MenpoWidgets Documentation

Release 0.2.1+0.g0c3dbe7.dirty

Joan Alabort-i-Medina, Epameinondas Antonakos, James Booth,

Jun 23, 2016

Contents

Welcome to the MenpoWidgets documentation!

MenpoWidgets is the Menpo Project's Python package for fancy visualization within the Jupyter notebook using interactive widgets. In the Menpo Project we take an opinionated stance that visualization is a key part of generating research. Therefore, we have tried to make the mental overhead of visualizing objects as low as possible. MenpoWidgets makes tasks like data exploration, model observation and results demonstration as simple as possible.

CHAPTER 1

API Documentation

In MenpoWidgets, we use legible docstrings, and therefore, all documentation should be easily accessible in any sensible IDE (or IPython) via tab completion. However, this section should make most of the core classes available for viewing online.

Main Widgets Functions for visualizing the various Menpo and MenpoFit objects using interactive widgets.

1.1 menpowidgets.base

Functions that can be used for visualizing the various Menpo objects using interactive widgets.

1.1.1 Shapes

visualize_pointclouds

menpowidgets.base.**visualize_pointclouds**(*pointclouds*, *figure_size=(10, 8)*, *style='coloured'*, *browser_style='buttons'*, *custom_info_callback=None*)

Widget that allows browsing through a *list* of *menpo.shape.PointCloud*, *menpo.shape.PointUndirectedGraph*, *menpo.shape.PointDirectedGraph*, *menpo.shape.PointTree*, *menpo.shape.TriMesh* or any subclass of those. Any instance of the above can be combined in the *list*.

The widget has options tabs regarding the renderer (lines, markers, numbering, zoom, axes) and saving the figure to file.

Parameters

- **pointclouds** (*list*) – The *list* of objects to be visualized. It can contain a combination of *menpo.shape.PointCloud*, *menpo.shape.PointUndirectedGraph*, *menpo.shape.PointDirectedGraph*, *menpo.shape.PointTree*, *menpo.shape.TriMesh* or subclasses of those.
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.
- **browser_style** (`{'buttons','slider'}`, optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.

- **custom_info_callback** (*function* or `None` , optional) – If not None, it should be a function that accepts a pointcloud and returns a list of custom messages to be printed per pointcloud. Each custom message will be printed in a separate line.

visualize_landmarkgroups

menpowidgets.base. **visualize_landmarkgroups** (*landmarkgroups*, *figure_size=(10, 8)*, *style='coloured'*, *browser_style='buttons'*, *custom_info_callback=None*)

Widget that allows browsing through a *list* of *menpo.landmark.LandmarkGroup* (or subclass) objects.

The landmark groups can have a combination of different attributes, e.g. different labels, number of points etc. The widget has options tabs regarding the landmarks, the renderer (lines, markers, numbering, legend, zoom, axes) and saving the figure to file.

Parameters

- **landmarkgroups** (*list* of *menpo.landmark.LandmarkGroup* or subclass) – The *list* of landmark groups to be visualized.
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.
- **browser_style** (`{'buttons','slider'}` , optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.
- **custom_info_callback** (*function* or `None` , optional) – If not None, it should be a function that accepts a landmark group and returns a list of custom messages to be printed per landmark group. Each custom message will be printed in a separate line.

visualize_landmarks

menpowidgets.base. **visualize_landmarks** (*landmarks*, *figure_size=(10, 8)*, *style='coloured'*, *browser_style='buttons'*, *custom_info_callback=None*)

Widget that allows browsing through a *list* of *menpo.landmark.LandmarkManager* (or subclass) objects.

The landmark managers can have a combination of different attributes, e.g. landmark groups and labels etc. The widget has options tabs regarding the landmarks, the renderer (lines, markers, numbering, legend, zoom, axes) and saving the figure to file.

Parameters

- **landmarks** (*list* of *menpo.landmark.LandmarkManager* or subclass) – The *list* of landmark managers to be visualized.
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.
- **browser_style** (`{'buttons','slider'}` , optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.

- **custom_info_callback** (*function* or `None` , optional) – If not None, it should be a function that accepts a landmark group and returns a list of custom messages to be printed per landmark group. Each custom message will be printed in a separate line.

1.1.2 Images

visualize_images

menpowidgets.base. **visualize_images** (*images*, *figure_size=(10, 8)*, *style='coloured'*, *browser_style='buttons'*, *custom_info_callback=None*)

Widget that allows browsing through a *list* of *menpo.image.Image* (or subclass) objects.

The images can have a combination of different attributes, e.g. masked or not, landmarked or not, without multiple landmark groups and labels etc. The widget has options tabs regarding the visualized channels, the landmarks, the renderer (lines, markers, numbering, legend, figure, axes) and saving the figure to file.

Parameters

- **images** (*list* of *menpo.image.Image* or subclass) – The *list* of images to be visualized.
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.
- **browser_style** (`{'buttons','slider'}` , optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.
- **custom_info_callback** (*function* or `None` , optional) – If not None, it should be a function that accepts an image and returns a list of custom messages to be printed per image. Each custom message will be printed in a separate line.

visualize_patches

menpowidgets.base. **visualize_patches** (*patches*, *patch_centers*, *figure_size=(10, 8)*, *style='coloured'*, *browser_style='buttons'*, *custom_info_callback=None*)

Widget that allows browsing through a *list* of patch-based images.

The patches argument can have any of the two formats that are returned from the *extract_patches()* and *extract_patches_around_landmarks()* methods of *menpo.image.Image*. Specifically it can be:

1. `(n_center,n_offset,self.n_channels,patch_shape)` *ndarray*
2. *list* of `n_center * n_offset` *menpo.image.Image* objects

The patches can have a combination of different attributes, e.g. number of centers, number of offsets, number of channels etc. The widget has options tabs regarding the visualized patches, channels, the renderer (lines, markers, numbering, figure, axes, image) and saving the figure to file.

Parameters

- **patches** (*list*) – The *list* of patch-based images to be visualized. It can consist of objects with any of the two formats that are returned from the *extract_patches()* and *extract_patches_around_landmarks()* methods. Specifically, it can either be an `(n_center,n_offset,self.n_channels,patch_shape)` *ndarray* or a *list* of `n_center * n_offset` *menpo.image.Image* objects.

- **patch_centers** (*list* of *menpo.shape.PointCloud*) – The centers to set the patches around. If the *list* has only one *menpo.shape.PointCloud* then this will be used for all patches members. Otherwise, it needs to have the same length as patches.
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.
- **browser_style** (`{'buttons','slider'}`, optional) – It defines whether the selector of the objects will have the form of plus/minus buttons or a slider.
- **custom_info_callback** (*function* or `None`, optional) – If not None, it should be a function that accepts an image and returns a list of custom messages to be printed per image. Each custom message will be printed in a separate line.

1.1.3 Models

visualize_shape_model

menpowidgets.base. **visualize_shape_model** (*shape_model*, *n_parameters=5*, *mode='multiple'*, *parameters_bounds=(-3.0, 3.0)*, *figure_size=(10, 8)*, *style='coloured'*)

Widget that allows the dynamic visualization of a multi-scale linear statistical shape model.

Parameters

- **shape_model** (*list* of *menpo.shape.PCAModel* or *subclass*) – The multi-scale shape model to be visualized. Note that each level can have different number of components.
- **n_parameters** (*int* or *list* of *int* or `None`, optional) – The number of principal components to be used for the parameters sliders. If *int*, then the number of sliders per level is the minimum between *n_parameters* and the number of active components per level. If *list* of *int*, then a number of sliders is defined per level. If `None`, all the active components per level will have a slider.
- **mode** (`{'single','multiple'}`, optional) – If `'single'`, then only a single slider is constructed along with a drop down menu. If `'multiple'`, then a slider is constructed for each parameter.
- **parameters_bounds** ((*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.
- **figure_size** ((*int*, *int*), optional) – The size of the plotted figures.
- **style** (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.

visualize_appearance_model

menpowidgets.base. **visualize_appearance_model** (*appearance_model*, *n_parameters=5*, *mode='multiple'*, *parameters_bounds=(-3.0, 3.0)*, *figure_size=(10, 8)*, *style='coloured'*)

Widget that allows the dynamic visualization of a multi-scale linear statistical appearance model.

Parameters

- **appearance_model** (*list* of *menpo.model.PCAModel* or subclass) – The multi-scale appearance model to be visualized. Note that each level can have different number of components.
- **n_parameters** (*int* or *list* of *int* or `None` , optional) – The number of principal components to be used for the parameters sliders. If *int*, then the number of sliders per level is the minimum between *n_parameters* and the number of active components per level. If *list* of *int*, then a number of sliders is defined per level. If `None` , all the active components per level will have a slider.
- **mode** (`{'single','multiple'}` , optional) – If `'single'` , then only a single slider is constructed along with a drop down menu. If `'multiple'` , then a slider is constructed for each parameter.
- **parameters_bounds** ((*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.
- **figure_size** ((*int*, *int*), optional) – The size of the plotted figures.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.

visualize_patch_appearance_model

menpowidgets.base. **visualize_patch_appearance_model** (*appearance_model*, *centers*, *n_parameters=5*, *mode='multiple'*, *parameters_bounds=(-3.0, 3.0)*, *figure_size=(10, 8)*, *style='coloured'*)

Widget that allows the dynamic visualization of a multi-scale linear statistical patch-based appearance model.

Parameters

- **appearance_model** (*list* of *menpo.model.PCAModel* or subclass) – The multi-scale patch-based appearance model to be visualized. Note that each level can have different number of components.
- **centers** (*list* of *menpo.shape.PointCloud* or subclass) – The centers to set the patches around. If the *list* has only one *menpo.shape.PointCloud* then this will be used for all appearance model levels. Otherwise, it needs to have the same length as *appearance_model*.
- **n_parameters** (*int* or *list* of *int* or `None` , optional) – The number of principal components to be used for the parameters sliders. If *int*, then the number of sliders per level is the minimum between *n_parameters* and the number of active components per level. If *list* of *int*, then a number of sliders is defined per level. If `None` , all the active components per level will have a slider.
- **mode** (`{'single','multiple'}` , optional) – If `'single'` , then only a single slider is constructed along with a drop down menu. If `'multiple'` , then a slider is constructed for each parameter.
- **parameters_bounds** ((*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.
- **figure_size** ((*int*, *int*), optional) – The size of the plotted figures.

- **style** (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.

1.1.4 Various

webcam_widget

menpowidgets.base. **webcam_widget** (*canvas_width=640*, *hd=True*, *n_preview_windows=5*, *style='coloured'*)

Webcam widget for taking snapshots. The snapshots are dynamically previewed in a FIFO stack of thumbnails.

Parameters

- **canvas_width** (*int*, optional) – The initial width of the rendered canvas. Note that this doesn't actually change the webcam resolution. It simply rescales the rendered image, as well as the size of the returned screenshots.
- **hd** (*bool*, optional) – If `True`, then the webcam will be set to high definition (HD), i.e. 720 x 1280. Otherwise the default resolution will be used.
- **n_preview_windows** (*int*, optional) – The number of preview thumbnails that will be used as a FIFO stack to show the captured screenshots. It must be at least 4.
- **style** (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.

Returnssnapshots (*list* of *menpo.image.Image*) – The list of captured images.

features_selection

menpowidgets.base. **features_selection** (*style='coloured'*)

Widget that allows selecting a features function and its options. The widget supports all features from *menpo.feature* and has a preview tab. It returns a *list* of length 1 with the selected features function closure.

Parametersstyle (`{'coloured','minimal'}`, optional) – If `'coloured'`, then the style of the widget will be coloured. If `minimal`, then the style is simple using black and white colours.

Returns

features_function (*list* of length `1`) – The function closure of the features function using *functools.partial*. So the function can be called as:

```
features_image = features_function[0](image)
```

plot_graph

menpowidgets.base. **plot_graph** (*x_axis*, *y_axis*, *legend_entries=None*, *figure_size=(10, 6)*, *style='coloured'*)

Widget that allows plotting various curves in a graph.

The widget has options tabs regarding the graph and the renderer (lines, markers, legend, figure, axes, grid) and saving the figure to file.

Parameters

- **x_axis** (*list* of *float*) – The values of the horizontal axis. Note that these values are common for all the curves.
- **y_axis** (*list* of *lists* of *float*) – A *list* that stores a *list* of values to be plotted for each curve.
- **legend_entries** (*list* or *str* or `None` , optional) – The *list* of names that will appear on the legend for each curve. If `None` , then the names format is `curve {}.format(i)` .
- **figure_size** ((*int*, *int*), optional) – The initial size of the rendered figure.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.

save_matplotlib_figure

`menpowidgets.base.` **save_matplotlib_figure** (*renderer*, *style='coloured'*)

Widget that allows to save a figure, which was generated with Matplotlib, to file.

Parameters

- **renderer** (*menpo.visualize.viewmatplotlib.MatplotlibRenderer*) – The Matplotlib renderer object.
- **style** (`{'coloured','minimal'}` , optional) – If `'coloured'` , then the style of the widget will be coloured. If `minimal` , then the style is simple using black and white colours.

1.2 menpowidgets.menpofit.base

Functions that can be used for visualizing the various MenpoFit objects using interactive widgets.

1.2.1 Active Appearance Model

visualize_aam

visualize_patch_aam

1.2.2 Active Template Model

visualize_atm

visualize_patch_atm

1.2.3 Constrained Local Model

visualize_clm

visualize_expert_ensemble

1.2.4 Fitting Result

visualize_fitting_result

plot_ced

Options Widgets Independent widget objects that can be used as the main components for designing high-level widget functions.

1.3 menpowidgets.options

1.3.1 Options

Independent widget classes that can be used as the main components for designing high-level widget functions, as the ones in *menpowidgets.base* and *menpowidgets.menpofit.base*.

AnimationOptionsWidget

class menpowidgets.options. **AnimationOptionsWidget** (*index*, *render_function=None*, *index_style='buttons'*, *interval=0.2*, *interval_step=0.05*, *description='Index: '*, *loop_enabled=True*, *style='minimal'*, *continuous_update=False*)

Bases: `MenpoWidget`

Creates a widget for animating through a list of objects. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*ToggleButton*	*play_stop_toggle*	The play/stop button
2	*Button*	*fast_forward_button*	Increase speed
3	*Button*	*fast_backward_button*	Decrease speed
4	*ToggleButton*	*loop_toggle*	Repeat mode
5	*HBox*	*animation_box*	Contains 1, 2, 3, 4
8	*IndexButtonsWidget* *IndexSliderWidget*	*index_wid*	The index selector

Note that:

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* and *predefined_style()* methods.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **index** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `min` : (*int*) The minimum value (e.g. `0`).
  - `max` : (*int*) The maximum value (e.g. `100`).
  - `step` : (*int*) The index step (e.g. `1`).
  - `index` : (*int*) The index value (e.g. `10`).
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute
  - `name` : the name of the modified trait attribute.

  If `None` , then nothing is assigned.
- **index_style** (`{'buttons','slider'}` , optional) – If `'buttons'` , then *IndexButtonsWidget* class is called. If `'slider'` , then *IndexSliderWidget* class is called.
- **interval** (*float*, optional) – The interval between the animation progress in seconds.
- **interval_step** (*float*, optional) – The interval step (in seconds) that is applied when fast forward/backward buttons are pressed.
- **description** (*str*, optional) – The title of the widget.
- **loop_enabled** (*bool*, optional) – If `True` , then after reach the minimum (maximum) index values, the counting will continue from the end (beginning). If `False` , the counting will stop at the minimum (maximum) value.

- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **continuous_update** (*bool*, optional) – If `True` and *index_style* is set to `'slider'`, then the render and update functions are called while moving the slider's handle. If `False`, then the the functions are called only when the handle (mouse click) is released.

Example

Let's create an animation widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import AnimationOptionsWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected index:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Selected index: {}".format(wid.selected_values)
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> index = {'min': 0, 'max': 100, 'step': 1, 'index': 10}
>>> wid = AnimationOptionsWidget(index, index_style='buttons',
>>>                              render_function=render_function,
>>>                              style='info')
>>> wid
```

By pressing the buttons (or simply pressing the Play button), the printed message gets updated. Finally, let's change the widget status with a new dictionary of options:

```
>>> new_options = {'min': 0, 'max': 20, 'step': 2, 'index': 16}
>>> wid.set_widget_state(new_options, allow_callback=False)
```

---

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*)
Function that sets a predefined style on the widget.

Parameters**style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature

can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

set_widget_state (*index*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *index* values are different than *self.selected_values*.

Parameters

- **index** (*dict*) – The selected options. It must be a *dict* with the following keys:
  - `min` : (*int*) The minimum value (e.g. `0`).
  - `max` : (*int*) The maximum value (e.g. `100`).
  - `step` : (*int*) The index step (e.g. `1`).
  - `index` : (*int*) The index value (e.g. `10`).
- **allow_callback** (*bool*, optional) – If `True`, it allows triggering of any callback functions.

stop_animation ( )

Method that stops an active annotation by setting `self.play_stop_toggle.value = False`.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

CameraSnapshotWidget

class menpowidgets.options. **CameraSnapshotWidget** (*canvas_width=640*, *hd=True*, *n_preview_windows=5*, *preview_windows_margin=3*, *render_function=None*, *style='minimal'*, *preview_style='minimal'*)

Bases: *MenpoWidget*

Creates a webcam widget for taking screenshots. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*CameraWidget*	*camera_wid*	The webcam widget
2	*Latex*	*n_snapshots_text*	Number of snapshots
3	*Button*	*snapshot_but*	Take snapshot button
4	*VBox*	*snapshot_box*	Contains 3, 2
5	*Button*	*close_but*	Close widget button
8	*ZoomOneScaleWidget*	*zoom_widget*	Resolution controller
9	*HBox*	*buttons_box*	Contains 3, 5, 8
10	*Image*	*preview_children*	List of preview images
11	*HBox*	*preview*	Contains all 10

Note that:

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the `style()` and `predefined_style()` methods.
- To update the handler callback function of the widget, please refer to the `replace_render_function()` method.

Parameters

- **`canvas_width`** (*int*, optional) – The initial width of the rendered canvas. Note that this doesn't actually change the webcam resolution. It simply rescales the rendered image, as well as the size of the returned screenshots.
- **`hd`** (*bool*, optional) – If `True` , then the webcam will be set to high definition (HD), i.e. 720 x 1280. Otherwise the default resolution will be used.
- **`n_preview_windows`** (*int*, optional) – The number of preview thumbnails that will be used as a FIFO stack to show the captured screenshots. It must be at least 4.
- **`preview_windows_margin`** (*int*, optional) – The margin between the preview thumbnails in pixels.
- **`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute
  - `name` : the name of the modified trait attribute.

  If `None` , then nothing is assigned.
- **`style`** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **`preview_style`** (*str* (see below), optional) – Sets a predefined style at the widget's preview box. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create a webcam widget. Firstly, we need to import it:

```
>>> from menpowidgets.options import CameraSnapshotWidget
```

Create the widget with some initial options and display it:

```
>>> wid = CameraSnapshotWidget(canvas_width=640, hd=True,
>>>                            n_preview_windows=5,
>>>                            preview_windows_margin=1, style='info')
>>> wid
```

By pressing the "Take snapshot" button, the snapshots appear in the thumbnails below the stream. The video stream can be interrupted by pressing the "Close" button.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type :`

the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*, *preview_style='minimal'*)

Function that sets a predefined style on the widget.

Parameters

- **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **preview_style** (*str* (see below)) – Preview box style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding='0.2cm'*, *margin=0*, *preview_box_style=None*, *preview_border_visible=True*, *preview_border_colour='black'*, *preview_border_style='solid'*, *preview_border_width=1*, *preview_border_radius=1*, *preview_padding=0*, *preview_margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **preview_box_style** (*str* or `None` (see below), optional) – Possible tab widgets style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **preview_border_visible** (*bool*, optional) – Defines whether to draw the border line around the preview.
- **preview_border_colour** (*str*, optional) – The color of the border around the preview.
- **preview_border_style** (*str*, optional) – The line style of the border around the preview.
- **preview_border_width** (*float*, optional) – The line width of the border around the preview.
- **preview_border_radius** (*float*, optional) – The radius of the corners of the box of the preview.
- **preview_padding** (*float*, optional) – The padding around the preview box.
- **preview_margin** (*float*, optional) – The margin around the preview box.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed

to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ChannelOptionsWidget

class menpowidgets.options. **ChannelOptionsWidget** (*n_channels*, *image_is_masked*, *render_function=None*, *style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting channel options for rendering an image. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*SlicingCommandWidget*	*channels_wid*	The channels selector
2	*Checkbox*	*masked_checkbox*	Controls masked mode
3	*Checkbox*	*rgb_checkbox*	View as RGB
4	*Checkbox*	*sum_checkbox*	View sum of channels
5	*Checkbox*	*glyph_checkbox*	View glyph
6	*BoundedIntText*	*glyph_block_size_text*	Glyph block size
7	*Checkbox*	*glyph_use_neg_checkbox*	Use negative values
8	*Latex*	*no_options_latex*	No options message
9	*VBox*	*glyph_options_box*	Contains 6, 7
10	*HBox*	*glyph_box*	Contains 5, 9
11	*HBox*	*rgb_masked_options_box*	Contains 3, 2
12	*HBox*	*glyph_sum_options_box*	Contains 4, 10
13	*VBox*	*checkboxes_box*	Contains 11, 12

Note that:

- To update the state of the widget, please refer to the *set_widget_state()* method.
- The widget has **memory** about the properties of the objects that are passed into it through *set_widget_state()* . Each image object has a unique key id assigned through *get_key()* . Then, the options that correspond to each key are stored in the `self.default_options` *dict*.
- The selected values of the current image object are stored in the `self.selected_values` *trait*. It is a *dict* with the following keys:
  - `channels` : (*list*) The selected channels.
  - `glyph_enabled` : (*bool*) Whether to render as glyph.

- `glyph_block_size` : (*int*) The glyph's block size.
- `glyph_use_negative` : (*bool*) Whether to use negative values in glyph
- `sum_enabled` : (*bool*) Whether to render as sum of channels.
- `masked_enabled` : (*bool*) Whether to render as masked.

•When an unseen image object is passed in (i.e. a key that is not included in the `self.default_options` *dict*), it gets the following initial options by default:

- `channels = [0] if n_channels == 3 else None`
- `glyph_enabled = False`
- `glyph_block_size = 3`
- `glyph_use_negative = False`
- `sum_enabled = False`
- `masked_enabled = image_is_masked`

•To set the styling of this widget please refer to the *`style()`* and *`predefined_style()`* methods.

•To update the handler callback function of the widget, please refer to the *`replace_render_function()`* method.

Parameters

•**`n_channels`** (*int*) – The number of channels of the initial image object.

•**`image_is_masked`** (*bool*) – Whether the initial image object is masked or not. If `True` , then the image is assumed to be a *menpo.image.MaskedImage* object.

•**`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:

- `type` : The type of notification (normally `'change'`).
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.

If `None` , then nothing is assigned.

•**`style`** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create a channels widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import ChannelOptionsWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected channels and masked flag:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Channels: {}, Masked: {}, Glyph: {}, Sum: {}".format(
>>>         wid.selected_values['channels'],
>>>         wid.selected_values['masked_enabled'],
>>>         wid.selected_values['glyph_enabled'],
>>>         wid.selected_values['sum_enabled'])
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> wid = ChannelOptionsWidget(n_channels=30, image_is_masked=True,
>>>                            render_function=render_function,
>>>                            style='warning')
>>> wid
```

By playing around with the widget, printed message gets updated. Finally, let's change the widget status with a new object:

```
>>> wid.set_widget_state(n_channels=10, image_is_masked=False,
>>>                      allow_callback=False)
```

Remember that the widget is **mnemonic**, i.e. it remembers the objects it has seen and their corresponding options. These can be retrieved as:

```
>>> wid.default_options
```

---

add_callbacks ( )

Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

get_default_options (*n_channels*, *image_is_masked*)
Function that returns a *dict* with default options given the properties of an image object, i.e. *n_channels* and *image_is_masked*. The function returns the *dict* of options but also updates the `self.default_options` *dict*.

Parameters
- **n_channels** (*int*) – The number of channels.
- **image_is_masked** (*bool*) – Whether the image object is masked or not. If `True` , then the image is assumed to be a *menpo.image.MaskedImage* object.

Returns

default_options (*dict*) – A *dict* with the default options. It contains:
- `channels` : (*list*) The selected channels.
- `glyph_enabled` : (*bool*) Whether to render as glyph.
- `glyph_block_size` : (*int*) The glyph's block size.
- `glyph_use_negative` : (*bool*) Whether to use negative values.
- `sum_enabled` : (*bool*) Whether to render as sum of channels.
- `masked_enabled` : (*bool*) Whether to render as masked.

If the object is not seen before by the widget, then it automatically gets the following default options:
- `channels = [0] if n_channels == 3 else None`
- `glyph_enabled = False`
- `glyph_block_size = 3`
- `glyph_use_negative = False`
- `sum_enabled = False`
- `masked_enabled = image_is_masked`

get_key (*n_channels*, *image_is_masked*)
Function that returns a unique key based on the properties of the provided image object.

Parameters
- **n_channels** (*int*) – The number of channels.
- **image_is_masked** (*bool*) – Whether the image object is masked or not. If `True` , then the image is assumed to be a *menpo.image.MaskedImage* object.

Returnskey (*str*) – The key that has the format `'{n_channels}_{image_is_masked}'` .

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters
- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` `is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'.

In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*)

Function that sets a predefined style on the widget.

Parameters **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )

Function that removes all the internal handler callback functions.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_visibility ( )

Function that sets the visibility of the various components of the widget, depending on the properties of the current image object, i.e. `self.n_channels` and `self.image_is_masked`.

set_widget_state (*n_channels*, *image_is_masked*, *allow_callback=True*)

Method that updates the state of the widget, if the key generated with *get_key()* based on the provided *n_channels* and *image_is_masked* is different than the current key based on `self.n_channels` and `self.image_is_masked`.

Parameters

- **n_channels** (*int*) – The number of channels of the initial image object.
- **image_is_masked** (*bool*) – Whether the initial image object is masked or not. If `True` , then the image is assumed to be a *menpo.image.MaskedImage* object.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *slider_width='4cm'*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **slider_width** (*str*, optional) – The width of the slider.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.

•**type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

FeatureOptionsWidget

class menpowidgets.options. **FeatureOptionsWidget** (*style='minimal'*)
Bases: FlexBox

Creates a widget for selecting feature options. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*RadioButtons*	*feature_radiobuttons*	Feature type selector
2	*DSIFTOptionsWidget*	*dsift_options_widget*	DSIFT options
3	*HOGOptionsWidget*	*hog_options_widget*	HOG options
4	*IGOOptionsWidget*	*igo_options_widget*	IGO options
5	*LBPOptionsWidget*	*lbp_options_widget*	LBP options
6	*DaisyOptionsWidget*	*daisy_options_widget*	Daisy options
7	*Latex*	*no_options_widget*	No options available
8	*Box*	*per_feature_options_box*	Contains 2 - 7
9	*Image*	*preview_image*	Contains 6, 7
10	*Latex*	*preview_input_latex*	Contains 5, 9
11	*Latex*	*preview_output_latex*	Contains 3, 2
12	*Latex*	*preview_time_latex*	Contains 4, 10
13	*VBox*	*preview_box*	Contains 9 - 12
14	*Tab*	*options_box*	Contains 1, 8, 13

Note that:

•To set the styling please refer to the *style()* and *predefined_style()* methods.

•The widget stores the features *function* to self.features_function , the features options *dict* in self.features_options and the *partial* function with the options as self.function .

Parameters
style (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
'minimal'	Simple black and white style
'success'	Green-based style
'info'	Blue-based style
'warning'	Yellow-based style
'danger'	Red-based style
''	No style

predefined_style (*style*)
Function that sets a predefined style on the widget.
Parameters
style (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

LandmarkOptionsWidget

class menpowidgets.options. **LandmarkOptionsWidget** (*group_keys*, *labels_keys*, *render_function=None*, *renderer_widget=None*, *style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting landmark options. The widget consists of the following objects from *ipywidgets*:

No	Object	Property (*self.*)	Description
1	*Latex*	*no_landmarks_msg*	No landmarks available msg.
2	*Checkbox*	*render_landmarks_checkbox*	Render landmarks checkbox
3	*Latex*	*group_description*	Landmark group title
4	*IntSlider*	*group_slider*	Landmark group selector
5	*Dropdown*	*group_dropdown*	Landmark group selector
6	*Latex*	*group_latex*	Landmark group text
7	*HBox*	*group_selection_box*	Contains 3, 4, 5, 6
8	*Latex*	*labels_text*	Labels title
9	*ToggleButtons*	*labels_toggles*	list with the labels per group
10	*HBox*	*labels_box*	Contains all 9
11	*HBox*	*labels_and_text_box*	Contains 8 and 10
12	*VBox*	*options_box*	Contains 7, 11
13	*HBox*	*render_and_options_box*	Contains 2, 12

Note that:

- To update the state of the widget, please refer to the `set_widget_state()` method.
- The widget has **memory** about the properties of the objects that are passed into it through `set_widget_state()`. Each landmarks object has a unique key id assigned through `get_key()`. Then, the options that correspond to each key are stored in the `self.default_options` *dict*.
- The selected values of the current landmarks object are stored in the `self.selected_values` *trait*. It is a *dict* with the following keys:
  - `group` : (*str* or `None`) The selected group.
  - `with_labels` : (*list* or `None`) The selected labels.
  - `render_landmarks` : (*bool*) Whether to render the landmarks.
- When an unseen landmarks object is passed in (i.e. a key that is not included in the `self.default_options` *dict*), it gets the following initial options by default:
  - `group = None if group_keys is None else group_keys[0]`
  - `with_labels = None if group_keys is None else labels_keys[0]`
  - `render_landmarks = False if group_keys is None else True`
- To set the styling of this widget please refer to the `style()` and `predefined_style()` methods.
- To update the handler callback function of the widget, please refer to the `replace_render_function()` method.

Parameters

- **`group_keys`** (*list* of *str* or `None`) – The *list* of landmark groups. If `None`, then no landmark groups are available.
- **`labels_keys`** (*list* of *list* of *str* or `None`) – The *list* of labels per landmark group. If `None`, then no labels are available.
- **`render_function`** (*callable* or `None`, optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance

– `old` : the old value of the modified trait attribute

– `new` : the new value of the modified trait attribute

– `name` : the name of the modified trait attribute.

If `None` , then nothing is assigned.

- **renderer_widget** (*RendererOptionsWidget* or `None` , optional) – The *RendererOptionsWidget* that is created and needs to be linked with this widget. If `None` , then nothing is assigned.
- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create a landmarks widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import LandmarkOptionsWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected index:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Group: {}, Labels: {}".format(
>>>         wid.selected_values['group'],
>>>         wid.selected_values['with_labels'])
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> wid = LandmarkOptionsWidget(
>>>             group_keys=['PTS', 'ibug_face_68'],
>>>             labels_keys=[['all'], ['jaw', 'eye', 'mouth']],
>>>             render_function=render_function, style='danger')
>>> wid
```

By playing around with the widget, the printed message gets updated. Finally, let's change the widget status with a new set of options:

```
>>> wid.set_widget_state(group_keys=['new_group'],
>>>                      labels_keys=[['1', '2', '3']],
>>>                      allow_callback=False)
```

Remember that the widget is **mnemonic**, i.e. it remembers the objects it has seen and their corresponding options. These can be retrieved as:

```
>>> wid.default_options
```

add_callbacks ( )
Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

get_default_options (*group_keys*, *labels_keys*)
Function that returns a *dict* with default options based on the properties of the provided landmarks. The function returns the *dict* of options but also updates the `self.default_options` *dict*.

Parameters
- **group_keys** (*list* of *str* or `None`) – The *list* of landmark groups. If `None` , then no landmark groups are available.
- **labels_keys** (*list* of *list* of *str* or `None`) – The *list* of labels per landmark group. If `None` , then no labels are available.

Returns

default_options (*dict*) – A *dict* with the default options. It contains:
- `group` : (*str* or `None`) The selected group.
- `with_labels` : (*list* or `None`) The selected labels.
- `render_landmarks` : (*bool*) Whether to render the landmarks.

If the object is not seen before by the widget, then it automatically gets the following default options:
- `group = None if group_keys is None else group_keys[0]`
- `with_labels = None if group_keys is None else labels_keys[0]`
- `render_landmarks = False if group_keys is None else True`

get_key (*group_keys*, *labels_keys*)
Function that returns a unique key based on the properties of the provided landmarks.

Parameters

- **group_keys** (*list* of *str* or `None`) – The *list* of landmark groups. If `None`, then no landmark groups are available.
- **labels_keys** (*list* of *list* of *str* or `None`) – The *list* of labels per landmark group. If `None`, then no labels are available.

Returns

key (*str*) – The key that has the format `'{group_keys}_{labels_keys}'`.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change``` is a dictionary. The change dictionary at least holds a `'type'` key. * ``type`: the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner`: the HasTraits instance * `old`: the old value of the modified trait attribute * `new`: the new value of the modified trait attribute * `name`: the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*)

Function that sets a predefined style on the widget.

Parameters

style (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )

Function that removes all the internal handler callback functions.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters

render_function (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner`: the *HasTraits* instance
- `old`: the old value of the modified trait attribute
- `new`: the new value of the modified trait attribute
- `name`: the name of the modified trait attribute.
- `type`: `'change'`

If `None` , then nothing is added.

set_visibility ( )
Function that sets the visibility of the various components of the widget, depending on the properties of the current landmarks, i.e. `self.group_keys` .

set_widget_state (*group_keys*, *labels_keys*, *allow_callback=True*)
Method that updates the state of the widget, if the key generated with *get_key()* based on the provided *group_keys* and *labels_keys* is different than the current key based on `self.group_keys` and `self.labels_keys` .

Parameters

- **group_keys** (*list* of *str* or `None`) – The *list* of landmark groups. If `None` , then no landmark groups are available.
- **labels_keys** (*list* of *list* of *str* or `None`) – The *list* of labels per landmark group. If `None` , then no labels are available.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *labels_buttons_style=''*)
Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **labels_buttons_style** (*str* or `None` (see below), optional) – Style options: ‘success’, ‘info’, ‘warning’, ‘danger’, ‘primary’, ‘’, None

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. The dictionary is keyed on the name and the values are the

TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)

Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

LinearModelParametersWidget

class menpowidgets.options. **LinearModelParametersWidget** (*n_parameters*, *render_function=None*, *mode='multiple'*, *params_str=''*, *params_bounds=(-3.0, 3.0)*, *params_step=0.1*, *plot_variance_visible=True*, *plot_variance_function=None*, *animation_visible=True*, *loop_enabled=False*, *interval=0.0*, *interval_step=0.05*, *animation_step=0.5*, *style='minimal'*, *continuous_update=False*)

Bases: *MenpoWidget*

Creates a widget for selecting parameters values when visualizing a linear model (e.g. PCA model). The widget has options for animating through various parameters values. It consists of the following objects from *ipywidgets*:

No	Object	Property (*self.*)	Description
1	*Button*	*plot_button*	The plot variance button
2	*Button*	*reset_button*	The reset button
3	*HBox*	*plot_and_reset*	Contains 1, 2
4	*ToggleButton*	*play_stop_toggle*	The play/stop button
5	*Button*	*fast_forward_button*	Increase speed
6	*Button*	*fast_backward_button*	Decrease speed
7	*ToggleButton*	*loop_toggle*	Repeat mode
8	*HBox*	*animation_buttons*	Contains 4, 5, 6, 7
9	*HBox*	*buttons_box*	Contains 3, 8

If `mode = 'single'` , then:

No	Object	Property (*self.*)	Description
4	*FloatSlider*	*slider*	The parameter value slider
5	*Dropdown*	*dropdown_params*	The parameter selector
6	*HBox*	*parameters_wid*	Contains 4, 5

If `mode = 'multiple'` , then:

No	Object	Property (*self.*)	Description
7	*FloatSlider*	*sliders*	*list* of all sliders
8	*VBox*	*parameters_wid*	Contains all 7

Note that:

- To update the state of the widget, please refer to the *`set_widget_state()`* method.
- The selected values are stored in the `self.selected_values` *trait* which is a *list*.
- To set the styling of this widget please refer to the *`style()`* and *`predefined_style()`* methods.
- To update the handler callback functions of the widget, please refer to the *`replace_render_function()`* and *`replace_variance_function()`* methods.

Parameters

- **`n_parameters`** (*int*) – The *list* of initial parameters values.
- **`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute
  - `name` : the name of the modified trait attribute.

  If `None` , then nothing is assigned.
- **`mode`** (`{'single','multiple'}` , optional) – If `'single'` , only a single slider is constructed along with a dropdown menu that allows the parameter selection. If `'multiple'` , a slider is constructed for each parameter.
- **`params_str`** (*str*, optional) – The string that will be used as description of the slider(s). The final description has the form `"{}{}".format(params_str,p)` , where `p` is the parameter number.

- **params_bounds** ((*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.
- **params_step** (*float*, optional) – The step, in std units, of the sliders.
- **plot_variance_visible** (*bool*, optional) – Defines whether the button for plotting the variance will be visible upon construction.
- **plot_variance_function** (*callable* or `None` , optional) – The plot function that is executed when the plot variance button is clicked. If `None` , then nothing is assigned.
- **animation_visible** (*bool*, optional) – Defines whether the animation options will be visible.
- **loop_enabled** (*bool*, optional) – If `True` , then the repeat mode of the animation is enabled.
- **interval** (*float*, optional) – The interval between the animation progress in seconds.
- **interval_step** (*float*, optional) – The interval step (in seconds) that is applied when fast forward/backward buttons are pressed.
- **animation_step** (*float*, optional) – The parameters step that is applied when animation is enabled.
- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **continuous_update** (*bool*, optional) – If `True` , then the render function is called while moving a slider's handle. If `False` , then the the function is called only when the handle (mouse click) is released.

Example

Let's create a linear model parameters values widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import LinearModelParametersWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected parameters:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Selected parameters: {}".format(wid.selected_values)
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> wid = LinearModelParametersWidget(n_parameters=5,
>>>                                   render_function=render_function,
>>>                                   params_str='Parameter ',
```

```
>>>                                     mode='multiple',
>>>                                     params_bounds=(-3., 3.),
>>>                                     plot_variance_visible=True,
>>>                                     style='info')
>>> wid
```

By moving the sliders, the printed message gets updated. Finally, let's change the widget status with a new set of options:

```
>>> wid.set_widget_state(n_parameters=10, params_str='',
>>>                      params_step=0.1, params_bounds=(-10, 10),
>>>                      plot_variance_visible=False,
>>>                      allow_callback=True)
```

---

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

add_variance_function (*variance_function*)

Method that adds a *variance_function()* to the *Variance* button of the widget. The given function is also stored in *self._variance_function*.

Parameters **variance_function** (*callable* or `None` , optional) – The variance function that behaves as a callback. If `None` , then nothing is added.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style(*style*)

Function that sets a predefined style on the widget.

Parameters **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_render_function( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

remove_variance_function( )

Method that removes the current *self._variance_function()* from the *Variance* button of the widget and sets `self._variance_function = None`.

replace_render_function(*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

replace_variance_function(*variance_function*)

Method that replaces the current *self._variance_function()* of the *Variance* button of the widget with the given *variance_function()*.

Parameters **variance_function** (*callable* or `None`, optional) – The variance function that behaves as a callback. If `None`, then nothing happens.

set_widget_state(*n_parameters=None*, *params_str=None*, *params_bounds=None*, *params_step=None*, *plot_variance_visible=True*, *animation_step=0.5*, *allow_callback=True*)

Method that updates the state of the widget with a new set of options.

Parameters

- **n_parameters** (*int*) – The *list* of initial parameters values.
- **params_str** (*str*, optional) – The string that will be used as description of the slider(s). The final description has the form `"{}{}".format(params_str,p)` , where `p` is the parameter number.
- **params_bounds** ((*float*, *float*), optional) – The minimum and maximum bounds, in std units, for the sliders.
- **params_step** (*float*, optional) – The step, in std units, of the sliders.
- **plot_variance_visible** (*bool*, optional) – Defines whether the button for plotting the variance will be visible upon construction.
- **animation_step** (*float*, optional) – The parameters step that is applied when animation is enabled.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *slider_width=''*, *slider_handle_colour=None*, *slider_bar_colour=None*, *buttons_style=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **slider_width** (*str*, optional) – The width of the slider(s).
- **slider_handle_colour** (*str*, optional) – The colour of the handle(s) of the slider(s).
- **slider_bar_colour** (*str*, optional) – The bar colour of the slider(s).
- **buttons_style** (*str* or `None` (see below), optional) – Style options:
  'success', 'info', 'warning', 'danger', 'primary', '', None

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

PatchOptionsWidget

class menpowidgets.options. **PatchOptionsWidget** (*n_patches*, *n_offsets*, *render_function=None*, *style='minimal'*, *subwidgets_style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting patches options when rendering a patch-based image. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*Dropdown*	*offset_dropdown*	Offset index
2	*Checkbox*	*render_centers_checkbox*	Render centers flag
3	*Checkbox*	*render_patches_checkbox*	Render patches flag
4	*ToggleButton*	*background_toggle*	Background colour
5	*Latex*	*background_title*	Background title
6	*SlicingCommandWidget*	*slicing_wid*	Patch index selector
7	*LineOptionsWidget*	*bboxes_line_options_wid*	Bboxes options
8	*HBox*	*background_box*	Contains 5, 4
9	*Box*	*render_checkboxes_box*	Contains 2, 3
10	*HBox*	*render_box*	Contains 8, 9
11	*VBox*	*offset_patches_box*	Contains 6, 1, 10

Note that:

- To update the state of the widget, please refer to the *set_widget_state()* method.
- The widget has **memory** about the properties of the objects that are passed into it through *set_widget_state()* . Each patches object has a unique key id assigned through *get_key()*

. Then, the options that correspond to each key are stored in the `self.default_options` *dict*.

- The selected values of the current patches object are stored in the `self.selected_values` *trait*. It is a *dict* with the following keys:
  - `patches_indices` : (*list* or *int*) The selected patches (e.g. `list(range(n_patches))`).
  - `offset_index` : (*int*) The selected offset
  - `background` : (*str*) The background colour (e.g. `'white'`).
  - `render_patches` : (*bool*) Whether to render the patches.
  - `render_patches_bboxes` : (*bool*) Whether to render boxes around the patches.
  - `bboxes_line_colour` : (*list*) The boxes line colour (e.g. `['red']`)
  - `bboxes_line_style` : (*str*) The boxes line style (e.g. `'-'`).
  - `bboxes_line_width` : (*float*) The boxes line width (e.g. `1`).
  - `render_centers` : (*bool*) Whether to render the patches' centers.
- When an unseen patches object is passed in (i.e. a key that is not included in the `self.default_options` *dict*), it gets the following initial options by default:
  - `patches_indices = list(range(n_patches))`
  - `offset_index = 0`
  - `background = 'white'`
  - `render_patches = True`
  - `render_patches_bboxes = True`
  - `bboxes_line_colour = ['red']`
  - `bboxes_line_style = '-'`
  - `bboxes_line_width = 1`
  - `render_centers = True`
- To set the styling of this widget please refer to the *`style()`* and *`predefined_style()`* methods.
- To update the handler callback function of the widget, please refer to the *`replace_render_function()`* method.

Parameters

- **`n_patches`** (*int*) – The number of patches of the initial object.
- **`n_offsets`** (*int*) – The number of offsets of the initial object.
- **`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute

– `name` : the name of the modified trait attribute.

If `None` , then nothing is assigned.

- **`style`** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **`subwidgets_style`** (*str* (see below), optional) – Sets a predefined style at the widget's patches and bboxes options. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create a patches widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import PatchOptionsWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected patches and bboxes flag:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Patches: {}, BBoxes: {}".format(
>>>         wid.selected_values['patches']['indices'],
>>>         wid.selected_values['bboxes']['render_lines'])
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> wid = PatchOptionsWidget(n_patches=68, n_offsets=5,
>>>                          render_function=render_function,
>>>                          style='info', subwidgets_style='danger')
>>> wid
```

By playing around with the widget, printed message gets updated. Finally, let's change the widget status with a new set of options:

```
>>> wid.set_widget_state(n_patches=49, n_offsets=1, allow_callback=False)
```

Remember that the widget is **mnemonic**, i.e. it remembers the objects it has seen and their corresponding options. These can be retrieved as:

```
>>> wid.default_options
```

---

add_callbacks ( )

Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parametersrender_function (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

get_default_options (*n_patches*, *n_offsets*)

Function that returns a *dict* with default options given the properties of a patches object, i.e. *n_patches* and *n_offsets*. The function returns the *dict* of options but also updates the `self.default_options` *dict*.

Parameters

- **n_patches** (*int*) – The number of patches.
- **n_offsets** (*int*) – The number of offsets.

Returns

default_options (*dict*) – A *dict* with the default options. It contains:

- `patches_indices` : (*list* or *int*) The selected patches.
- `offset_index` : (*int*) The selected offset.
- `background` : (*str*) The background colour.
- `render_patches` : (*bool*) Whether to render the patches.
- `render_patches_bboxes` : (*bool*) Whether to render boxes around the patches.
- `bboxes_line_colour` : (*list*) The boxes line colour.
- `bboxes_line_style` : (*str*) The boxes line style.
- `bboxes_line_width` : (*float*) The boxes line width.
- `render_centers` : (*bool*) Whether to render the patches centers

If the object is not seen before by the widget, then it automatically gets the following default options:
- `patches_indices = list(range(n_patches))`
- `offset_index = 0`
- `background = 'white'`
- `render_patches = True`
- `render_patches_bboxes = True`
- `bboxes_line_colour = ['red']`
- `bboxes_line_style = '-'`
- `bboxes_line_width = 1`
- `render_centers = True`

get_key (*n_patches*, *n_offsets*)

Function that returns a unique key based on the properties of the provided patches object.

Parameters
- **n_patches** (*int*) – The number of patches.
- **n_offsets** (*int*) – The number of offsets.

Returnskey (*str*) – The key that has the format `'{n_patches}_{n_offsets}'`.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters
- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change``` is a dictionary. The change dictionary at least holds a `'type'` key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*, *subwidgets_style*)

Function that sets a predefined style on the widget.

Parameters
- **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **subwidgets_style** (*str* (see below)) – Sub-widgets (patches and bounding boxes) style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )
Function that removes all the internal handler callback functions.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*n_patches*, *n_offsets*, *allow_callback=True*)
Method that updates the state of the widget, if the key generated with *get_key()* based on the provided *n_patches* and *n_offsets* is different than the current key based on `self.n_patches` and `self.n_offsets`.

Parameters

- **n_patches** (*int*) – The number of patches.
- **n_offsets** (*int*) – The number of offsets.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='dashed'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *bboxes_box_style=None*, *bboxes_border_visible=False*, *bboxes_border_colour='black'*, *bboxes_border_style='solid'*, *bboxes_border_width=1*, *bboxes_border_radius=0*, *bboxes_padding=0*, *bboxes_margin=0*, *patches_box_style=None*, *patches_border_visible=False*, *patches_border_colour='black'*, *patches_border_style='solid'*, *patches_border_width=1*, *patches_border_radius=0*, *patches_padding=0*, *patches_margin=0*)
Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.

- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **bboxes_box_style** (*str* or `None` (see below), optional) – Style options for the bounding boxes:
  'success', 'info', 'warning', 'danger', '', None
- **bboxes_border_visible** (*bool*, optional) – Defines whether to draw the border line around the bounding boxes options.
- **bboxes_border_colour** (*str*, optional) – The color of the border around the bounding boxes options.
- **bboxes_border_style** (*str*, optional) – The line style of the border around the bounding boxes options.
- **bboxes_border_width** (*float*, optional) – The line width of the border around the bounding boxes options.
- **bboxes_border_radius** (*float*, optional) – The radius of the corners of the box of the bounding boxes options.
- **bboxes_padding** (*float*, optional) – The padding around the bounding boxes options.
- **bboxes_margin** (*float*, optional) – The margin around the bounding boxes options.
- **patches_box_style** (*str* or `None` (see below), optional) – Style options of the patches and offset options:
  'success', 'info', 'warning', 'danger', '', None
- **patches_border_visible** (*bool*, optional) – Defines whether to draw the border line around the patches and offset options.
- **patches_border_colour** (*str*, optional) – The color of the border around the patches and offset options.
- **patches_border_style** (*str*, optional) – The line style of the border around the patches and offset options.
- **patches_border_width** (*float*, optional) – The line width of the border around the patches and offset options.
- **patches_border_radius** (*float*, optional) – The radius of the corners of the box of the patches and offset options.
- **patches_padding** (*float*, optional) – The padding around the patches and offset options.
- **patches_margin** (*float*, optional) – The margin around the patches and offset options.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

PlotOptionsWidget

class menpowidgets.options. **PlotOptionsWidget** (*legend_entries*, *render_function=None*, *style='minimal'*, *tabs_style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting options for rendering various curves in a graph. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*LineOptionsWidget*	*lines_wid*	Line options widget
2	*MarkerOptionsWidget*	*markers_wid*	Marker options widget
3	*Dropdown*	*curves_dropdown*	Curve selector
4	*Tab*	*lines_markers_tab*	Contains 1, 2
5	*VBox*	*lines_markers_box*	Contains 3, 4
6	*LegendOptionsWidget*	*legend_wid*	Legend options widget
7	*AxesOptionsWidget*	*axes_wid*	Axes options widget
8	*ZoomTwoScalesWidget*	*zoom_wid*	Zoom options widget
9	*GridOptionsWidget*	*grid_wid*	Grid options widget
10	*Text*	*x_label*	X label text
11	*Text*	*y_label*	Y label text
12	*Text*	*title*	Title text
13	*Textarea*	*legend_entries_text*	Legend entries text
14	*VBox*	*plot_related_options*	Contains 10 - 13
15	*Tab*	*options_tab*	Contains 14, 5 - 9

Note that:

- The widget has **memory** about the properties of the objects that are passed into it through *legend_entries*.
- The selected values of the current object object are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *`style()`* and *`predefined_style()`* methods.
- To update the handler callback function of the widget, please refer to the *`replace_render_function()`* method.

Parameters

- **`legend_entries`** (*list* of *str*) – The *list* of legend entries per curve.
- **`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute
  - `name` : the name of the modified trait attribute.

  If `None` , then nothing is assigned.
- **`style`** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **`tabs_style`** (*str* (see below), optional) – Sets a predefined style at the tabs of the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create a plot options widget. Firstly, we need to import it:

```
>>> from menpowidgets.options import PlotOptionsWidget
```

Let's set some legend entries:

```
>>> legend_entries = ['method_1', 'method_2']
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected marker face colour and line width:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Marker edge colours: {}, Line widths: {}".format(
>>>         wid.selected_values['marker_edge_colour'],
>>>         wid.selected_values['line_width'])
>>>     print_dynamic(s)
```

Create the widget with the initial options and display it:

```
>>> wid = PlotOptionsWidget(legend_entries,
>>>                         render_function=render_function,
>>>                         style='danger', tabs_style='info')
>>> wid
```

By playing around, the printed message gets updated. The style of the widget can be changed as:

```
>>> wid.predefined_style('minimal', 'info')
```

---

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

create_default_options ( )

Function that returns a *dict* with default options. The returned *dict* has the following default keys and values:

- `title = ''`
- `x_label = ''`
- `y_label = ''`
- `render_legend = True`
- `legend_title = ''`
- `legend_font_name = 'sans-serif'`
- `legend_font_style = 'normal'`
- `legend_font_size = 10`
- `legend_font_weight = 'normal'`
- `legend_marker_scale = 1.`
- `legend_location = 2`
- `legend_bbox_to_anchor = (1.05,1.)`
- `legend_border_axes_pad = 1.`
- `legend_n_columns = 1`
- `legend_horizontal_spacing = 1.`
- `legend_vertical_spacing = 1.`
- `legend_border = True`
- `legend_border_padding = 0.5`
- `legend_shadow = False`
- `legend_rounded_corners = False`
- `render_axes = True`
- `axes_font_name = 'sans-serif'`
- `axes_font_size = 10`
- `axes_font_style = 'normal'`
- `axes_font_weight = 'normal'`
- `axes_x_limits = None`
- `axes_y_limits = None`
- `axes_x_ticks = None`
- `axes_y_ticks = None`
- `render_grid = True`
- `grid_line_style = '--'`
- `grid_line_width = 0.5`
- `render_lines = [True] * self.n_curves`
- `line_width = [1] * self.n_curves`
- `line_colour = colours if self.n_curves > 1 else ['red']`
- `line_style = ['-'] * self.n_curves`
- `render_markers = [True] * self.n_curves`
- `marker_size = [7] * self.n_curves`
- `marker_face_colour = ['white'] * self.n_curves`
- `marker_edge_colour = colours if self.n_curves > 1 else ['red']`
- `marker_style = ['s'] * self.n_curves`
- `marker_edge_width = [2.] * self.n_curves`
- `zoom = [1.,1.]`

where `colours = sample_colours_from_colourmap(self.n_curves,'Paired')`.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature

should be `handler(change)`, where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*, *tabs_style='minimal'*)

Function that sets a predefined style on the widget.

Parameters

- **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **tabs_style** (*str* (see below)) – Tabs style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding='0.2cm'*, *margin=0*, *tabs_box_style=None*, *tabs_border_visible=True*, *tabs_border_colour='black'*, *tabs_border_style='solid'*, *tabs_border_width=1*, *tabs_border_radius=1*, *tabs_padding=0*, *tabs_margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **tabs_box_style** (*See Below, optional*) – Possible tab widgets style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **tabs_border_visible** (*bool*, optional) – Defines whether to draw the border line around the tab widgets.
- **tabs_border_colour** (*str*, optional) – The colour of the border around the tab widgets.
- **tabs_border_style** (*str*, optional) – The line style of the border around the tab widgets.
- **tabs_border_width** (*float*, optional) – The line width of the border around the tab widgets.
- **tabs_border_radius** (*float*, optional) – The radius of the corners of the box of the tab widgets.
- **tabs_padding** (*float*, optional) – The padding around the tab widgets.
- **tabs_margin** (*float*, optional) – The margin around the tab widgets.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns

False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

RendererOptionsWidget

class menpowidgets.options. **RendererOptionsWidget** (*options_tabs*, *labels*, *axes_x_limits=None*, *axes_y_limits=None*, *render_function=None*, *style='minimal'*, *tabs_style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting rendering options. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*LineOptionsWidget* *MarkerOptionsWidget* *ImageOptionsWidget* *NumberingOptionsWidget* *ZoomOneScaleWidget* *ZoomTwoScalesWidget* *AxesOptionsWidget* *LegendOptionsWidget* *GridOptionsWidget*	*options_widgets*	*list* that contains the rendering sub-options widgets
2	Tab	*suboptions_tab*	Contains all 2

Note that:

- To update the state of the widget, please refer to the *set_widget_state()* method.

- The widget has **memory** about the properties of the objects that are passed into it through *set_widget_state()* . Each object has a unique key id assigned through *get_key()* . Then, the options that correspond to each key are stored in the `self.default_options` *dict*.
- The selected values of the current object object are stored in the `self.selected_values` *trait*.
- When an unseen image object is passed in (i.e. a key that is not included in the `self.default_options` *dict*), it gets the following initial options by default:

  -lines
    *render_lines = True
    *line_width = 1
    *line_style = '-
    *line_colour = ['red'] if labels is None else colours
  -markers
    *render_markers = True
    *marker_size = 5
    *marker_style = 'o'
    *marker_face_colour = ['red'] if labels is None else colours
    *marker_edge_colour = ['black'] if labels is None else colours
    *marker_edge_width = 1

  where `colours = sample_colours_from_colourmap(len(labels),'jet')`

  -image
    *interpolation = 'bilinear'
    *cmap_name = None
    *alpha = 1.
  -numbering
    *render_numbering = False
    *numbers_font_name = 'sans-serif'
    *numbers_font_size = 10
    *numbers_font_style = 'normal'
    *numbers_font_weight = 'normal'
    *numbers_font_colour = ['black']
    *numbers_horizontal_align = 'center'
    *numbers_vertical_align = 'bottom'
  -zoom_one = 1.
  -zoom_two = [1.,1.]
  -axes
    *render_axes = False
    *axes_font_name = 'sans-serif'

```
    *axes_font_size = 10
    *axes_font_style = 'normal'
    *axes_font_weight = 'normal'
    *axes_x_ticks = None
    *axes_y_ticks = None
    *axes_x_limits = axes_x_limits
    *axes_y_limits = axes_y_limits
  -legend
    *render_legend = False
    *legend_title = ''
    *legend_font_name = 'sans-serif'
    *legend_font_style = 'normal'
    *legend_font_size = 10
    *legend_font_weight = 'normal'
    *legend_marker_scale = 1.
    *legend_location = 2
    *legend_bbox_to_anchor = (1.05,1.)
    *legend_border_axes_pad = 1.
    *legend_n_columns = 1
    *legend_horizontal_spacing = 1.
    *legend_vertical_spacing = 1.
    *legend_border = True
    *legend_border_padding = 0.5
    *legend_shadow = False
    *legend_rounded_corners = False
  -grid
    *render_grid = False
    *grid_line_width = 0.5
    *grid_line_style = '--'
```

- To set the styling of this widget please refer to the *style()* and *predefined_style()* methods.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **options_tabs** (*list* of *str*) – *List* that defines the ordering of the options tabs. Possible values are:

Value	Returned object
'lines'	*LineOptionsWidget*
'markers'	*MarkerOptionsWidget*
'numbering'	*NumberingOptionsWidget*
'zoom_one'	*ZoomOneScaleWidget*
'zoom_two'	*ZoomTwoScalesWidget*
'legend'	*LegendOptionsWidget*
'grid'	*GridOptionsWidget*
'image'	*ImageOptionsWidget*
'axes'	*AxesOptionsWidget*

- **labels** (*list* or None , optional) – The *list* of labels used in all *ColourSelectionWidget* objects.

- **axes_x_limits** (*float* or (*float*, *float*) or None , optional) – The limits of the x axis. If *float*, then it sets padding on the right and left as a percentage of the rendered object's width. If *tuple* or *list*, then it defines the axis limits. If None , then the limits are set automatically.

- **axes_y_limits** ((*float*, *float*) *tuple* or None , optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom as a percentage of the rendered object's height. If *tuple* or *list*, then it defines the axis limits. If None , then the limits are set automatically.

- **render_function** (*callable* or None , optional) – The render function that is executed when a widgets' value changes. It must have signature render_function(change) where change is a *dict* with the following keys:

  - type : The type of notification (normally 'change').
  - owner : the *HasTraits* instance
  - old : the old value of the modified trait attribute
  - new : the new value of the modified trait attribute
  - name : the name of the modified trait attribute.

  If None , then nothing is assigned.

- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
'minimal'	Simple black and white style
'success'	Green-based style
'info'	Blue-based style
'warning'	Yellow-based style
'danger'	Red-based style
''	No style

- **tabs_style** (*str* (see below), optional) – Sets a predefined style at the tabs of the widget. Possible options are:

Style	Description
'minimal'	Simple black and white style
'success'	Green-based style
'info'	Blue-based style
'warning'	Yellow-based style
'danger'	Red-based style
''	No style

Example

Let's create a rendering options widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import RendererOptionsWidget
```

Let's set some initial options:

```
>>> options_tabs = ['markers', 'lines', 'grid']
>>> labels = ['jaw', 'eyes']
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected marker face colour and line width:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Marker face colour: {}, Line width: {}".format(
>>>         wid.selected_values['markers']['marker_face_colour'],
>>>         wid.selected_values['lines']['line_width'])
>>>     print_dynamic(s)
```

Create the widget with the initial options and display it:

```
>>> wid = RendererOptionsWidget(options_tabs, labels=labels,
>>>                             render_function=render_function,
>>>                             style='info')
>>> wid
```

By playing around, the printed message gets updated. The style of the widget can be changed as:

```
>>> wid.predefined_style('minimal', 'info')
```

Finally, let's change the widget status with a new set of labels:

```
>>> wid.set_widget_state(labels=['1'], allow_callback=True)
```

Remember that the widget is **mnemonic**, i.e. it remembers the objects it has seen and their corresponding options. These can be retrieved as:

```
>>> wid.default_options
```

add_callbacks()

Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_render_function(*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.

- `type : 'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

get_default_options (*labels*)
Function that returns a *dict* with default options given a *list* of labels. The function returns the *dict* of options but also updates the `self.default_options` *dict*.

Parameters

labels (*list* or `None` , optional) – The *list* of labels used in all *ColourSelectionWidget* objects

Returns

default_options (*dict*) – A *dict* with the default options. It contains:

- `lines` : (*dict*) It has the following keys:
  - `render_lines` : (*bool*) Whether to render the lines.
  - `line_width` : (*float*) The width of the lines.
  - `line_style` : (*str*) The style of the lines.
  - `line_colour` : (*list*) The colour per label.
- `markers` : (*dict*) It has the following keys:
  - `render_markers` : (*bool*) Whether to render the markers.
  - `marker_size` : (*int*) The size of the markers.
  - `marker_style` : (*str*) The style of the markers.
  - `marker_face_colour` : (*list*) The face colour per label.
  - `marker_edge_colour` : (*list*) The edge colour per label.
  - `marker_edge_width` : (*float*) The edge width of the markers.

If the object is not seen before by the widget, then it automatically gets the following default options:

```
- lines
  - render_lines = True
  - line_width = 1
  - line_style = '-
  - line_colour = ['red'] if labels is None else colours
- markers
  - render_markers = True
  - marker_size = 5
  - marker_style = 'o'
  - marker_face_colour = ['red'] if labels is None else
    colours
  - marker_edge_colour = ['black'] if labels is None else
    colours
  - marker_edge_width = 1
```

where `colours = sample_colours_from_colourmap(len(labels),'jet')`

get_key (*labels*)

Function that returns a unique key based on the provided labels.

Parameters **labels** (*list* or `None` , optional) – The *list* of labels used in all `ColourSelectionWidget` objects

Returns **key** (*str*) – The key that has the format `'{labels}'` .

has_trait (*name*)

Returns True if the object has a trait with the specified name.

initialise_global_options (*axes_x_limits*, *axes_y_limits*)

Function that returns a *dict* with global options, i.e. options that do not depend on *labels*. The functions updates `self.global_options` *dict* with:

- `image` : (*dict*) It has the following keys:
  - `interpolation` : (*str*) The interpolation method.
  - `cmap_name` : (*str*) The colourmap.
  - `alpha` : (*float*) The alpha transparency value.
- `numbering` : (*dict*) It has the following keys:
  - `render_numbering` : (*bool*) Flag for rendering the numbers.
  - `numbers_font_name` : (*str*) The font name.
  - `numbers_font_size` : (*int*) The font size.
  - `numbers_font_style` : (*str*) The font style.
  - `numbers_font_weight` : (*str*) The font weight.
  - `numbers_font_colour` : (*list*) The font colour.
  - `numbers_horizontal_align` : (*str*) The horizontal alignment.
  - `numbers_vertical_align` : (*str*) The vertical alignment.
- `zoom_one` : (*float*) The zoom value.
- `zoom_two` : (*list* of *float*) The zoom values.
- `axes` : (*dict*) It has the following keys:
  - `render_axes` : (*bool*) Flag for rendering the axes.
  - `axes_font_name` : (*str*) The axes font name.
  - `axes_font_size` : (*int*) The axes font size.
  - `axes_font_style` : (*str*) The axes font style
  - `axes_font_weight` : (*str*) The font weight.
  - `axes_x_ticks` : (*list* or `None`) The x ticks.
  - `axes_y_ticks` : (*list* or `None`) The y ticks.
  - `axes_x_limits` : (*float* or [*float*, *float*] or `None`) The x limits.
  - `axes_y_limits` : (*float* or [*float*, *float*] or `None`) The y limits.
- `legend` : (*dict*) It has the following keys:
  - `render_legend` : (*bool*) Flag for rendering the legend.
  - `legend_title` : (*str*) The legend title.
  - `legend_font_name` : (*str*) The font name.
  - `legend_font_style` : (*str*) The font style.
  - `legend_font_size` : (*str*) The font size.
  - `legend_font_weight` : (*str*) The font weight.
  - `legend_marker_scale` : (*float*) The marker scale.
  - `legend_location` : (*int*) The legend location.
  - `legend_bbox_to_anchor` : (*tuple*) Bbox to anchor.
  - `legend_border_axes_pad` : (*float*) Border axes pad.
  - `legend_n_columns` : (*int*) The number of columns.
  - `legend_horizontal_spacing` : (*float*) Horizontal spacing.
  - `legend_vertical_spacing` : (*float*) Vetical spacing.
  - `legend_border` : (*bool*) Flag for adding border to the legend
  - `legend_border_padding` : (*float*) The border padding
  - `legend_shadow` : (*bool*) Flag for adding shadow to the legend
  - `legend_rounded_corners` : (*bool*) Flag for adding rounded corners to the legend.

- `gird` : (*dict*) It has the following keys:
  - `render_grid` : (*bool*) Flag for rendering the grid.
  - `grid_line_width` : (*int*) The line width.
  - `grid_line_style` : (*str*) The line style.

If the object is not seen before by the widget, then it automatically gets the following default options:

- `image`
  - `interpolation = 'bilinear'`
  - `cmap_name = None`
  - `alpha = 1.`
- `numbering`
  - `render_numbering = False`
  - `numbers_font_name = 'sans-serif'`
  - `numbers_font_size = 10`
  - `numbers_font_style = 'normal'`
  - `numbers_font_weight = 'normal'`
  - `numbers_font_colour = ['black']`
  - `numbers_horizontal_align = 'center'`
  - `numbers_vertical_align = 'bottom'`
- `zoom_one = 1.`
- `zoom_two = [1.,1.]`
- `axes`
  - `render_axes = False`
  - `axes_font_name = 'sans-serif'`
  - `axes_font_size = 10`
  - `axes_font_style = 'normal'`
  - `axes_font_weight = 'normal'`
  - `axes_x_ticks = None`
  - `axes_y_ticks = None`
  - `axes_x_limits = axes_x_limits`
  - `axes_y_limits = axes_y_limits`
- `legend`
  - `render_legend = False`
  - `legend_title = ''`
  - `legend_font_name = 'sans-serif'`
  - `legend_font_style = 'normal'`
  - `legend_font_size = 10`
  - `legend_font_weight = 'normal'`
  - `legend_marker_scale = 1.`
  - `legend_location = 2`
  - `legend_bbox_to_anchor = (1.05,1.)`
  - `legend_border_axes_pad = 1.`
  - `legend_n_columns = 1`
  - `legend_horizontal_spacing = 1.`
  - `legend_vertical_spacing = 1.`
  - `legend_border = True`
  - `legend_border_padding = 0.5`
  - `legend_shadow = False`
  - `legend_rounded_corners = False`
- `grid`
  - `render_grid = False`
  - `grid_line_width = 0.5`
  - `grid_line_style = '--'`

Parameters

- **axes_x_limits** (*float* or (*float*, *float*) or `None` , optional) – The limits of the x axis. If *float*, then it sets padding on the right and left as a percentage of the rendered object's width. If *tuple* or *list*, then it defines the axis limits. If `None` , then the limits are set automatically.
- **axes_y_limits** ((*float*, *float*) *tuple* or `None` , optional) – The limits of the y axis. If *float*, then it sets padding on the top and bottom as a percentage of the rendered object's height. If *tuple* or *list*, then it defines the axis limits. If `None` , then the limits are set automatically.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*, *tabs_style='minimal'*)

Function that sets a predefined style on the widget.

Parameters

- **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **tabs_style** (*str* (see below)) – Tabs style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )

Function that removes all the internal handler callback functions.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters`render_function` (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*labels*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *labels* are different than `self.labels` .

Parameters
- **labels** (*list* or `None` , optional) – The *list* of labels used in all *ColourSelectionWidget* objects
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding='0.2cm'*, *margin=0*, *tabs_box_style=None*, *tabs_border_visible=True*, *tabs_border_colour='black'*, *tabs_border_style='solid'*, *tabs_border_width=1*, *tabs_border_radius=1*, *tabs_padding=0*, *tabs_margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters
- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **tabs_box_style** (*str* or `None` (see below), optional) – Possible tab widgets style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **tabs_border_visible** (*bool*, optional) – Defines whether to draw the border line around the tab widgets.
- **tabs_border_colour** (*str*, optional) – The color of the border around the tab widgets.
- **tabs_border_style** (*str*, optional) – The line style of the border around the tab widgets.
- **tabs_border_width** (*float*, optional) – The line width of the border around the tab widgets.
- **tabs_border_radius** (*float*, optional) – The radius of the corners of the box of the tab widgets.
- **tabs_padding** (*float*, optional) – The padding around the tab widgets.
- **tabs_margin** (*float*, optional) – The margin around the tab widgets.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

SaveFigureOptionsWidget

class menpowidgets.options. **SaveFigureOptionsWidget** (*renderer=None*, *file_format='png'*, *dpi=None*, *orientation='portrait'*, *paper_type='letter'*, *transparent=False*, *face_colour='white'*, *edge_colour='white'*, *pad_inches=0.0*, *overwrite=False*, *style='minimal'*)

Bases: `FlexBox`

Creates a widget for saving a figure to file. The widget consists of the following objects from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*Select*	*file_format_select*	Image format selector
2	*FloatText*	*dpi_text*	DPI selector
3	*Dropdown*	*orientation_dropdown*	Paper orientation
4	*Select*	*papertype_select*	Paper type selector
5	*Checkbox*	*transparent_checkbox*	Transparency setter
6	*ColourSelectionWidget*	*facecolour_widget*	Face colour selector
7	*ColourSelectionWidget*	*edgecolour_widget*	Edge colour selector
8	*FloatText*	*pad_inches_text*	Padding in inches
9	*Text*	*filename_text*	Path and filename
10	*Checkbox*	*overwrite_checkbox*	Overwrite flag
11	*Latex*	*error_latex*	Error message area
12	*Button*	*save_button*	Save button
13	*VBox*	*path_box*	Contains 9, 1, 10, 4
14	*VBox*	*page_box*	Contains 3, 2, 8
15	*VBox*	*colour_box*	Contains 6, 7, 5
16	*Tab*	*options_tabs*	Contains 13, 14, 15
17	*HBox*	*save_box*	Contains 12, 11
18	*VBox*	*options_box*	Contains 16, 17

To set the styling of this widget please refer to the *style()* and *predefined_style()* methods.

Parameters

- **renderer** (*menpo.visualize.Renderer* or subclass or `None`) – The renderer object that was used to render the figure.
- **file_format** (*str*, optional) – The initial value of the file format.
- **dpi** (*float* or `None` , optional) – The initial value of the dpi. If `None` , then dpi is set to `0` .
- **orientation** (`{'portrait','landscape'}` , optional) – The initial value of the paper orientation.
- **paper_type** (*str*, optional) – The initial value of the paper type. Possible options are:

```
'letter', 'legal', 'executive', 'ledger', 'a0', 'a1', 'a2', 'a3',
'a4', 'a5', 'a6', 'a7', 'a8', 'a9', 'a10', 'b0', 'b1', 'b2', 'b3',
'b4', 'b5', 'b6', 'b7', 'b8', 'b9', 'b10'
```

- **transparent** (*bool*, optional) – The initial value of the transparency flag.
- **face_colour** (*str* or *list* of *float*, optional) – The initial value of the face colour.
- **edge_colour** (*str* or *list* of *float*, optional) – The initial value of the edge colour.
- **pad_inches** (*float*, optional) – The initial value of the figure padding in inches.
- **overwrite** (*bool*, optional) – The initial value of the overwrite flag.
- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

predefined_style (*style*)

Function that sets a predefined style on the widget.

Parameters **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

TextPrintWidget

class menpowidgets.options. **TextPrintWidget** (*text_per_line*, *style='minimal'*)

Bases: FlexBox

Creates a widget for printing text. Specifically, it consists of a *list* of *ipywidgets.Latex* objects, i.e. one per text line.

Note that:

- To set the styling please refer to the *style()* and *predefined_style()* methods.
- To update the state of the widget, please refer to the *set_widget_state()* method.

Parameters

- **text_per_line** (*list* of *str*) – The text to be printed per line.
- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
'minimal'	Simple black and white style
'success'	Green-based style
'info'	Blue-based style
'warning'	Yellow-based style
'danger'	Red-based style
''	No style

Example

Let's create an text widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.options import TextPrintWidget
```

Create the widget with some initial options and display it:

```
>>> text_per_line = ['> The', '> Menpo', '> Team']
>>> wid = TextPrintWidget(text_per_line, style='success')
>>> wid
```

The style of the widget can be changed as:

```
>>> wid.predefined_style('danger')
```

Update the widget state as:

```
>>> wid.set_widget_state(['M', 'E', 'N', 'P', 'O'])
```

---

predefined_style (*style*)

Function that sets a predefined style on the widget.

Parameters **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

set_widget_state (*text_per_line*)

Method that updates the state of the widget with a new *list* of lines.

Parameters **text_per_line** (*list* of *str*) – The text to be printed per line.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

1.4 menpowidgets.menpofit.options

1.4.1 Options

Independent widget classes that can be used as the main components for designing high-level widget functions, as the ones in *menpowidgets.menpofit.base*.

ResultOptionsWidget

class menpowidgets.menpofit.options. **ResultOptionsWidget** (*has_gt_shape*, *has_initial_shape*, *has_image*, *render_function=None*, *style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting options when visualizing a fitting result. The widget consists of the following parts from *ipywidgets*:

No	Object	Property (*self.*)	Description
1	*RadioButtons*	*mode*	Subplot mode flag
2	*Checkbox*	*render_image*	Image rendering flag
3	*VBox*	*mode_render_image_box*	Contains 1, 2
4	*Latex*	*shape_buttons[0]*	'Shape:' str
5	*ToggleButton*	*shape_buttons[1]*	Initial shape toggle
6	*ToggleButton*	*shape_buttons[2]*	Final shape toggle
7	*ToggleButton*	*shape_buttons[3]*	Ground truth shape toggle
8	*HBox*	*shape_selection*	Contains 4, 5, 6, 7

Note that:

- To update the state of the widget, please refer to the *set_widget_state()* method.
- The selected values are stored in the `self.selected_values` *trait* which is a *list*.
- To set the styling of this widget please refer to the *style()* and *predefined_style()* methods.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **has_gt_shape** (*bool*) – Whether the fitting result object has the ground truth shape.
- **has_initial_shape** (*bool*) – Whether the fitting result object has the initial shape.
- **has_image** (*bool*) – Whether the fitting result object has the image.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:
  - `type` : The type of notification (normally `'change'`).
  - `owner` : the *HasTraits* instance
  - `old` : the old value of the modified trait attribute
  - `new` : the new value of the modified trait attribute
  - `name` : the name of the modified trait attribute.

  If `None` , then nothing is assigned.
- **style** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
'minimal'	Simple black and white style
'success'	Green-based style
'info'	Blue-based style
'warning'	Yellow-based style
'danger'	Red-based style
''	No style

Example

Let's create a fitting result options widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.menpofit.options import ResultOptionsWidget
```

Now let's define a render function that will get called on every widget change and will dynamically print the selected options:

```
>>> from menpo.visualize import print_dynamic
>>> def render_function(change):
>>>     s = "Final: {}, Initial: {}, GT: {}, Image: {}, Subplots: {}".format(
>>>         wid.selected_values['render_final_shape'],
>>>         wid.selected_values['render_initial_shape'],
>>>         wid.selected_values['render_gt_shape'],
>>>         wid.selected_values['render_image'],
>>>         wid.selected_values['subplots_enabled'])
>>>     print_dynamic(s)
```

Create the widget with some initial options and display it:

```
>>> wid = ResultOptionsWidget(has_gt_shape=True,
>>>                           has_initial_shape=True, has_image=True,
>>>                           render_function=render_function,
>>>                           style='info')
>>> wid
```

By changing the various widgets, the printed message gets updated. Finally, let's change the widget status with a new set of options:

```
>>> wid.set_widget_state(has_gt_shape=True, has_initial_shape=False,
>>>                      has_image=False, allow_callback=True)
```

add_callbacks ( )

Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.

•`type : 'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*)

Function that sets a predefined style on the widget.

Parametersstyle (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )

Function that removes all the internal handler callback functions.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_visibility ( )

Function that sets the visibility of the various components of the widget, depending on the properties of the current image object, i.e. `self.n_channels` and `self.image_is_masked` .

set_widget_state (*has_gt_shape*, *has_initial_shape*, *has_image*, *allow_callback=True*)

Method that updates the state of the widget.

Parameters

- **has_gt_shape** (*bool*) – Whether the fitting result object has the ground truth shape.
- **has_initial_shape** (*bool*) – Whether the fitting result object has the initial shape.
- **has_image** (*bool*) – Whether the fitting result object has the image.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *buttons_style=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **buttons_style** (*str* or `None` (see below), optional) – Style options: 'success', 'info', 'warning', 'danger', 'primary', '', None

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

IterativeResultOptionsWidget

class menpowidgets.menpofit.options. **IterativeResultOptionsWidget** (*has_gt_shape*, *has_initial_shape*, *has_image*, *n_shapes*, *has_costs*, *render_function=None*, *tab_update_function=None*, *displacements_function=None*, *errors_function=None*, *costs_function=None*, *style='minimal'*, *tabs_style='minimal'*)

Bases: *MenpoWidget*

Creates a widget for selecting options when visualizing an iterative fitting result. The widget consists of the following parts from *ipywidgets* and *menpowidgets.tools*:

No	Object	Property (*self.*)	Description
1	*RadioButtons*	*mode*	Subplot mode
2	*Checkbox*	*render_image*	Image rendering
3	*VBox*	*mode_render_image_box*	Contains 1, 2
4	*Latex*	*shape_buttons[0]*	'Shape:' str
5	*ToggleButton*	*shape_buttons[1]*	Initial shape
6	*ToggleButton*	*shape_buttons[2]*	Final shape
7	*ToggleButton*	*shape_buttons[3]*	Ground truth
8	*HBox*	*result_box*	Contains 4-7
9	*RadioButtons*	*iterations_mode*	'Animation' or 'Static'
10	*AnimationOptionsWidget*	*index_animation*	Animation wid
11	*SlicingCommandWidget*	*index_slicing*	Slicing wid
12	*Button*	*plot_errors_button*	Errors plot
13	*Button*	*plot_displacements_button*	Displacements
14	*Button*	*plot_costs_button*	Costs plot
15	*HBox*	*buttons_box*	Contains 12-14
16	*VBox*	*index_buttons_box*	10,11,15
17	*HBox*	*mode_index_buttons_box*	Contains 9, 16
18	*Latex*	*no_iterations_text*	No iterations
19	*VBox*	*iterations_box*	Contains 17, 18
20	*Tab*	*result_iterations_tab*	Contains 8, 19

Note that:

- To update the state of the widget, please refer to the `set_widget_state()` method.
- The selected values are stored in the `self.selected_values` *trait* which is a *list*.
- To set the styling of this widget please refer to the `style()` and `predefined_style()` methods.
- To update the handler callback function of the widget, please refer to the `replace_render_function()` method.
- To update the handler callback plot functions of the widget, please refer to `replace_plots_function()`, `replace_errors_function()` and `replace_displacements_function()` methods.

Parameters

- **`has_gt_shape`** (*bool*) – Whether the fitting result object has the ground truth shape.
- **`has_initial_shape`** (*bool*) – Whether the fitting result object has the initial shape.
- **`has_image`** (*bool*) – Whether the fitting result object has the image.
- **`n_shapes`** (*int* or `None`) – The total number of shapes. If `None`, then it is assumed that no iteration shapes are available.
- **`has_costs`** (*bool*) – Whether the fitting result object has costs attached.
- **`render_function`** (*callable* or `None`, optional) – The render function that is executed when a widgets' value changes. It must have signature `render_function(change)` where `change` is a *dict* with the following keys:

– `type` : The type of notification (normally `'change'`).

– `owner` : the *HasTraits* instance

– `old` : the old value of the modified trait attribute

– `new` : the new value of the modified trait attribute

– `name` : the name of the modified trait attribute.

If `None` , then nothing is assigned.

- **`tab_update_function`** (*callable* or `None` , optional) – A function that gets called when switching between the 'Result' and 'Iterations' tabs. If `None` , then nothing is assigned.

- **`displacements_function`** (*callable* or `None` , optional) – The function that is executed when the 'Displacements' button is pressed. It must have signature `displacements_function(name)` . If `None` , then nothing is assigned and the button is invisible.

- **`errors_function`** (*callable* or `None` , optional) – The function that is executed when the 'Errors' button is pressed. It must have signature `errors_function(name)` . If `None` , then nothing is assigned and the button is invisible.

- **`costs_function`** (*callable* or `None` , optional) – The function that is executed when the 'Errors' button is pressed. It must have signature `displacements_function(name)` . If `None` , then nothing is assigned and the button is invisible.

- **`style`** (*str* (see below), optional) – Sets a predefined style at the widget. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **`tabs_style`** (*str* (see below), optional) – Sets a predefined style at the tab widgets. Possible options are:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

Example

Let's create an iterative result options widget and then update its state. Firstly, we need to import it:

```
>>> from menpowidgets.menpofit.options import IterativeResultOptionsWidget
```

Now let's define a render function that will get called on every widget change and will print the selected options:

```
>>> def render_function(change):
>>>     print(wid.selected_values)
```

Let's also define a plot function that will get called when one of the 'Errors', 'Costs' or 'Displacements' buttons is pressed:

```
>>> def plot_function(name):
>>>     print(name)
```

Create the widget with some initial options and display it:

```
>>> wid = IterativeResultOptionsWidget(
>>>         has_gt_shape=True, has_initial_shape=True, has_image=True,
>>>         n_shapes=20, has_costs=True, render_function=render_function,
>>>         displacements_function=plot_function,
>>>         errors_function=plot_function, costs_function=plot_function,
>>>         style='info', tabs_style='danger')
>>> wid
```

By changing the various widgets, the printed message gets updated. Finally, let's change the widget status with a new set of options:

```
>>> wid.set_widget_state(has_gt_shape=False, has_initial_shape=True,
>>>                      has_image=True, n_shapes=None, has_costs=False,
>>>                      allow_callback=True)
```

---

add_callbacks ( )

Function that adds the handler callback functions in all the widget components, which are necessary for the internal functionality.

add_costs_function (*costs_function*)

Method that adds the provided *costs_function* as a callback handler to the *click* event of `self.plot_costs_button`. The given function is also stored in `self._costs_function`.

Parameters **costs_function** (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_costs_button`. Its signature is `costs_function(name)`. If `None`, then nothing is added.

add_displacements_function (*displacements_function*)

Method that adds the provided *displacements_function* as a callback handler to the *click* event of `self.plot_displacements_button`. The given function is also stored in `self._displacements_function`.

Parameters **displacements_function** (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_displacements_button`. Its signature is `displacements_function(name)`. If `None`, then nothing is added.

add_errors_function (*errors_function*)

Method that adds the provided *errors_function* as a callback handler to the *click* event of `self.plot_errors_button`. The given function is also stored in `self._errors_function`.

Parameters **errors_function** (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_errors_button`. Its signature is `errors_function(name)`. If `None`, then nothing is added.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

predefined_style (*style*, *tabs_style*)

Function that sets a predefined style on the widget.

Parameters

- **style** (*str* (see below)) – Style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

- **tabs_style** (*str* (see below)) – Tabs style options:

Style	Description
`'minimal'`	Simple black and white style
`'success'`	Green-based style
`'info'`	Blue-based style
`'warning'`	Yellow-based style
`'danger'`	Red-based style
`''`	No style

remove_callbacks ( )
Function that removes all the internal handler callback functions.

remove_costs_function ( )
Method that removes the current `self._costs_function` as a callback handler to the *click* event of `self.plot_costs_button` and sets `self._costs_function = None`.

remove_displacements_function ( )
Method that removes the current `self._displacements_function` as a callback handler to the *click* event of `self.plot_displacements_button` and sets `self._displacements_function = None`.

remove_errors_function ( )
Method that removes the current `self._errors_function` as a callback handler to the *click* event of `self.plot_errors_button` and sets `self._errors_function = None`.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_costs_function (*costs_function*)
Method that replaces the current `self._costs_function` with the given *costs_function* as a callback handler to the *click* event of `self.plot_costs_button`.

Parameterscosts_function (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_costs_button`. Its signature is `costs_function(name)`. If `None`, then nothing is added.

replace_displacements_function (*displacements_function*)
Method that replaces the current `self._displacements_function` with the given *displacements_function* as a callback handler to the *click* event of `self.plot_displacements_button`.

Parametersdisplacements_function (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_displacements_button`. Its signature is `displacements_function(name)`. If `None`, then nothing is added.

replace_errors_function (*errors_function*)
Method that replaces the current `self._errors_function` with the given *errors_function* as a callback handler to the *click* event of `self.plot_errors_button`.

Parameterserrors_function (*callable* or `None`, optional) – The function that behaves as a callback handler of the *click* event of `self.plot_errors_button`. Its signature is `errors_function(name)`. If `None`, then nothing is added.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_visibility ( )

Function that sets the visibility of the various components of the widget, depending on the properties of the current image object, i.e. `self.n_channels` and `self.image_is_masked` .

set_widget_state (*has_gt_shape*, *has_initial_shape*, *has_image*, *n_shapes*, *has_costs*, *allow_callback=True*)

Method that updates the state of the widget with a new set of values.

Parameters

- **has_gt_shape** (*bool*) – Whether the fitting result object has the ground truth shape.
- **has_initial_shape** (*bool*) – Whether the fitting result object has the initial shape.
- **has_image** (*bool*) – Whether the fitting result object has the image.
- **n_shapes** (*int* or `None`) – The total number of shapes. If `None` , then it is assumed that no iteration shapes are available.
- **has_costs** (*bool*) – Whether the fitting result object has the costs attached.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *buttons_style=''*, *tabs_box_style=None*, *tabs_border_visible=False*, *tabs_border_colour='black'*, *tabs_border_style='solid'*, *tabs_border_width=1*, *tabs_border_radius=0*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **buttons_style** (*str* or `None` (see below), optional) – Style options:
  'success', 'info', 'warning', 'danger', 'primary', '', None
- **tabs_box_style** (*str* or `None` (see below), optional) – Possible tab widgets style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **tabs_border_visible** (*bool*, optional) – Defines whether to draw the border line around the tab widgets.
- **tabs_border_colour** (*str*, optional) – The colour of the border around the tab widgets.
- **tabs_border_style** (*str*, optional) – The line style of the border around the tab widgets.
- **tabs_border_width** (*float*, optional) – The line width of the border around the tab widgets.
- **tabs_border_radius** (*float*, optional) – The radius of the border around the tab widgets.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

Tools Widgets Low-level widget objects that can be used as the main ingredients for creating more complex widgets.

1.5 menpowidgets.abstract

Main abstract class for defining a Menpo widget.

1.5.1 MenpoWidget

class menpowidgets.abstract.**MenpoWidget**(*children*, *trait*, *trait_initial_value*, *render_function=None*, *orientation='vertical'*, *align='start'*)

Bases: FlexBox

Base class for defining a Menpo widget.

The widget has a *selected_values* trait that can be used in order to inspect any changes that occur to its children. It also has functionality for adding, removing, replacing or calling the handler callback function of the *selected_values* trait.

Parameters

- **children** (*list* of *ipywidgets*) – The *list* of *ipywidgets* objects to be set as children in the *ipywidgets.FlexBox*.
- **trait** (*traitlets.TraitType* subclass) – The type of the *selected_values* object that gets added as a trait in the widget. Possible options from *traitlets* are {Int, Float, Dict, List, Tuple}.
- **trait_initial_value** (*int* or *float* or *dict* or *list* or *tuple*) – The initial value of the *selected_values* trait.
- **render_function** (*callable* or None, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be render_function() or render_function(change), where change is a *dict* with the following keys:
  - owner : the *HasTraits* instance
  - old : the old value of the modified trait attribute
  - new : the new value of the modified trait attribute
  - name : the name of the modified trait attribute.
  - type : 'change'

  If None, then nothing is added.
- **orientation** ({'horizontal', 'vertical'}, optional) – The orientation of the *ipywidgets.FlexBox*.
- **align** ({'start', 'center', 'end'}, optional) – The alignment of the children of the *ipywidgets.FlexBox*.

add_render_function(*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or None, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be render_function() or render_function(change), where change is a *dict* with the following keys:
- owner : the *HasTraits* instance
- old : the old value of the modified trait attribute

- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

1.6 menpowidgets.tools

Low-level widget classes that can be used as the main ingredients for creating more complex widgets, as the ones in *menpowidgets.options* and *menpowidgets.menpofit.options*.

1.6.1 Logo

LogoWidget

class menpowidgets.tools. **LogoWidget** (*style='minimal'*)
Bases: `FlexBox`

Creates a widget with Menpo's logo image. The widget stores the image in `self.image` using *ipywidgets.Image*. To set the styling of this widget please refer to the *style()* method.

Parameters**style** ({'minimal','danger','info','warning','success'} , optional) – Defines the styling of the logo widget, i.e. the colour around the logo image.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *image_width='50px'*)
Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **image_width** (*str*, optional) – The width of the image object

1.6.2 Indexing

IndexSliderWidget

class menpowidgets.tools. **IndexSliderWidget** (*index*, *description='Index: '*, *continuous_update=False*, *render_function=None*)

Bases: *MenpoWidget*

Creates a widget for selecting an index using a slider.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **index** (*dict*) – The *dict* with the default options:
  - `min` : (*int*) The minimum value (e.g. `0`).
  - `max` : (*int*) The maximum value (e.g. `100`).
  - `step` : (*int*) The index step (e.g. `1`).
  - `index` : (*int*) The index value (e.g. `10`).
- **description** (*str*, optional) – The title of the widget.
- **continuous_update** (*bool*, optional) – If `True` , then the render and update functions are called while moving the slider's handle. If `False` , then the the functions are called only when the handle (mouse click) is released.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters
- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change`

is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*index*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *index* values are different than *self.selected_values*.

Parameters

- **index** (*dict*) – The *dict* with the selected options:
  - `min` : (*int*) The minimum value (e.g. `0`).
  - `max` : (*int*) The maximum value (e.g. `100`).
  - `step` : (*int*) The index step (e.g. `1`).
  - `index` : (*int*) The index value (e.g. `10`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *slider_width='6cm'*, *slider_bar_colour=None*, *slider_handle_colour=None*, *slider_text_visible=True*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **slider_width** (*float*, optional) – The width of the slider
- **slider_bar_colour** (*str*, optional) – The colour of the slider's bar.
- **slider_handle_colour** (*str*, optional) – The colour of the slider's handle.

- **slider_text_visible** (*bool*, optional) – Whether the selected value of the slider is visible.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

IndexButtonsWidget

class menpowidgets.tools. **IndexButtonsWidget** (*index*, *render_function=None*, *description='Index: '*, *minus_description='fa-minus'*, *plus_description='fa-plus'*, *loop_enabled=True*, *text_editable=True*)

Bases: *MenpoWidget*

Creates a widget for selecting an index using plus/minus buttons.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **index** (*dict*) – The *dict* with the default options:
  - `min` : (*int*) The minimum value (e.g. `0`).

– `max` : (*int*) The maximum value (e.g. `100`).

– `step` : (*int*) The index step (e.g. `1`).

– `index` : (*int*) The index value (e.g. `10`).

- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.
- **description** (*str*, optional) – The title of the widget.
- **minus_description** (*str*, optional) – The text/icon of the button that decreases the index. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **plus_description** (*str*, optional) – The title of the button that increases the index. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **loop_enabled** (*bool*, optional) – If `True` , then if by pressing the buttons we reach the minimum (maximum) index values, then the counting will continue from the end (beginning). If `False` , the counting will stop at the minimum (maximum) value.
- **text_editable** (*bool*, optional) – Flag that determines whether the index text will be editable.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

set_widget_state (*index*, *loop_enabled*, *text_editable*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *index* value is different than *self.selected_values*.

Parameters

- **index** (*dict*) – The *dict* with the selected options:
  - `min` : (*int*) The minimum value (e.g. `0`).
  - `max` : (*int*) The maximum value (e.g. `100`).
  - `step` : (*int*) The index step (e.g. `1`).
  - `index` : (*int*) The index value (e.g. `10`).
- **loop_enabled** (*bool*, optional) – If `True`, then if by pressing the buttons we reach the minimum (maximum) index values, then the counting will continue from the end (beginning). If `False`, the counting will stop at the minimum (maximum) value.
- **text_editable** (*bool*, optional) – Flag that determines whether the index text will be editable.
- **allow_callback** (*bool*, optional) – If `True`, it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *minus_style=''*, *plus_style=''*, *text_colour=None*, *text_background_colour=None*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **minus_style** (*str* or `None` (see below), optional) – Style options

```
'success', 'info', 'warning', 'danger', 'primary', '', None
```

- **plus_style** (*str* or `None` (see below), optional) – Style options

```
'success', 'info', 'warning', 'danger', 'primary', '', None
```

- **text_colour** (*str*, optional) – The text colour of the index text.
- **text_background_colour** (*str*, optional) – The background colour of the index text.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.

- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)

Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ListWidget

class menpowidgets.tools. **ListWidget** (*selected_list, mode='float', description='Command:', render_function=None, example_visible=True*)

Bases: *MenpoWidget*

Creates a widget for selecting a *list* of numbers. It supports both *int* and *float*.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **selected_list** (*list*) – The initial list of numbers.
- **mode** (`{'int','float'}` , optional) – Defines the data type of the list members.
- **description** (*str*, optional) – The description of the command text box.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **example_visible** (*bool*, optional) – If *True*, then a line with command examples is printed below the main text box.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*selected_list*, *allow_callback=True*)
Method that updates the state of the widget if the provided *selected_list* value is different than *self.selected_values*.

Parameters

- **selected_list** (*list*) – The selected list of numbers.

- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *text_box_style=None*, *text_box_background_colour=None*, *text_box_width=None*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **text_box_style** (*str* or `None` (see below), optional) – Command text box style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **text_box_background_colour** (*str*, optional) – The background colour of the command text box.
- **text_box_width** (*str*, optional) – The width of the command text box.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

SlicingCommandWidget

class menpowidgets.tools. **SlicingCommandWidget** (*slice_options*, *description='Command:'*, *render_function=None*, *example_visible=True*, *continuous_update=False*, *orientation='horizontal'*)

Bases: *MenpoWidget*

Creates a widget for selecting a slicing command.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **slice_options** (*dict*) – The initial slicing options. It must be a *dict* with:
  - `command` : (*str*) The slicing command (e.g. `':3'`).
  - `length` : (*int*) The maximum length (e.g. `68`).
- **description** (*str*, optional) – The description of the command text box.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **example_visible** (*bool*, optional) – If `True` , then a line with command examples is printed below the main text box.
- **continuous_update** (*bool*, optional) – If `True` , then the render and update functions are called while moving the slider's handle. If `False` , then the the functions are called only when the handle (mouse click) is released.
- **orientation** (`{'horizontal','vertical'}` , optional) – The orientation between the command text box and the sliders.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parametersrender_function (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*slice_options*, *allow_callback=True*)

Method that updates the state of the widget if the provided *slice_options* value is different than *self.selected_values*.

Parameters
- **slice_options** (*dict*) – The new slicing options. It must be a *dict* with:
  - `command` : (*str*) The slicing command (e.g. `':3'`).
  - `length` : (*int*) The maximum length (e.g. `68`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *text_box_style=None*, *text_box_background_colour=None*, *text_box_width=None*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters
- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **text_box_style** (*str* or `None` (see below), optional) – Command text box style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **text_box_background_colour** (*str*, optional) – The background colour of the command text box.
- **text_box_width** (*str*, optional) – The width of the command text box.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

•**font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

•**handler** (*callable*) – The callable called when a trait attribute changes.

•**names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.

•**type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

1.6.3 Rendering

AxesLimitsWidget

class menpowidgets.tools. **AxesLimitsWidget** (*axes_x_limits*, *axes_y_limits*, *render_function=None*)

Bases: *MenpoWidget*

Creates a widget for selecting the axes limits.

•The selected values are stored in the `self.selected_values` *trait*.

•To set the styling of this widget please refer to the *style()* method.

•To update the state of the widget, please refer to the *set_widget_state()* method.

•To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

•**axes_x_limits** (*float* or [*float*, *float*] or `None`) – The limits of the x axis.

- **axes_y_limits** (*float* or [*float*, *float*] or `None`) – The limits of the y axis.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values*

trait of the widget and sets `self._render_function = None`.

replace_render_function(*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

set_widget_state(*axes_x_limits*, *axes_y_limits*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *axes_y_limits* and *axes_x_limits* values are different than *self.selected_values*.

Parameters

- **axes_x_limits** (*float* or [*float*, *float*] or `None`) – The limits of the x axis.
- **axes_y_limits** (*float* or [*float*, *float*] or `None`) – The limits of the y axis.
- **allow_callback** (*bool*, optional) – If `True`, it allows triggering of any callback functions.

style(*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *toggles_style=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

•**toggles_style** (*str* or `None` (see below), optional) – Style options

```
'success', 'info', 'warning', 'danger', 'primary', '', None
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

•**handler** (*callable*) – The callable called when a trait attribute changes.

•**names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.

•**type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

AxesOptionsWidget

class menpowidgets.tools. **AxesOptionsWidget** (*axes_options*, *render_function=None*, *render_checkbox_title='Render axes'*)

Bases: *MenpoWidget*

Creates a widget for selecting axes rendering options.

•The selected values are stored in the `self.selected_values` *trait*.

•To set the styling of this widget please refer to the *style()* method.

•To update the state of the widget, please refer to the *set_widget_state()* method.

•To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

•**axes_options** (*dict*) – The initial axes options. It must be a *dict* with the following keys:

–`render_axes` : (*bool*) Flag for rendering the axes.

–`axes_font_name` : (*str*) The axes font name (e.g. `'serif'`).

– `axes_font_size` : (*int*) The axes font size (e.g. `10`).

– `axes_font_style` : (*str*) The axes font style (e.g. `'normal'`)

– `axes_font_weight` : (*str*) The font weight (e.g. `'normal'`).

– `axes_x_ticks` : (*list* or `None`) The x ticks (e.g. `[10,20]`)

– `axes_y_ticks` : (*list* or `None`) The y ticks (e.g. `None`).

– `axes_x_limits` : (*float* or [*float*, *float*] or `None`) The x limits (e.g. `None`).

– `axes_y_limits` : (*float* or [*float*, *float*] or `None`) The y limits (e.g. `1.`).

• **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

• **render_checkbox_title** (*str*, optional) – The description of the show line checkbox.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type :`

the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*axes_options*, *allow_callback=True*)

Method that updates the state of the widget if the provided *axes_options* are different than *self.selected_values*.

Parameters

- **axes_options** (*dict*) – The selected axes options. It must be a *dict* with the following keys:
  - `render_axes` : (*bool*) Flag for rendering the axes.
  - `axes_font_name` : (*str*) The axes font name (e.g. `'serif'`).
  - `axes_font_size` : (*int*) The axes font size (e.g. `10`).
  - `axes_font_style` : (*str*) The axes font style (e.g. `'normal'`)
  - `axes_font_weight` : (*str*) The font weight (e.g. `'normal'`).
  - `axes_x_ticks` : (*list* or `None`) The x ticks (e.g. `[10,20]`).
  - `axes_y_ticks` : (*list* or `None`) The y ticks (e.g. `None`)
  - `axes_x_limits` : (*float* or [*float*, *float*] or `None`) The x limits (e.g. `None`).
  - `axes_y_limits` : (*float* or [*float*, *float*] or `None`) The y limits (e.g. `1.`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.

- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

AxesTicksWidget

class menpowidgets.tools. **AxesTicksWidget** (*axes_ticks*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting the axes ticks.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **axes_ticks** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `x` : (*list* or `None`) The x ticks (e.g. `[10,20]`).
  - `y` : (*list* or `None`) The y ticks (e.g. `None`).
- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)`, where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None`, optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)`, where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None`, then nothing is added.

set_widget_state (*axes_ticks*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *axes_ticks* values are different than *self.selected_values*.

Parameters

- **axes_ticks** (*dict*) – The selected options. It must be a *dict* with the following keys:
  - `x` : (*list* or `None`) The x ticks (e.g. `[10,20]`).
  - `y` : (*list* or `None`) The y ticks (e.g. `None`).
- **allow_callback** (*bool*, optional) – If `True`, it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *toggles_style=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.

- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **toggles_style** (*str* or `None` (see below), optional) – Style options

```
'success', 'info', 'warning', 'danger', 'primary', '', None
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ColourSelectionWidget

class menpowidgets.tools. **ColourSelectionWidget** (*colours_list*, *render_function=None*, *description='Colour'*, *labels=None*)

Bases: *MenpoWidget*

Creates a widget for colour selection of a single or multiple objects.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **colours_list** (*list* of *str* or [*float*, *float*, *float*]) – A *list* of colours. If a colour is defined as an *str*, then it must either be a hex code or a colour name, such as

```
'blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black',
'white', 'pink', 'orange'
```

  If a colour has the form [*float*, *float*, *float*], then it defines an RGB value and must have length 3.

- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **description** (*str*, optional) – The description of the widget.
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels' names. If `None` , then a *list* of the form `label {}` is automatically defined.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_colours (*colours_list*, *allow_callback=True*)
Method that updates the colour values of the widget.

Parameters

- **colours_list** (*list* of *str* or [*float*, *float*, *float*]) – A *list* of colours. If a colour is defined as an *str*, then it must either be a hex code or a colour name, such as

```
'blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black',
'white', 'pink', 'orange'
```

  If a colour has the form [*float*, *float*, *float*], then it defines an RGB value and must have length 3.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

Raises`ValueError` – You must provide a colour per label.

set_widget_state (*colours_list*, *labels=None*, *allow_callback=True*)
Method that updates the state of the widget, if the provided *colours_list* and *labels* values are different than *self.selected_values* and *self.labels* respectively.

Parameters

- **colours_list** (*list* of *str* or [*float*, *float*, *float*]) – A *list* of colours. If a colour is defined as an *str*, then it must either be a hex code or a colour name, such as

```
'blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black',
'white', 'pink', 'orange'
```

  If a colour has the form [*float*, *float*, *float*], then it defines an RGB value and must have length 3.
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels' names. If `None` , then a *list* of the form `label {}` is automatically defined.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *label_colour=None*, *label_background_colour=None*, *picker_colour=None*, *picker_background_colour=None*, *apply_to_all_style=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **label_colour** (*str*, optional) – The text colour of the labels dropdown selection.
- **label_background_colour** (*str*, optional) – The background colour of the labels dropdown selection.
- **picker_colour** (*str*, optional) – The text colour of the colour picker.
- **picker_background_colour** (*str*, optional) – The background colour of the colour picker.
- **apply_to_all_style** (*str*,) – Style options

```
'success', 'info', 'warning', 'danger', 'primary', '', None
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

GridOptionsWidget

class menpowidgets.tools. **GridOptionsWidget** (*grid_options*, *render_function=None*, *render_checkbox_title='Render grid'*)

Bases: *MenpoWidget*

Creates a widget for selecting grid rendering options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **grid_options** (*dict*) – The initial grid options. It must be a *dict* with the following keys:
  - `render_grid` : (*bool*) Flag for rendering the grid.
  - `grid_line_width` : (*int*) The line width (e.g. `1`).

–`grid_line_style` : (*str*) The line style (e.g. `'-'`).

•**`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

•**`render_checkbox_title`** (*str*, optional) – The description of the show line checkbox.

`add_render_function` (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**`render_function`** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

•`owner` : the *HasTraits* instance

•`old` : the old value of the modified trait attribute

•`new` : the new value of the modified trait attribute

•`name` : the name of the modified trait attribute.

•`type` : `'change'`

If `None` , then nothing is added.

`add_traits` (***traits*)

Dynamically add trait attributes to the Widget.

`call_render_function` (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

•**`old_value`** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.

•**`new_value`** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.

•**`type_value`** (*str*, optional) – The trait event type.

`close` ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

`has_trait` (*name*)

Returns True if the object has a trait with the specified name.

`observe` (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

•**`handler`** (*`callable`*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

•**`names`** (*`list, str, All`*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.

•**`type`** (*`str, All (default: 'change')`*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*grid_options*, *allow_callback=True*)
Method that updates the state of the widget with a new set of values.

Parameters
- **grid_options** (*dict*) – The selected grid options. It must be a *dict* with the following keys:
  - `render_grid` : (*bool*) Flag for rendering the grid.
  - `grid_line_width` : (*int*) The line width (e.g. `1`).
  - `grid_line_style` : (*str*) The line style (e.g. `'-'`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)
Function that defines the styling of the widget.

Parameters
- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ImageOptionsWidget

class menpowidgets.tools. **ImageOptionsWidget** (*image_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting image rendering options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **image_options** (*dict*) – The initial image options. It must be a *dict* with the following keys:
  - `alpha` : (*float*) The alpha value (e.g. `1.`).
  - `interpolation` : (*str*) The interpolation (e.g. `'bilinear'`).

– `cmap_name` : (*str*) The colourmap (e.g. `'gray'`).

• **`render_function`** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

`add_render_function` (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **`render_function`** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

`add_traits` (***traits*)

Dynamically add trait attributes to the Widget.

`call_render_function` (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **`old_value`** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **`new_value`** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **`type_value`** (*str*, optional) – The trait event type.

`close` ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

`has_trait` (*name*)

Returns True if the object has a trait with the specified name.

`observe` (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **`handler`** (*`callable`*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **`names`** (*`list, str, All`*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **`type`** (*`str, All (default: 'change')`*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

`remove_render_function` ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values*

trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*image_options*, *allow_callback=True*)

Method that updates the state of the widget if the provided *image_options* are different than *self.selected_values*.

Parameters

- **image_options** (*dict*) – The selected image options. It must be a *dict* with the following keys:
  - `alpha` : (*float*) The alpha value (e.g. `1.`).
  - `interpolation` : (*str*) The interpolation (e.g. `'bilinear'`)
  - `cmap_name` : (*str*) The colourmap (e.g. `'gray'`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

LegendOptionsWidget

class menpowidgets.tools. **LegendOptionsWidget** (*legend_options*, *render_function=None*, *render_checkbox_title='Render legend'*)

Bases: *MenpoWidget*

Creates a widget for selecting legend rendering options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **legend_options** (*dict*) – The initial legend options. It must be a *dict* with the following keys:
  - `render_legend` : (*bool*) Flag for rendering the legend.

- `legend_title` : (*str*) The legend title (e.g. `''`).
- `legend_font_name` : (*str*) The font name (e.g. `'serif'`).
- `legend_font_style` : (*str*) The font style (e.g. `'normal'`).
- `legend_font_size` : (*str*) The font size (e.g. `10`).
- `legend_font_weight` : (*str*) The font weight (e.g. `'normal'`).
- `legend_marker_scale` : (*float*) The marker scale (e.g. `1.`).
- `legend_location` : (*int*) The legend location (e.g. `2`).
- `legend_bbox_to_anchor` : (*tuple*) Bbox to anchor (e.g. `(1.05,1.)`).
- `legend_border_axes_pad` : (*float*) Border axes pad (e.g. `1.`).
- `legend_n_columns` : (*int*) The number of columns (e.g. `1`).
- `legend_horizontal_spacing` : (*float*) Horizontal spacing (e.g. `1.`).
- `legend_vertical_spacing` : (*float*) Vetical spacing (e.g. `1.`).
- `legend_border` : (*bool*) Flag for adding border to the legend.
- `legend_border_padding` : (*float*) The border padding (e.g. `0.5`)
- `legend_shadow` : (*bool*) Flag for adding shadow to the legend.
- `legend_rounded_corners` : (*bool*) Flag for adding rounded corners to the legend.

•render_function (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

•render_checkbox_title (*str*, optional) – The description of the render legend checkbox.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

•old_value (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.

•new_value (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.

•type_value (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*legend_options*, *allow_callback=True*)
Method that updates the state of the widget if the provided *legend_options* are different than *self.selected_values*.

Parameters

- **legend_options** (*dict*) – The selected legend options. It must be a *dict* with the following keys:
  - `render_legend` : (*bool*) Flag for rendering the legend.
  - `legend_title` : (*str*) The legend title (e.g. `''`).
  - `legend_font_name` : (*str*) The font name (e.g. `'serif'`).
  - `legend_font_style` : (*str*) The font style (e.g. `'normal'`).
  - `legend_font_size` : (*str*) The font size (e.g. `10`).
  - `legend_font_weight` : (*str*) The font weight (e.g. `'normal'`).

- `legend_marker_scale` : (*float*) The marker scale (e.g. `1.`).
- `legend_location` : (*int*) The legend location (e.g. `2`).
- `legend_bbox_to_anchor` : (*tuple*) Bbox to anchor (e.g. `(1.05,1.)`).
- `legend_border_axes_pad` : (*float*) Border axes pad (e.g. `1.`).
- `legend_n_columns` : (*int*) The number of columns (e.g. `1`).
- `legend_horizontal_spacing` : (*float*) Horizontal spacing (e.g. `1.`).
- `legend_vertical_spacing` : (*float*) Vetical spacing (e.g. `1.`)
- `legend_border` : (*bool*) Flag for adding border to the legend.
- `legend_border_padding` : (*float*) The border padding (e.g. `0.5`)
- `legend_shadow` : (*bool*) Flag for adding shadow to the legend.
- `legend_rounded_corners` : (*bool*) Flag for adding rounded corners to the legend.

• **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

• **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

• **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
• **border_colour** (*str*, optional) – The colour of the border around the widget.
• **border_style** (*str*, optional) – The line style of the border around the widget.
• **border_width** (*float*, optional) – The line width of the border around the widget.
• **border_radius** (*float*, optional) – The radius of the border around the widget.
• **padding** (*float*, optional) – The padding around the widget.
• **margin** (*float*, optional) – The margin around the widget.
• **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

• **font_size** (*int*, optional) – The font size.
• **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

• **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

LineOptionsWidget

class menpowidgets.tools. **LineOptionsWidget** (*line_options*, *render_function=None*, *render_checkbox_title='Render lines'*, *labels=None*)

Bases: *MenpoWidget*

Creates a widget for selecting line rendering options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **line_options** (*dict*) – The initial line options. It must be a *dict* with the following keys:
  - `render_lines` : (*bool*) Flag for rendering the lines.
  - `line_width` : ('float') The width of the lines (e.g. `1.`)
  - `line_colour` : (*str*) The colour of the lines (e.g. `'blue'`).
  - `line_style` : (*str*) The style of the lines (e.g. `'-'`).
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **render_checkbox_title** (*str*, optional) – The description of the show line checkbox.
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels' names that get passed in to the *ColourSelectionWidget*. If `None` , then a *list* of the form `label {}` is automatically

defined. Note that the labels are defined only for the colour option and not the rest of the options.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parametersrender_function (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*line_options*, *labels=None*, *allow_callback=True*)

Method that updates the state of the widget if the provided *line_options* are different than *self.selected_values*.

Parameters

- **line_options** (*dict*) – The selected line options. It must be a *dict* with the following keys:
  - `render_lines` : (*bool*) Flag for rendering the lines.
  - `line_width` : (‘float‘) The width of the lines (e.g. `1.`)
  - `line_colour` : (*str*) The colour of the lines (e.g. `'blue'`).
  - `line_style` : (*str*) The style of the lines (e.g. `'-'`).
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels’ names that get passed in to the *ColourSelectionWidget*. If `None` , then a *list* of the form `label {}` is automatically defined.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

•**font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

•**handler** (*callable*) – The callable called when a trait attribute changes.

•**names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.

•**type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

MarkerOptionsWidget

class menpowidgets.tools. **MarkerOptionsWidget** (*marker_options*, *render_function=None*, *render_checkbox_title='Render markers'*, *labels=None*)

Bases: *MenpoWidget*

Creates a widget for selecting marker rendering options.

•The selected values are stored in the `self.selected_values` *trait*.

•To set the styling of this widget please refer to the *style()* method.

•To update the state of the widget, please refer to the *set_widget_state()* method.

•To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **marker_options** (*dict*) – The initial marker options. It must be a *dict* with the following keys:
  - `render_markers` : (*bool*) Flag for rendering the markers.
  - `marker_size` : (*int*) The size of the markers (e.g. `20`).
  - `marker_face_colour` : (*list*) The colours list. (e.g. `['red','blue']`).
  - `marker_edge_colour` : (*list*) The edge colours list. (e.g. `['black','white']`).
  - `marker_style` : (*str*) The size of the markers. (e.g. `'o'`).
  - `marker_edge_width` : (*int*) The esdge width of the markers. (e.g. `1`).
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **render_checkbox_title** (*str*, optional) – The description of the render marker checkbox.
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels' names that get passed in to the *ColourSelectionWidget*. If `None` , then a *list* of the form `label {}` is automatically defined. Note that the labels are defined only for the colour option and not the rest of the options.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*marker_options*, *labels=None*, *allow_callback=True*)

Method that updates the state of the widget if the provided *marker_options* are different than *self.selected_values*.

Parameters

- **marker_options** (*dict*) – The seected marker options. It must be a *dict* with the following keys:
  - `render_markers` : (*bool*) Flag for rendering the markers.
  - `marker_size` : (*int*) The size of the markers (e.g. `20`).
  - `marker_face_colour` : (*list*) The colours list. (e.g. `['red','blue']`).
  - `marker_edge_colour` : (*list*) The edge colours list. (e.g. `['black','white']`).
  - `marker_style` : (*str*) The size of the markers. (e.g. `'o'`).
  - `marker_edge_width` : (*int*) The esdge width of the markers. (e.g. `1`).
- **labels** (*list* of *str* or `None` , optional) – A *list* with the labels' names that get passed in to the *ColourSelectionWidget*. If `None` , then a *list* of the form `label {}` is automatically defined.
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding

to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

NumberingOptionsWidget

class menpowidgets.tools. **NumberingOptionsWidget** (*numbers_options*, *render_function=None*, *render_checkbox_title='Render numbering'*)

Bases: *MenpoWidget*

Creates a widget for selecting numbering rendering options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **numbers_options** (*dict*) – The initial numbering options. It must be a *dict* with the following keys:
  - `render_numbering` : (*bool*) Flag for rendering the numbers.
  - `numbers_font_name` : (*str*) The font name (e.g. `'serif'`).
  - `numbers_font_size` : (*int*) The font size (e.g. `10`).
  - `numbers_font_style` : (*str*) The font style (e.g. `'normal'`).
  - `numbers_font_weight` : (*str*) The font weight (e.g. `'normal'`).
  - `numbers_font_colour` : (*list*) The font colour (e.g. `['black']`)
  - `numbers_horizontal_align` : (*str*) The horizontal alignment (e.g. `'center'`).
  - `numbers_vertical_align` : (*str*) The vertical alignment (e.g. `'bottom'`).
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.
- **render_checkbox_title** (*str*, optional) – The description of the render numbering checkbox.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute

- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*numbers_options*, *allow_callback=True*)

Method that updates the state of the widget if the given *numbers_options* are different than *self.selected_values*.

Parameters

- **numbers_options** (*dict*) – The selected numbering options. It must be a *dict* with the following keys:
  - `render_numbering` : (*bool*) Flag for rendering the numbers.
  - `numbers_font_name` : (*str*) The font name (e.g. `'serif'`).
  - `numbers_font_size` : (*int*) The font size (e.g. `10`).
  - `numbers_font_style` : (*str*) The font style (e.g. `'normal'`).
  - `numbers_font_weight` : (*str*) The font weight (e.g. `'normal'`).
  - `numbers_font_colour` : (*colour*) The font colour (e.g. `'black'`)
  - `numbers_horizontal_align` : (*str*) The horizontal alignment (e.g. `'center'`).
  - `numbers_vertical_align` : (*str*) The vertical alignment (e.g. `'bottom'`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the

TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)

Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ZoomOneScaleWidget

class menpowidgets.tools. **ZoomOneScaleWidget** (*zoom_options*, *render_function=None*, *description='Figure scale: '*, *minus_description='fa-search-minus'*, *plus_description='fa-search-plus'*, *continuous_update=False*)

Bases: *MenpoWidget*

Creates a widget for selecting zoom options with a single scale.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **zoom_options** (*dict*) – The *dict* with the default options. It must have the following keys:
  - `min` : (*float*) The minimum value (e.g. `0.1`).
  - `max` : (*float*) The maximum value (e.g. `4.`).
  - `step` : (*float*) The zoom step (e.g. `0.05`).
  - `zoom` : (*float*) The zoom value (e.g. `1.`).

- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.
- **description** (*str*, optional) – The title of the widget.
- **minus_description** (*str*, optional) – The text/icon of the button that zooms_out. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **plus_description** (*str*, optional) – The title of the button that zooms in. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **continuous_update** (*bool*, optional) – If `True` , then the render and update functions are called while moving the zoom slider's handle. If `False` , then the the functions are called only when the handle (mouse click) is released.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the

HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*zoom_options*, *allow_callback=True*)

Method that updates the state of the widget, if the provided *zoom_options* value is different than *self.selected_values*.

Parameters

- **zoom_options** (*dict*) – The *dict* with the selected options. It must have the following keys:
  - `min` : (*float*) The minimum value (e.g. `0.1`).
  - `max` : (*float*) The maximum value (e.g. `4.`).
  - `step` : (*float*) The zoom step (e.g. `0.05`).
  - `zoom` : (*float*) The zoom value (e.g. `1.`).
- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *slider_width='6cm'*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example

options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **slider_width** (*float*, optional) – The width of the slider

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

ZoomTwoScalesWidget

class menpowidgets.tools. **ZoomTwoScalesWidget** (*zoom_options*, *render_function=None*, *description='Figure scale: '*, *minus_description='fa-search-minus'*, *plus_description='fa-search-plus'*, *continuous_update=False*)

Bases: *MenpoWidget*

Creates a widget for selecting zoom options with a single scale.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the state of the widget, please refer to the *set_widget_state()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **zoom_options** (*dict*) – The *dict* with the default options. It must have the following keys:
  - `min` : (*float*) The minimum value (e.g. `0.1`).
  - `max` : (*float*) The maximum value (e.g. `4.`).
  - `step` : (*float*) The zoom step (e.g. `0.05`).
  - `zoom` : (*float*) The zoom value (e.g. `1.`).
  - `lock_aspect_ratio` : (*bool*) Flag that locks the aspect ratio.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when the index value changes. If `None` , then nothing is assigned.
- **description** (*str*, optional) – The title of the widget.
- **minus_description** (*str*, optional) – The text/icon of the button that zooms_out. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **plus_description** (*str*, optional) – The title of the button that zooms in. If the *str* starts with *'fa-'*, then a font-awesome icon is defined.
- **continuous_update** (*bool*, optional) – If `True` , then the render and update functions are called while moving the zoom slider's handle. If `False` , then the the functions are called only when the handle (mouse click) is released.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

set_widget_state (*zoom_options*, *allow_callback=True*)
Method that updates the state of the widget, if the provided *zoom_options* value is different than *self.selected_values*.

Parameters

- **zoom_options** (*dict*) – The *dict* with the selected options. It must have the following keys:
  - `min` : (*float*) The minimum value (e.g. `0.1`).
  - `max` : (*float*) The maximum value (e.g. `4.`).
  - `step` : (*float*) The zoom step (e.g. `0.05`).
  - `zoom` : (*float*) The zoom value (e.g. `1.`).
  - `lock_aspect_ratio` : (*bool*) Flag that locks the aspect ratio.

- **allow_callback** (*bool*, optional) – If `True` , it allows triggering of any callback functions.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*, *slider_width='6cm'*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

- **slider_width** (*float*, optional) – The width of the slider

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.

- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

1.6.4 Features

DaisyOptionsWidget

class menpowidgets.tools. **DaisyOptionsWidget** (*daisy_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting Daisy options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **daisy_options** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `step` : (*int*) The sampling step (e.g. `1`).
  - `radius` : (*int*) The radius value (e.g. `15`).
  - `rings` : (*int*) The number of rings (e.g. `2`).
  - `histograms` : (*int*) The number of histograms (e.g. `2`).
  - `orientations` : (*int*) The number of orientation bins (e.g. `8`).
  - `normalization` : (*str*) The normalisation method (e.g. `'l1'`).
  - `sigmas` : (*list* or `None`)
  - `ring_radii` : (*list* or `None`)
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute

- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding

to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

DSIFTOptionsWidget

class menpowidgets.tools. **DSIFTOptionsWidget** (*dsift_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting desnse SIFT options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **dsift_options** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `window_step_horizontal` : (*int*) The horizontal window step (e.g. `1`).
  - `window_step_vertical` : (*int*) The vertical window step (e.g. `1`).
  - `num_bins_horizontal` : (*int*) The horizontal number of spatial bins (e.g. `2`).
  - `num_bins_vertical` : (*int*) The vertical number of spatial bins (e.g. `2`).
  - `num_or_bins` : (*int*) The number of orientation bins (e.g. `9`).
  - `cell_size_horizontal` : (*int*) The horizontal cell size in pixels (e.g. `6`).
  - `cell_size_vertical` : (*int*) The vertical cell size in pixels (e.g. `6`).
  - `fast` : (*bool*) Flag for fast approximation.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

HOGOptionsWidget

class menpowidgets.tools. **HOGOptionsWidget** (*hog_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting HOG options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **hog_options** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `mode` : (*str*) `'dense'` or `'sparse'` .
  - `algorithm` : (*str*) `'dalaltriggs'` or `'zhuramanan'` .
  - `num_bins` : (*int*) The number of orientation bins (e.g. `9`).
  - `cell_size` : (*int*) The cell size in pixels (e.g. `8`).
  - `block_size` : (*int*) The block size in cells (e.g. `2`).
  - `signed_gradient` : (*bool*) Whether to use signed gradients.
  - `l2_norm_clip` : (*float*) L2 norm clipping threshold (e.g `0.2`).
  - `window_height` : (*int*) The sliding window height (e.g. `1`).
  - `window_width` : (*int*) The sliding window width (e.g. `1`).
  - `window_unit` : (*str*) The window size unit (e.g. `'blocks'`).
  - `window_step_vertical` : (*int*) The vertical window step (e.g. `1`).
  - `window_step_horizontal` : (*int*) The horizontal window step (e.g. `1`).
  - `window_step_unit` : (*str*) The window step unit (e.g. `'pixels'`)
  - `padding` : (*bool*) Whether to pad the final image.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)
Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)

Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)

Remove trait change handlers of any type for the specified name. If name is not specified, removes all

trait notifiers.

IGOOptionsWidget

class menpowidgets.tools. **IGOOptionsWidget** (*igo_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting IGO options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **igo_options** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `double_angles` : (*bool*) Whether to use the cos and sin of the double angles as well.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)
Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:
- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)
Method that calls the existing *render_function()* callback handler.

Parameters
- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``` is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )
Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None` .

replace_render_function (*render_function*)
Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)
Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

LBPOptionsWidget

class menpowidgets.tools. **LBPOptionsWidget** (*lbp_options*, *render_function=None*)
Bases: *MenpoWidget*

Creates a widget for selecting LBP options.

- The selected values are stored in the `self.selected_values` *trait*.
- To set the styling of this widget please refer to the *style()* method.
- To update the handler callback function of the widget, please refer to the *replace_render_function()* method.

Parameters

- **lbp_options** (*dict*) – The initial options. It must be a *dict* with the following keys:
  - `radius` : (*list*) The radius values list (e.g. `[0,1,2,3]`).
  - `samples` : (*list*) The sampling poitns list (e.g. `[8] * 4`).
  - `mapping_type` : (*str*) The mapping type (e.g. `'u2'`).
  - `window_step_vertical` : (*int*) The vertical window step (e.g. `1`),
  - `window_step_horizontal` : (*int*) The horizontal window step (e.g. `1`)
  - `window_step_unit` : (*str*) The window step unit (e.g. `'pixels'`)
  - `padding` : (*bool*) Whether to pad the final image.
- **render_function** (*callable* or `None` , optional) – The render function that is executed when a widgets' value changes. If `None` , then nothing is assigned.

add_render_function (*render_function*)

Method that adds the provided *render_function()* as a callback handler to the *selected_values* trait of the widget. The given function is also stored in *self._render_function*.

Parameters**render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

add_traits (***traits*)

Dynamically add trait attributes to the Widget.

call_render_function (*old_value*, *new_value*, *type_value='change'*)

Method that calls the existing *render_function()* callback handler.

Parameters

- **old_value** (*int* or *float* or *dict* or *list* or *tuple*) – The old *selected_values* value.
- **new_value** (*int* or *float* or *dict* or *list* or *tuple*) – The new *selected_values* value.
- **type_value** (*str*, optional) – The trait event type.

close ( )

Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)

Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)

Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change``is a dictionary. The change dictionary at least holds a 'type' key. * ``type :`

the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.

- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

remove_render_function ( )

Method that removes the current *self._render_function()* as a callback handler to the *selected_values* trait of the widget and sets `self._render_function = None`.

replace_render_function (*render_function*)

Method that replaces the current *self._render_function()* with the given *render_function()* as a callback handler to the *selected_values* trait of the widget.

Parameters **render_function** (*callable* or `None` , optional) – The render function that behaves as a callback handler of the *selected_values* trait for the *change* event. Its signature can be `render_function()` or `render_function(change)` , where `change` is a *dict* with the following keys:

- `owner` : the *HasTraits* instance
- `old` : the old value of the modified trait attribute
- `new` : the new value of the modified trait attribute
- `name` : the name of the modified trait attribute.
- `type` : `'change'`

If `None` , then nothing is added.

style (*box_style=None*, *border_visible=False*, *border_colour='black'*, *border_style='solid'*, *border_width=1*, *border_radius=0*, *padding=0*, *margin=0*, *font_family=''*, *font_size=None*, *font_style=''*, *font_weight=''*)

Function that defines the styling of the widget.

Parameters

- **box_style** (*str* or `None` (see below), optional) – Possible widget style options:

```
'success', 'info', 'warning', 'danger', '', None
```

- **border_visible** (*bool*, optional) – Defines whether to draw the border line around the widget.
- **border_colour** (*str*, optional) – The colour of the border around the widget.
- **border_style** (*str*, optional) – The line style of the border around the widget.
- **border_width** (*float*, optional) – The line width of the border around the widget.
- **border_radius** (*float*, optional) – The radius of the border around the widget.
- **padding** (*float*, optional) – The padding around the widget.
- **margin** (*float*, optional) – The margin around the widget.
- **font_family** (*str* (see below), optional) – The font family to be used. Example options:

```
'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace',
'helvetica'
```

- **font_size** (*int*, optional) – The font size.
- **font_style** (*str* (see below), optional) – The font style. Example options:

```
'normal', 'italic', 'oblique'
```

- **font_weight** (*See Below, optional*) – The font weight. Example options:

```
'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'
```

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

1.6.5 Webcam

CameraWidget

class menpowidgets.tools. **CameraWidget** (*canvas_width=640*, *hd=True*, **args*)
Bases: `DOMWidget`

Creates a webcam widget.

Parameters

- **canvas_width** (*int*, optional) – The initial width of the rendered canvas. Note that this doesn't actually change the webcam resolution. It simply rescales the rendered image, as well as the size of the returned screenshots.
- **hd** (*bool*, optional) – If `True` , then the webcam will be set to high definition (HD), i.e. 720 x 1280. Otherwise the default resolution will be used.

add_traits (***traits*)
Dynamically add trait attributes to the Widget.

close ( )
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

has_trait (*name*)
Returns True if the object has a trait with the specified name.

observe (*handler*, *names=traitlets.All*, *type='change'*)
Setup a handler to be called when a trait changes.

This is used to setup dynamic notifications of trait changes.

Parameters

- **handler** (*callable*) – A callable that is called when a trait changes. Its signature should be `handler(change)` , where `change```is a dictionary. The change dictionary at least holds a 'type' key. * ``type` : the type of notification. Other keys may be passed depending on the value of 'type'. In the case where type is 'change', we also have the following keys: * `owner` : the HasTraits instance * `old` : the old value of the modified trait attribute * `new` : the new value of the modified trait attribute * `name` : the name of the modified trait attribute.
- **names** (*list, str, All*) – If names is All, the handler will apply to all traits. If a list of str, handler will apply to all names in the list. If a str, the handler will apply just to that name.
- **type** (*str, All (default: 'change')*) – The type of notification to filter by. If equal to All, then all notifications are passed to the observe handler.

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. The dictionary is keyed on the name and the values are the TraitType objects.

The TraitTypes returned don't know anything about the values that the various HasTrait's instances are holding.

The metadata kwargs allow functions to be passed in which filter traits based on metadata values. The functions should take a single value as an argument and return a boolean. If any function returns False, then the trait is not included in the output. If a metadata key doesn't exist, None will be passed to the function.

unobserve (*handler*, *names=traitlets.All*, *type='change'*)
Remove a trait change handler.

This is used to unregister handlers to trait change notificiations.

Parameters

- **handler** (*callable*) – The callable called when a trait attribute changes.
- **names** (*list, str, All (default: All)*) – The names of the traits for which the specified handler should be uninstalled. If names is All, the specified handler is uninstalled from the list of notifiers corresponding to all changes.
- **type** (*str or All (default: 'change')*) – The type of notification to filter by. If All, the specified handler is uninstalled from the list of notifiers corresponding to all types.

unobserve_all (*name=traitlets.All*)
Remove trait change handlers of any type for the specified name. If name is not specified, removes all trait notifiers.

CHAPTER 2

Usage Example

A short example is often more illustrative than a verbose explanation. Let's assume that you want to quickly explore a folder of numerous annotated images, without the overhead of waiting to load them and writing code to view them. The images can be easily loaded using the Menpo package and then visualized using an interactive widget as:

```
import menpo.io as mio
from menpowidgets import visualize_images

images = mio.import_images('/path/to/images/')
visualize_images(images)
```

Similarly, the fitting result of a deformable model from the MenpoFit package can be demonstrated as:

```
result = fitter.fit_from_bb(image, initial_bounding_box)
result.view_widget()
```

A

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z