

---

# **joystick Documentation**

***Release 0.1.2***

**Guillaume Schworer**

September 27, 2016



## CONTENTS

<b>1</b>	<b>joystick package</b>	<b>3</b>
1.1	Submodules . . . . .	3
1.2	joystick.core module . . . . .	3
1.3	joystick.deco module . . . . .	3
1.4	joystick.example module . . . . .	3
1.5	joystick.frame module . . . . .	3
1.6	joystick.graph module . . . . .	4
1.7	joystick.joystick module . . . . .	5
1.8	joystick.text module . . . . .	5
1.9	Module contents . . . . .	5
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>







## JOYSTICK PACKAGE

### 1.1 Submodules

### 1.2 joystick.core module

### 1.3 joystick.deco module

`joystick.deco.deco_infinite_loop(wait_time=0.5)`

This decorator creates a daemon-thread to call a decorated joystick method in an infinite loop every `wait_time` seconds, as long as the `joystick.running` attribute is `True`, or until the end of the universe, whichever is first.

This is a self-aware decorator, recording all function names decorated with itself, such that all threads can be launched simultaneously with the `joystick.start()` method.

However, it must be initialized at run-time before use:

```
>>> class yuhu(joystick.Joystick):
>>>     _infinite_loop = joystick.deco_infinite_loop()
>>>     ...
(the reason is that it must get a memory copy of the decorator
function in order to record the decorated functions in the
desired scope, not in the package import scope.
In short, just initilize it as above and it will work)
```

It then can be used normally:

```
>>> @_infinite_loop(wait_time=0.5) # in sec
>>> def repetitive_task():
>>>     print("Next time I'm done I swear.")
```

### 1.4 joystick.example module

### 1.5 joystick.frame module

```
class joystick.frame.Frame(daddy, name, freq_up, pos=(50, 50), size=(400, 400),
                           screen_relative=False, **kwargs)
```

Bases: object

**exit()**  
Terminates the frame

**freq\_up**  
Update frequency (Hz) of the frame. Set between 1e-3 and 1e3 Hz, or `None` for no updating

**reinit(\*\*kwargs)**  
Re-initializes the frame, i.e. closes the current frame if necessary and creates a new one. Uses the parameters of initialization by default or anything provided through `kwargs`. See [Frame](#) for the description of input parameters.

**running**  
Returns `True` if the frame should update Set to `True/False` to start/stop the updating

**start()**  
Starts updating the frame

**stop()**  
Stops updating the frame

**typ**  
Returns the type of the frame, e.g. `Graph`. Read-only.

**visible**  
Returns `True` if the frame has not been closed Read-only.

## 1.6 joystick.graph module

**class** `joystick.graph.Graph`(*daddy, name, freq\_up=1, pos=(50, 50), size=(400, 400), screen\_relative=False, xnpts=30, fmt='ro-', bgcol='w', axrect=(0.1, 0.1, 0.9, 0.9), grid='k', xlim=(0.0, None, 0.0, None), xnptsmax=50, \*\*kwargs*)

Bases: `joystick.frame.Frame`

**get\_xydata()**  
Returns the x and y data of the graph

**get\_xlim()**  
Returns the (xmin, xmax, ymin, ymax) limits of the graph

**reinit(\*\*kwargs)**  
Re-initializes the frame, i.e. closes the current frame if necessary and creates a new one. Uses the parameters of initialization by default or anything provided through `kwargs`. See [Graph](#) for the description of input parameters.

**set\_xydata(x, y)**  
Sets the x and y data of the graph. Give x and y vectors as numpy arrays; only the last `self.xnpts` data-points will be displayed

**set\_xlim(xlim=(None, None, None, None))**  
Sets the (xmin, xmax, ymin, ymax) limits of the graph. Set one or several values to `None` to auto-adjust the limits of the graph to its x- or y-data.

**show()**  
Updates the graph

**xnpts**

**xnptsmax**



## 1.7 joystick.joystick module

```
class joystick.joystick.Joystick (**kwargs)
    Bases: object

    exit()
        Terminates the simulation

    running
        Returns True if the simulation is running Set to True/False to start/stop the simulation

    start()
        Starts the simulation

    start_frames()
        Turns on the updating of all frames, keeps the simulation as it was, running or not

    stop()
        Stops the simulation and all frames

    stop_frames()
        Stops all frames from updating, the simulation continues running
```

## 1.8 joystick.text module

```
class joystick.text.Text (daddy, name, freq_up=1, pos=(50, 50), size=(400,
400), screen_relative=False, background='black', foreground='green', rev=True, font=('consolas', 11),
mark_line=True, mark_fmt='%H:%M:%S > ', scrollbar=True,
**kwargs)
    Bases: joystick.frame.Frame

    add_text (txt='', end=None, newline=True, mark_line=None)
        Adds the text txt to the frame, on a newline if newline is True. The new txt
        is prepended using the format in self.mark_fmt if mark_line is True, default is
        self.mark_line. It is added at the end of the frame text if rev is True, default is not
        self.rev.

    clear()
        Flushes the text in the frame

    reinit (**kwargs)
        Re-initializes the frame, i.e. closes the current frame if necessary and creates a new one. Uses
        the parameters of initialization by default or anything provided through kwargs. See Text for
        the description of input parameters.

    show()
        Updates the text
```

## 1.9 Module contents

**Name** joystick

**Website** <https://github.com/ceyzeriat/joystick>

**Author** Guillaume Schworer

**Version** 0.1

Joystick provides a light-weight and simple framework to real-time data-plotting and logging, while the console remains accessible to manage the on-going simulation and data acquisition.

In some ways, this framework can replace a Graphical User Interface (GUI) on many projects, as long as 1) the user is comfortable enough with managing the simulation using command-lines, and 2) the display of the real-time data is not too complex.

Allright. Let's say you have some data-stream (serial port, web scraping, on-going simulation or experiment, etc), and you would like to plot or log in real-time whatever is happening. In addition you would also like to send commands to interact with the mechanisms producing the data... without having to build a GUI (which looks pretty to your boss, but is time-consuming both in initial design and maintenance).

Then, this package is for you.

Note that Joystick is based on Tkinter to display frames of text or graph, and that it is released under the GNU General Public License v3 or later (GPLv3+).

Straight to the point: check-out this example. It generates fake random data (ydata) between 0 and 1.05 every 0.2 second, displayed as a function of time in a graph-frame. Whenever there is a datapoint above 1, it drops a warning in the text-frame.

```
import joystick as jk
import numpy as np
import time

class test(jk.Joystick):
    # initialize the infinite loop decorator
    _infinite_loop = jk.deco_infinite_loop()

    def __init__(self, *args, **kwargs):
        """
        Function called at initialization, don't bother why for now
        """
        self._t0 = time.time() # initialize time
        self.xdata = np.array([self._t0]) # time x-axis
        self.ydata = np.array([0.0]) # fake data y-axis
        # create a graph frame
        self.mygraph = jk.Graph(daddy=self, name="test", size=(500, 500),
                                pos=(50, 50), fmt="go-", xnpts=15,
                                freq_up=7, bgcol="y", xylim=(0,10,0,1))
        # create a text frame
        self.mytext = jk.Text(daddy=self, name="Y-overflow",
                              size=(500, 250), pos=(600, 50), freq_up=1)

    @_infinite_loop(wait_time=0.2)
    def _generate_fake_data(self): # function looped every 0.2 second
        """
        Loop starting with simulation start, getting data and
        pushing it to the graph every 0.2 seconds
        """
        # concatenate data on the time x-axis
        self.xdata = jk.core.add_datapoint(self.xdata,
                                            time.time(),
                                            xnptsmax=self.mygraph.xnptsmax)
        # concatenate data on the fake data y-axis
        self.ydata = jk.core.add_datapoint(self.ydata,
                                            np.random.random()*1.05,
```

```

                                xnptsmax=self.mygraph.xnptsmax)
    # check overflow for the last data point added
    if self.ydata[-1] > 1:
        # send warning to the text-frame
        self.mytext.add_text('Some data bumped into the ceiling: '
                              '{:.3f}'.format(self.ydata[-1]))

    # prepare the time axis
    t = np.round(self.xdata-self._t0, 1)
    # push new data to the graph
    self.mygraph.set_xydata(t, self.ydata)

t = test()
t.start()

```

Now you should see a ‘snake’ going through the graph-frame, but after 10 seconds it is gone (that was on purpose, for the sake of the demo!). Type (line by line):

```

t.mygraph.xnpts = 50
t.mygraph.freq_up = 2
t.mygraph.xyylim = (None, None, 0, 1)

```

Now that should be better, displaying the latest 50 points at a slower pace (twice a second), and the x-axis is auto-adjusting. Here is what it should look like:

Let’s stop and reinitialize the graph with slightly different parameters:

```

t.stop()
t.mygraph.reinit(bgcol='w', axrect=(0,0,1,1), xyylim=(None, None, 0, 1))
t.start()
t.stop()
t.exit()

```

Too easy!

## 1.9.1 Documentation

Refer to this page, <http://pythonhosted.org/joystick/joystick.html>

## 1.9.2 Requirements

Joystick requires the following Python packages:

- NumPy: for basic numerical routines
- matplotlib: for plotting

## 1.9.3 Installation

The easiest and fastest way for you to get the package and run is to install joystick through pip:

```
$ pip install joystick
```

You can also download joystick source from GitHub and type:

```
$ python setup.py install
```

Dependencies will not be installed automatically. Refer to the requirements section. If you have an anaconda distribution, you will be good to go.

## 1.9.4 Contributing

### Code writing

Code contributions are welcome! Just send a pull request on GitHub and we will discuss it. In the [issue tracker](#) you may find pending tasks.

### Bug reporting

If you think you've found one please refer to the [issue tracker](#) on GitHub.

### Additional options

You can either send me an e-mail or add it to the issues/wishes list on GitHub.

## 1.9.5 Citing

If you use joystick on your project, please *drop me a line* [<mailto:{my first name}.{my family name}@gmail.com>](mailto:{my first name}.{my family name}@gmail.com), you will get fixes and additional options earlier.

## 1.9.6 License

Joystick is released under the GNU General Public License v3 or later (GPLv3+). Please refer to the LICENSE file.

j

- joystick, 5
- joystick.core, 3
- joystick.deco, 3
- joystick.frame, 3
- joystick.graph, 4
- joystick.joystick, 5
- joystick.text, 5



**A**

add\_text() (joystick.text.Text method), 5

**C**

clear() (joystick.text.Text method), 5

**D**

deco\_infinite\_loop() (in module joystick.deco), 3

**E**

exit() (joystick.frame.Frame method), 3

exit() (joystick.joystick.Joystick method), 5

**F**

Frame (class in joystick.frame), 3

freq\_up (joystick.frame.Frame attribute), 4

**G**

get\_xydata() (joystick.graph.Graph method), 4

get\_xylim() (joystick.graph.Graph method), 4

Graph (class in joystick.graph), 4

**J**

Joystick (class in joystick.joystick), 5

joystick (module), 5

joystick.core (module), 3

joystick.deco (module), 3

joystick.frame (module), 3

joystick.graph (module), 4

joystick.joystick (module), 5

joystick.text (module), 5

**R**

reinit() (joystick.frame.Frame method), 4

reinit() (joystick.graph.Graph method), 4

reinit() (joystick.text.Text method), 5

running (joystick.frame.Frame attribute), 4

running (joystick.joystick.Joystick attribute), 5

**S**

set\_xydata() (joystick.graph.Graph method), 4

set\_xylim() (joystick.graph.Graph method), 4

show() (joystick.graph.Graph method), 4

show() (joystick.text.Text method), 5

start() (joystick.frame.Frame method), 4

start() (joystick.joystick.Joystick method), 5

start\_frames() (joystick.joystick.Joystick method), 5

stop() (joystick.frame.Frame method), 4

stop() (joystick.joystick.Joystick method), 5

stop\_frames() (joystick.joystick.Joystick method), 5

**T**

Text (class in joystick.text), 5

typ (joystick.frame.Frame attribute), 4

**V**

visible (joystick.frame.Frame attribute), 4

**X**

xnpts (joystick.graph.Graph attribute), 4

xnptsmax (joystick.graph.Graph attribute), 4