TIDAL API

Revision 1 - Software v2023.2.2

irw

Summary

The following includes a description of the available methods used to interact with the TIDAL system via Python. This is an abstraction used to integrate the GUI with the physical components.

Contents

Lobe class

Instantiate an object of the `Lobe` class with the following:

```
lobe = Lobe('RU') # One lobe
lobes = [Lobe(name) for name in ['RU', 'RM', 'RL', 'LU', 'LL']] # All lobes
```

The available methods are listed below, including an example and description of each.

name

```
lobe.name = 'RU'
print(lobe.name)
```

Get the name of the lobe or rename it.

---

gui_checkbox

```
lobe.gui_checkbox = tk.IntVar()
lobe.gui_checkbox.get()
```

The GUI checkbox associated with the lobe.

---

gui_constant_step_entry

```
lobe.gui_constant_step_entry = tk.Entry()
lobe.gui_constant_step_entry.get()
```

The GUI entry associated with the number of steps in a constant-delay maneuver.

---

gui_constant_delay_entry

```
lobe.gui_constant_delay_entry = tk.Entry()
lobe.gui_constant_delay_entry.get()
```

The GUI entry associated with the step delay in a constant-delay maneuver.

---

gui_inhale_bezier

```
lobe.gui_inhale_bezier = BezierPanel(fr_inhale, plot_title="Inhale")
```

The GUI frame containing the interactive Bezier plot for setting variable delay values for an inhalation maneuver.

---

gui_exhale_bezier

```
lobe.gui_exhale_bezier = BezierPanel(fr_exhale, plot_title="Exhale")
```

The GUI frame containing the interactive Bezier plot for setting variable delay values for an exhalation maneuver.

---

step_count_constant

```
lobe.step_count_constant = 200
```

The number of steps associated with a constant-delay breathing maneuver.

---

step_delay_constant

```
lobe.step_delay_constant = 1000
```

The delay value associated with a constant-delay breathing maneuver.

---

inhale_control_points

```
lobe.inhale_control_points = {'x': inhale_interactor.control_x, 'y': inhale_interactor.control_y}

lobe.gui_inhale_bezier.interactor.control_x = lobe.inhale_control_points['x']
lobe.gui_inhale_bezier.interactor.control_y = lobe.inhale_control_points['y']
```

A `dict` of control points associated with the inhalation Bezier curve.

---

inhale_delays

```
lobe.inhale_delays = [1,2,3,...,700]
```

Integer value list of the delay values used for a variable-delay inhalation maneuver.

---

inhale_delays_formatted

```
print(lobe.inhale_delays_formatted)
```

A pretty-printed list of delay values used for a variable-delay inhalation maneuver.

---

exhale_control_points

```
lobe.exhale_control_points = {'x': exhale_interactor.control_x, 'y': exhale_interactor.control_y}

lobe.gui_exhale_bezier.interactor.control_x = lobe.exhale_control_points['x']
lobe.gui_exhale_bezier.interactor.control_y = lobe.exhale_control_points['y']
```

A `dict` of control points associated with the exhalation Bezier curve.

---

exhale_delays

```
lobe.exhale_delays = [1,2,3,...,700]
```

Integer value list of the delay values used for a variable-delay exhalation maneuver.

---

exhale_delays_formatted

```
print(lobe.exhale_delays_formatted)
```

A pretty-printed list of delay values used for a variable-delay exhalation maneuver.

---

imin

An integer value of the minimum delay value for variable-delay inhalation.

---

imax

An integer value of the maximum delay value for variable-delay inhalation.

---

emin

An integer value of the minimum delay value for variable-delay exhalation.

---

emax

An integer value of the maximum delay value for variable-delay exhalation.

---

step_count_variable

The number of steps associated with a variable-delay breathing maneuver.

---

update_variable_profile_params(self)

```
lobe.update_variable_profile_params()
```

Retrieves and stores the values of `step_count_variable`, `imin`, `imax`, `emin`, `emax`, `inhale_control_points`, and `exhale_control_points` from the GUI.

Calculates and stores the variable delay values for inhalation and exhalation.

---

calculate_relative_flow_delays(self)

```
lobe.calculate_relative_flow_delays()
```

Calculates and stores the variable delay values for inhalation and exhalation, based on the associated Bezier curves.

TIDAL class

Instantiate an object of the TIDAL class with the following:

```
tidal = TIDAL(tsi_port='/dev/ttyUSB0', motor_port='/dev/ttyACM0')
```

The TIDAL object is initialized with the following:

- Log, run, and data directories
- Log file (general-purpose)
- Lobes (RU, RM, RL, LU, LL)
- Motor controller, if port supplied
- Flow meter, if port supplied

All other values are set to None or False by default. These values are populated later by other methods or classes.

The available methods are listed below, including an example and description of each.

---

load(self)

```
tidal.load()
```

Opens a file selection dialog window and sets the default values for the TIDAL instance and GUI objects. Will load files of the .tidal extension.

---

save(self)

```
tidal.save()
```

Opens a file selection dialog window and saves the current values for the TIDAL instance and GUI objects. Will save files of the .tidal extension.

---

set_tsi_port(self, port)

```
tidal.set_tsi_port('COM5')
```

Set the tsi_port variable. If there is not a flow meter associated with the TIDAL object, creates one with the new port. Otherwise, updates the flow meter port.

---

get_tsi_port(self)

```
tidal.get_tsi_port()
```

Returns the port associated with the flow meter for the TIDAL object. Better practice would be to use properties for this purpose.

---

get_tsi(self)

```
tidal.get_tsi()
```

Returns the flow meter associated with the `TIDAL` object. Better practice would be to use properties for this purpose.

---

connect_tsi(self)

```
tidal.connect_tsi()
```

Attempts to connect to the flow meter. Reports errors.

---

disconnect_tsi(self)

```
tidal.disconnect_tsi()
```

Attempts to disconnect from the flow meter. Reports errors.

---

set_motor_port(self, port)

```
tidal.set_motor_port('/dev/ttyACM0')
```

Set the `motor_port` variable. If there is not a motor controller associated with the `TIDAL` object, creates one with the new port. Otherwise, updates the motor controller port.

---

get_motor_port(self)

```
tidal.get_motor_port()
```

Returns the port associated with the motor controller for the `TIDAL` object. Better practice would be to use properties for this purpose.

---

get_motors(self)

```
tidal.get_motors()
```

Returns the motor controller associated with the `TIDAL` object. Better practice would be to use properties for this purpose.

---

connect_motors(self)

```
tidal.connect_motors()
```

Attempts to connect to the motor controller. Reports errors.

---

disconnect_motors(self,)

```
tidal.disconnect_motors()
```

Attempt to disconnect from the motor controller. Reports errors.

set_log_dir(self, dir)

```
tidal.set_log_dir(r'./logs')
```

Set the directory where information on each run will be recorded.

Better to use properties.

---

get_log_dir(self)

```
tidal.get_log_dir()
```

Returns the log directory.

Better to use properties.

---

set_run_id(self, id)

```
tidal.set_run_id('run-2023-10')
```

The run ID is a ~unique identifier for each set of experimental conditions. Can be made with the `unique_id()` function based on a given set of properties.

---

get_run_id(self)

```
tidal.get_run_id()
```

Returns the run id.

Better to use properties.

---

set_run_dir(self, dir)

```
tidal.set_run_dir(r'./logs/run')
```

Set the folder to store logs and data for the current run (usually based on run ID).

Better to use properties.

---

get_run_dir(self)

```
tidal.get_run_dir()
```

Returns the folder for the current run.

Better to use properties.

---

set_data_dir(self, dir)

```
tidal.set_run_dir(r'./logs/run/data')
```

Set the folder to save data for the current run. Associated with the flow meter data recording process.

Better to use properties.

---

get_data_dir(self)

```
tidal.get_data_dir()
```

Returns the folder where data is stored for the current run.

Better to use properties.

---

logger

```
tidal.logger = Logger()
tidal.logger
```

Get and set the `Logger` object associated with the `TIDAL` instance. Used for printing to GUI console and recording printed statements in the general or run-specific log file.

---

update_variable_profile(self)

```
tidal.update_variable_profile()
```

Attempts to update the variable-delay profile settings on the motor controller and reconnect to the controller.

Reports errors.

---

substitute_params(self)

```
tidal.substitute_params()
```

Internal

Writes the motor controller template routine with the updated values for the variable-delay profiles (inhalation and exhalation for each lobe).

---

push_arduino(self)

```
tidal.push_arduino()
```

Internal

Attempts to push the control routine to the motor controller, recording the log information from the Arduino CLI. Performs verification and upload for an Arduino MEGA 2560 at the instance `motor_port`.