2019

Coat APP – Documentation

Nakhaee-Zadeh Gutierrez, Daniel (ARTORG)

Contents

[1. OT-2 Robot 4](#_Toc17825244)

[1.1. Hardware 4](#_Toc17825245)

[1.1.1. Rasphberry Pi 3 Model B+ 4](#_Toc17825246)

[1.1.2. Pipettes 4](#_Toc17825247)

[1.2. Software 5](#_Toc17825248)

[1.2.1. Balena OS 5](#_Toc17825249)

[1.2.2. Opentrons Docker Container 5](#_Toc17825250)

[2. Opentrons API 6](#_Toc17825251)

[2.1. Installation of the API 6](#_Toc17825252)

[2.1.1. Anaconda Installation 6](#_Toc17825253)

[2.1.2. Simulating Your Scripts 6](#_Toc17825254)

[2.2. Limitations of Opentrons API 7](#_Toc17825255)

[3. Opentrons Desktop App 8](#_Toc17825256)

[3.1. Using the Opentrons Desktop App 8](#_Toc17825257)

[3.2. Robot Tab 8](#_Toc17825258)

[3.2.1. Searching and Connecting to a robot 8](#_Toc17825259)

[3.3. Protocol Tab 9](#_Toc17825260)

[3.3.1. Uploading protocols 9](#_Toc17825261)

[3.4. Calibrate Tab 9](#_Toc17825262)

[3.5. Run Tab 9](#_Toc17825263)

[4. Connectivity 10](#_Toc17825264)

[4.1. Accessing the OT-2 Robot 10](#_Toc17825265)

[4.1.1. Secure Shell – SSH 10](#_Toc17825266)

[4.2. Transferring files to and from the OT-2 11](#_Toc17825267)

[5. Protocols 12](#_Toc17825268)

[5.1. Protocols Design 12](#_Toc17825269)

[5.1.1. Imports 12](#_Toc17825270)

[5.1.2. Metadata 12](#_Toc17825271)

[5.1.3. Labware 12](#_Toc17825272)

[5.1.4. Pipettes 13](#_Toc17825273)

[5.1.5. Commands 13](#_Toc17825274)

[5.1.6. Final Protocol 13](#_Toc17825275)

[5.2. Creating Protocols 13](#_Toc17825276)

[5.3. Simulating Protocols 13](#_Toc17825277)

[6. Labware 14](#_Toc17825278)

[6.1. Labware Definitions 14](#_Toc17825279)

[6.1.1. Ordering 15](#_Toc17825280)

[6.1.2. Brand 15](#_Toc17825281)

[6.1.3. Metadata 15](#_Toc17825282)

[6.1.4. Dimensions 16](#_Toc17825283)

[6.1.5. Corner Offset from Slot 16](#_Toc17825284)

[6.1.6. Namespace 17](#_Toc17825285)

[6.1.7. Version 17](#_Toc17825286)

[6.1.8. Schema version 17](#_Toc17825287)

[6.1.9. Parameters 17](#_Toc17825288)

[6.1.10. Wells 17](#_Toc17825289)

[7. Coat APP 18](#_Toc17825290)

[7.1. Interface 18](#_Toc17825291)

[7.1.1. Connection 18](#_Toc17825292)

[7.1.2. Calibration 18](#_Toc17825293)

[7.1.3. Control Hardware 18](#_Toc17825294)

[7.1.4. Labware Setup (On Development) 18](#_Toc17825295)

[7.1.5. Protocols 18](#_Toc17825296)

[7.2. Set-Up 18](#_Toc17825297)

[7.3. Calibration 18](#_Toc17825298)

[7.4. Coating and Washing 18](#_Toc17825299)

[7.5. Development 18](#_Toc17825300)

[7.5.1. 18](#_Toc17825301)

[7.5.2. Communication Protocols 18](#_Toc17825302)

[7.5.3. Creating New Forms 18](#_Toc17825303)

[7.5.4. Creating 18](#_Toc17825304)

[8. CHIPPIE Python Module 19](#_Toc17825305)

[8.1. Functions: 19](#_Toc17825306)

[8.1.1. Pipette Tip Management 19](#_Toc17825307)

[8.1.2. Aspirate all volume on well (On development) 19](#_Toc17825308)

[8.1.3. Calibration – Updating offset 20](#_Toc17825309)

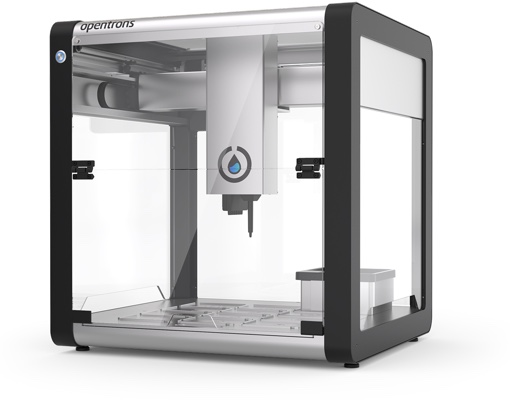
[8.2. Adding new Functions to the module (On development) 20](#_Toc17825310)

[8.3. Adding new Classes to the module (On development) 20](#_Toc17825311)

[9. Coat APP GitHub Repository 21](#_Toc17825312)

# OT-2 Robot

The OT-2 robot is a pipetting robot developed by Opentrons. The OT-2 robot allows the automatization of protocols with increase speed, control and flexibility.



## Hardware

The hardware of the robot is composed of several mechanical and electrical components which are unlisted by the company. However, the main piece of hardware on the robot is the Raspberry Pi 3, which stores the container or operating system of the robot.

### Rasphberry Pi 3 Model B+

The raspberry Pi 3 is a single-board compact computer developed by a RaspberryPi. The computer platform is intended for learning programming and developing electronic projects. One great benefit of this platform are the scalability and the open-source nature of the platform.

### Pipettes

The OT-2 pipettes can be purchased directly from Opentrons. This allow the automation of precise liquid transfers using the OT-2 robot. The robot purchased by Alveolix (Robot ID: OT2CEP20181107A03) has two pipette modules on the mount: P300 and P50. The mount system has two slots to install pipettes, label with “Left” and “Right”. Pipette modules can be easily interchange or replace (See Opentrons [support page](https://support.opentrons.com/en/articles/2067321-a-attaching-pipettes)).

#### P300 – Single Channel

The P300 is a single channel automated pipette. On the Alveolix Robot (Robot ID: OT2CEP20181107A03) this pipette module is installed on the “Left” slot of the mount. The P300 has the following specifications:

* Available Range: 30 – 300 ul
* Systematic Error at 300 ul: ±1.8 ul
* Systematic Error at 30 ul: ±0.9 ul

#### P50 – Single Channel

The P50 is a single channel automated pipette. On the Alveolix Robot (Robot ID: OT2CEP20181107A03) this pipette module is installed on the “Right” slot of the mount. The P50 has the following specifications:

* Available Range: 5 – 50 ul

## Software

### Balena OS

The RaspberryPi runs on a custom OS called Resin OS 2.7.5 (recently renamed to Balena OS (<https://www.balena.io/os/)>. Balena OS is a host OS designed for deploying docker containers.

### Opentrons Docker Container

A container in software terms is similar to a virtual machine. Containers contain applications in a way that keep them isolated from the host system that they run on. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. Containers are designed to make it easier to provide a consistent experience as developers and system administrators move code from development environments into production in a fast and replicable way.

The Container on the OT-2 robot can be access using an SSH OR SFTP (Port: 22) connection, or alternatively the containers python’s instance can be accessed by launching the Jupyter Notebook web-based development platform, advertised on Port: 48888 from a web browser. Please, refer to section [4. Connectivity](#_Connectivity), to learn more about how to connect to the OT-2 robot.

By default, the Opentrons container initialises every time the robot is turned on or rebooted. The Opentrons programmatically deletes all the new files added to the container

# Opentrons API

The Opentrons API is the library developed by Opentrons to connect with the OT-2 robot and be able to execute commands. The Opentrons API is built using Python 3.6.

## Installation of the API

The Opentrons Module and API is only required to be downloaded on the local machine for simulation purposes. This means simulating protocols without being connecting to the OT-2 robot or using the Opentrons Desktop APP. The Opentrons container on the OT-2 robot already contains all the necessary packages for executing the python protocols.

To install the Opentrons package, you must install it from Python’s package manager, pip. The recommended installation is using the Anaconda distribution of python 3.6 or higher.

***Note for Alveolix Users****: If you are using the Laptop (DESKTOP-KH87S7D) already connected to the OT-2 robot, this step is not required as the setup is already done on your machine using Anaconda 3. To check if the package is installed open the Anaconda 3 prompt and type:*

*pip freeze | findstr opentrons*

*This should return the Opentrons module and its version.*

*opentrons==3.10.3*

*If the command returns an error, type the following command to install the Opentrons module.*

*pip install opentrons*

### Anaconda Installation

Download the Anaconda installer 3.6 or higher for your operating system ([Windows, Mac or Linux)](https://www.anaconda.com/distribution/#download-section).

Once the installer is done, make sure that Python is properly installed by opening the Anaconda Prompt or the Windows Command Prompt, if you added the Anaconda 3 to the system Path and typing python --version. The version installed should be 3.6 or higher.

Once python is installed, install the opentrons package using pip:

pip install opentrons

You should see some output that ends with Successfully installed Opentrons

### Simulating Your Scripts

Simulating the python scripts is a good method to validate, troubleshoot and fix bugs on your protocol code. Once the Opentrons Python package is installed, you can simulate protocols in your terminal using the opentrons\_simulate command:

opentrons\_simulate.exe my\_protocol.py

or, on OS X or linux:

opentrons\_simulate my\_protocol.py

The simulator will print out a log of the actions the protocol will cause, similar to the Opentrons app; it will also print out any log messages caused by a given command next to that list of actions. If there is a problem with the protocol, the simulation will stop, and the error will be printed.

The simulation script can also be invoked through python with:

python -m opentrons.simulate /path/to/protocol.

This also provides an entry point to use the Opentrons simulation package from other Python contexts such as an interactive prompt or Jupyter. To simulate a protocol in python, open a file containing a protocol and pass it to opentrons.simulate.simulate:

import opentrons.simulate

protocol\_file = open('/path/to/protocol.py')

runlog = opentrons.simulate.simulate(protocol\_file)

print(format\_runlog(runlog))

The opentrons.simulate.simulate() method does the work of simulating the protocol and returns the run log, which is a list of structured dictionaries. opentrons.simulate.format\_runlog() turns that list of dictionaries into a human readable string, which is then printed out. For more information on the protocol simulator, see Simulation.

## Limitations of Opentrons API

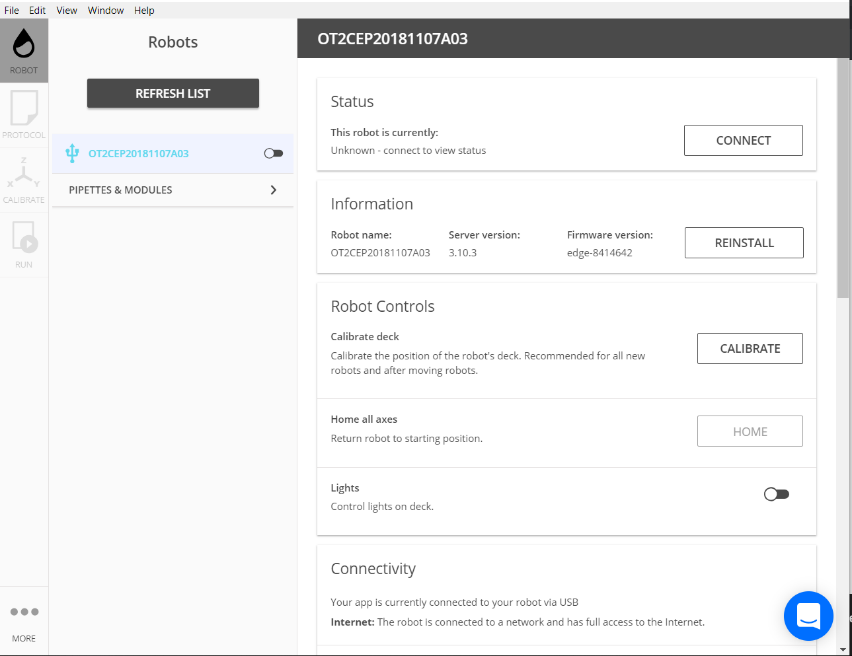
Although the Opentrons API supports a great variety of task and commands for the OT-2, there are a few limitations that. Some of the general limitations can be found below.

1. Variable storage. EX pipette tip, position of labware, …
2. Limited labware Creation.

# Opentrons Desktop App

The Opentrons APP is the user interface that allows communication and the ability to run protocols on the OT-2 robot. The application is built using Electron and is based on JavaScript, HTML and CSS code stack. More information about the Opentrons desktop app and how to build or develop can be found on the [Opentrons GitHub Repository](https://github.com/Opentrons/opentrons/). This part will focus on the UI design and usability of the Opentrons application.

## Using the Opentrons Desktop App

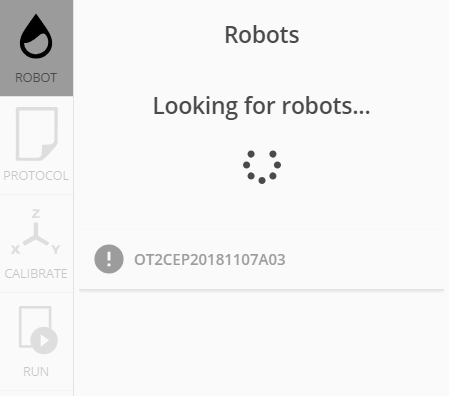
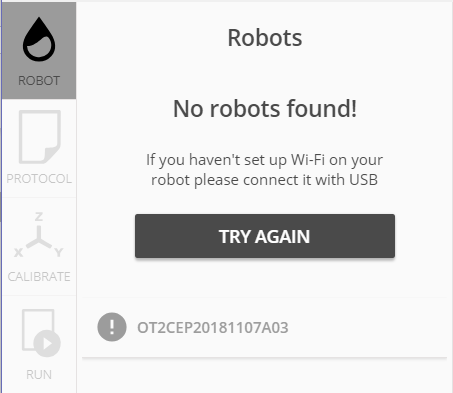
The user interface of the Opentrons Desktop App is divided in four different tabs, each with specific function.

## Robot Tab

When opening the Opentrons app the user is prompted automatically to the Robot Tab. This tab allows the user to search for an available OT-2 robot on the network connections, secure a connection to the robot and access several properties of the robot’s hardware and Opentrons Container.

### Searching and Connecting to a robot

Automatically when launching the instance, the Opentrons app will try to search for any available OT-2 robots on the network ()

## Protocol Tab

### Uploading protocols

## Calibrate Tab

## Run Tab

# Connectivity

The communication protocols are key to understand how to



## Accessing the OT-2 Robot

Accessing the OT-2 operating system can help you troubleshooting or modifying files directly on to the stored memory of the robot. There all several communication protocols that can be used to access the Linux Alpine container on the OT-2 robot from a computer.

### Secure Shell – SSH

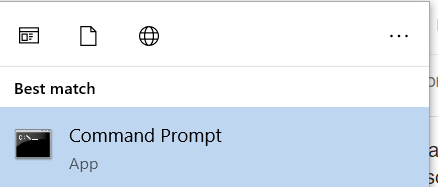
The SSH command provides a secure encrypted connection between two hosts over an insecure network. This connection can also be used for terminal access, file transfers, and for tunnelling other applications. By default, most Linux machines advertised their SSH server on port 22, this is the case for the OT-2 robot. There are common functions that we can perform using a SSH connection:

* Simulating and Running protocols on the OT-2 Robot.
* Creating and editing directories and files.
* Copying and moving files from and to remote computer.
* Updating

To connect to the OT-2 from a windows machine you can use one of the following clients:

#### Open-SSH

In the latest updates (From 2018) Windows has integrated support for OpenSSH client and it is enabled by default in any PC running Windows 10. To use the OpenSSH in Windows, launch the Command Prompt by clicking on the Windows Start Icon, and typing “**CMD**” (Instead of the Command Prompt you can alternatively use any other Console such as Anaconda Prompt, Microsoft Azure, PowerShell, …).





Then typing on the console, where <host> is the host server of the OT-2 (By default the only available user on the OT-2 robot is the root user: “root” ) and the where <ip\_address> is the IP address of the robot (Unfortunately, depending on the network you are using the OT-2 is unable to maintain a static IP address, meaning that the IP address will change constantly between launching the container):

ssh <host>@<ip\_address>

Example: ssh root@169.256.119.61

To obtain the current IP address of the robot:

* Using the Opentrons APP: On the Opentrons APP, Connect the OT-2 robot using the USB connection or wireless connection. Once the OT-2 robot has been discovered click on it, to view its settings and properties. Under the Connectivity tab, the Wireless and wired IPs are advertised.
* Using the Coat APP: Refer to section [Obtaining IP Address from Coat APP](#ObtainingIP)

#### Putty - Recommended

Putty is another SSH client that can be used to create a secure connection between the robot and desktop.

## Transferring files to and from the OT-2

Transferring files securely between the desktop and the OT-2 robot local storage sometimes might be necessary in order to add new protocols or download recorded images or videos from the robot.

# Protocols

Protocols are the main method to interface between the Opentrons API and the different hardware (lights, X-Y stage, pipette modules, etc). Similarly, to a standard biological protocol or SOP, the protocols on the Opentrons API work by defining a series of materials and equipment and then iterating step by step through the different task in the protocol. However, instead of designing the protocols using standard language, the user needs to use the Opentrons Python Package to write the protocol, so that the OT-2 robot can understand the commands. This manual only, explains a general overview of how to create and edit protocols using the Opentrons API. For an in-detail explanation of all the functions available please refer to the [Opentrons Docs Website](https://docs.opentrons.com/).

## Protocols Design

A protocol is a python (.PY) script containing the dependencies, labware and commands of the intended protocol. The scripts contain 5 main components as outline below:

### Imports

When writing in Python, you must always include the Opentrons API within your file. This is required to be able to access all the commands from the Opentrons module. We most commonly use the labware and instruments sections of the API. Other sections include:

* robot: to control the hardware of the robot directly.
* containers: container specifications for the Opentrons Container.

The imports section looks like:

from opentrons import labware, instruments

### Metadata

Metadata is a dictionary of data that is read by the server and returned to client applications (such as the Opentrons App). It is not needed to run a protocol (and is entirely optional), but if present can help the client application display additional data about the protocol currently being executed.

The fields above (“protocolName”, “author”, and “description”) are the recommended fields, but the metadata dictionary can contain fewer or additional fields as desired (though non-standard fields may not be rendered by the client, depending on how it is designed).

metadata = {

'protocolName': 'My Protocol',

'author': 'Name <email@address.com>',

'description': 'Simple protocol to get started using OT2',

}

### Labware

While the imports section is usually the same across protocols, the labware section is different depending on the labware (tip racks, well plates, troughs, or tubes) you’re using on the robot.

Each labware is given a name (ex: '96-flat'), and the slot on the robot it will be placed (ex: '2').

The Labware section looks like:

plate = labware.load('96-flat', '2') #Loads '96-flat' on the Slot 2

tiprack = labware.load('tiprack-200ul', '1') #Loads 'tiprack-200 ul' on the Slot 1

### Pipettes

Pipettes are created and attached to a specific mount on the OT-2 ('left' or 'right'). There are other parameters for pipettes, but the most important are the tip rack(s) it will use during the protocol.

The pipettes section looks like:

pipette = instruments.P300\_Single(mount='left', tip\_racks=[tiprack])

### Commands

The command section is where the user should write the actual protocol and orders to the robot. The most common commands are transfer(), aspirate(), dispense(), pick\_up\_tip(), drop\_tip().

The commands section looks like:

pipette.transfer(100, plate.wells('A1'), plate.wells('B1'))

# This will transfer 100 ul of volume from plate well A1 to plate well B1.

### Final Protocol

Combining all the different parts of the protocol, a final protocol should look:

### Imports

from opentrons import labware, instruments

### Metadata

metadata = {

'protocolName': 'My Protocol',

'author': 'Name <email@address.com>',

'description': 'Simple protocol to get started using OT2',

}

### Labware

plate = labware.load('96-flat', '2') #Loads '96-flat' on the Slot 2

tiprack = labware.load('tiprack-200ul', '1') #Loads 'tiprack-200 ul' on the Slot 1

### Pipettes

pipette = instruments.P300\_Single(mount='left', tip\_racks=[tiprack])

### Commands

pipette.transfer(100, plate.wells('A1'), plate.wells('B1'))

## Creating Protocols

This section explains how and where to create a new protocol depending on the final application routine or

## Simulating Protocols

Once your protocol is finished you can simulate it offline and locally using the Opentrons API. Please refer to section [2.1.2. Simulating your Scripts.](#_Simulating_Your_Scripts)

# Labware

Labware in the context of the Opentrons app is any specific piece of equipment that interfaces with the OT-2 robot. Labware include well plates, tip racks, Eppendorf and falcon tube holders, etc. The Opentrons API comes with many common labware built in. These can be loaded into your Python protocol by using the labware.load() method with the specific load name of the labware you need. The list below contains

|  |
| --- |
| Labware Names |
| corning\_6\_wellplate\_16.8ml\_flat |
| corning\_12\_wellplate\_6.9ml\_flat |
| corning\_24\_wellplate\_3.4ml\_flat |
| corning\_48\_wellplate\_1.6ml\_flat |
| corning\_384\_wellplate\_112ul\_flat |
| usascientific\_96\_wellplate\_2.4ml\_deep |
| corning\_96\_wellplate\_360ul\_flat |
| biorad\_96\_wellplate\_200ul\_pcr |
| biorad\_96\_wellplate\_200ul\_pcr |
| opentrons\_40\_aluminumblock\_eppendorf\_24x2ml\_safelock\_snapcap\_generic\_16x0.2ml\_pcr\_strip |
| biorad\_96\_wellplate\_200ul\_pcr |
| opentrons\_24\_aluminumblock\_generic\_2ml\_screwcap |
| opentrons\_24\_aluminumblock\_generic\_2ml\_screwcap |
| opentrons\_96\_aluminumblock\_biorad\_wellplate\_200ul |
| opentrons\_96\_aluminumblock\_generic\_pcr\_strip\_200ul |
| opentrons\_96\_tiprack\_300ul |
| opentrons\_24\_tuberack\_eppendorf\_1.5ml\_safelock\_snapcap |
| opentrons\_10\_tuberack\_falcon\_4x50ml\_6x15ml\_conical |
| opentrons\_15\_tuberack\_falcon\_15ml\_conical |
| opentrons\_24\_tuberack\_eppendorf\_2ml\_safelock\_snapcap |
| opentrons\_24\_tuberack\_generic\_2ml\_screwcap |
| opentrons\_6\_tuberack\_falcon\_50ml\_conical |
| opentrons\_96\_aluminumblock\_generic\_pcr\_strip\_200ul |
| opentrons\_96\_tiprack\_10ul |
| tipone\_96\_tiprack\_200ul |
| opentrons\_96\_tiprack\_1000ul |
| agilent\_1\_reservoir\_290ml |
| usascientific\_12\_reservoir\_22ml |
| opentrons\_24\_tuberack\_generic\_0.75ml\_snapcap\_acrylic |
| opentrons\_24\_tuberack\_eppendorf\_2ml\_safelock\_snapcap\_acrylic |
| opentrons\_10\_tuberack\_falcon\_4x50ml\_6x15ml\_conical\_acrylic |

## Labware Definitions

The dimensions and characteristics of each loaded labware on the robot are saved under a labware definition. The labware definitions consists on a .JSON file which typically is used to store data structures and objects in JavaScript Object annotation. .JSON are text-based and human readable and can easily be edited using a text editor. The labware definitions are stored on the OT-2 locally on the folder:

Each piece of labware contains an individual labware definition. The labware definitions contain several different attributes:

/data/packages/usr/local/lib/python3.6/sitepackages/opentrons/shared\_data/labware/definitions/2/

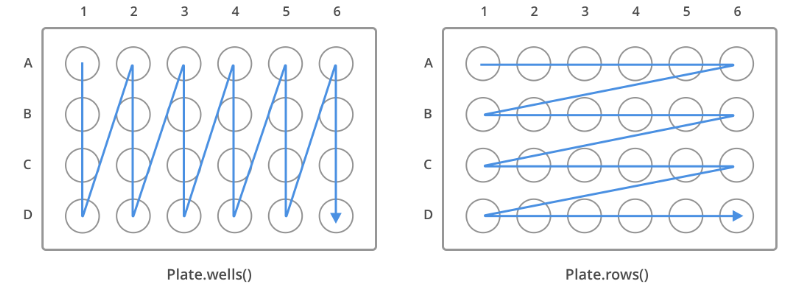
### Ordering

Ordering attribute often appears the first element of the .JSON file. This attribute shows the order in which the robot should iterate through the wells of the labware. By default, the OT-2 deck and labware are all set-up with the same coordinate system, starting from the left-top corner of the labware:

* Rows: using letters, IE. [“A”] – [“END”]
* Columns: using numbers, IE. [“1”] – [“END”]

The ordering attribute contains a matrix with the specific order of each, column and row, within the labware. An example of a labware ordering attribute of a 3x3 well plate with a total of 9 wells can be found below:

"ordering":[["A1","B1","C1"],["A2","B2","C2"],["A3","B3","C3"]] #3x3 well plate

The ordering of any labware definition can be changed to for example modify the order in which pipette tips are pickup.

### Brand

Contains specific information about the manufacturer or producer and ID or Serial numbers of the labware. The brand attribute contains with in it three variables, “brand”, which contains the name of the manufacturer of the labware, “brandId” which shows the different serial numbers or IDs that relate to the labware and “links” which contain any web link to the labware, its dimensions or characteristics.

Below there is an example of a labware brand attribute:

"brand": {

"brand": "Alveolix",

"brandId": "AX6",

"links": "None" }

### Metadata

The metadata attribute contains general information about the labware, which includes:

* "displayName": This is the unofficial name of the labware and usually used to display the labware on the
* "displayCatergory": contains the type of labware that the definition is referring to. The system can identify several types of labware category:
  + "wellPlate": for a well plate which has accessible wells, for dispensing or aspirating volume.
  + "tipRack": for a tip rack labware that stores tips or other pipette complements.
  + "trash": for a tip trash definition
  + "reservoir": for a pipette reservoir with individual channels.
* "displayVolumeUnits":
* "tags":

An example of the metadata from a well plate labware can be found below:

"metadata": {

"displayName": "AX6 LOC",

"displayCategory": "wellPlate",

"displayVolumeUnits": "\\u00b5L",

"tags": "None" }

### Dimensions

Contains the physical dimensions of the labware (width, length and height). The attribute has three different values corresponding to each dimension of the labware and are always displayed on millimetres [mm]:

* "xDimension": length of the labware
* "yDimension": width of the labware
* "zDimension": height or thickness of the labware

Image with dimensions

An example of the dimension attribute can be found below:

"dimensions": {

"xDimension": 120.0,

"yDimension": 56.0,

"zDimension": 56.0 }

### Corner Offset from Slot

This attribute contains the distance from the top-left corner of labware to the centre-bottom of the first slot, typically well A1.

"cornerOffsetFromSlot": {

"x": 0,

"y": 0,

"z": 0 }

### Namespace

ss

"namespace": "opentrons"

### Version

**cfsdf**

"version": 1

### Schema version

sdfsd

"schemaVersion": 2

### Parameters

sfsfs

"parameters":{

"format": "trough",

"isTiprack": false,

"isMagneticModuleCompatible": false,

"loadName": "usascientific\_12\_reservoir\_22ml" }

### Wells

sfsdfsdf

"A1": {

"depth": 2,

"shape": "circular",

"diameter": 3,

"totalLiquidVolume": 0.07,

"x": 13.5,

"y": 65,

"z":60

},

# Coat APP

The coat app is a .NET framework (C#) build to interface between the python protocols and the Opentrons API. The Coat app is a client application meaning that is able to connect and get information and services from the server running on the OT-2 robot. The Coat app is designed for the purpose of coating and pre-processing the AX6 platform from Alveolix AG. For this reason, the user interface has limited functionality. Nevertheless, the Coat app is open-sourced allowing developers to use the application framework and tools to add new protocols or

## Interface

The application main form includes 5 distinct panels to connect and run protocols using the OT-2 robot.

### Connection

This panel allows connection with the OT-2 Robot. If the robot is available and the IP address is correct when clicking the button.

### Calibration

This panel contains all the controls necessary to calibrate the labware with reference to the Ot-2 robot system. The

### Control Hardware

Allows control of external hardware including the Chip Holder rotator or the lights of the robot.

### Labware Setup (On Development)

### Protocols

This panel includes all the controls to coat and wash the chips using the OT-2 robot and the external hardware.

## Set-Up

In order to install the Coat app please visit the Alveolix AG GitHub repository to find the coat app. Clone the repository to the desktop.

***Note for Alveolix Users****: If you are using the Laptop already connected to the OT-2 robot, this step is not required as the setup is already done on your machine.*

On the repository find

## Calibration

## Coating and Washing

## Development

### 

### Communication Protocols

### Creating New Forms

### Creating

# CHIPPIE Python Module

As coating protocols become more complex and functions get changed over time, it is important to be able to ensure that functions are simultaneously updated on each script. Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module. A module is a file containing Python definitions and statements. The CHIPPIE Python Module contains all the necessary functions to perform coating and washing protocols and ensuring the well-functioning of the OT-2 setup.

## Functions:

### Pipette Tip Management

Although the Opentrons API supports pipette tip management within a single protocol, the API is unable to store the number and location of the tips that remain on the tip rack. This is a big problem since when running multiple protocols consecutively, the user will have to edit the accessible pipette tip repeatedly. In order to create a single place for storing and accessing the available tips on the tip rack the function usetip() can be used when calling a pickup or drop tip action on the Opentrons API. The usetip() function has a dependencies with the python module pickle, which is use to store the current pipette tip available on the system.

usetip(val, rst)

**Reset Tips, val = 0**

usetip(0) #resets the tip of the tip rack on the deck to the first tip on the top left corner (A1)

usetip(0, 5) #resets the tip of the tip rack on the deck to the 5th tip on the tip rack (E1).

**Access Next Available tip, val = 1**

usetip(1) #Returns the next tip available in the tip rack.

Return Tip, val = 2

usetip(2) #used after pipette.return\_tip() command, to set the next available pipette tip to the return pipette tip.

**Pick up new tip, val = 3**

usetip(3) or usetip() #this command returns the next available tip on the tip rack and updates the tips used. Use this command whenever you use the command pipette.pick\_up\_tip(), to ensure that already used pipette tips are not callable.

**Python Dependencies**

import pickle

### Aspirate all volume on well (On development)

### Calibration – Updating offset

The calibration functions are necessary to ensure that the robot can perform the movements and protocols accurately, avoiding damage on the labware or pipette modules. Calibration through the Opentrons APP is summarized in section, Calibration of the OT-2 – Opentrons APP. However, this system of implementing the offset is cumbersome and can cause conflicts between the labware. At the same time, it is not easy from the user to update or visualize the current positioning of the robot. For this reason the function update\_offset(), was created. This function accesses the labware definition for

update\_offset(),

## Adding new Functions to the module (On development)

## Adding new Classes to the module (On development)

# Coat APP GitHub Repository

The coat app GitHub repository contains all the tools, files and. Furthermore, the creation of a git environment that connects the development location and the OT-2 robot allows, instant and rapid deployment of new protocols remotely. This is ideal to be able to produce protocols off-line, simulate them locally and once verified executing them on the robot using the coat app.