secv_guis Documentation

Release 0.1.0

Andres F. R.

Mar 31, 2020

CONTENTS:

CHAPTER

ONE

SECV_GUIS PACKAGE

1.1 Subpackages

1.1.1 secv_guis.bimask_app package

1.1.1.1 Submodules

1.1.1.2 secv_guis.bimask_app.dialogs module

This module contains definitions for different kinds of dialogs and related components that are specific for this application.

class secv_guis.bimask_app.dialogs.**AboutDialog**
Bases: *secv_guis.dialogs.InfoDialog*

Info dialog showing about section

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.dialogs.**InstructionsDialog**
Bases: *secv_guis.dialogs.InfoDialog*

Info dialog showing instructions

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.dialogs.**KeymapsDialog**(*mappings*, *parent=None*)
Bases: *secv_guis.dialogs.FlexibleDialog*

Info dialog showing keymap list

setup_ui_body(*widget*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.dialogs.**SaveWarningDialog**
Bases: *secv_guis.dialogs.InfoDialog*

A dialog to be prompted when trying to delete unsaved changes. Usage example:

```
self.dialog = SaveWarningDialog()
user_wants_to_remove = bool(self.dialog.exec_())
...
```

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.dialogs.**SavedInfoDialog**(*save_dict*, *timeout_ms=500*)
Bases: *secv_guis.dialogs.InfoDialog*

Informative dialog telling about saved paths.

static save_dict_to_str(*save_dict*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.dialogs.**SavedStateTracker**
Bases: object

Create one of these every time a new state is loaded, call `edited` when the state has been changed, and `saved` when saved.

The `saved` function will optionally show an informative dialog.

Then call `delete` when the state is intended to be deleted. The method makes sure that unsaved changes are only deleted with user's confirmation.

Note for developers:

State machines with callbacks are a classic recipe for spaghetti alla callback inferno. Here we didn't provide a structured way to handle GUI state, so we are applying this as high in the API as possible. It works NOW, but consider restructuring it if it gets in the way.

delete()
Call this when we intend to delete the information that we are tracking. If unsaved changes, it will prompt the user to continue.

edit()
Call this any time the state that we want to track has been edited

save(*saved_dict=None*, *ok_dialog_ms=1000*)
Call this any time the state that we want to track has been saved

1.1.1.3 secv_guis.bimask_app.main_window module

This module contains the logic and widgets pertaining to the main window of the bimask app: An app that allows displaying an image, editing a mask on it and also displaying/editing a preannotation mask.

It can be used to efficiently annotate large images with pixel precision. Check instructions.txt for more details.

class secv_guis.bimask_app.main_window.**CrackAnnotPaintForm**(*main_window*, *brushes*, *max_brush_size=100*, *parent=None*, *thresh_min=0*, *thresh_max=1*, *thresh_num_steps=100*)
Bases: *secv_guis.base_widgets.MaskPaintForm*

A `MaskPaintForm` that holds a reference to the app's main window and connects its callbacks with the main window's corresponding components.

brush_size_changed(*sz*)
Setter

brush_type_changed(*idx*)
Setter

button_pressed(*but*)
Setter

rgba_box_changed(*idx*, *r*, *g*, *b*, *a*)
Update corresponding mask with new RGBA color.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

threshold_slider_changed(*idx*, *val*)

Parameters

- **idx** (*int*) – The mask index. 0 is the index of the preannotation, 1 for annotation.
- **val** – The new p-value.

Update preannotation mask with new p-value by calling the `change_preannot_val` method of the view. Only works if idx is 0.

class secv_guis.bimask_app.main_window.**FileLists**(*parent=None, img_extensions=['.png', '.jpg', '.jpeg'], mask_extensions=None, preannot_extensions=None*)

Bases: `PySide2.QtWidgets.QWidget`

A cluster of 3 file lists: one for images, one for masks and one for preannotations.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.main_window.**IntegratedDisplayView**(*main_window*, *scale_percent=15*)

Bases: *secv_guis.masked_scene.DisplayView*

This class implements the main component of the main window: it features a view of the image and the masks, together with a set of operations that can be done on them (painting, updating...), and the callback mechanisms to trigger those operations.

change_annot_rgba(*rgba*)
Updates the annot mask color.

change_preannot_pval(*keep_p_value*, *discard_p_value=0.5*)
Updates the preannot->mask threshold.

change_preannot_rgba(*rgba*)
Updates the preannot mask color.

mask_from_path(*mask_path*, *rgba*)

Parameters

- **mask_path** – Path to an image containing a binary mask, where zero pixels are considered false and non-zero true.
- **rgba** – Color of the loaded mask

Loads a binary mask into the scene as an RGBA-colored mask.

new_image(*img_path*, *initial_mask_color=(219, 54, 148, 150)*, *initial_preannot_color=(102, 214, 123, 100)*)
If successful, removes all elements from the scene and the undo stack, and loads a fresh image and masks. If there are unsaved changes, a dialog asking for confirmation will pop up.

Returns True if the action completed successfully, False if the user decides to abort.

on_left_press(*event*)
Callback implementation, calls `paint_scene`

on_left_release(*event*)
If there is an open macro command, closes it and adds it to the undo stack

on_move(*event*, *has_left*, *has_mid*, *has_right*, *this_pos*, *last_pos*)
Callback implementation, calls `paint_scene` if moving while pressing left.

paint_scene(*x*, *y*)
Paint to the currently selected mask, with the currently selected brush type, at the given position. The given `x, y` position is in 'scene coordinates', i.e. the position from a mouse event has to be translated as follows:

```
xpos, ypos = self.mapToScene(event.pos()).toTuple()
self.paint_scene(xpos, ypos)
```

preannot_from_path(*preannot_path*, *rgba*, *keep_p_value=0.05*, *discard_p_value=0.5*, *normalize=False*)
This method is prototype-ish: It loads an `.npz` file with and 'entropy' field, expected to have a numpy float matrix with same shape as the image.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.main_window.**IntegratedSaveForm**(*main_window*, *default_path=None*, *save_dialog_timeout_ms=1000*)
Bases: *secv_guis.base_widgets.SaveForm*

A `SaveForm` that implements this app's logic, namely, it features 2 masks, one for annot and one for preannot, and saves them as B&W png.

save_masks(*states*, *suffixes*, *overwrite*)
Overriden method that we don't call directly. See `SaveForm` for interface details.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.bimask_app.main_window.**MainWindow**(*parent=None*, *initial_mask_color=(255, 54, 76, 150)*, *initial_preannot_color=(102, 214, 123, 100)*, *max_brush_size=200*)
Bases: `PySide2.QtWidgets.QMainWindow`

This is the central widget for the bimask application. It is a composition of all the used elements, together with the logic that binds them.

DISCARD_P_VALUE = 0.5

ERASER_TXT = 'Eraser'

MASKED_PAINTER_TXT = 'Masked painter'

PAINTER_TXT = 'Painter'

THRESH_MAX = 1e-09

THRESH_MIN = 1e-07

THRESH_NUM_STEPS = 400

keymaps()

Returns A dictionary in the form `name: QtGui.QKeySequence`, where the

Define this GUI's specific key mappings. Note that this method can be overriden to return a different mapping, but the `name``s have to remain identical, in order to be recognized by ``_add_keymaps`.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

wheelEvent(*event*)

The `DisplayView` has zoom functionality associated to the wheel. Here we associate 'brush size change' functionality when the wheel is rolled while pressing Control.

`secv_guis.bimask_app.main_window.`**exp_lambda_estimator**(*elts*)

Parameters **elts** – A collection of elements (can also be numpy).

Given a set of elements, assumed to be exponentially distributed, returns the unbiased estimator for the lambda parameter of the exp distribution.

`secv_guis.bimask_app.main_window.`**exp_threshold**(*keep_p_value*, *lmbd*)

Parameters

- **keep_p_value** – Scalar in range `(0, 1]`
- **elts** – collection of values, assumed to be sampled from an exponential distribution.

Returns A threshold `t`, so that the integral for `exp(lambda)` from `t` to infinity equals `keep_p_value`.

This function assumes that the given `elts` have been sampled from an exponential distribution. Then inferes lambda, using the unbiased ML estimator (see https://en.wikipedia.org/wiki/Exponential_distribution) and returns the threshold `t` that fulfills `keep_p_value` for the distribution above `t`.

`secv_guis.bimask_app.main_window.`**pmap_to_mask**(*pmap*, *keep_highest_pval=0.05*, *discard_lowest_pval=0.5*)

This method performs the following steps:

1. **Assuming that pmap values are exponentially distributed, extracts the** unbiased lambda parameter and the cumulative distribution.
2. Applies threshold to the given low/high p-values

This method uses the standard connected component extraction mechanism in Python, i.e. `scipy.ndimage.measurements.label` and `skimage.measure.regionprops`.

Parameters

- **pmap** – A float array of shape `h, w`.
- **keep_highest_pval** – The p-value designing the amount of top 'pmap' scores to be surely kept.
- **discard_lowest_pval** – The p-value designing the amount of bottom 'pmap' scores to be surely discarded.

Returns The output mask

1.1.1.4 Module contents

1.2 Submodules

1.3 secv_guis.base_widgets module

This module is a library of reusable, extendable widgets.

class secv_guis.base_widgets.**CheckBoxGroup**(*parent=None*, *horizontal=False*)
Bases: PySide2.QtWidgets.QWidget

A group of CheckBox es

add_box(*name*, *tristate=False*, *initial_val=True*)

Parameters **tristate** (*bool*) – If true, the added check box will have 3 states.

remove_box(*idx*)

Parameters **idx** (*int*) – Boxes are added in increasing index order, so this the lower this index the 'older' the box that is being removed.

state()

Returns A list with all the current states, in index order.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.base_widgets.**FileList**(*label*, *parent=None*, *default_path=None*, *extensions=None*, *sort=True*)
Bases: PySide2.QtWidgets.QWidget

A file dialog button followed by a list that shows the files in the selected folder.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_path(*dirname*)

Parameters **dirname** (*str*) – The new directory path to be listed.

class secv_guis.base_widgets.**MaskPaintForm**(*brush_names*, *max_brush_size=100*, *parent=None*, *thresh_min=0*, *thresh_max=1*, *thresh_num_steps=100*, *min_alpha=1*, *max_alpha=255*)
Bases: PySide2.QtWidgets.QWidget

This widget contains one section for the masks and one for the painter. The mask section contains a set of elements, one per mask. A radio button selects the currently active mask, and each mask features an RGBA box and a thershold slider. The painter section contains a ComboBox to select the painter type, and a slider for the painter size.

To use it in specific applications override `button_pressed`, `combo_box_changed`, `rgba_box_changed...`

add_item(*name*, *rgba*, *slider_visible=True*, *activate=False*)
Add an element to the 'mask' section, with the given name and color.

Parameters

- **slider_visible** – If false, the slider will be still there but hidden.
- **activate** – Once created, select this item in the radio buttons.

brush_size_changed(*sz*)
Override me!

brush_type_changed(*idx*)
Override me!

Parameters **idx** (*int*) – Starts with 0 and respects ordering given at construction. So when overriding this method, you can assume that 0 will correspond to the firstly added element, and so on.

button_pressed(*but*)
Override me!

Parameters **idx** (*int*) – Starts with 0 and respects ordering given at construction. So when overriding this method, you can assume that 0 will correspond to the firstly added element, and so on. Implementation example:

```
i = self._buttons.index(but)
print("button pressed: >>>", i, but.text())
```

remove_item(*idx*)
Remove an element from the 'mask' section by index. Indexes are in increasing order, so lowest is oldest.

rgba_box_changed(*idx*, *r*, *g*, *b*, *a*)
Override me!

slider_to_p_val(*sl_val*)
Since the slider goes from 0 to `thresh_num_steps`, this function linearly interpolates the, so that 0 maps to `thresh_min` and `thresh_num_steps` maps to `thresh_max`. Note that min does not neccesarily have to be smaller than max.

Parameters **sl_val** (*int*) – The actual slider value from 0 to num_steps.

Returns The converted and interpolated value.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

threshold_slider_changed(*idx*, *val*)
Override me!

class secv_guis.base_widgets.**RGBASpinbox**(*initial_rgba=(0, 255, 0, 100)*, *parent=None*, *min_alpha=1*, *max_alpha=255*)
Bases: `PySide2.QtWidgets.QWidget`

A cluster of 4 [0-255] spin boxes, representing (and having) an RGBA color. Use `self.connect` to wire this widget to any method.

connect(*fn*)

Parameters **fn** – A function to connect this widget to. It must have the following signature''fn(idx, r, g, b, a)''.

When calling `self.connect(f)`, any value changes in R, G, B or A will trigger `f(r, g, b, a)` with the changed values

get_current_rgba()

Returns Current state as `(r, g, b, a)` numeric tuple.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.base_widgets.**SaveForm**(*parent=None*, *default_path=None*)
Bases: `PySide2.QtWidgets.QWidget`

A formulary providing functionality for selecting what to save, where to save, the output suffix and overwriting policy.

DIALOG_TEXT = 'Output\nfolder'

OVERWRITE_TEXT = 'Overwrite\nsaved'

SAVE_TEXT = 'Save\nselected'

add_checkbox(*checkbox_name*, *initial_val=True*, *initial_txt=None*)
Adds an element that can be selected to be saved.

Parameters

- **checkbox_name** – The element identifier
- **initial_txt** – The initial suffix to be appended to the files. If none is given, the `checkbox_name` is picked as default. The user can change this from the GUI.

save_masks(*states*, *suffixes*, *overwrite*)

Parameters

- **states** – A list with booleans, representing the checkbox states for the contained elements.
- **suffixes** – A list with the corresponding suffixes
- **overwrite** – A boolean determining whether the 'overwrite' checkbox has been activated.

Override me!

staticMetaObject = <PySide2.QtCore.QMetaObject object>

1.4 secv_guis.commands module

This module contains all the 'undoable' actions. They must implement a way to undo and redo them.

Composite commands deserve a special mention: they are trains of actions that only track, store and report the initial and final state. They are particularly useful when performing interactive editings on big datastructures like pixmaps, to prevent memory bloating.

class secv_guis.commands.**CompositeCommand**(*parent=None*)

Bases: `PySide2.QtWidgets.QUndoCommand`

In some cases like painting a stroke into a pixmap, it doesn't make sense to store every single update: rather, the prior and finished states only. This class provides a structure for such cases:

1. **Instantiate the command with the parameters that belong to the whole** composite action.
2. Call `action` for every desired update of the finished state
3. **Call `finish` to crystalize the final state. No further `action` s** will be allowed, and (optionally) the action will be added to a Qt `UndoStack`.

The following is required to extend the class: 1. Define a `COMMAND_NAME` 2. Extend the `__init__`, `action`, `undo` and `redo` methods.

COMMAND_NAME = NotImplemented

action()

Extend me!

finish(*undo_stack=None*)

Parameters **undo_stack** – If given, this command will be added to the stack, wich then allows to undo/redo.

Call this function once you are done with the `action` s. Once finish is called, no more `action` s are possible, so that the undo/redo actions stay frozen.

class secv_guis.commands.**DrawCommand**(*pmi*, *rgba*, *diameter*, *comp_mode=PySide2.QtGui.QPainter.CompositionMode.CompositionMode_So*, *parent=None*)

Bases: *secv_guis.commands.CompositeCommand*

A composite command to draw a stroke of circles into a `PixmapIten`.

COMMAND_NAME = 'Draw'

action(*x_pos*, *y_pos*)
Once the object has been constructed **and ‘‘finish()‘‘ hasn’t been called yet**, Call this function to paint a circle at given position. Check constructor for further variables.

finish(*undo_stack=None*)
Usually we don’t override `finish`, but since pixmaps are so big, we don’t want to store the command if original and final are equal.

redo()
This function implements the interface for the UndoStack. Don’t call this directly.

undo()
This function implements the interface for the UndoStack. Don’t call this directly.

class secv_guis.commands.**DrawOverlappingCommand**(*pmi*, *ref_pmi*, *rgba*, *diameter*, *comp_mode=PySide2.QtGui.QPainter.CompositionMode.Comp*, *parent=None*)

Bases: *secv_guis.commands.DrawCommand*

Like `DrawCommand`, but accepts 2 PixmapItems instead of one, so that the drawing onto the first is only allowed if the same pixel is active in the second.

COMMAND_NAME = 'Draw Overlapping'

action(*x_pos*, *y_pos*)
Paint a circle on `pmi` at given position, masked by `ref_pmi`.

class secv_guis.commands.**EraseCommand**(*pmi*, *diameter*, *comp_mode=PySide2.QtGui.QPainter.CompositionMode.Comp*, *parent=None*)

Bases: *secv_guis.commands.DrawCommand*

A composite command to erase a stroke of circles into a `PixmapIten`. See `DrawCommand` docstrings for more info.

COMMAND_NAME = 'Erase'

1.5 secv_guis.dialogs module

This module defines several reusable dialog types.

class secv_guis.dialogs.**ExceptionDialog**(*error_msg*, *timeout_ms=None*, *parent=None*)

Bases: *secv_guis.dialogs.InfoDialog*

This class is intended to be used at the main loop level, to catch any exceptions that the app may have and show them in a Dialog. To do that, it suffices to put the following line anywhere before `app.exec_()`:

```
sys.excepthook = ExceptionDialog.excepthook
```

Source: https://stackoverflow.com/a/55819545/4511978

DEACTIVATE = False

ERROR_TXT = '\nERROR!\nIf you think this is an app error consider reporting the follow

classmethod excepthook(*exc_type*, *exc_value*, *exc_tb*)
Set this method as `sys.excepthook = <THIS_CLASS>.excepthook` somewhere before `app.exec_()` to wrap all Python exceptions with this dialog.

on_reject()
If the user presses on don't show errors again, the whole class gets deactivated, so further created instances won't pop up.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.dialogs.**FlexibleDialog**(*accept_button_name=None*, *reject_button_name=None*, *timeout_ms=None*, *parent=None*)
Bases: `PySide2.QtWidgets.QDialog`

Dialog class that allows for OK, Yes/No, and timeout interactions. To extend this dialog class, override `setup_ui_body`, `on_accept` and `on_reject`, store the instance and call it with `show` or `exec_`.

Note that `setup_ui_body` is being called IN the constructor, so any variables that it may need when extending the class need to be set before `super().__init__` is called.

As it can be seen here, https://stackoverflow.com/questions/56449605/pyside2-qdialog-possible-bug

implementing a Dialog in PySide2 is a little tricky. These are some things to consider:

- Do not implement `accept, reject` directly. Rather, connect the buttons to `accept, reject`, and then connect the `accepted, rejected` signals to custom methods (in this case `on_accept, on_reject`).
- When calling the Dialog from the main window, the dialog must be persistently stored as a field of the main window `i.e. self.d = ...`. Otherwise it will not show up. Then it can be called in modal or modeless way, as follows: `XXX.connect(self.d.show), ...(self.d.exec_)`.

TIMEOUT_LBL_TXT = 'Closing in {} seconds...'

exec_(**args*, ***kwargs*)
Start the dialog in 'exclusive' way, blocking the rest of the app.

on_accept()
This method will be called if the user presses the (optional) accept button.

on_reject()
This method will be called if the user presses the (optional) reject button.

setup_ui_body(*widget*)
Populate the widget with your desired contents. The widget will be above the buttons.

show(**args*, ***kwargs*)
Start the dialog in parallel to the rest of the app.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class secv_guis.dialogs.**InfoDialog**(*header*, *message*, *accept_button_name=None*, *reject_button_name=None*, *timeout_ms=None*, *parent=None*, *print_msg=True*, *header_style='font-weight: bold; color: black'*)
Bases: *secv_guis.dialogs.FlexibleDialog*

A type of dialog that shows a header and body strings.

INTERACT_BODY = True

INTERACT_HEADER = False

exec_(**args*, ***kwargs*)

setup_ui_body(*widget*)

show(**args*, ***kwargs*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

1.6 secv_guis.masked_scene module

This module contains the `QGraphicsScene+QGraphicsView` binomial, typically used in Qt apps to display and navigate images, together with some specific functionality to annotate.

class secv_guis.masked_scene.**DisplayView**(*scene=None*, *parent=None*, *scale_percent=15*)

Bases: *secv_guis.mouse_event_manager.MouseEventManager*, PySide2.QtWidgets.QGraphicsView

In Qt applications, it is usual to wrap a scene with a view. This allows to dynamically and easily change the perspective on the scene.

fit_in_scene()
Moves perspective so that the whole scene can be seen in view.

on_mid_press(*event*)
Override me

on_move(*event*, *has_left*, *has_mid*, *has_right*, *this_pos*, *last_pos*)
Override me

on_right_press(*event*)
Override me

on_right_release(*event*)
Override me

on_wheel(*event*, *has_ctrl*, *has_alt*, *has_shift*)
Override me

shift_view(*delta_x*, *delta_y*)
Move perspective to shift through the scene

staticMetaObject = <PySide2.QtCore.QMetaObject object>

zoom(*pos_x*, *pos_y*, *zoom_out=False*)
Source for wheel zoom: https://stackoverflow.com/a/29026916/4511978

class secv_guis.masked_scene.**MaskedImageScene**(*img_arr=None*, *parent=None*)

Bases: PySide2.QtWidgets.QGraphicsScene

Basic area that allows to display a color image, together with a set of binary masks on top of it.

DEFAULT_MASK_ALPHA = 100

add_mask_pmi(*mask_arr*, *rgba=None*, *item_on_top=None*)

Parameters

- **mask_arr** – A `np.bool(h, w)` array.
- **item_on_top** – If given, mask will be added underneath that item. Otherwise will be added on top of item stack.

Returns The added `PixmapItem`.

items(*ascending=True*)

Returns This scene's items.

Parameters **ascending** – If true, items are given from background to foreground.

num_items()
Number of items in this scene, ordered from foreground to background.

pixmap_item_to_np_mask(*pmi*, *pixmap_format=PySide2.QtGui.QImage.Format.Format_RGBA8888*)
Converts given pixmap to np.bool, assuming all non-zero values are True. Returns the bool array.

remove_mask_pmi(*pmi*)

Parameters **pmi** – The PixmapItem to remove. It has to be a mask added via `add_mask`

replace_mask_pmi(*pmi*, *new_mask_arr*, *new_rgba=None*)
If we call `remove_mask` and then `add_mask` fails, we will lose the removed mask forever. This method updates the mask in an atomary way: either succeeds or does nothing.

> **Warning:** The input `pmi` gets removed from the scene and the reference becomes invalid. Use the reference returned by this function instead.

save_mask_as_image(*pmi*, *outpath*, *overwrite_existing=False*, *verbose=True*)
Output: RGB PNG image where false is black (0, 0, 0) and true is white (255, 255, 255).

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_image(*img_arr*)
Clears whole scene, and adds the given numpy array as Pixmap.

Parameters **img_arr** – A `np.uint8(h, w [, ?])` array.

1.7 secv_guis.mouse_event_manager module

This module contains a convenience mixin that provides the following functionality for mouse tracking:

- Record previous mouse states
- Demultiplex mouse events and preprocess relevant informations

To make a widget responsive to mouse events, simply this class there and override the desired methods.

class secv_guis.mouse_event_manager.**MouseEventManager**(*track=True*)
Bases: `object`

Extend this class and then instantiate it once as a property in your desired widget.

> **Warning:** The Mixin is compatible with multiple inheritance, but not all initializations work. The following does:
>
> **class A(MouseEventManager, QtWidgets.XXX):**
>
> **def __init__(self, ...):** QtWidgets.XXX.__init__(self, ...) MouseEventManager.__init__(self, ...)

In this example, class `A` will respond to the overriden `on_move`, etc... methods.

mouseMoveEvent(*event*)

mousePressEvent(*event*)
Wheel click event handler

mouseReleaseEvent(*event*)
Wheel release event handler

on_left_press(*event*)
Override me

on_left_release(*event*)
Override me

on_mid_press(*event*)
Override me

on_mid_release(*event*)
Override me

on_move(*event*, *has_left*, *has_mid*, *has_right*, *this_pos*, *last_pos*)
Override me

on_right_press(*event*)
Override me

on_right_release(*event*)
Override me

on_wheel(*event*, *has_ctrl*, *has_alt*, *has_shift*)
Override me

wheelEvent(*event*)
Override me. This is a simple wrapper, but may include functionality like storing positions if needed.

1.8 secv_guis.utils module

This module contains helper functions and utilities that may be used anywhere else in the project.

class secv_guis.utils.**RandomColorGenerator**(*seed=None*)
Bases: randomcolor.RandomColor

Flexible generator for nice random colors. For more details check https://pypi.org/project/randomcolor/

Usage example:: r, g, b = next(RandomColorGenerator().generate(form="rgbArray"))

generate(*hue=None*, *luminosity=None*, *count=1*, *form='rgbArray'*)

Parameters **form** – Popular ones: `rgbArray, rgba, hex, rgb`

Returns A generator with `count` random colors.

Overriden to return a generator instead of a list. Source: https://github.com/kevinwuhoo/randomcolor-py

secv_guis.utils.**bool_arr_to_rgba_pixmap**(*arr*, *rgba=(255, 0, 0, 255)*)

Parameters

- **arr** – Expects a `np.bool(h, w)` array.
- **rgba** – 4 values between 0 and 255. Alpha=255 means full opacity.

Returns A `QtGui.QPixmap` in format `RGBA8888(w, h)`, where the `false` values are all zeros and the `true` values have the specified `rgba` color.

secv_guis.utils.**load_exif**(*img_path*)

Returns A dictionary with the EXIF data contained at `img_path`.

`secv_guis.utils.`**`load_img_and_exif`**(*img_path: str*, *as_np_array=True*, *ignore_alpha=True*)

Loads the image at given path using PIL, and its EXIF data. If the EXIF data contains extra info about orientation, also rotates the image accordingly.

Parameters

- **`as_np_array`** – If true, the image will be converted from PIL format to np via `np.asarray(image)`
- **`ignore_alpha`** – If the type of the image is `RGBA`, it will be converted to `RGB`.

Returns A tuple `(image, exif_dict)`.

Inspired in https://stackoverflow.com/a/26928142

`secv_guis.utils.`**`pixmap_to_arr`**(*pm*, *img_format=PySide2.QtGui.QImage.Format.Format_RGBA8888*)

Parameters

- **`pm`** – A Pixmap to be converted
- **`img_format`** – The `QtGui.QImage` format that `pm` corresponds to. https://doc.qt.io/qtforpython/PySide2/QtGui/QImage.html#image-formats

Returns A `np.uint8(h, w, C)` array, where the number of channels `C` depends on the image format.

..note:: Pixmaps are in format (w, h, ...) but arrays are returned in `(h, w, ...)`, as usual for numpy

`secv_guis.utils.`**`rgb_arr_to_rgb_pixmap`**(*arr*)

Parameters **`arr`** – Expects a `np.uint8(h, w, 3)` array.

Returns A `QtGui.QPixmap` in format `RGB888(w, h)`.

`secv_guis.utils.`**`unique_filename`**(*path*, *suffix='_({})'*, *max_iters=10000*)

Given a path, returns the same path if unique, or adds `(N)` before the extension to make it unique, for `N` being the lowest integer possible starting from 1.

1.9 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

F

G

I

K

L

M

N

O

P

R

S

T

U

W

Z