opentrons

You are currently viewing the documentation for the Opentrons OT-1. To view documentation for the OT-2, click here.

API Reference

If you are reading this, you are probably looking for an in-depth explanation of API classes and methods to fully master your protocol development skills.

Robot

All protocols are set up, simulated and executed using a Robot class.

class `opentrons.`**`Robot`**

This class is the main interface to the robot.

Through this class you can can:

define your **`opentrons.Deck`**
`connect()` to Opentrons physical robot
`home()` axis, move head (**`move_to()`**)
`pause()` and **`resume()`** the protocol run
set the **`head_speed()`** of the robot

Each Opentrons protocol is a Python script. When evaluated the script creates an execution plan which is stored as a list of commands in Robot's command queue.

Here are the typical steps of writing the protocol:

Using a Python script and the Opentrons API load your containers and define instruments (see **`Pipette`**).
Call **`reset()`** to reset the robot's state and clear commands.
Write your instructions which will get converted into an execution plan.
Review the list of commands generated by a protocol **`commands()`**.
`connect()` to the robot and call **`run()`** it on a real robot.

See **`Pipette`** for the list of supported instructions.

Examples

```
>>> from opentrons import robot, instruments, containers
>>> robot.reset()
<opentrons.robot.robot.Robot object at ...>
>>> plate = containers.load('96-flat', 'A1', 'plate')
>>> p200 = instruments.Pipette(axis='b', max_volume=200)
>>> p200.aspirate(200, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>
>>> robot.commands()
['Aspirating 200 at <Deck><Slot A1><Container plate><Well A1>']
```

`actions`

Return a copy of a raw list of commands in the Robot's queue.

attached to.
instrument (*Instrument*) – An instance of a **Pipette** to attached to the axis.

Notes

A canonical way to add to add a Pipette to a robot is:

```
from opentrons.instruments.pipette import Pipette
p200 = Pipette(axis='a')
```

This will create a pipette and call **add_instrument()** to attach the instrument.

commands()

Access the human-readable list of commands in the robot's queue.

Returns: *A list of string values for each command in the queue, for example* `'Aspirating 200uL at <Deck>/<Slot A1>/<Container plate>/<Well A1>'`

connect(*port=None*, *options=None*)

Connects the robot to a serial port.

Parameters: port (*str*) – OS-specific port name or `'Virtual Smoothie'`
options (*dict*) – if **port** is set to `'Virtual Smoothie'`, provide the list of options to be passed to **get_virtual_device()**

Returns:
Return type: `True` for success, `False` for failure.

containers()

Returns the dict with all of the containers on the deck.

diagnostics()

Access diagnostics information for the robot.

Returns: `axis_homed` — axis that are currently in home position.
`switches` — end stop switches currently hit.
`steps_per_mm` — steps per millimeter calibration values for `x` and `y` axis.

Return type: Dictionary with the following keys

disconnect()

Disconnects from the robot.

get_mosfet(*mosfet_index*)

Get MOSFET for a MagBead (URL).

Parameters: mosfet_index (*int*) – Number of a MOSFET on MagBead.
Returns:
Return type: Instance of **InstrumentMosfet**.

get_motor(*axis*)

opentrons

get_warnings()

Get current runtime warnings.

Returns: Runtime warnings accumulated since the last **run()** or **simulate()**.

head_speed(**args*, ***kwargs*)

Set the XY axis speeds of the robot, set in millimeters per minute

Parameters: rate (*int*) – An integer setting the mm/minute rate of the X and Y axis. Speeds too fast (around 6000 and higher) will cause the robot to skip step, be careful when using this method

Examples

```
>>> from opentrons import robot
>>> robot.connect('Virtual Smoothie')
>>> robot.home()
>>> robot.head_speed(4500)
>>> robot.move_head(x=200, y=200)
```

home(**args*, ***kwargs*)

Home robot's head and plunger motors.

Parameters: *args –
A string with axes to home. For example `'xyz'` or `'ab'`.

If no arguments provided home Z-axis then X, Y, B, A

Notes

Sometimes while executing a long protocol, a robot might accumulate precision error and it is recommended to home it. In this scenario, add `robot.home('xyzab')` into your script.

Examples

```
>>> from opentrons import Robot
>>> robot.connect('Virtual Smoothie')
>>> robot.home()
```

move_to(*location*, *instrument=None*, *strategy='arc'*, ***kwargs*)

Move an instrument to a coordinate, container or a coordinate within a container.

Parameters: location (*one of the following:*) – 1. **Placeable** (i.e. Container, Deck, Slot, Well) — will move to the origin of a container. 2. **Vector** move to the given coordinate in Deck coordinate system. 3. (**Placeable**, **Vector**) move to a given coordinate within object's coordinate system.
instrument – Instrument to move relative to. If `None`, move relative to the center of a gantry.
strategy (*{'arc', 'direct'}*) –
`arc` : move to the point using arc trajectory avoiding

Examples

```
>>> from opentrons import Robot
>>> robot.reset()
<opentrons.robot.robot.Robot object at ...>
>>> robot.connect('Virtual Smoothie')
>>> robot.home()
>>> plate = robot.add_container('96-flat', 'A1', 'plate')
>>> robot.move_to(plate[0])
>>> robot.move_to(plate[0].top())
```

pause()

Pauses execution of the protocol. Use **resume()** to resume

reset()

Resets the state of the robot and clears:

Deck
Instruments
Command queue
Runtime warnings

resume()

Resume execution of the protocol after **pause()**

stop()

Stops execution of the protocol.

Pipette

class opentrons.instruments.**Pipette**(*robot*, *axis*, *name=None*, *channels=1*, *min_volume=0*, *max_volume=None*, *trash_container=None*, *tip_racks=[]*, *aspirate_speed=300*, *dispense_speed=500*)

Through this class you can can:

Handle liquids with **aspirate()**, **dispense()**, **mix()**, and **blow_out()**
Handle tips with **pick_up_tip()**, **drop_tip()**, and **return_tip()**
Calibrate this pipette's plunger positions
Calibrate the position of each **Container** on deck

Here are the typical steps of using the Pipette:

Instantiate a pipette with a maximum volume (uL)

and an axis (*a* or *b*) * Design your protocol through the pipette's liquid-handling commands

Parameters:

axis (*str*) – The axis of the pipette's actuator on the Opentrons robot ('a' or 'b')
name (*str*) – Assigns the pipette a unique name for saving it's calibrations
channels (*int*) – The number of channels on this pipette (Default: *1*)
min_volume (*int*) – The smallest recommended uL volume for this pipette (Default: *0*)
max_volume (*int*) – The largest uL volume for this pipette (Default: *min_volume* + 1)
trash_container (*Container*) – Sets the default location **drop_tip()** will put tips (Default: *None*)
tip_racks (*list*) – A list of Containers for this Pipette to track tips when calling **pick_up_tip()** (Default: [])
aspirate_speed (*int*) – The speed (in mm/minute) the

Return type: A new instance of **Pipette**.

Examples

```
>>> from opentrons import instruments, containers
>>> p1000 = instruments.Pipette(axis='a', max_volume=1000)
>>> tip_rack_200ul = containers.load('tiprack-200ul', 'A1')
>>> p200 = instruments.Pipette(
...     name='p200',
...     axis='b',
...     max_volume=200,
...     tip_racks=[tip_rack_200ul])
```

aspirate(*volume=None*, *location=None*, *rate=1.0*)

Aspirate a volume of liquid (in microliters/uL) using this pipette

Notes

If no *location* is passed, the pipette will aspirate from it's current position. If no *volume* is passed, *aspirate* will default to it's *max_volume*

Parameters: volume (*int or float*) – The number of microliters to aspirate (Default: self.max_volume)
location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the aspirate. Can also be a tuple with first item **Placeable**, second item relative **Vector**
rate (*float*) – Set plunger speed for this aspirate, where speed = rate * aspirate_speed (see **set_speed()**)

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> p200 = instruments.Pipette(
...     name='p200', axis='a', max_volume=200)


>>> # aspirate 50uL from a Well
>>> p200.aspirate(50, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>


>>> # aspirate 50uL from the center of a well
>>> p200.aspirate(50, plate[1].bottom())
<opentrons.instruments.pipette.Pipette object at ...>


>>> # aspirate 20uL in place, twice as fast
>>> p200.aspirate(20, rate=2.0)
<opentrons.instruments.pipette.Pipette object at ...>


>>> # aspirate the pipette's remaining volume (80uL) from a Well
>>> p200.aspirate(plate[2])
<opentrons.instruments.pipette.Pipette object at ...>
```

blow_out(*location=None*)

Force any remaining liquid to dispense, by moving this pipette's plunger to the calibrated *blow_out* position

Parameters: location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the blow_out. Can also be a tuple with first item **Placeable**, second item relative **Vector**

Returns:

Return type: This instance of **Pipette**.

Examples

```
>>> p200 = instruments.Pipette(name='p200', axis='a', max_volume=200)
>>> p200.aspirate(50).dispense().blow_out()
<opentrons.instruments.pipette.Pipette object at ...>
```

calibrate(*position*)

Calibrate a saved plunger position to the robot's current position

Notes

This will only work if the API is connected to a robot

Parameters: position (*str*) – Either "top", "bottom", "blow_out", or "drop_tip"

Returns:

Return type: This instance of **Pipette**.

Examples

```
>>> robot = Robot()
>>> p200 = instruments.Pipette(axis='a')
>>> robot.move_plunger(**{'a': 10})
>>> # save plunger 'top' to coordinate 10
>>> p200.calibrate('top')
<opentrons.instruments.pipette.Pipette object at ...>
```

calibrate_position(*location*, *current=None*)

Save the position of a **Placeable** (usually a **Container**) relative to this pipette.

Notes

The saved position will be persisted under this pipette's *name* and *axis* (see **Pipette**)

Parameters: location (tuple(**Placeable**, **Vector**)) – A tuple with first item **Placeable**, second item relative **Vector**
current (**Vector**) – The coordinate to save this container to (Default: robot current position)

Returns:

Return type: This instance of **Pipette**.

Examples

```
>>> robot.reset()
<opentrons.robot.robot.Robot object at ...>
>>> tiprack = containers.load('tiprack-200ul', 'A1')
```

opentrons

delay(*seconds=0*, *minutes=0*)

Parameters: seconds (*float*) – The number of seconds to freeeze in place.

dispense(*volume=None*, *location=None*, *rate=1.0*)

Dispense a volume of liquid (in microliters/uL) using this pipette

Notes

If no *location* is passed, the pipette will dispense from it's current position. If no *volume* is passed, *dispense* will default to it's *current_volume*

Parameters: volume (*int or float*) – The number of microliters to dispense (Default: self.current_volume)
location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the dispense. Can also be a tuple with first item **Placeable**, second item relative **Vector**
rate (*float*) – Set plunger speed for this dispense, where speed = rate * dispense_speed (see **set_speed()**)

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> p200 = instruments.Pipette(name='p200', axis='a', max_volume=200)
>>> # fill the pipette with liquid (200uL)
>>> p200.aspirate(plate[0])
<opentrons.instruments.pipette.Pipette object at ...>

>>> # dispense 50uL to a Well
>>> p200.dispense(50, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>

>>> # dispense 50uL to the center of a well
>>> relative_vector = plate[1].center()
>>> p200.dispense(50, (plate[1], relative_vector))
<opentrons.instruments.pipette.Pipette object at ...>

>>> # dispense 20uL in place, at half the speed
>>> p200.dispense(20, rate=0.5)
<opentrons.instruments.pipette.Pipette object at ...>

>>> # dispense the pipette's remaining volume (80uL) to a Well
>>> p200.dispense(plate[2])
<opentrons.instruments.pipette.Pipette object at ...>
```

drop_tip(*location=None*, *home_after=True*)

Drop the pipette's current tip

Notes

If no location is passed, the pipette defaults to its *trash_container* (see **Pipette**)

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> robot.reset()
<opentrons.robot.robot.Robot object at ...>
>>> tiprack = containers.load('tiprack-200ul', 'A1')
>>> trash = containers.load('point', 'A1')
>>> p200 = instruments.Pipette(axis='a', trash_container=trash)
>>> p200.pick_up_tip(tiprack[0])
<opentrons.instruments.pipette.Pipette object at ...>
>>> # drops the tip in the trash
>>> p200.drop_tip()
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.pick_up_tip(tiprack[1])
<opentrons.instruments.pipette.Pipette object at ...>
>>> # drops the tip back at its tip rack
>>> p200.drop_tip(tiprack[1])
<opentrons.instruments.pipette.Pipette object at ...>
```

home()

Home the pipette's plunger axis during a protocol run

Notes

Pipette.home() homes the *Robot*

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> p200 = instruments.Pipette(axis='a')
>>> p200.home()
<opentrons.instruments.pipette.Pipette object at ...>
```

mix(*repetitions=1*, *volume=None*, *location=None*, *rate=1.0*)

Mix a volume of liquid (in microliters/uL) using this pipette

Notes

If no *location* is passed, the pipette will mix from it's current position. If no *volume* is passed, *mix* will default to it's *max_volume*

Parameters:
- repetitions (*int*) – How many times the pipette should mix (Default: 1)
- volume (*int or float*) – The number of microliters to mix (Default: self.max_volume)
- location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the mix. Can also be a tuple with first item **Placeable**, second item relative **Vector**
- rate (*float*) – Set plunger speed for this mix, where speed = rate * (aspirate_speed or dispense_speed) (see **set_speed()**)

Returns:

Examples

```
>>> p200 = instruments.Pipette(name='p200', axis='a', max_volume=200)


>>> # mix 50uL in a Well, three times
>>> p200.mix(3, 50, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>


>>> # mix 3x with the pipette's max volume, from current position
>>> p200.mix(3)
<opentrons.instruments.pipette.Pipette object at ...>
```

move_to(*location*, *strategy='arc'*)

Move this **Pipette** to a **Placeable** on the **Deck**

Notes

Until obstacle-avoidance algorithms are in place, **Robot** and **Pipette** **move_to()** use either an "arc" or "direct"

Parameters: location (**Placeable** or tuple(**Placeable**, **Vector**)) – The destination to arrive at
strategy (*"arc" or "direct"*) – "arc" strategies (default) will pick the head up on Z axis, then over to the XY destination, then finally down to the Z destination. "direct" strategies will simply move in a straight line from the current position

Returns:
Return type: This instance of **Pipette**.

pick_up_tip(*location=None*, *presses=3*)

Pick up a tip for the Pipette to run liquid-handling commands with

Notes

A tip can be manually set by passing a *location*. If no location is passed, the Pipette will pick up the next available tip in it's *tip_racks* list (see **Pipette**)

Parameters: location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the pick_up_tip. Can also be a tuple with first item **Placeable**, second item relative **Vector**

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> robot.reset()
<opentrons.robot.robot.Robot object at ...>
>>> tiprack = containers.load('tiprack-200ul', 'A1')
>>> p200 = instruments.Pipette(axis='a', tip_racks=[tiprack])
>>> p200.pick_up_tip(tiprack[0])
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.return_tip()
<opentrons.instruments.pipette.Pipette object at ...>
>>> # `pick_up_tip` will automatically go to tiprack[1]
>>> p200.pick_up_tip()
```

return_tip(*home_after=True*)

Drop the pipette's current tip to it's originating tip rack

Notes

This method requires one or more tip-rack **Container** to be in this Pipette's *tip_racks* list (see **Pipette**)

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> tiprack = containers.load('tiprack-200ul', 'A1')
>>> p200 = instruments.Pipette(axis='a',
...     tip_racks=[tiprack], max_volume=200, name='p200')
>>> p200.pick_up_tip()
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.aspirate(50, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.dispense(plate[1])
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.return_tip()
<opentrons.instruments.pipette.Pipette object at ...>
```

set_speed(***kwargs*)

Set the speed (mm/minute) the **Pipette** plunger will move during **aspirate()** and **dispense()**

Parameters: kwargs (*Dict*) – A dictionary who's keys are either "aspirate" or "dispense", and who's values are int or float (Example: *{"aspirate": 300}*)

touch_tip(*location=None*, *radius=1.0*)

Touch the **Pipette** tip to the sides of a well, with the intent of removing left-over droplets

Notes

If no *location* is passed, the pipette will touch_tip from it's current position.

Parameters: location (**Placeable** or tuple(**Placeable**, **Vector**)) – The **Placeable** (**Well**) to perform the touch_tip. Can also be a tuple with first item **Placeable**, second item relative **Vector**
radius (*float*) – Radius is a floating point number between 0.0 and 1.0, describing the percentage of a well's radius. When radius=1.0, **touch_tip()** will move to 100% of the wells radius. When radius=0.5, **touch_tip()** will move to 50% of the wells radius.

Returns:
Return type: This instance of **Pipette**.

Examples

```
>>> p200 = instruments.Pipette(name='p200', axis='a', max_volume=200)
>>> p200.aspirate(50, plate[0])
<opentrons.instruments.pipette.Pipette object at ...>
>>> p200.dispense(plate[1]).touch_tip()
<opentrons.instruments.pipette.Pipette object at ...>
```

opentrons

Sign Up For Our Newsletter

Email**

SUBMIT