

2019

Coat APP – Documentation

aLVEOLIX ag

Nakhaee-Zadeh Gutierrez, Daniel

# Contents

[Contents 1](#_Toc22593269)

[1. OT-2 Robot 4](#_Toc22593270)

[1.1. Hardware 4](#_Toc22593271)

[1.1.1. Rasphberry Pi 3 Model B+ 4](#_Toc22593272)

[1.1.2. Pipettes 4](#_Toc22593273)

[1.2. Software 5](#_Toc22593274)

[1.2.1. Balena OS 5](#_Toc22593275)

[1.2.2. Opentrons Docker Container 5](#_Toc22593276)

[2. Opentrons API 6](#_Toc22593277)

[2.1. Installation of the API 6](#_Toc22593278)

[2.1.1. Anaconda Installation 6](#_Toc22593279)

[2.1.2. Simulating Your Scripts 6](#_Toc22593280)

[2.2. Limitations of Opentrons API 7](#_Toc22593281)

[3. Opentrons Desktop App 8](#_Toc22593282)

[3.1. Using the Opentrons Desktop App 8](#_Toc22593283)

[3.2. Robot Tab 8](#_Toc22593284)

[3.2.1. Searching and Connecting to a robot 8](#_Toc22593285)

[3.3. Protocol Tab 9](#_Toc22593286)

[3.4. Calibrate Tab 9](#_Toc22593287)

[3.5. Run Tab 9](#_Toc22593288)

[4. Connectivity 10](#_Toc22593289)

[4.1. Accessing the OT-2 Robot 10](#_Toc22593290)

[4.1.1. Secure Shell – SSH 10](#_Toc22593291)

[4.2. Transferring files to and from the OT-2 11](#_Toc22593292)

[5. Protocols 12](#_Toc22593293)

[5.1. Protocols Design 12](#_Toc22593294)

[5.1.1. Imports 12](#_Toc22593295)

[5.1.2. Metadata 12](#_Toc22593296)

[5.1.3. Labware 12](#_Toc22593297)

[5.1.4. Pipettes 13](#_Toc22593298)

[5.1.5. Commands 13](#_Toc22593299)

[5.1.6. Final Protocol 13](#_Toc22593300)

[5.2. Creating Protocols 13](#_Toc22593301)

[5.3. Simulating Protocols 13](#_Toc22593302)

[6. Labware 14](#_Toc22593303)

[6.1. Labware Definitions 14](#_Toc22593304)

[6.1.1. Ordering 15](#_Toc22593305)

[6.1.2. Brand 15](#_Toc22593306)

[6.1.3. Metadata 15](#_Toc22593307)

[6.1.4. Dimensions 16](#_Toc22593308)

[6.1.5. Corner Offset from Slot 17](#_Toc22593309)

[6.1.6. Namespace 17](#_Toc22593310)

[6.1.7. Version 17](#_Toc22593311)

[6.1.8. Schema version 17](#_Toc22593312)

[6.1.9. Parameters 17](#_Toc22593313)

[6.1.10. Wells and wells objects 18](#_Toc22593314)

[7. Coat APP 19](#_Toc22593315)

[7.1. Interface 19](#_Toc22593316)

[7.1.1. Connection 19](#_Toc22593317)

[7.1.2. Calibration 19](#_Toc22593318)

[7.1.3. Control Hardware 19](#_Toc22593319)

[7.1.4. Labware Setup (On Development) 19](#_Toc22593320)

[7.1.5. Protocols 20](#_Toc22593321)

[7.1.6. Robot 20](#_Toc22593322)

[7.1.7. SSH 20](#_Toc22593323)

[7.2. Set-Up 20](#_Toc22593324)

[7.3. Connecting to the Robot 20](#_Toc22593325)

[7.3.1. Issues with connectivity 21](#_Toc22593326)

[7.4. Labware Setup 21](#_Toc22593327)

[7.4.1. Labware Setup Panel (On development) 22](#_Toc22593328)

[7.4.2. Create new labware panel 24](#_Toc22593329)

[7.5. Tip Set-up 25](#_Toc22593330)

[7.6. Calibration 26](#_Toc22593331)

[7.6.1. Editing the Calibration Protocol 27](#_Toc22593332)

[7.7. Coating and Washing 27](#_Toc22593333)

[7.7.1. Running a Protocol 27](#_Toc22593334)

[7.7.2. Advanced Protocols 28](#_Toc22593335)

[7.7.3. Modifying a Protocol 28](#_Toc22593336)

[7.8. Additional Functions 28](#_Toc22593337)

[7.9. Development 28](#_Toc22593338)

[7.9.1. Launching the Coat App on Development Mode 28](#_Toc22593339)

[7.9.2. Source Code 28](#_Toc22593340)

[8. CHIPPIE Python Module 29](#_Toc22593341)

[8.1. Functions: 29](#_Toc22593342)

[8.1.1. Pipette Tip Management 29](#_Toc22593343)

[8.1.2. Calibration – Updating offset 29](#_Toc22593344)

[8.1.3. Aspirate all volume on well (On development) 30](#_Toc22593345)

[8.2. Adding new Functions to the module (On development) 30](#_Toc22593346)

[8.3. Adding new Classes to the module (On development) 30](#_Toc22593347)

[9. Coat APP GitHub Repository 31](#_Toc22593348)

[9.1. File Location 31](#_Toc22593349)

[9.1.1. Protocols 31](#_Toc22593350)

[9.1.2. Library (On development) 31](#_Toc22593351)

[9.1.3. Documentation 31](#_Toc22593352)

[9.1.4. Patches 31](#_Toc22593353)

[9.1.5. bin 31](#_Toc22593354)

[9.1.6. Coat-APP 31](#_Toc22593355)

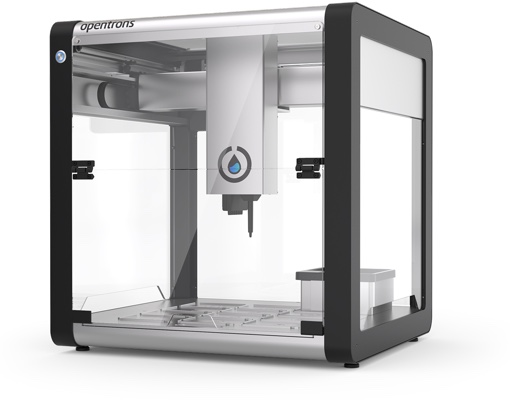
[Appendix 1 – AX protocols 32](#_Toc22593356)

[Coating Protocol 32](#_Toc22593357)

[Washing Protocol 33](#_Toc22593358)

# 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

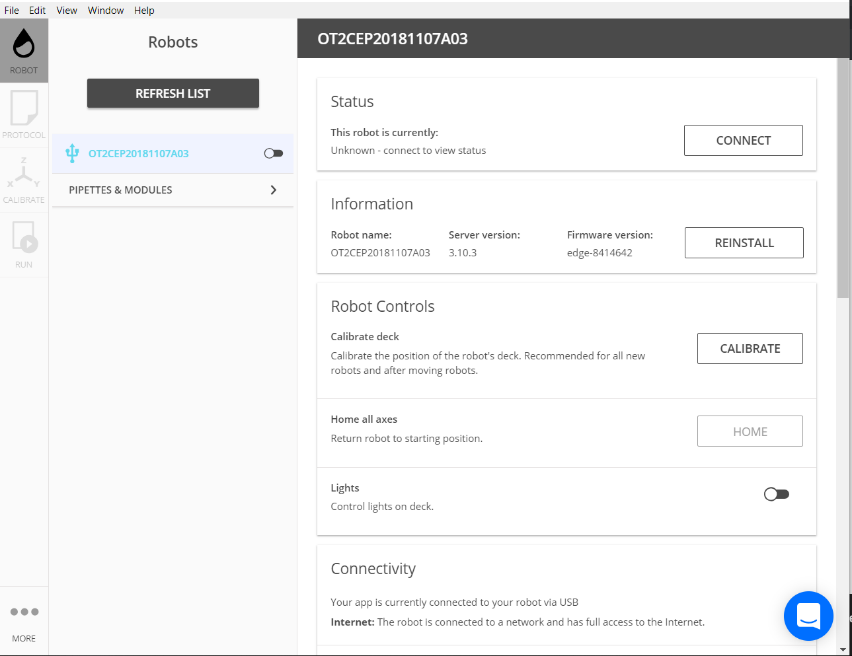
Although the Opentrons API supports a great variety of task and commands for the OT-2, there are a few limitations that. Some of the general limitations can be found below.

1. Variable storage. Storing current pipette tip, position of labware,
2. Limited labware Creation.
3. Unable to control external hardware inside protocols

# 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

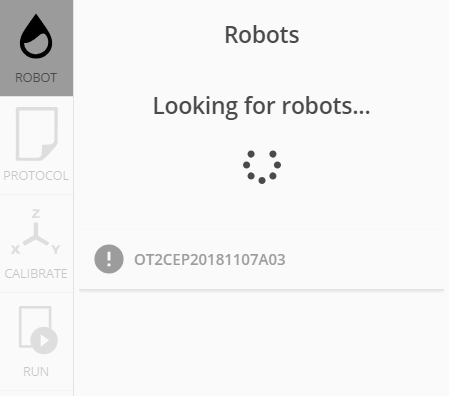
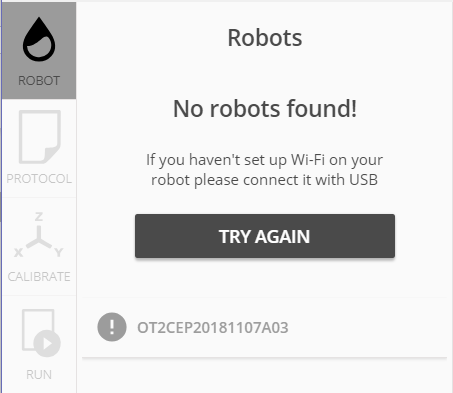
The user interface of the Opentrons Desktop App is divided in four different tabs, each with specific function.

## 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 (both wired and wirelessly). The available robots are displayed under the robot list using its serial number.

## Protocol Tab

The protocol page can be used to upload and simulate the designed python protocols.

## Calibrate Tab

This tab manages the labware found on the protocol and allows the user to individually calibrate each labware.

## Run Tab

In this tab the user can launch or stop a protocol. Furthermore, there is a summary of the task that the OT-2 has to perform. This tab also displays the current status of the robot.

# Connectivity



## 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

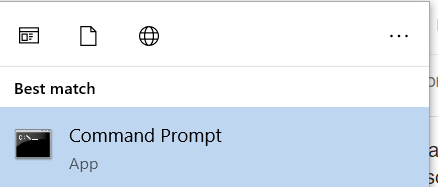
The SSH command provides a secure encrypted connection between two hosts over an insecure network. This connection can also be used for terminal access, file transfers, and for tunnelling other applications. By default, most Linux machines advertised their SSH server on port 22, this is the case for 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. For this purpose, it is possible to use a Secure File Transfer Protocol (SFTP). SFTP protocol is part of the Secure Shell protocol and works in a similar way. You can find all the information on how to use SFTP to transfer files to an external machine in the following [link](https://www.maketecheasier.com/use-sftp-transfer-files-linux-servers/).

# 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 or routine. To create a protocol simply open a new python script and use the Opentrons API to start creating your commands.

## 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 the provided labware by Opentrons:

|  |
| --- |
| 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 files 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/

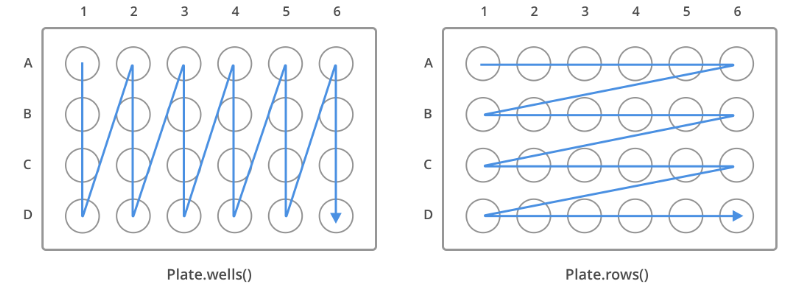
### Ordering

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": Represents the notation of the volume units used inside the labware definition. By default, the standard unit for the labware is in microliters (µl) which is represented in the Unicode system by the expression: \\u00b5L
* "tags": Additional tags that can be used to sort the protocols.

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

The description of the dimensions of a well plate are shown below:

A close up of a logo

Description automatically generated

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 the labware to the centre of the bottom of the first slot, typically well A1. As default, the offsets are set the zero for all directions. The offsets can be modified manually by editing the labware JSON file to adjust the position of the labware. Alternatively, the offset can be adjusted using the calibration function of the Coat APP or Opentrons App.

"cornerOffsetFromSlot": {

"x": 0,

"y": 0,

"z": 0 }

### Namespace

The namespace definition includes a string referring to the description of the protocols. This field is mandatory and should contain the

"namespace": "opentrons"

### Version

The version is used to identify the iteration of the protocol.

"version": 1

### Schema version

The schema version refers to the format of the JSON file of the labware. Prior to version 3.10.0 of the Opentrons API, opentrons used a completely different set of labware load names. On version 4.0.0 the new JSON format, known as Schema Version 2 was introduced. When developing, protocols using this documentation always set the schemaVersion to 2.

"schemaVersion": 2

### Parameters

The parameter field contains important information related to the type of labware, its compatibility with other modules and its load name, which is the name of the labware that is used to define it within the protocols. The parameter field includes the following terms:

* "format": Property to determine compatibility with multichannel pipette. The possible fields are: "96Standard", "384Standard", "trough", "irregular" or "trash".
* "isTiprack": Flag marking whether a labware is a tiprack or not. The field should include a Boolean (True or False).
* "isMagneticModuleCompatible": Flag marking whether a labware is compatible by default with the Magnetic Module. The field should include a Boolean (True or False).
* "loadName": Name used to reference a labware definition. This is the name that should be used on the protocols to load the labware.

"parameters":{

"format": "trough",

"isTiprack": false,

"isMagneticModuleCompatible": false,

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

### Wells and wells objects

The wells field contains the unordered list of well objects with position and dimensional information. Each well object is defined by its ordering name or well name (ex. A1) and contains multiple fields:

* "depth": depth of the well. A positive number.
* "shape": Shape of the weels in the labware. Currently there are only two shapes supported: circular or rectangular. If 'rectangular', use xDimension and yDimension; if 'circular' use diameter:
  + "diameter": diameter of circular wells
  + "xDimension": x dimension of rectangular wells.
  + "yDimension": y dimension of rectangular wells.
* "totalLiquidVolume": Total well, tube, or tip volume in microliters
* "x": x location of centre-bottom of well in reference to left-front-bottom of labware. A positive number.
* "y": y location of centre-bottom of well in reference to left-front-bottom of labware. A positive number.
* "z": z location of centre-bottom of well in reference to left-front-bottom of labware. A positive number.

"wells": {

"A1": {

"depth": 2,

"shape": "circular",

"diameter": 3,

"totalLiquidVolume": 0.07,

"x": 13.5,

"y": 65,

"z": 60 }

}

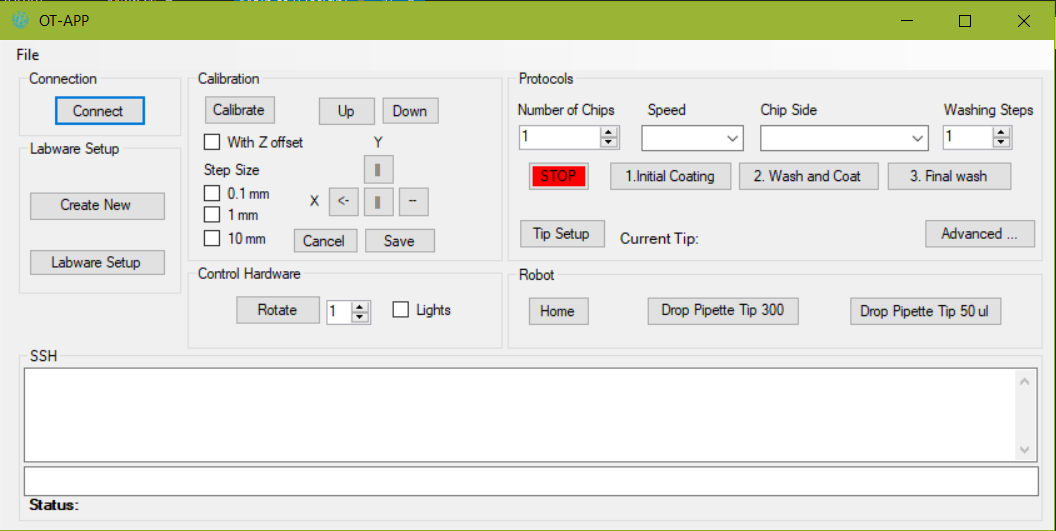
When combining all the definitions into the JSON file, the user can create new custom labware. The descriptions included here of the labware is limited and does not include more complex definitions and options. For an in-depth description of the labware definitions and the built functions please refer to the Schema 2 file on the [Opentrons GitHub Repository](https://github.com/Opentrons/opentrons/blob/edge/shared-data/labware/schemas/2.json).

# 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. The Coat APP (V1.0) has limited functionality as many functions are still on development. Nevertheless, the Coat app is open-sourced allowing developers to use the application framework and tools to add new protocols or routines.

## Interface

The Coat APP (V1.0) of the robot is currently created for the specific task of coating and washing the AX6 chip platform. The application main form includes seven 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 visible the coat app can detect automatically the address of the robot and connect to it.

### Calibration

This panel contains all the controls necessary to calibrate the labware with reference to the OT-2 robot system. In V1.0 the robot is only capable of calibrating the Chip Rotator on Slot 6 of the Deck. To calibrate other labware please refer to Section [3.4 Calibrate Tab](#_Calibrate_Tab).

### Control Hardware

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

### Labware Setup (On Development)

In V1.0 this module is currently on development. This module stores the current deck layout (positioning of the labware with respect to the robot’s deck) and allows the creation of new custom labware for the robot. Furthermore, this panel will be able to also edit existing labware. Currently, it is possible to create new labware and edit exiting one, however is only recommended to be used with the debug or development version of the coat app.

### Protocols

This panel includes all the controls to coat and wash the chips using the OT-2 robot and the external hardware. The user can select several options such as motor speed, number of chips to coat or the side to be coated.

### Robot

This panel has some functions to control the hardware of the robot, including the option of homing all axes of the robot or dropping pipette tips.

### SSH

The SSH panel is currently used for development on the coat app V1.0. The first box of the SSH panel displays a stream writer of the SSH connection. Within some protocols, this box will update the user of the status of the protocol. The second text box allows the user to manually send commands to the OT-2 robot via SSH connection. The SSH panel also contains a Status indicator, which changes its status when the robot successfully connects.

## Set-Up

***Note for All Users****: The coat app V1.0 is still on development. For this reason, son functions of the app might be unresponsive or cause the application to freeze. We are working on it. The Recommended method of using the app is on development mode using Visual Studio. Please refer to section* [*7.6 Development*](#_Development) *to start using the coat app*

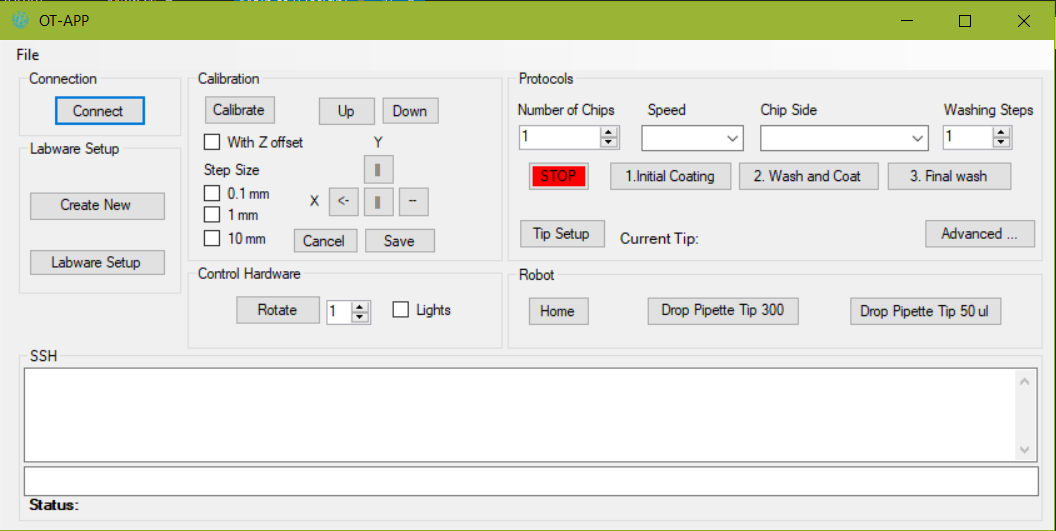
***Note for Alveolix Users****: If you are using the Laptop already connected to the OT-2 robot, this step is not required as the setup is already done on your machine.*

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 and open the SetupOTApp application. The installer will prompt the user to select an installation folder (the default is recommended). Once the application is installed launch it.

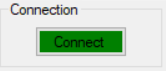
## Connecting to the Robot

Connecting to the OT-2 robot is simple and automatic. Follow the indications below:

1. Turn on the OT-2 robot, by activating the I/O switch on the back side of the robot.
2. Wait until the container has initialise on the robot. (Hint: The robot homes the pipettes always when initialising the container)
3. Connect the USB port to one of the USB ports on your machine.
4. Launch the Coat APP (This will take a few seconds)
5. On the Connection Panel press the **Connect** Button.



1. If, the connection of robot was successful the **Connect** button should turn green and the Status indicator on the SSH connection should say: “Status: Connected”



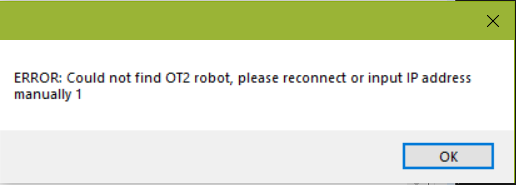


### Issues with connectivity

There can be several issues related to the connectivity of the robot. Below there is a list of the common issues that can come up when attempting connection to the robot:

#### Connection Error Box when launching the App

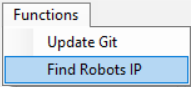
If the robot is not detected correctly or the IP address of the robot is not correct the following dialog box will show up:



This problem might be due to one of the following problems:

* The robot is not correctly connected to the host machine. Please, unplug the OT-2 robot and plug it in one of the USB ports.
* The Opentrons App is not installed. The Coat APP uses the Opentrons desktop application to find the IP address of the robot please ensure that the Opentrons App is installed and that the path of the app is found on a path: C:\Users\<Username>\AppData\Local\Programs.
* The robot has connectivity issues. Please restart the robot by switching it on and off. After switching it on wait a minute until the robot is fully restarted (Hint: this typically happens when the robot tries to home the axes), then connect the robot to the machine.

Once the robot is correctly connected, then reopen the Coat App or alternatively, use the *Find Robots IP* function, which can be found on the *Functions*>*Find Robots IP* on the menu bar.



This function will automatically find the IP Address of the robot. Then the user can connect to the robot using the *Connect Bottom.*

## Labware Setup

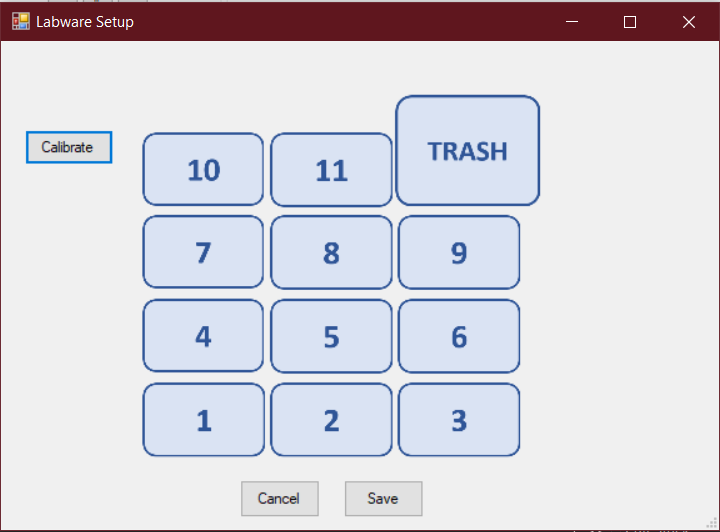
***Note for Alveolix Users****: In the Coat App V1.X the labware setup only supports one AX6 chip holder (3 chips) place on Slot 6, 300 microliter pipette tip rack on Slot 1 and a 24 well Opentrons 2ml Eppendorf rack on Slot 4.*

Before starting to run the protocols, it is necessary to ensure that all the labware is place in the correct slot on the Opentrons bed. The slot and location of the labware is by default defined on each individual python protocol in the labware section (refer to section [5.1.3. Labware](#_Labware)). In the Coat App V1.0, needs to edit the protocol individually to add or modify a new labware and its position on the opentrons robot. This method of implementation of the labware setup is optimal when the user runs routines that do not require any changes on the labware. In order to optimize the protocols for future implementations a set of function has been included in the Coat App to interact with the labware.

### Labware Setup Panel (On development)

***Note for Alveolix Users****: This function is not fully functional on the Coat App V1.X, please modify labware setup directly on the protocols. Refer to* [5.1.3. Labware](#_Labware).

The Labware Setup panel allows the user to interactively and visually design the layout of their experiment and labware setup. The layout is then saved and updated on all the protocols. The Labware setup UI main part is composed of a numbered grid that represents each of the slots on the OT-2 table.

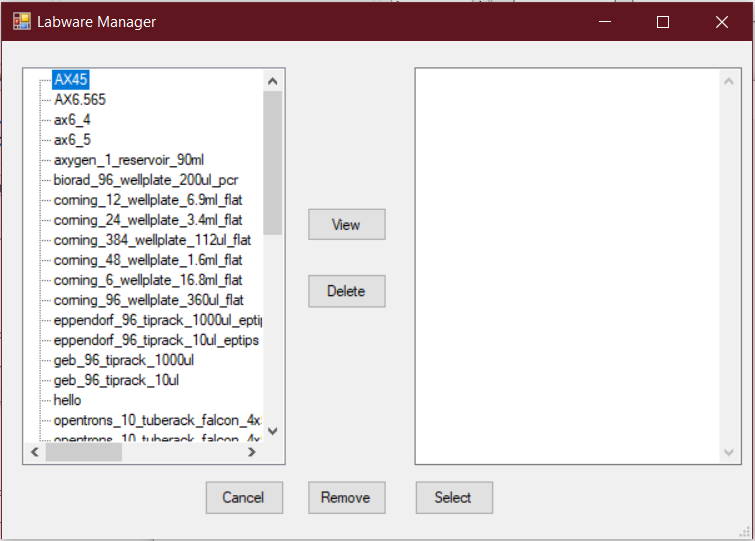


#### Adding and removing a labware to the setup

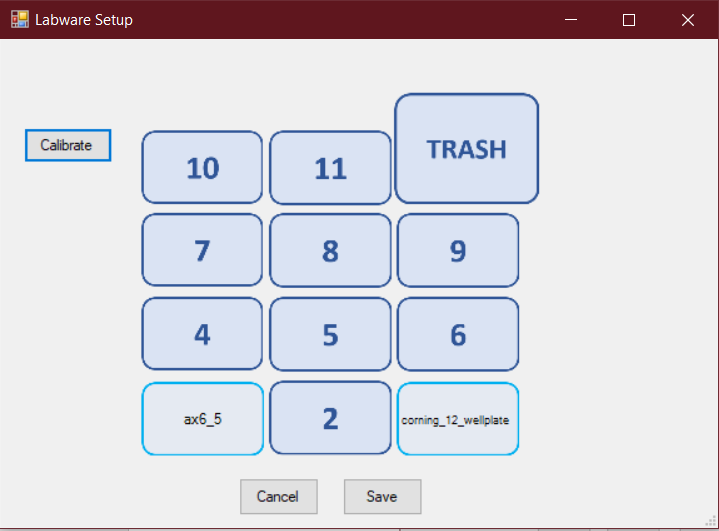
In order to add a new labware:

1. Select the desire slot on the grid. (The Labware Manager will open up)
2. Select the desire labware from the left panel and click the “Select” Button. Alternatively, the user can use the “View” Button to view the labware definition. In addition, if the slot selected already contained a labware it is possible to clear the slot using the “Remove” Button.

**Note:** Do not use the “Delete” button to remove labware from Slots. this will permanently delete the labware definition



1. The labware is added to the setup.
2. Once all labware is added, the “Save” button will implement the changes.



There are other functions contained within the Labware Function which are still on development as in V1.3. This include the possibility of visualizing and labelling individual wells within labware (This implementation is related to the possibility of having different solutions within each well or Eppendorf in a holder (Ex. AX-Sense), and being able to assign them on demand, instead of having to modify them on the protocols) and the function to calibrate all the labware, once they are added to the setup.

### Create new labware panel

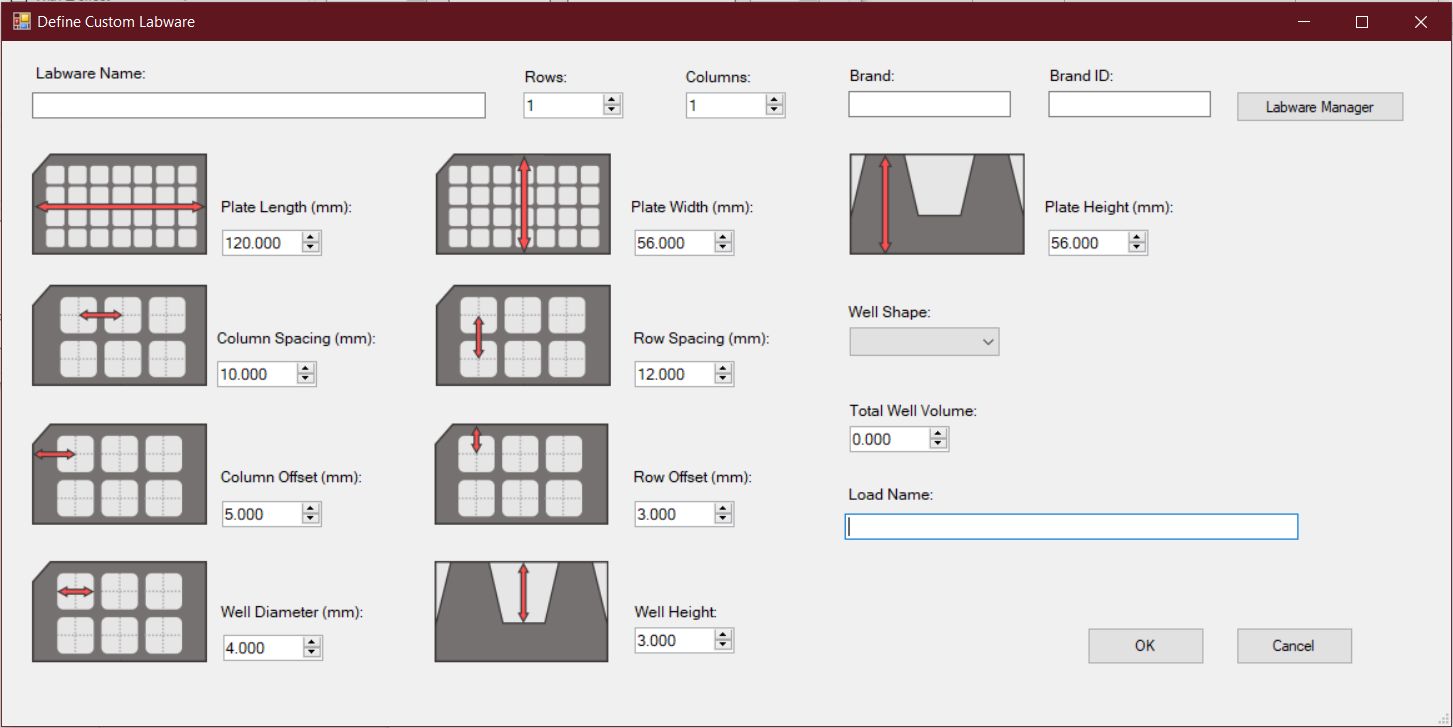
***Note for Users****: This function can only create a new wellplate type of labware (Labware that has a regular grid of wells holding an identical volume). This function can create new chip designs and modify old versions. In order to create other types of labware such as tipracks or containers please use the labware template and the manual on section* [*6. Labware*](#_Labware_1)*, to create a new labware definition*

The “Create New Labware” function allows users to define a new labware definition. To launch the “Create New Labware Panel”, click the button on the Labware Setup module.

In order to create a new labware please fill all the fields on the “Create New Labware Panel”.

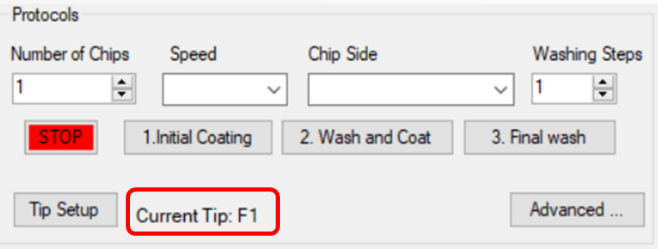
* Labware Name: Name of the labware. Can include spaces and symbols.
* Rows: Number of rows in the labware (Horizontal).
* Columns: Number of Columns in the labware (Vertical).
* Brand: Brand or manufacturer of the labware. Not Mandatory.
* BrandID: Identification number used to track the labware. Can also be used to keep track of the version of a new labware. Not mandatory.
* Plate Length: Total length of the labware in millimetres.
* Plate Width: Total width of the labware in millimetres.
* Plate Height: Total height of the labware in millimetres. Maximum height.
* Column Spacing: Space between each column, from the centre of the wells in millimetres.
* Row Spacing: Space between each row, from the centre of the wells in millimetres.
* Column Offset: Distance from the left edge of the labware to the first column in millimetres.
* Row Offset: Distance from the top edge of the labware to the first row in millimetres.
* Well diameter: Diameter of each well for circular wells, or length of the rectangle for rectangular wells in millimetres.
* Well height: Distance between the highest point of the well to the bottom of the well.
* Well Shape: Shape of the well, between Circular or Rectangular.
* Total Well Volume: Volume of each well. Can be used by the Opentrons software to limit the dispensing volume on each well.
* Load Name: This is the name used to load the labware on the protocols. The load name should not include symbols or spaces.

Alternatively, the user can modify an existing labware definition by selecting it from the “Labware Manager”. Once the new labware is created it will automatically update on the OT-2 machine and the user can start defining it in protocols.

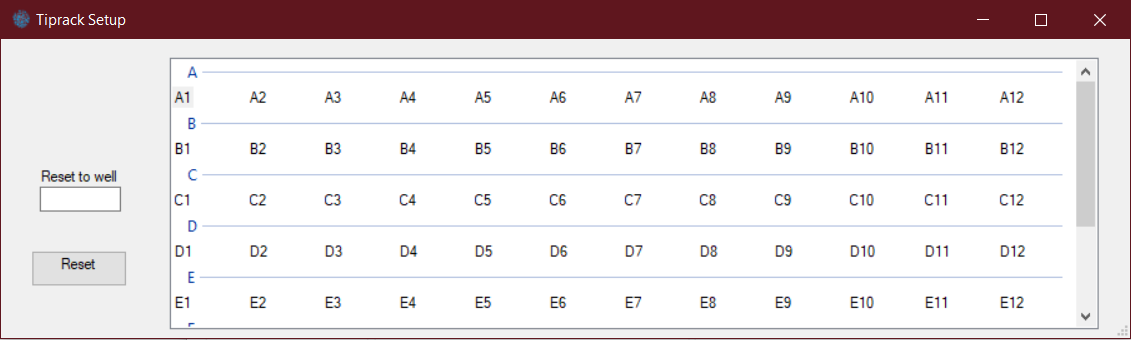


## Tip Set-up

When adding a new tip rack to the setup or in order to adjust the next pipette tip available for the protocol, a tip set-up is required. At all times the user can check the next available pipette tip on rack in Slot 1 by checking the **“Current Tip:”** status on the Protocol Panel:



If the **“Current Tip:”** Status is not correct and doesn’t represent the pipette tip selected. Then the user can launch the Tip Setup by pressing the **Tip Setup** button on the Protocols panel. In order to set a new current tip definition (Next available tip) on the Tip Setup:



1. Select your slot on the tip rack (Ex. A1 to reset to the tip located on the tip rack at position A1). The selected tip should appear on the *Reset to well* box.
2. Hit Reset.
3. Close the Tiprack Setup dialog.

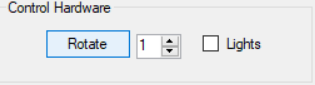
The new selected tip should update on the Coat App main window.

## Calibration

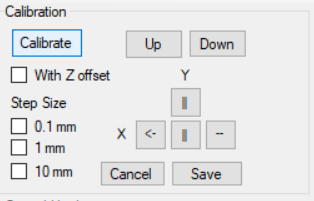
***Note for Users****: In V1.X of the Coat App the Calibration function can only perform a calibration on the custom design labware (AX6) and only on the Slot* ***6*** *of the table of the OT-2 robot. In order to modify the labware to calibrate please refer to* [*7.6.1. Editing the Calibration Protocol*](#_Editing_the_Calibration)

From run to run there might be small fluctuations on the position of the chip rotator and other hardware within the deck of the robot. The calibrate panel allows you to calibrate the AX6 accurately to ensure a good performance of the protocols and avoiding breaking the membranes or the pipette modules. The calibration process is recommended to be done with a dummy chip, before using the newly assembled chip, to avoid breaking it. The calibration function only works when the coat app is connected to the OT-2 robot. Please ensure that the robot is connected before attempting to calibrate (Section [7.3 Connecting to the robot](#_Connecting_to_the)) In order to calibrate the Chip Rotator:

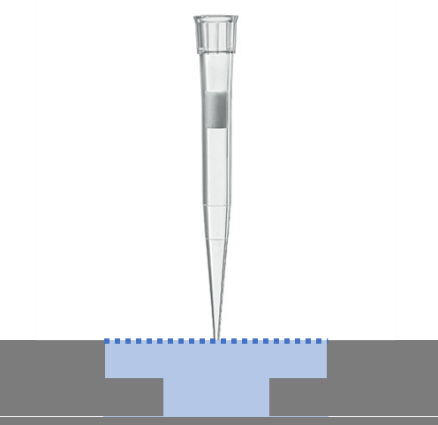
1. Insert a dummy chip onto the 1st slot of the Chip Rotator Holder.
2. Place the holder onto the Chip Rotator.
3. On the **Hardware Panel** of the coat app, on the number box set the value to **1 or “Apical” (V1.3)** and press the **Rotate Button.** This will home the chip rotator and move it to its apical position.



1. On the **Calibration panel** of the coat app, Press the **Calibrate Button**. This will launch the Calibration Setup. The robot will pick up a new pipette tip and move to well A1 on the chip rotator. Note, if you are unsure of accuracy of the labware definition you are using or, please launch the calibration using the “With Z offset” (Clicking the box), to avoid damage to the robot or labware. The “With Z offset” will launch the calibration setup and it will move the to the top of well A1 with an increase clearance to avoid contacting with any element.



1. In order to jog the pipette in one of the three axes, please select a step size between **0.1 mm, 1 mm or 10 mm**, by ticking the boxes. Once the step size is selected move the robot by pressing the intended direction and axis. The **Up** and **Down** buttons move the Z axis of the robot, the X buttons move the X axis of the robot (Ex. To move from Slot 1 to 3 on the deck) and the Y buttons move the Y axis of the robot (Ex. To move from Slot 1 to 10 on the deck). The step size can be modified at any time during the Calibration Setup.
2. For a good calibration, move the pipette tip so that its end sits flush with the top of well A1 as shown in the diagram below. Also ensure that the pipette tip is located on the centre of the A1 well.



1. Once the optimal position is reached, press the button **Save**. This will change the cornerOffsetFromSlot property on the Labware definition of the robot, to match the new position.
2. Otherwise, if the calibration was not successful or the user wants to reset the previous position press the **Cancel** Button. This will not save the new added changes.

### Editing the Calibration Protocol

The calibration protocol can be modified to allow the calibration of additional labware or to modify the current definition of the AX6 used. In order to modify the calibration, edit the python protocol “calibration\_pro1.py”. In order to update the new calibration settings, use the update\_offset function on the protocol. Please refer to section [8.1.2. Calibration – Updating offset](#_Calibration_–_Updating).

## Coating and Washing

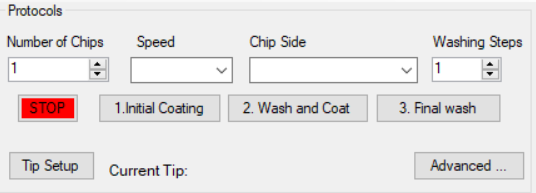
The main purpose of the Coat App is to allow to coat and wash the AX6 chip effectively. Once the labware setup and calibration are performed correctly for a newly added setup the protocols can be run. The Coat App V1.X supports three protocols according to the requirements of the AX6 coating.

* Coating of the Membrane with AX Treat (Initial Coating): The initial coating with the AX Treat is done by dispensing a small drop of the solution over the chip membrane.
* Washing and Coating with AX Sense (Wash and Coat): This protocol allows the wash of AX Treat from the membrane and the dispensing of the AX Coating.
* Washing the AX Sense (Final Wash): This final protocol removes the remaining free coating.

A schematic representation of the steps of each protocol can be found on the [Appendix 1 – AX Protocols](#_Appendix_1_–).

### Running a Protocol

In order to run any of the protocols, follow the instructions below:



1. Ensure that all components are correctly place on each slot as described by the protocol and that there are no lids of elements on the way of the OT-2 robot.
2. Ensure that the highlighted tip on the Protocols Module is correct.
3. Select the “Number of Chips”, “Speed” (Recommended Medium) and the “Side of Chip” to coat or wash. Additionally, if you are washing the chips with the “Final Wash” Protocol you can select the Number of washes to perform by modifying the “Washing Steps” on the module.
4. Once all parameters are selected select the desired protocol by clicking on its button.

During the runtime of the protocol the “Stop” Button can be pressed at any point to stop or forced shutdown the protocol.

### Advanced Protocols

In order to finely optimized the conditions of new protocols or to run pilot experiments the advanced protocol function is available. The Advanced Protocols panel allows to coat or wash each individual well with a set of parameters. This include the

### Modifying a Protocol

If the protocol is not running as expected or changes are made to the setup the user can modify the protocol easily by editing it.

## Additional Functions

The Coat App includes other functions to control the hardware of the OT-2 robot. These include:

* Home Function: The home function resets the axis of the robot and moves the pipette module and axis to its home position.
* Drop Pipette Tip 50: Drops the pipette tip attached to the P50 pipette. Then homes the robot.
* Drop Pipette Tip 300: Drops the pipette tip attached to the P300 pipette. Then homes the robot.

## Development

### Launching the Coat App on Development Mode

**Note**: It is recommended that in order to modify the source code of the Coat App the user has some knowledge on .NET, C# Programming and Windows Forms.

In order to continue developing or adding new functions to the Coat App the whole application can be edited and launched using Windows Visual Studio. This will allow the user to edit, change and modify the behaviour of the application, connectivity with the robot, change settings. To launch it open Windows Visual Studio and find the Coat App Project, “Coat-APP.sln”. Refer to [9.1.6. Coat-App](#_Coat-APP)

### Source Code

The source code on the windows forms consists in different windows also known as forms that can receive and display inputs and send outputs to other forms. Most of the code is located under the Main form.

# 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

### 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 and returns the current calibration offset of the labware or updates the calibration offset after the calibration command is run.

update\_offset(labware\_name, method, x\_val=0.0, y\_val=0.0, z\_val=0.0)

**Reading Mode, method = “True”**

update\_offset(labware\_name,True) #returns the saved offset for the labware.

**Writing Mode, method = “False”**

update\_offset(labware\_name, False, 2, 3, 5) #updates the offset for the labware name to (2, 3, 5)[x, y, z]

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

## 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.

## File Location

This guide will help you find the file within the Coat App repository. The following

### Protocols

This folder contains all the python protocols that allow to run the OT-2 robot. Below there is a list with the most useful protocols and a short description.

|  |  |
| --- | --- |
| Protocol Name | Description |
| calibration\_pro1.py | Calibration protocol, updates labware offset |
| drop\_300\_tip.py | Drop P300 pipette tip |
| drop\_50\_tip.py | Drop P50 pipette tip |
| home.py | Homes the robot axis |
| switch\_lights.py | Controls lights inside the OT-2 |
| Initial\_Coating\_Protocol.py | Initial Coating Protocol, coats the chip with AX-Treat solution |
| washing\_and\_coating\_protocol.py | Washing and Coating Protocol, washes, dries, and coats the chip with AX-Sense |
| chip\_washing\_rotator.py | Final Washing, washes and dries well |

### Library (On development)

Contains the CHIPPIE Python Module. All the functions and configuration files are stored in this location.

### Documentation

Contains all the documentation and tutorials on the usage of the Coat App, OT-2 Robot and Opentrons API.

### Patches

Contains all the software and update patches for updating the local git inside the OT-2 robot. This will be use in a future update of the Coat App to update the protocols and files of the OT-2 without an online connection.

### bin

Contains security and configuration binary files use to run the Coat App.

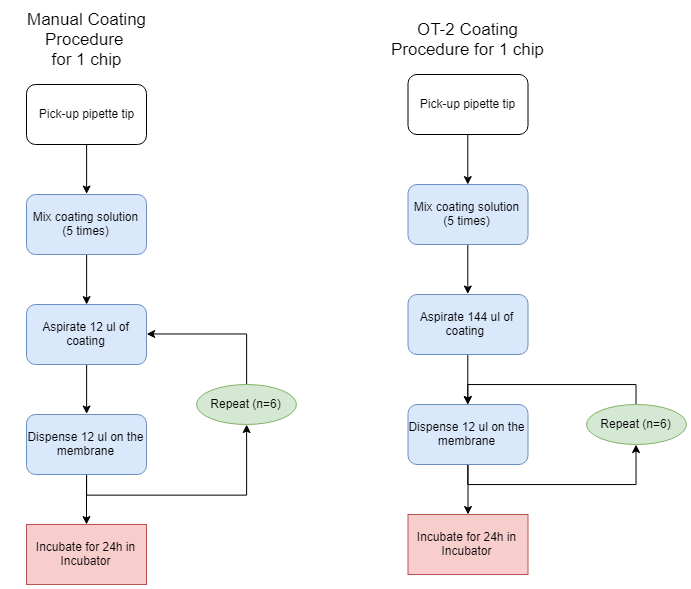
### Coat-APP

This folder contains all the source files and project files used to develop the Coat App. In this location reside the Visual Studio Solution (.SLN) file that users can use to edit and develop the Coat App in Visual Studio.

# Appendix 1 – AX protocols

## Coating Protocol

The following flowchart summarizes the current coating protocol used in V1.3 of the Coat App and compares it to the Manual Coating Procedure used in Alveolix.



## Washing Protocol

The following flowchart summarizes the current washing protocol used in V1.3 of the Coat App and compares it to the Manual Washing Procedure used in Alveolix.

