Automate Biology Lab Protocols with the OT-2 Python Protocol API

Scientist-To-Scientist Webinar Series – September 22, 2021

opentrons

PRLNYC
Pandemic Response Lab

Speakers

Presenter: Nick Diehl

Applications Engineer, Opentrons

Nick graduated from Vanderbilt University with a Biomedical Engineering degree in 2018. He joined Opentrons in 2019 to deliver bespoke solutions to OT-2 users around the world.

Moderator: Mike Asham

Director of Applications Sciences and Solutions, Opentrons

Mike joined Opentrons in 2021 as the Director of Applications Sciences & Solutions. He has a diverse automation background with over 10+ years in the field using a variety of high throughput platforms and research experience specializing in Genomics & Molecular Oncology.

Webinar Overview

- Opentrons API overview
- Accessing and modifying protocols from the Opentrons Protocol Library
- Integrating any piece of labware into your OT-2 workflow
- Writing your own Python protocols from scratch

Creating protocols for the OT-2

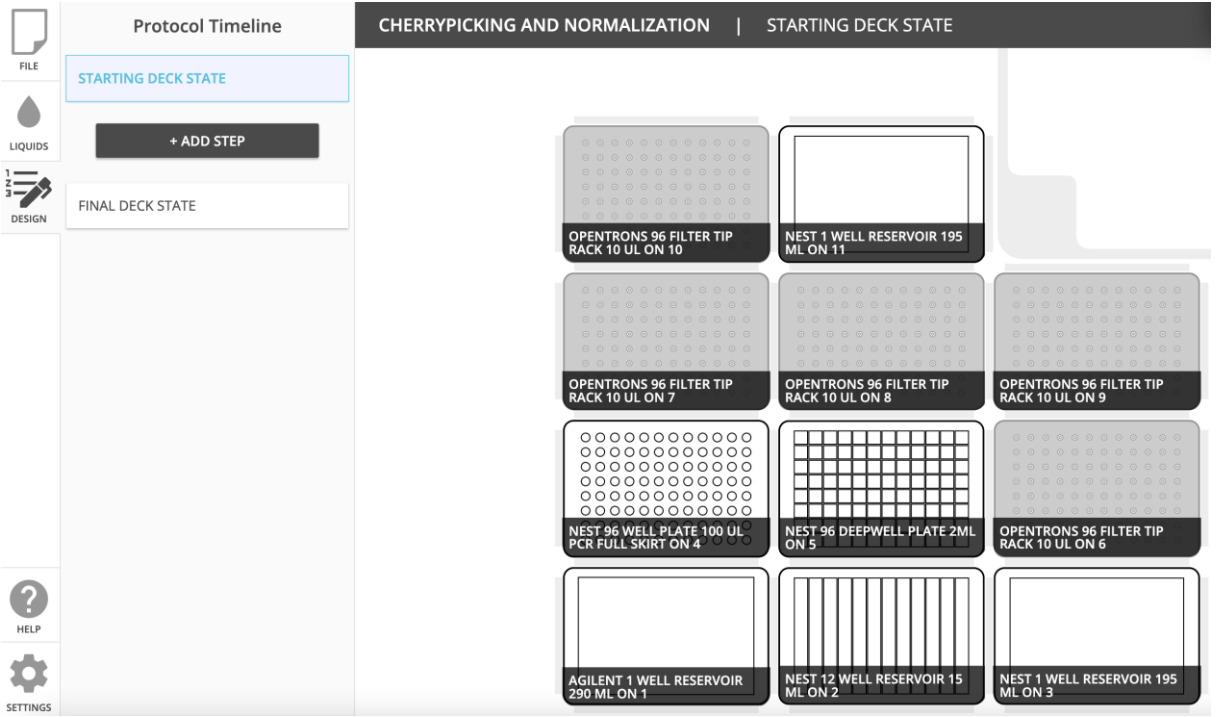
A **protocol** for the OT-2 is a set of instructions written in code that gives the OT-2 a step-by-step procedure to execute

- Liquid handling steps
- Non-liquid handling steps

There are currently 2 main paths to create protocols on the OT-2:

1. Protocol Designer (.json files)
2. Python API (.py files)

Protocol Designer



Protocol Designer

PROTOCOL DESIGNER		PYTHON API
✓	Intuitive drag and drop graphical interface	
✓	Visual labware and liquid management	
✓	Ability to customize pipetting technique	✓
✓	Opentrons Standard Labware	✓
✓	Custom Labware	✓
✓	Opentrons Modules	✓
	CSV Import	✓
	User Defined Variables and Conditional Logic	✓

Opentrons Python API

What is an API?

- Application Programming Interface
- A type of software interface, offering a service to other pieces of software
- Often allows a user to write code in a high-level programming language to interact with lower-level code

Opentrons Python API

What is the Opentrons Python API?

- Simple Python framework designed to make writing automated biology lab protocols easy
- Accessible to anyone with basic Python and wetlab skills
- Allows user to code automated protocols in a way that reads like a lab notebook
- Open-source and well-documented

OT-2 API V2

Python Protocol API

Watch 36

Table Of Contents

Welcome to version 2 of the Opentrons OT-2 Python Protocol API!

This is the new API for writing Python protocols for the Opentrons OT-2. It has all of the same features as version 1 of the Python Protocol API, plus support for new products like the Opentrons Thermocycler Module.

OT-2 Python Protocol API Version 2

The OT-2 Python Protocol API is a simple Python framework designed to make writing automated biology lab protocols easy.

We've designed it in a way we hope is accessible to anyone with basic Python and wetlab skills. As a bench scientist, you should be able to code your automated protocols in a way that reads like a lab notebook.

Version 2 of the API is a new way to write Python protocols. It is more reliable, simpler, and better able to be supported. Unlike version 1, it has support for new modules like the Thermocycler. While version 1 will still recieve bug fixes, new features and improvements will land in version 2. For a guide on transitioning your protocols from version 1 to version 2 of the API, see this article on migration. For a more in-depth discussion of why version 2 of the API was developed and what is different about it compared to version 1, see this article on why we wrote API V2.

Getting Started

New to Python? Check out our Using Python For Protocols page first before continuing. To get a

What can our Python API do?

- Automate a wide variety of scientific workflows:
  - NGS prep
  - Nucleic acid extractions
  - Serial dilutions
  - End-to-end PCR
  - Cherrypicking and concentration normalization
- Read and write to local files on the robot
- Integrating custom labware (tips, plates, tubes, etc.) on the robot deck

Protocol Library

Protocol Library

COVID WORKSTATION

- qPCR Setup
- Sample Plating
- RNA Extraction

FEATURED

- NGS Library Prep: Swift 2S Turbo
- Cherrypicking
- Nucleic Acid Purification with Magnetic Beads (Universal)
- NGS Cleanup and Size Selection with Omega Bio-tek Mag-Bind® TotalPure
- PCR Prep
- NGS Library Prep: Illumina Nextera XT
- Serial Dilution

Featured Protocols

Beckman Coulter RNAdvance Viral RNA Isolation

Author: Opentrons | Partner: Beckman Coulter Life Sciences

Nextera XT DNA Library Prep Kit Protocol: Part 1/4 - Tagment Genomic DNA and Amplify Libraries

Author: Opentrons | Partner:

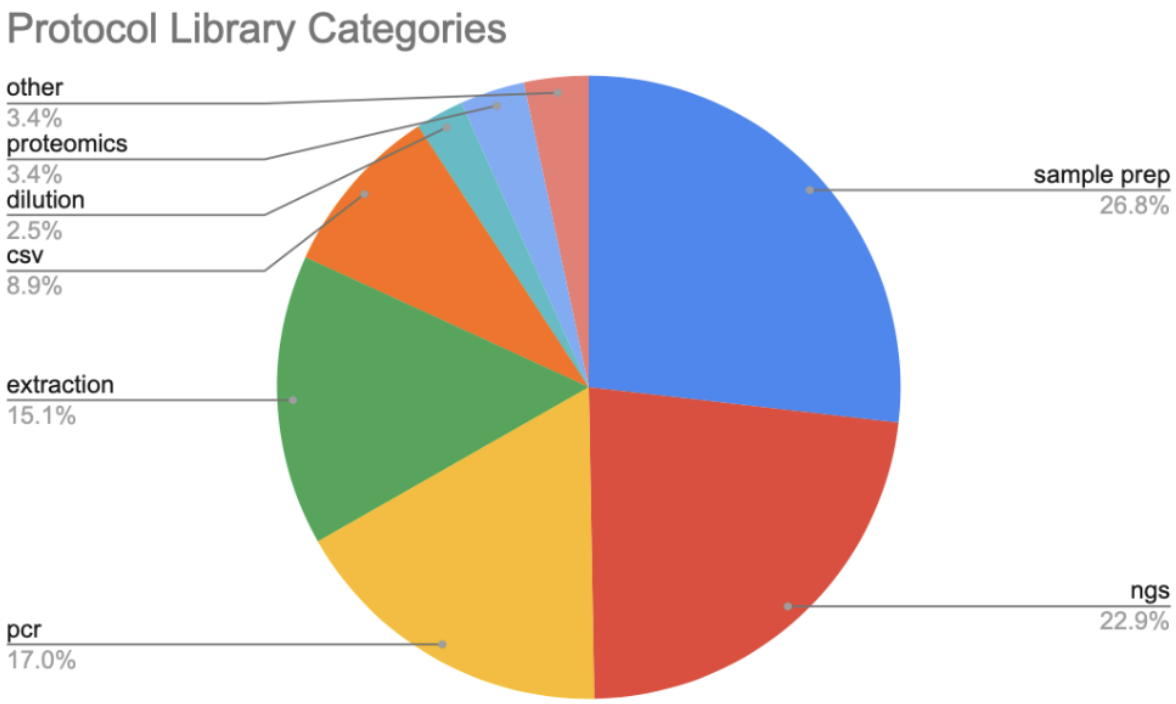
With this series of protocols and the Opentrons Magnetic Module, your robot can complete a library prep using the Illumina Nextera XT DNA Library Prep Kit. This library prep protocol comes in four parts: Tagment and Amplify, Clean Up Libraries, Normalize Libraries, and Pool Libraries.

NGS Cleanup and Size Selection with Omega Bio-tek Mag-Bind® TotalPure NGS

Author: Opentrons (verified) | Partner: Omega Bio-tek

With this protocol, you can perform high-quality nucleic acid purifications using the Opentrons Magnetic Module and Omega Bio-tek Mag-Bind® TotalPure NGS magnetic beads. This kit is widely used in NGS cleanup for its affordability and simplicity. You can select specific sizes of nucleic acids by varying the bead-to-DNA ratio across a wide array of fragment sizes. For reagent and module purchasing details contact info@opentrons.com.

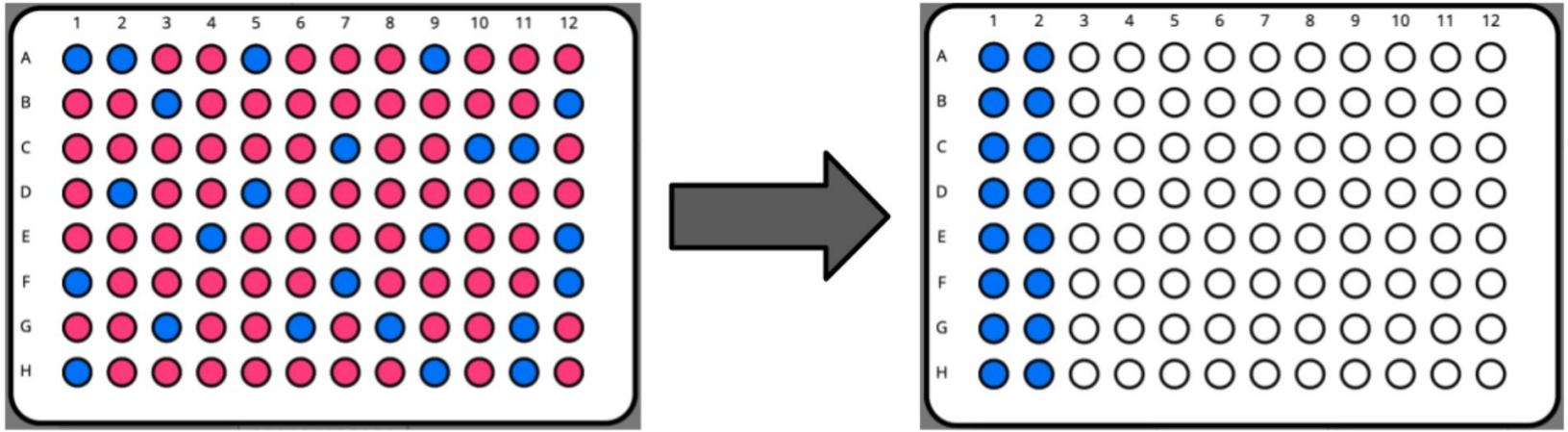
Protocol Library



Cherrypicking

Author : Opentrons

Our most robust cherrypicking protocol. Specify aspiration height, labware, pipette, as well as source and destination wells with this all inclusive cherrypicking protocol.



Explanation of complex parameters below:

- `input .csv file`: Here, you should upload a .csv file formatted in the following way, making sure to include headers in your csv file. Refer to our Labware Library to copy API names for labware to include in the `Source Labware` and `Dest Labware` columns of the .csv.
- `Pipette Model`: Select which pipette you will use for this protocol.
- `Pipette Mount`: Specify which mount your single-channel pipette is on (left or right)
- `Tip Type`: Specify whether you want to use filter tips.

Protocol Structure: Overview

Organization:

1. Metadata and Version Selection
2. Run function
3. Labware
4. Pipettes
5. Commands

```python
from opentrons import protocol_api

# metadata
metadata = {
    'protocolName': 'My Protocol',
    'author': 'Name <email@address.com>',
    'description': 'Simple protocol to get started using OT2',
    'apiLevel': '2.10'
}

# protocol run function. the part after the colon lets your editor know
# where to look for autocomplete suggestions
def run(protocol: protocol_api.ProtocolContext):

    # labware
    plate = protocol.load_labware('corning_96_wellplate_360ul_flat', '2')
    tiprack = protocol.load_labware('opentrons_96_tiprack_300ul', '1')

    # pipettes
    left_pipette = protocol.load_instrument(
         'p300_single', 'left', tip_racks=[tiprack])

    # commands
    left_pipette.pick_up_tip()
    left_pipette.aspirate(100, plate['A1'])
    left_pipette.dispense(100, plate['B2'])
    left_pipette.drop_tip()
```

Protocol Structure: Metadata + Version Selection

```
metadata = {
    'protocolName': 'My Protocol',
    'author': 'Name <email@address.com>',
    'description': 'Simple protocol to get started using OT2',
    'apiLevel': '2.10'
}
```

metadata

- Dictionary of data that is read by the server and returned to client applications
- Required element of the metadata is apiLevel
- Other elements shown here are useful but not required by the robot server

Protocol Structure: Run Function and the Protocol Context

```
def run(protocol: protocol_api.ProtocolContext):
```

run() function

- Must be named exactly run
- Exactly one mandatory argument...

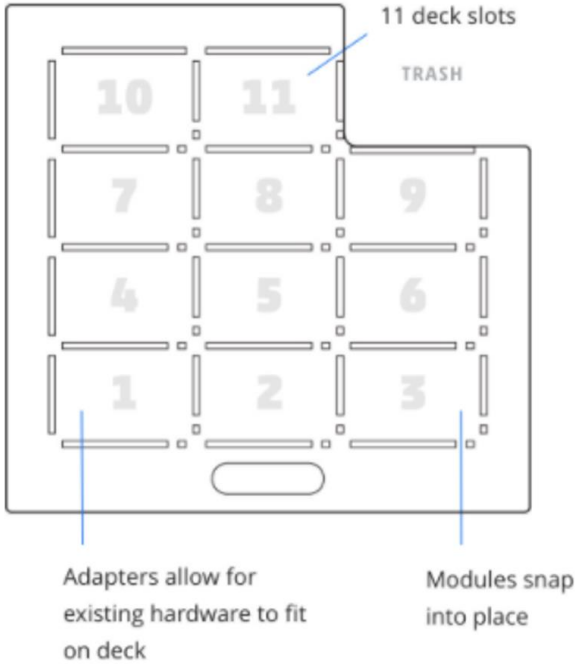
protocol context

- Represents the robot and its capabilities
- Main responsibilities:
  1. Remember, track, and check the robot's state
  2. Expose the functions that make the robot execute actions

Protocol Structure: Labware

```
tiprack = protocol.load_labware('opentrons_96_tiprack_300ul', '1')
```

Labware refers to plates, reservoirs, tipracks, tuberacks, and any other static products used in liquid handling

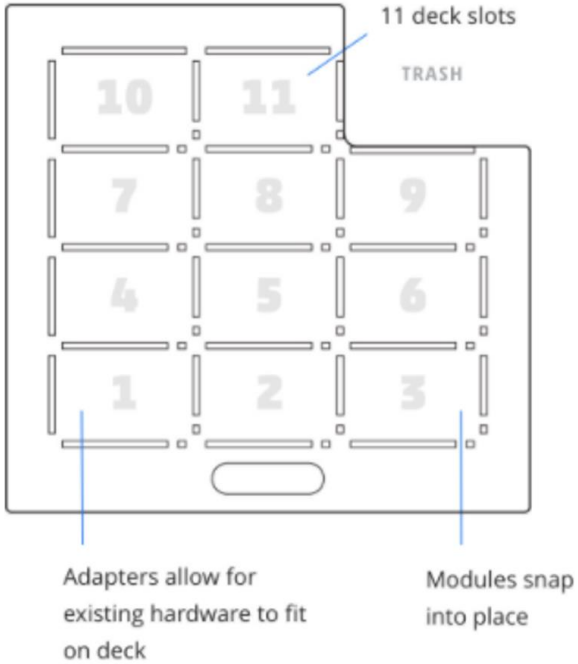


Protocol Structure: Labware

```
tiprack = protocol.load_labware('opentrons_96_tiprack_300ul', '1')
```

Loading

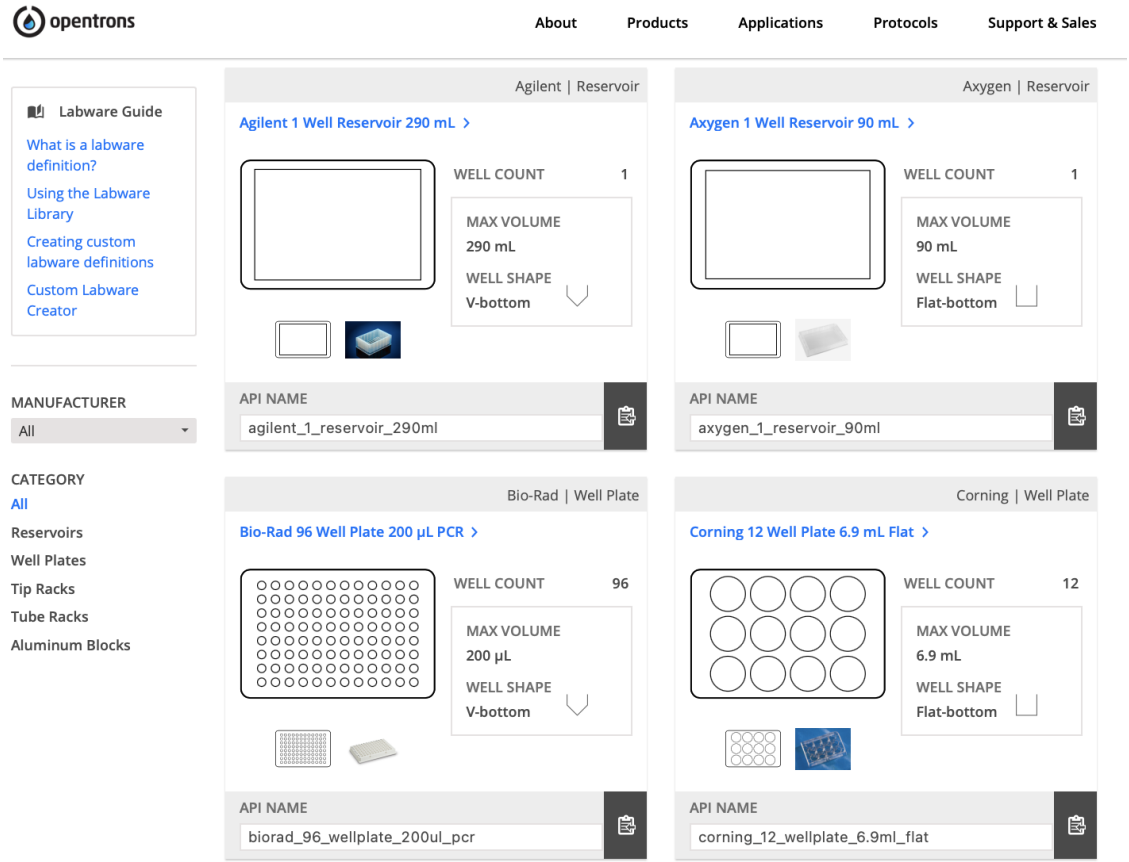
- Each labware is loaded onto 1 of 11 **slots** on the Opentrons **deck**, or liquid handling surface (the 12th slot is reserved for the built-in Opentrons trash container)
- The load_labware() method is called on the protocol context we've just discussed. It takes these arguments:
  1. A **loadname** that points to information about the physical dimensions of the labware and its comprising wells
  2. A **slot** (1-11) that points to the labware's physical location on the deck
  3. (optional) a display name



Protocol Structure: Labware

Labware Library

- All of Opentrons standard labware can be found here
- If your labware is not found here, you can create your own definition with our Labware Creator tool— it's best to have a technical drawing on hand for this
- This generates a .json labware file that can be added to the Opentrons App for protocol use



Labware Guide

- What is a labware definition?
- Using the Labware Library
- Creating custom labware definitions
- Custom Labware Creator

MANUFACTURER

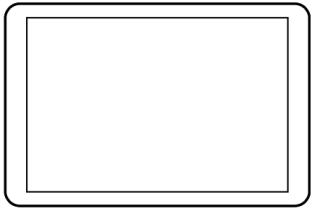
All

CATEGORY

- All
- Reservoirs
- Well Plates
- Tip Racks
- Tube Racks
- Aluminum Blocks

Agilent | Reservoir

Agilent 1 Well Reservoir 290 mL >



WELL COUNT 1

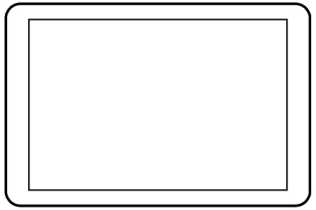
MAX VOLUME	WELL SHAPE
290 mL	V-bottom

API NAME

agilent_1_reservoir_290ml

Axygen | Reservoir

Axygen 1 Well Reservoir 90 mL >



WELL COUNT 1

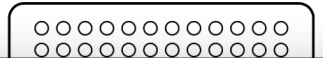
MAX VOLUME	WELL SHAPE
90 mL	Flat-bottom

API NAME

axygen_1_reservoir_90ml

Bio-Rad | Well Plate

Bio-Rad 96 Well Plate 200 µL PCR >



WELL COUNT 96

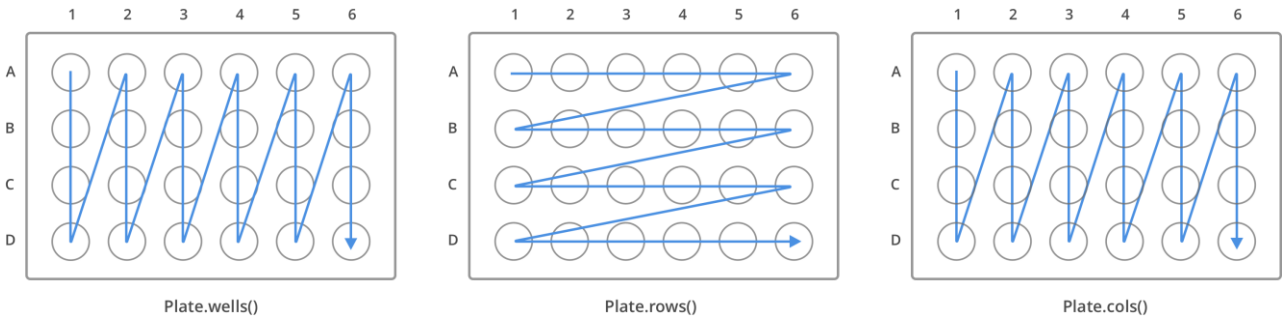
Corning | Well Plate

Corning 12 Well Plate 6.9 mL Flat >

WELL COUNT 12

Protocol Structure: Labware

Accessing groups of wells



Returns 1-d list

Return 2-d list

Return dictionary

`Labware.wells()`	List of all wells, i.e. `[labware:A1, labware:B1, labware:C1...]`
`Labware.rows()`	List of a list ordered by row, i.e `[[labware:A1, labware:A2...], [labware:B1, labware:B2..]]`
`Labware.columns()`	List of a list ordered by column, i.e. `[[labware:A1, labware:B1..], [labware:A2, labware:B2..]]`
`Labware.wells_by_name()`	Dictionary with well names as keys, i.e. `{'A1': labware:A1, 'B1': labware:B1}`
`Labware.rows_by_name()`	Dictionary with row names as keys, i.e. `{'A': [labware:A1, labware:A2..], 'B': [labware:B1, labware:B2]}`
`Labware.columns_by_name()`	Dictionary with column names as keys, i.e. `{'1': [labware:A1, labware:B1..], '2': [labware:A2, labware:B2..]}`

PRLNYC opentrons

```python
metadata = {
    'protocolName': 'Well accession',
    'author': 'Opentrons <protocols@opentrons.com>',
    'apiLevel': '2.10'
}


# Start protocol
def run(ctx):

  plate = ctx.load_labware(
    'nest_96_wellplate_100ul_pcr_full_skirt', '1',
    'Nick\'s plate')

  # groups
  all_wells = plate.wells()
  all_rows = plate.rows()
  all_columns = plate.columns()

  for well in all_wells:
    print(well)
  # for row in all_rows:
  #   print(row)
  #   print('\n')
  # print(len(all_columns))

  # individual
  well_H3 = plate.wells_by_name()['H3']
  wells_10_thru_12 = plate.wells()[9:12]
```

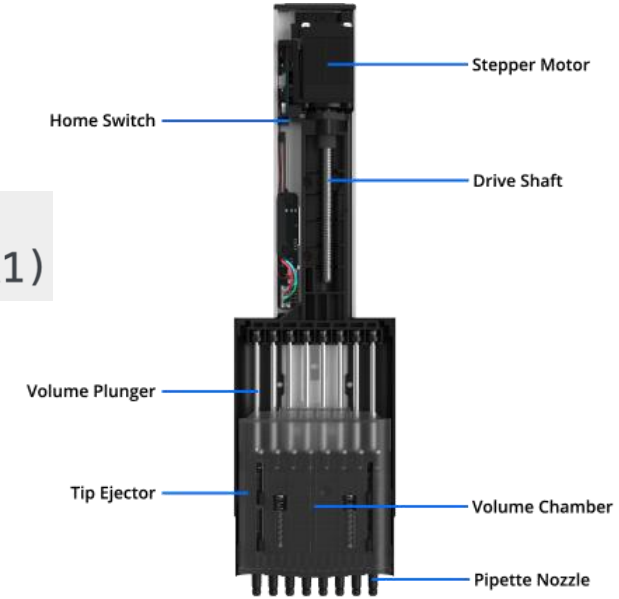
```
If you're looking to install packages, try our packager. Learn more.
~/webinar$ opentrons_simulate main.py
```

Protocol Structure: Pipettes

```
right = protocol.load_instrument(
    'p300_multi_gen2', 'right', tip_rack=tiprack1)
```

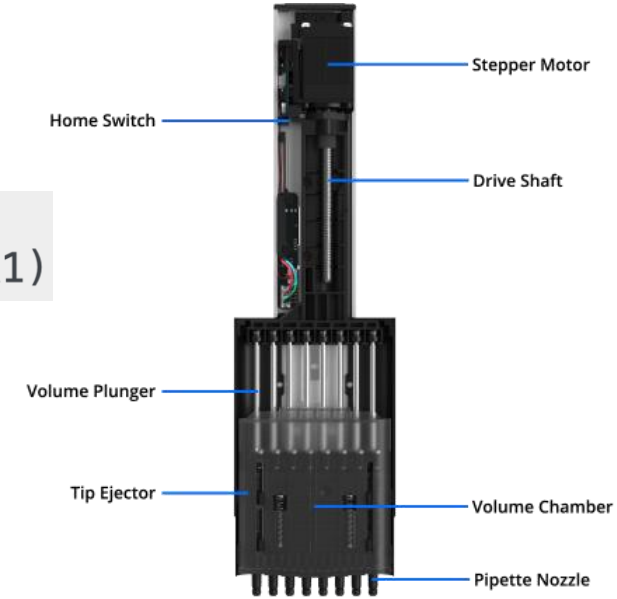
Pipettes refer to the Opentrons fleet of electronic pipettes

Pipette Type	Model Name
`P20 Single GEN2` (1 - 20 µL)	`'p20_single_gen2'`
`P300 Single GEN2` (20 - 300 µL)	`'p300_single_gen2'`
`P1000 Single GEN2` (100 - 1000 µL)	`'p1000_single_gen2'`
`P300 Multi GEN2` (20-300 µL)	`'p300_multi_gen2'`
`P20 Multi GEN2` (1-20 µL)	`'p20_multi_gen2'`



Protocol Structure: Pipettes

```
right = protocol.load_instrument(
    'p300_multi_gen2', 'right', tip_rack=tiprack1)
```



Pipettes refer to the Opentrons fleet of electronic pipettes

Loading

- The OT-2 gantry houses **2 pipette mounts** (left and right) that move in 3D space
- The load_instrument() method is called on the protocol context we've just discussed. It takes these arguments:
    1. A **model** pointing to the range and type (single- or multi-channel) of the pipette
    2. A **mount** (left or right)
    3. (optional) tipracks (an instance of **labware**) to automatically iterate over tip pickups

Protocol Structure: Pipettes

Building block commands:

```
tipracks300 = [
    ctx.load_labware('opentrons_96_tiprack_300ul', slot)
    for slot in ['2', '3']]
m300 = ctx.load_instrument('p300_multi_gen2', 'right',
                           tip_racks=tipracks300)
m300.pick_up_tip()
m300.flow_rate.aspirate = 300
m300.flow_rate.dispense = 300
m300.mix(10, 200, source)
m300.aspirate(100, source)
m300.dispense(100, dest)
m300.blow_out(dest)
m300.drop_tip()
```

Protocol Structure: Pipettes

Complex commands:

- transfer() (1 to 1)
- distribute() (1 to many)
- consolidate() (many to 1)

Keyword arguments:

new_tip → str {'once', 'always', 'never}
mix_before, mix_after → tuple (mix repetitions, mix volume)
touch_tip → bool {True, False}
air_gap → float
blow_out → bool {True, False}
disposal_volume → float

```
m300.transfer(100, sources, dests, air_gap=100, new_tip='always',
              mix_after=(5, 50))
m300.distribute(100, source, dests, touch_tip=True)
m300.consolidate(100, sources, dest, blow_out=True)
```

Protocol Structure: Pipettes

```python
metadata = {
    'title': 'Inheritance',
    'author': 'Nick Diehl',
    'apiLevel': '2.10'
}


def run(ctx):

  number_of_wells = 20
  plate = ctx.load_labware(
      'nest_96_wellplate_100ul_pcr_full_skirt', '4', 'PCR plate')
  reservoir = ctx.load_labware(
    'nest_12_reservoir_15ml', '2', 'reagent reservoir')
  tipracks = [ctx.load_labware('opentrons_96_tiprack_300ul', '1')]

  p300 = ctx.load_instrument(
    'p300_single_gen2', 'right', tip_racks=tipracks)

  volume = 30
  source = reservoir.wells_by_name()['A2']
  destinations = plate.wells()[:number_of_wells]

  p300.distribute(volume, source, destinations, disposal_volume=20)
```

```
~/webinar$ opentrons_simulate main.py
```

Protocol Structure: Modules

Modules are peripherals that attach to the OT-2 to extend its capabilities

Temperature Module	On Deck Thermocycler	Magnetic Module	HEPA module
			

Protocol Structure: Modules

Loading

- Modules are connected to the OT-2 via USB connection
- The load_module() method is called on the protocol context we've just discussed. It takes these arguments:
    1. A **model** pointing to the type of module
    2. A **slot** (1-11)
- Labware are loaded *directly onto the module*, rather than on the protocol context

```
tempdeck = ctx.load_module('temperature module gen2', '1')
magdeck = ctx.load_module('magnetic module gen2', '4')
thermocycler = ctx.load_module('thermocycler')

temp_plate = tempdeck.load_labware(
    'opentrons_96_aluminumblock_nest_wellplate_100ul')
mag_plate = magdeck.load_labware(
    'nest_96_wellplate_100ul_pcr_full_skirt')
tc_plate = thermocycler.load_labware(
    'nest_96_wellplate_100ul_pcr_full_skirt')
```

Protocol Structure: Modules

Commands

- Temperature and magnetic module

```
tempdeck.set_temperature(4)
magdeck.engage(height=6.8)
magdeck.disengage()
```

Protocol Structure: Modules

Commands

- Thermocycler

```
tempdeck.set_temperature(4)
magdeck.engage(height=6.8)
magdeck.disengage()

thermocycler.open_lid()
thermocycler.set_lid_temperature(95)
thermocycler.set_block_temperature(4)
thermocycler.close_lid()
profile = [
    {'temperature': 10, 'hold_time_seconds': 30},
    {'temperature': 60, 'hold_time_seconds': 45}]
thermocycler.execute_profile(steps=profile, repetitions=100,
                             block_max_volume=30)
thermocycler.open_lid()
```

Protocol Structure: Non-liquid handling commands

These commands, including comment(), delay(), and pause(), are called on the protocol context as shown here:

```
ctx.comment('This is a comment. Robot function will continue!')
ctx.delay(minutes=1, seconds=30, msg='Delaying for 90 seconds.')
ctx.pause('Delay over. Pausing until user input.')
```

Protocol Structure: High-touch control

0.1mm movement resolution in all 3 dimensions

Well position modifiers:

```
top_loc = well.top(-2)
bottom_loc = well.bottom(0.5)
center = well.center()
pipette.move_to(bottom_loc)
```

Advanced use cases

Custom 3rd-party hardware integrations (ex: heater shakers, linear actuators)

HTTP Requests (live data reading and writing)

Liquid Height tracking

```python
from opentrons.protocol_api.labware import Well
import math

metadata = {
    'title': 'inheritance',
    'author': 'Nick Diehl',
    'apiLevel': '2.10'
}


def run(ctx):

    class WellH(Well):
        def __init__(self, well, height=0, min_height=5, comp_coeff=1.15,
                     current_volume=0):
            super().__init__(well._impl)
            self.well = well
            self.height = height
            self.min_height = min_height
            self.comp_coeff = comp_coeff
            self.radius = self.diameter/2
            self.current_volume = current_volume

        def height_dec(self, vol):
            dh = (vol/(math.pi*(self.radius**2)))*self.comp_coeff
            if self.height - dh > self.min_height:
                self.height = self.height - dh
            else:
                self.height = self.min_height
```

If you're looking to install packages, try our packager. Learn more.

```
~/webinar$ opentrons_simulate main.py
```

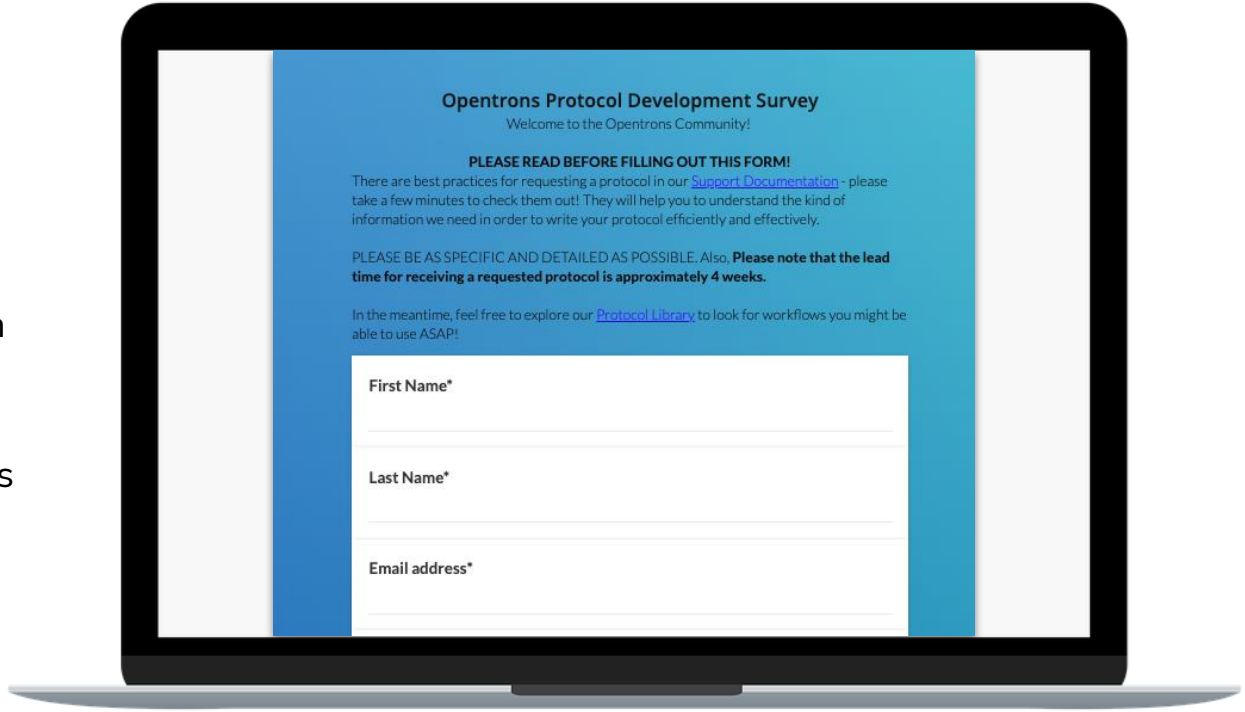
Request a Custom Protocol

[Custom protocol submission form.]

Lead time is in under 2 weeks from submission time.

Protocols are delivered to our users via the [Protocol Library]

We will help you work with your automation sales executive to purchase



Opentrons Protocol Development Survey

Welcome to the Opentrons Community!

PLEASE READ BEFORE FILLING OUT THIS FORM!

There are best practices for requesting a protocol in our Support Documentation - please take a few minutes to check them out! They will help you to understand the kind of information we need in order to write your protocol efficiently and effectively.

PLEASE BE AS SPECIFIC AND DETAILED AS POSSIBLE. Also, **Please note that the lead time for receiving a requested protocol is approximately 4 weeks.**

In the meantime, feel free to explore our Protocol Library to look for workflows you might be able to use ASAP!

First Name*

Last Name*

Email address*

Resources

Opentrons API reference

API Source Code (Github)

Applications Engineering team email

Simulating protocols article

Protocol Library

Example code

Questions?

Thanks for Coming!

ndiehl@opentrons.com

michael.asham@opentrons.com

Check out more Opentrons webinars at https://blog.opentrons.com/webinars/