fastKris User's guide

Fast automated protocol generation for **crys**tallisation screens

Sophie Colette, Lucas De Vrieze, Judith Raskin, Kristof Van Assche

December 17, 2022

Contents

1 Introduction

1.1 Crystallisation

X-ray crystallography is a technique to examine crystals of, for example, proteins for their molecular structure. The electrons present in a crystal cause the X-rays to diffract in a certain pattern. As such, a three-dimensional figure of the electron density is created, from which the position of the atoms in the crystal can be determined, next to other information. Even though crystallography was discovered more than 150 years ago, there is still no general theory to find the best crystallisation conditions for a particular molecule. Hence, a trial-and-error approach is usually needed to crystallise a protein. To do this, mixtures of crystallisation-inducing compounds are first tested over rather broad concentration ranges. When a crystal-like structure is detected in a certain mixture, the compounds and concentration ranges are refined around the conditions of that mixture. Often multiple runs are needed to find the optimal conditions. Continuously recalculating concentrations and pipette aspiration volumes, and manual pipetting make this approach time-consuming and labour-intensive.

The *fastKris* tool was created to tackle both time and labour issues. It automatically determines the concentrations for each compound in a crystallisation screening experiment and executes the experiment by steering an Opentrons robot for the pipetting work.

Often four different categories of compounds are mixed to create a protein crystal: a salt, a precipitant, a buffer, and a diluent such as Milli-Q water.

Different screen types are available in this protocol: one-dimensional, two-dimensional, and three-dimensional screens, as illustrated for a well plate in Figure 1. In a one-dimensional screen, only one out of the three compounds is varied according to a concentration range. In a two-dimensional screen, one compound is varied horizontally on the well plate in one concentration range and another compound is adjusted vertically in another range. In a one- and a two-dimensional screen, the concentration of the third compound remains constant. Finally, in a three-dimensional screen, the well plate is divided into four compartments and, hence, four separate two-dimensional screens. The difference between these is the concentration of the third component, which is varied according to a third concentration range. In all cases, the diluent completes the mixture up to a specified working volume.
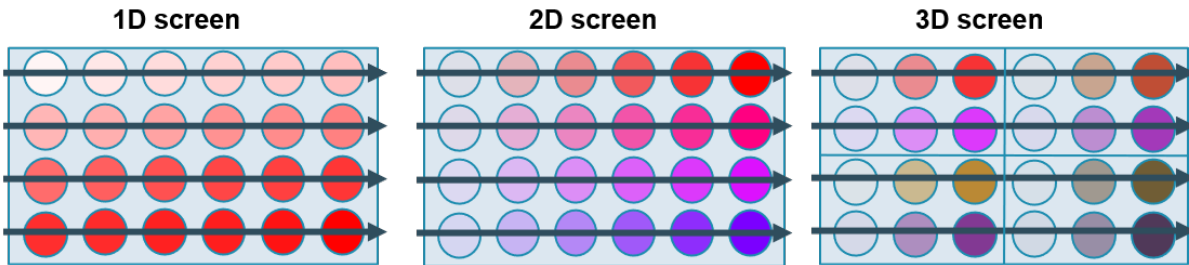

Figure 1: The three different screen types supported by this tool. The black arrows indicate the order of preparation on well plates, i.e. from top left to bottom right, row by row.

1.2 Opentrons

The Opentrons is a versatile pipetting robot designed for replicative work such as serial dilutions and PCR preparations. It can greatly reduce the time spent on executing protocols with much repetitive manual work. The Opentrons is open-source, which increases its flexibility and lowers its cost. There are four ways to use protocols. Users can 1) search in the protocol libraries for existing protocols, 2) use an online protocol designer, or 3) buy a custom protocol. Finally, users can create a protocol themselves with the Python Protocol API (OT-2 API V2). The OT-2 Python Protocol API from Opentrons is written in Python and is used to create automated lab protocols. In each protocol, it is important to give the Opentrons specific instructions and the location of each labware, pipette or module. Many functions have already been defined in the Opentrons API.
The Opentrons has 11 deck slots and a trash container, as depicted in Figure 2. Each deck slot can hold a module or labware such as a well plate or a pipette tip rack. The Opentrons can mount two pipette heads at the same time. It is compatible with both single-channel and 8-channel pipettes. Single-channel and 8-channel pipettes exist in 20 µl and 300 µl. On top of this, single-channel ones also exist in 1000 µl.
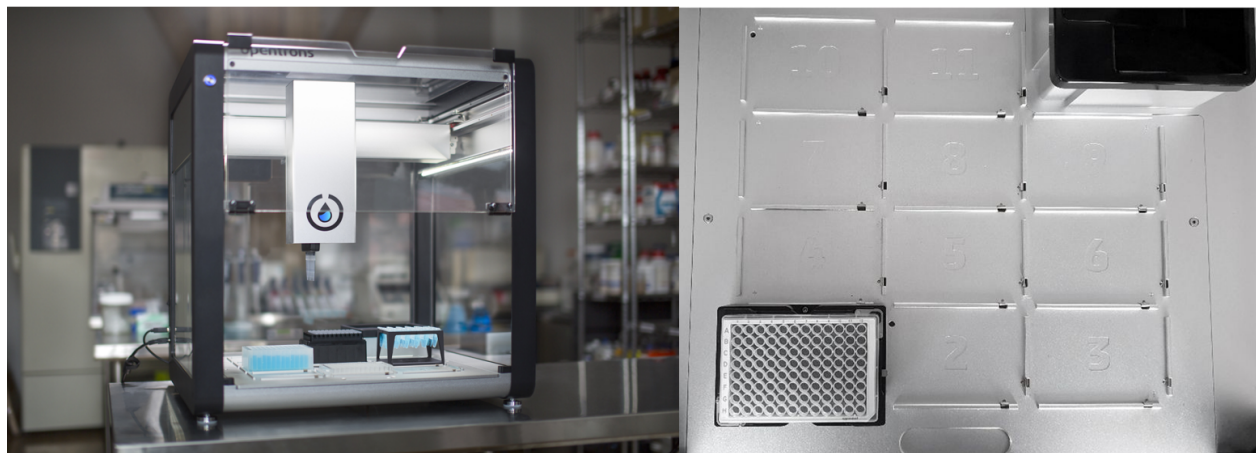

Figure 2: Opentrons with 11 deck slots and one trash container.
Figures are taken from fiercebiotech.com (left) and qinstruments.com (right).

The type of labware used by the Opentrons for crystallisation is not hard-coded and can be adjusted according to the protocol of the user. The fastKris protocol was tested with an "Opentrons 6 Tube Rack with NEST 50 mL Conical" tube rack for the stock solutions of the different compounds. 24 and 96 well plates are often used for crystallisation, so in our test protocol, the "Corning 24 Well Plate 3.4 mL Flat" was used. Two tip racks were also defined: the "Opentrons 96 Tip Rack 1000 µL" and "Opentrons 96 Tip Rack 300 µL". Plenty more options can be found at the Opentrons website (https://labware.opentrons.com/). Only the 300 µl and 1000 µl single-channel pipettes were mounted for this protocol. All this information can be written in a structured parameter text file, which is further explained in Table 4 in Subsection 4.2.2.

2 Installation

This package is entirely written in Python 3.8.10. You can check whether you have a sufficiently high version of Python installed on your system by typing `python --version` in a terminal. If not, please upgrade your Python version, for example, using Anaconda.

On Windows systems, you can get a terminal by, for example, pressing Windows+R and entering `cmd` in the field that pops up. You can also search for `cmd` or `command prompt` via the search bar of the start menu. On Unix systems, press Ctrl+Alt+T. On OS X systems, search for 'Terminal'.

For some **Linux distributions** that recently transitioned from Python 2 to Python 3, first check whether you do not have a system with two versions installed for back-compatibility by typing `python3 --version`. If you are in this case, just use `python3` and `pip3` instead of `python` or `pip` for any terminal command below.
Don't forget to do the same update in the shebang of `gui.py` and `ScriptBuilder.py` as well, if you want to keep running these by double-clicking! This is the first line in the script and should look like

```
#!/usr/bin/env python
```

Installing **fastKris** only encompasses downloading the contents of our GitHub repository (https://github.com/LucoDevro/fastKris) to a location which you can easily find back. Make sure the following dependencies are installed using the terminal commands below.

- NumPy ($\geq$ 1.17.4) - `pip install numpy`
- Opentrons API ($\geq$ 6.1.0) - `pip install opentrons`
- customTkinter ($\geq$ 5.0.1) - `pip install customtkinter`
- Pillow ($\geq$ 9.3.0) - `pip install Pillow`
- XlsxWriter ($\geq$ 3.0.3) - `pip install XlsxWriter`
- easygui ($\geq$ 0.98.3) - `pip install easygui`

Now, you are already fully set up to use **fastKris**!
Most scripts start by double-clicking them, so you can conveniently make links to them. Alternatively, everything should run from a terminal as well. To check if everything works fine, proceed to the tutorial below and try the commands using our example protocol.

If your **Linux system** fails to run a script despite a correct setup of Python and the dependencies, and a correct shebang, it is possible that your system cannot cope with DOS text files produced by Windows systems. In that case, convert the script to Unix encoding using the `dos2unix` tool in a terminal.

```
dos2unix <script>.py
```

3 Tutorial

3.1 Define the experiment using the GUI

To start the GUI, navigate to the fastKris folder and double-click on the `gui.py` file. On Windows systems, the first time you try to open the file you may be prompted to select a program. Alternatively, the default option may not display the GUI. In both cases, select python.exe (with console) or pythonw.exe (without console) to run the script.

First, the GUI asks you to select a working directory in which all temporary files and output files will be written. Ensure that the GUI has read/write permissions to the selected directory. You should now see the following window:
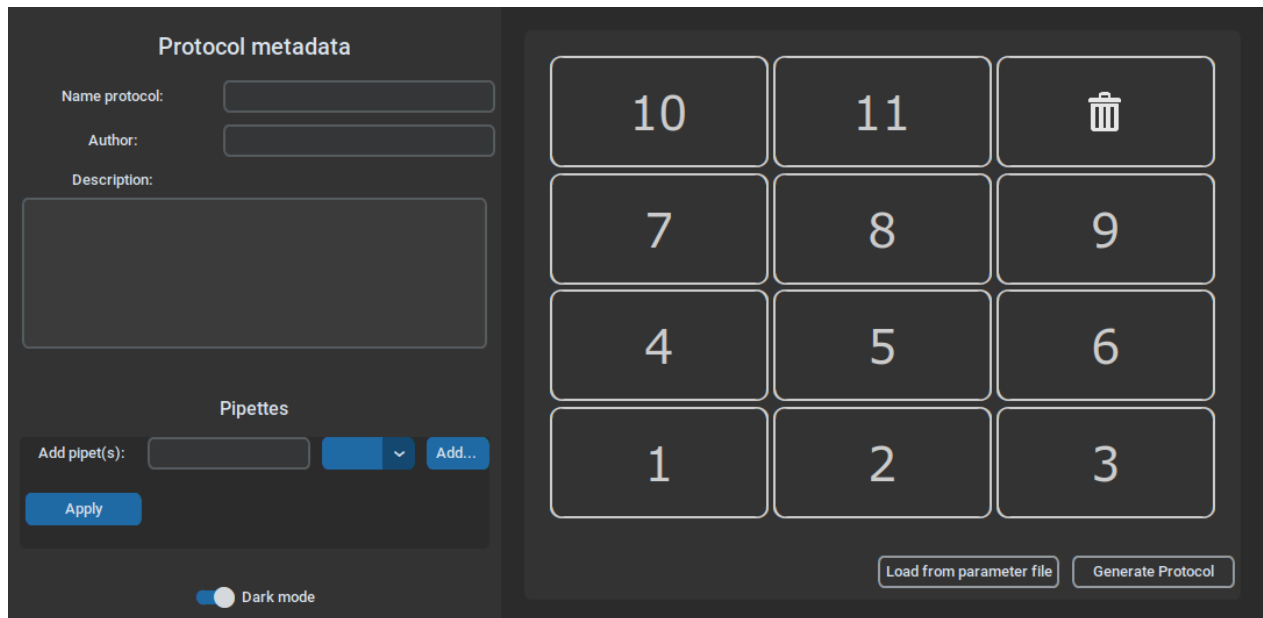

Figure 3: start screen of the GUI

3.1.1 Adding information to a slot

To add information to a slot, simply click on it. An input frame will appear.

Figure 4: input frame for slot 1

Next, select the type of container you want to add. This will create extra input fields based on your choice of 'Tube rack', 'Tip rack', or 'Well plate' (see table 1). Fill in all information you have at the moment and click on 'apply' to store the information. You can find the specified information back in the working directory's 'inputs' folder. The filename is Input_plateX, with X the number of the slot. Alternatively, 'cancel' will discard all information that you entered or changed and no changes will be made to previously created files in the 'inputs' folder. To delete the temporary file for this slot, click on 'reset'.

Key	Comments
(All types) name container	Label of the container as defined in the Opentrons labware library (see 3.3)
add to slot	type of container
(Tip racks) Assigned to pipet	Pipette associated with the tip rack. Either 'left' or 'right'.
(Well plates, upper frame) working volume (ul)	Final volume that should be present in each well
tube rack position	Number of the slot with a tube rack containing all the compounds *Note: maximum one slot*
Milli-Q position	position of the stock solution of the diluent in the tube rack
(Well plates, compounds) label	Name of the compound *Note: label in compound library without concentration*
conc	The concentration
range	Start and end point of the concentration range *Note: "." as decimal separator*
drop-down menu with options M and %	Unit of both concentration and ranges
position	position of the stock solution of the compound in the tube rack

Table 1: Input fields

For the pipettes, fill in the labels and positions of at least one and at most two pipettes in the left frame and click on 'Apply'. Optionally, you can fill in protocol-specific metadata: the name of the protocol, the author, and a short description. The values displayed are the ones that will be written to the protocol file. Blank entries will be replaced by the default values in Table 2.

descriptor	default value
Name protocol	Crystallization screen
Author	fastKris 1.0
Description	This protocol has been automatically generated using fastKris. Manual modifications will be overwritten.

Table 2: Default values for the metadata descriptors

3.1.2 Generate protocol

To generate the parameter file and protocol file, click on 'generate protocol' at the main window and enter a valid filename. If all required information is present, the files will appear in the working directory. If some essential information is missing, read the error message carefully and make the necessary changes. In case you specified one or more 1D or 2D screens, an additional Excel file will be created with the concentration gradients of these screens. For this, the GUI assumes that the name of the well plate follows the Opentrons API naming convention i.e., the first group of integers in the name represents the number of wells. An example is "corning_24_wellplate_3.4ml_flat" for a 24-well plate. Concentration gradients for all screen types and well plates can be retrieved by simulating the protocol in a terminal (see Section 3.3.1).

3.1.3 Load from parameter file

Information contained within a parameter file can directly be loaded in fastKris by clicking 'Load from parameter file' at the bottom of the main window. You will be prompted to select a parameter file from your file system. Note that this does not change the working directory.

3.1.4 Problems

In case unwanted behaviour occurs such as inputs that keep disappearing or repeated crashing of the app, the GUI is probably trying to read incorrectly provided or stored data. This may be caused by encoding issues (the tool itself was tested with utf-8 encoding). In case you loaded a parameter file, make sure it follows the template exactly. If you are unable to overwrite the input, the easiest solution is to delete the inputs folder in the working directory and start over. Beware that in that case all information that you already entered will be lost.

3.2 Build the protocol

The first step after setting up your experiment is generating the protocol by running `ScriptBuilder.py`. The easiest way is to run it from the GUI by pressing the 'Generate protocol' button after you have set up all the parameters. This method first creates a parameter file and then calls `ScriptBuilder.py` to generate the protocol. You can also browse to the script builder Python file and double-click it to run with default settings (see box below).

Another way is to run it from a terminal. Fire up one and browse to the location of the script builder using `cd '<location>'`, with `<location>` being the path to the folder containing the script builder.

Running the script builder is done by typing

```
python ScriptBuilder.py <params.txt> <protocol.py>
```

in which `<params.txt>` is the path to your parameter text file and `<protocol.py>` the location to get the newly generated protocol. By running

```
python ScriptBuilder.py
```

the script builder assumes that you want to generate a protocol named `protocol.py` from a parameter text file `params.txt` in the same folder in which the builder itself is located.


The compound library `compLibrary.txt` should be located in the same folder as the script builder. Generating it does not require any tools provided by us. It is sufficient to save the library somewhere as an Excel sheet and to export it to that folder as a tab-separated text file (the `*.txt` option) from the 'Save as' menu in Excel.

> The compound library should be saved in the **same** folder as the script builder and the GUI. Use a `Label - Type - Stock concentration` column structure for the library Excel sheet!

3.3 Execute the protocol

After generating the protocol script, the next step is to upload it to the Opentrons. Therefore, we highly recommend the Opentrons app designed by the Opentrons development team. It provides the possibility to calibrate the pipetting instruments and carries out some checks of the protocol script to which we have hooked up some additional ones that are specific for screening experiments.

When running the Opentrons app for the first time, you see the following window. Here, you can upload new protocols and consult previously known ones.
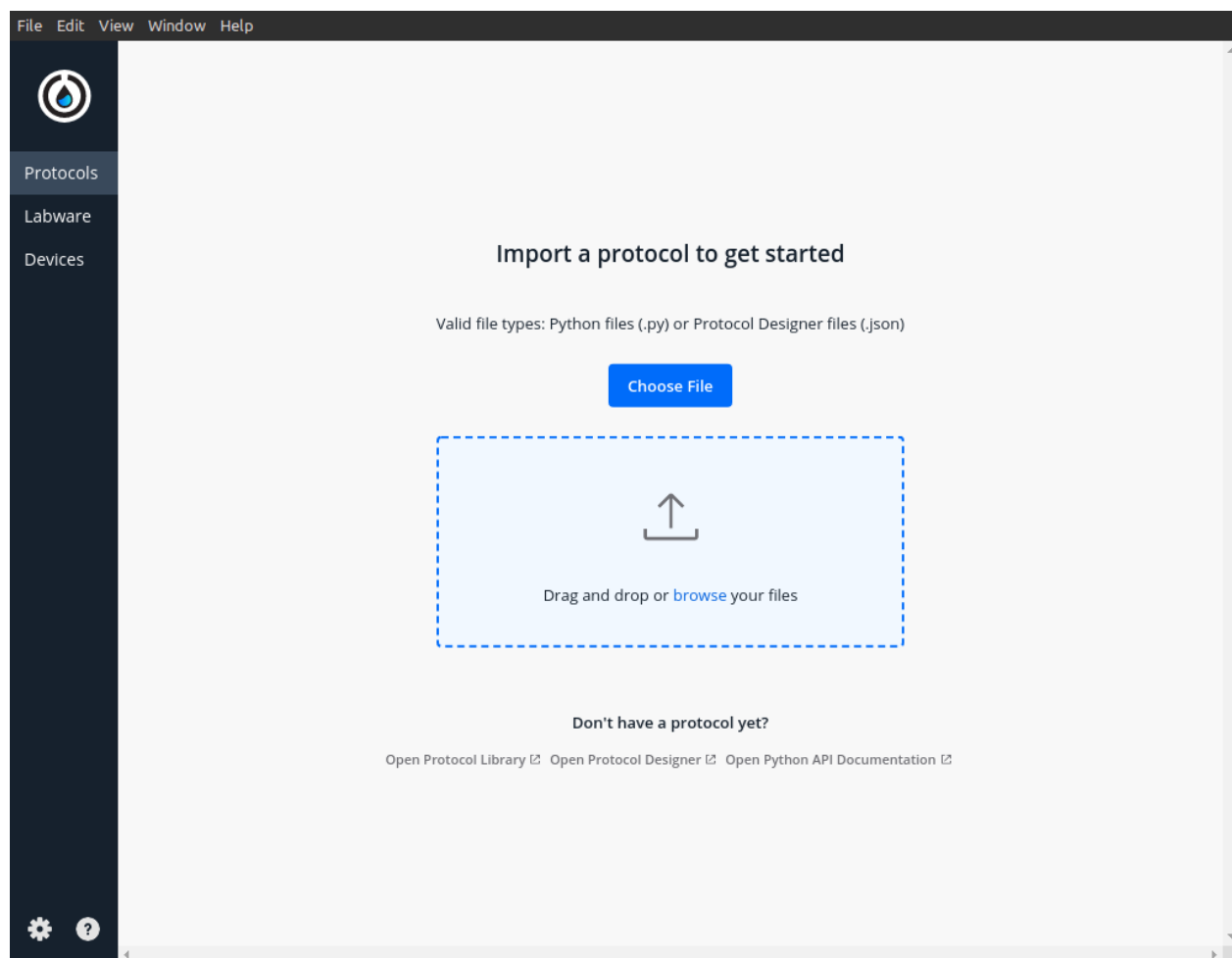
At the Labware tab at the left, you find all labware defined in the labware library. Every entry in this list shows the API name of that particular labware. **This is the label you should use to include this labware in your experimental setup!** At the Devices tab, you find all Opentrons devices that are known by your workstation.
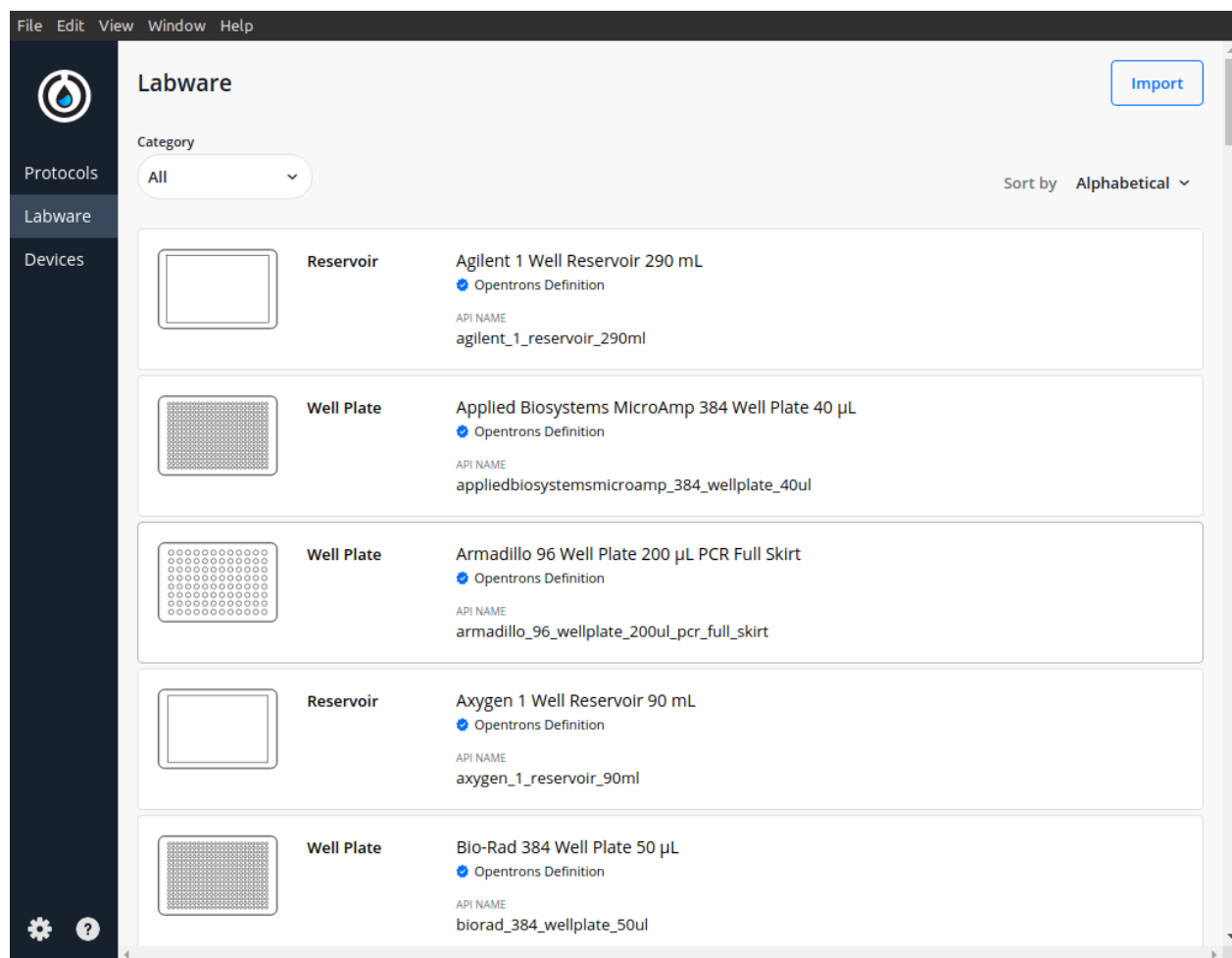
Return to the Protocols tab and click 'Choose file' to upload the generated protocol. When you do so, the app copies the protocol script to its own work folder, which you can access via the 'Show in folder' option at the top right menu after clicking on your protocol. Updating this protocol can be done via deleting and reimporting it in the app, but it can also be done by overwriting the protocol file in its work folder and clicking the 'reanalyze' option in that same menu top right.

While loading the protocol, under the hood, the app checks the protocol by executing it in a simulated Opentrons environment and throws an error message if it encounters a problem. We have defined additional exceptions next to the native ones that will trigger such an error message. For example, in the window below, we uploaded our example protocol, with the slight modification of changing the position of a stock tube from a tube rack to a well plate, which obviously is an inconsistent setup.

After correcting, rebuilding and copying the protocol to the Opentrons work folder, you can reanalyse the script from the menu at the top right to redo the checks.
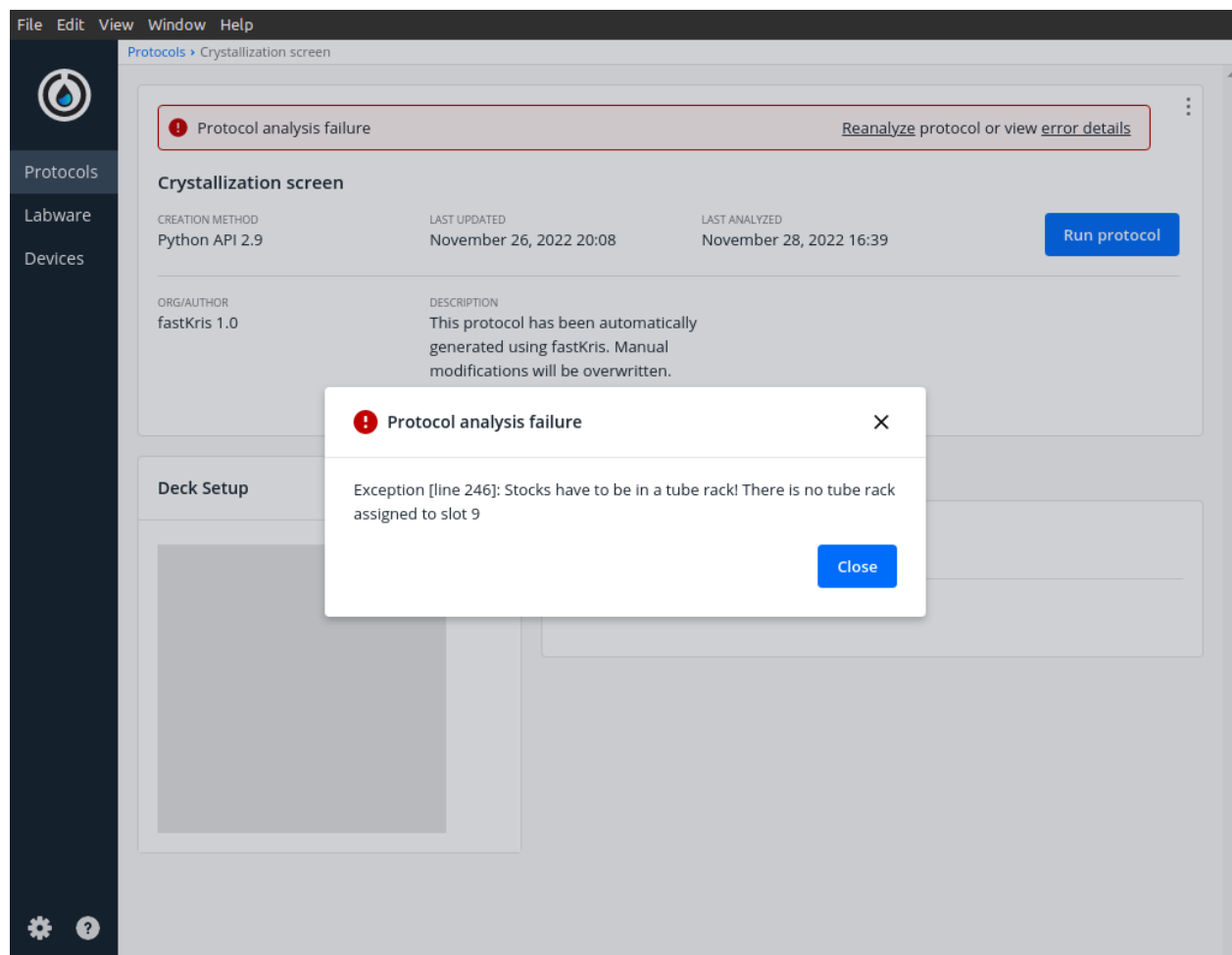
When the checks succeed, click the 'Run protocol' button and select your Opentrons device. The app will now upload the script to the Opentrons. Before executing the protocol, scroll down and open the calibration menu, which will trigger a pop-up prompt to apply any known instrument calibration settings. It is recommended to do a calibration check every time the Opentrons has been shut down. You can launch this from this particular menu.

3.3.1 Simulating the protocol & the verbose shell

It is possible to access that simulation environment from a terminal as well. Moreover, we have stuffed the protocol script with informative logs, which consequently will be displayed at the terminal output. As such, you find matrices depicting for every compound the concentration that will be prepared in each well. The total volume requirements are reported as well, allowing you to check whether there is enough stock solution available in your lab inventory. Underneath, a summary line is printed for every transferred volume. Finally, the output of the Opentrons simulation kernel itself is returned, which reflects every step the Opentrons would undertake while executing this protocol.

To get into simulation mode, fire up a terminal, browse to the location of your protocol using `cd` and, on Unix and OS X systems, type

```
opentrons_simulate '<protocol.py>'
```

or on Windows systems

```
opentrons_simulate.exe '<protocol.py>'
```

You can also avoid `cd`'ing by typing immediately after firing up the terminal

```
python -m opentrons.simulate '<path-to-your-protocol.py>'
```

Check out all the other options to simulate a protocol Opentrons provides, at this link.
The window below illustrates the terminal simulation mode for our example protocol.

```
File  Edit  View  Search  Terminal  Help
/home/lucas/.opentrons/robot_settings.json not found. Loading defaults
/home/lucas/.opentrons/deck_calibration.json not found. Loading defaults
Compound library loaded...
Screens parsed...

Executing Screen 1
Screen type: twoD

EDTA (10%) (Precipitant):
[[1.   1.2 1.4 1.6 1.8 2. ]
 [1.   1.2 1.4 1.6 1.8 2. ]
 [1.   1.2 1.4 1.6 1.8 2. ]
 [1.   1.2 1.4 1.6 1.8 2. ]]


NaCl (1M) (Salt):
[[0.1        0.1        0.1        0.1        0.1        0.1       ]
 [0.13333333 0.13333333 0.13333333 0.13333333 0.13333333 0.13333333]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]
 [0.2        0.2        0.2        0.2        0.2        0.2       ]]


Acetate buffer (1M) (Buffer):
[[0.1 0.1 0.1 0.1 0.1 0.1]
 [0.1 0.1 0.1 0.1 0.1 0.1]
 [0.1 0.1 0.1 0.1 0.1 0.1]
 [0.1 0.1 0.1 0.1 0.1 0.1]]


This screen needs:
EDTA (10%) (Precipitant):      3600.0 uL
NaCl (1M) (Salt):       3600.0 uL
Acetate buffer (1M) (Buffer):  2400.0 uL
MQ (Diluent):   14400.0 uL


Adding compound 1
Transferred 100.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A1 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 120.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A2 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 140.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A3 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 160.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A4 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 180.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A5 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 200.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 A6 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 100.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 B1 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 120.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 B2 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 140.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 B3 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 160.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 B4 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 180.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
 B5 of Corning 24 Well Plate 3.4 mL Flat on 1 using P300 Single-Channel GEN2 on right mount
Transferred 200.0 uL of EDTA (10%) (Precipitant) from B1 of Opentrons 6 Tube Rack with Falcon 50 mL Conical on 7 into
```

4 Program pipeline

This section provides an overview of the program workflow. It explains how it proceeds in more detail, which assumptions are made and which limitations there may be.

4.1 General overview

Let's assume you have a correctly made parameter text file, for example, created by our bundled GUI (by default, `params.txt`), and that you have a text file containing a correctly formatted compound library as well (`compLibrary.txt`). This package provides three Python scripts: `gui.py`, `ScriptBuilder.py` and `template.py`. As the filename already indicates, the latter is not a functional Python script. On the contrary, it is the starting point from which new functional scripts will be built, so you should **never** modify this file, unless you are sure about what you are doing!

Every screening experiment is characterised by different parameters. Therefore, `ScriptBuilder.py` first reads the contents of the two parameter files defining your experimental settings (by default, `params.txt` and `compLibrary.txt`). From this, it creates an experiment-specific preamble of Python code and joins it with `template.py` to create a functional Opentrons protocol script (by default, `protocol.py`), which is ready to send to the Opentrons using, for example, the Opentrons app.

A functional protocol script consists of three parts: setting up the experimental environment, calculating the concentrations and the associated pipette volumes, and carrying out the actual operations using Opentrons API commands. Figure 5 below visualises the workflow described here.
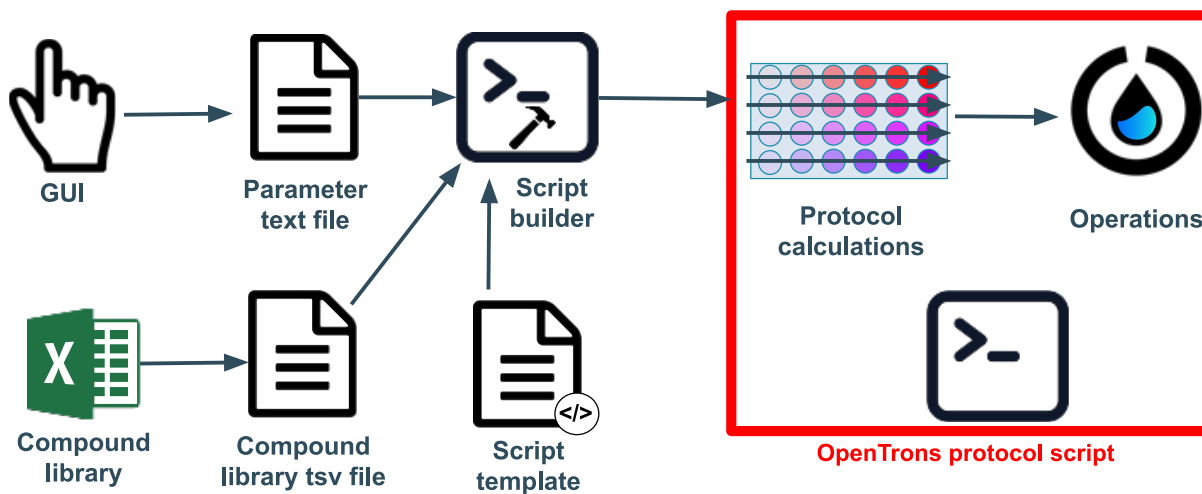
Figure 5: Schematic representation of the fastKris workflow. With our bundled GUI, you can produce the experiment parameter file (by default, `params.txt`). Using this and the compound library (`compLibrary.txt`), `ScriptBuilder.py` produces a functional Opentrons protocol script (by default, `protocol.py`) using the template. This protocol calculates the screening concentrations and pipette volumes and generates a sequence of operations for the Opentrons.

4.2 Defining the experimental setup

4.2.1 Generating the parameter text files with the GUI

All pipette and slot information defined by the user in the GUI goes through multiple stages. First, after clicking 'apply', the information is stored in temporary files in a new folder 'inputs' in the working directory. During the generation of the output, an intermediary parameter file is created, containing all information from the 'inputs' folder except the types of the compounds. This information is sourced from the `compLibrary.txt` file and, hence, is not required to be written to the parameter file. Finally, the information in the parameter file, the current entries in the metadata fields and `template.py` are combined by a dedicated `BuildWithMetadata` function in the script builder to generate the protocol file.

Temporary text files for the slots are named `Input_plateX.txt` with X the number of the slot. These files consist of two lines, of which the first one identifies the type of labware. The second line contains a corresponding Python dictionary definition. The keys of this dictionary are for a Well plate: label, index, dimension, names_conc, positions, ranges, WorkingVolume, Tuberack and types_compounds. For a tube rack, these are: label and index, while for a tip rack: label, index and AssignedPipetOption. More detailed information can be found in Table 3.

Key	Comments
(All types)	
label	Label of the container as defined in the Opentrons labware library (see 3.3)
index	Slot number
(Tip racks)	
AssignedPipetOption	Pipette associated with the tip rack. Either 'left' or 'right'.
(Well plates)	
dimension	Screen dimension
names_conc	String of compound labels in the format 'name (concentration unit)' and "MQ", comma-separated, e.g. 'NaCl (1M),EDTA (10%),Acetate buffer (1M),MQ'
positions	String of positions of the compounds in the format 'Tuberack/position label', comma-separated, e.g. '7/A1,7/B1,7/A2,7/B2'
ranges	String of concentration ranges of the compounds in the format 'min-max', comma-separated, e.g. '0.1-0.5,1-1,0.1-0.1' for a 1D screen
WorkingVolume	Final volume that should be present in each well
Tuberack	Number of the slot with a tube rack containing all the compounds. *Note: maximum one slot*
types_compounds	String with capitalised types of the compounds, comma-separated, e.g. 'Salt,Precipitant,Buffer'

Table 3: Structure of the temporary input text files

The information about the pipettes is stored in a temporary `pipets.txt` file. Each group of two lines contains the label (line 1) and the position (line 2; left or right) of a pipette. There should be at least one and at most two pipettes defined, thus `pipets.txt` is 2 or 4 lines long.
Clicking 'Generate protocol' transfers all information contained within these temporary input text files to the eventual parameter text file.

4.2.2   Parsing the text files

The procedure of parsing a text file is heavily dependent on its structure. We have designed a structure that is relatively easy to parse using Python code. As it concerns a text file, the user can still inspect it and do some quick modifications in a simple text editor program, if familiar with the file structure. To facilitate this, we have included a text file designating the different parts of the parameter file schematically (`params_Structure.txt`), and an example file (`params.txt`). An example compound library file has been added too (`compLibrary.txt`) as well as an example protocol script (`protocol.py`).

In the parameter text file, the parameters are grouped by type in the following order: tipracks - tuberacks - pipettes - well plates - screens. These groups are separated by blank lines (thus two newline characters `\n`), while the subsequent fields of one entry are defined line by line and, hence, are separated by one newline character. The compound library file is a simple tab-separated text file, in which each entry consists of three columns. Tables 4 and 5 below give an overview of which parameters are contained within these structured text files.

4.2.3   Auxiliary Python classes

To distinguish between the different types of compounds and screens during the experiment definition and the protocol calculations later on, we have chosen to set up two families of classes. The use of classes also allows the more programming-proficient user to define and use their own custom compound types and screen types. Below, UML class diagrams are given in Figures 6 and 7. If one would like to add more compound and/or screen types, at least the mentioned attributes and methods should be defined as the processing pipelines depend on them.

The screen types that are predefined in this package are the 1D screen, the 2D screen and a 3D screen with a fixed number of compartments, as described in Section 1.1. The plates are prepared horizontally, row by row. The 1D screen varies only the concentration of the first component in the `compounds` attribute of the OneD class according to the first interval in the `range` attribute. The TwoD class does so for the first two components. Our ThreeD class splits the well plate into four compartments or 'subplates'. Mind here that the well plate has to have an even number of wells in both dimensions! In each compartment, it carries out a TwoD screen with the concentration of the third compartment temporarily fixed at a certain value within the third concentration range.

The compound types that are included in this package as a subclass, are salts, precipitants, buffers and diluents. This distinction is not very strict in the sense that the first three compound types are free to be used as the first, second or third component. It only serves the user to recognise compounds as these are commonly categorised as one of these types. The behaviour of these compound classes is identical as the dilution calculations are equivalent. Hence, a compound can be defined in any class with the sole condition that it is differently labelled for each definition.
The Diluent class deviates from this as it is not being diluted itself, but serves to dilute the prepared compound mixture to the well working volume. Obviously, there can only be one diluent compound.

Parameter file	Comments
Tiprack label	Name of the tip rack in the Opentrons Labware Library
Position	Assigned slot number at the Opentrons' deck
Associated instrument	Position of the pipette that uses this tiprack (left or right)
(Blank line)	
Tuberack label	Name of the tube rack in the Opentrons Labware Library
Position	Assigned slot number at the Opentrons' deck
(Blank line)	
Pipette	Name of the instrument in the Opentrons Instrument Library
Left or right	The head to which the pipette is mounted.
(Blank line)	
Plate	Name of the well plate in the Opentrons Labware Library
Position	Assigned slot number at the Opentrons' deck
(Blank line)	
Number of dimensions	One of the supported screen types (1D, 2D or 3D screen)
Compounds	Names of the screen compounds in the compound library, comma-separated
Reservoir tubes	Position in a tuberack of the stock solution for the compound at the same index in the row above (in slash-and-comma-separated format '...,slotnumber/location of tube in the tuberack on that slot,...') (e.g. ...,7/A1,...)
Ranges	Concentration ranges of the compounds of this screen in the appropriate units and in the same order as in the Compounds line (format: ...,min - max,...) (e.g. ...,0.1-0.5,...). If the concentration should be kept constant, min and max are equal.
Plate number	Slot number of the well plate on which this screen is executed
Working volume	Total volume to be prepared in each well in μL
(Blank line)	

Table 4: Structure of the parameter text file

Compound library file	Label	Type	Stock concentration
Comments	Label of the compound	Compound type (salt, precipitant, buffer, diluent)	Concentration of the stock solution[a]

[a] Make sure you know in which unit this concentration is expressed! The ranges of the parameter file should be in the same unit whenever you refer to this compound in a screen definition! You can remind yourself by, for example, including the unit in the compound label.

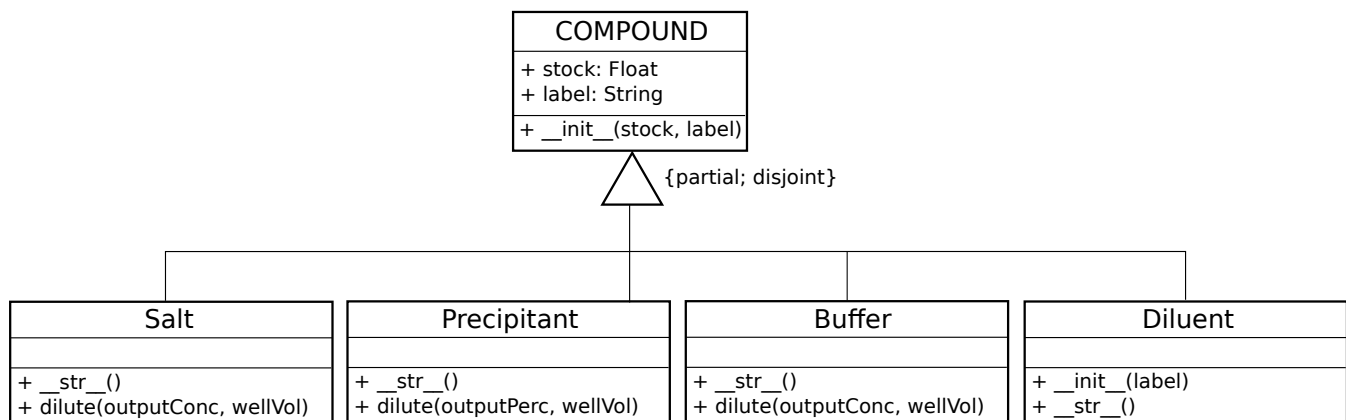**Table 5:** Structure of the compound library text file

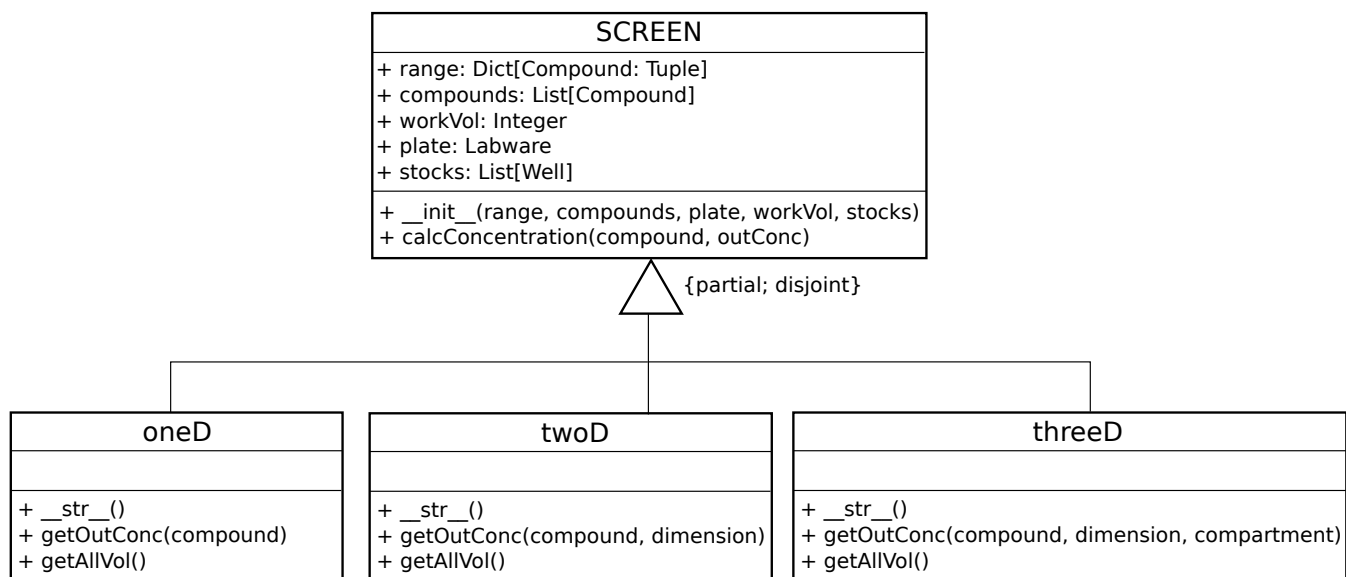**Figure 6:** UML class diagram of the Compound class family

Figure 7: UML class diagram of the Screen class family

4.3 Building the protocol script

In an ideal setting, doing a separate script building step would not be required as parsing parameter files can be easily integrated with downstream pipelines into one script. The communication with the Opentrons, however, has an annoying limitation: there is no support for file I/O to an external device while running a script[1]. As a result, it is not possible to read and parse parameter text files during a run: it has to be done by another device than the Opentrons. Hence, building the script is done by the user's workstation before interacting with the Opentrons.

Every experimental setup is different, as defined by the parameter file, while the way it is processed is similar. This is the underlying idea behind building the protocol script from the template. The fixed part of a protocol script is the setup of the experimental framework based on the parsed parameters, the protocol calculations and the Opentrons operations. The code that does exactly this, is collected in `template.py`. `ScriptBuilder.py` reads the parameter file, does some initial processing and merges the results with the template into a newly built script.
More specifically, it writes a preamble of string variable declarations in Python code at the top of a copy of the template script. These string variables facilitate the Opentrons to pick up the workflow where the script builder stopped without the need for file I/O. When opening a newly generated script in an editor, at the top, you would find these string variable declarations and probably recognise the parameter values you would have submitted through the parameter file.

4.4 Protocol calculations

Calculating the pipette volumes is executed in three steps. First, each compound is matched with a certain dimension of the well plate. The first compound always matches with the horizontal direction and the second with the vertical one, while the third one is matched row by row to all wells. As such, the requested concentration range is converted to a list of concentrations depending on the number of wells there are available in that dimension of the well plate. For example, a 2D screen on a 24 well (6x4) plate would result in a list of six concentrations for the first compound and one of four for the second, while the third compound would match with a list of 24 constant concentrations. This step is carried out by the respective `getOutConc()` method of each Screen subclass.

Next, these lists of concentrations are converted to equally sized lists of pipette volumes using the associated dilution calculation method `dilute()` of the compound. The values are rounded to 0.001 μL. These volumes V_{stock} are obtained through the simple formula below, with C_{stock} the stock concentration of that compound, C_{req} a requested concentration from the list and V_{well} the well working volume. This formula works equally well for C's expressed in M as well as in percentages, but this also shows why it is important to express both of them in the same unit!

$$C_{stock} \cdot V_{stock} = C_{req} \cdot V_{well} \iff V_{stock} = \frac{C_{req} \cdot V_{well}}{C_{stock}}$$

Thirdly, the `getAllVol()` method of the Screen subclasses rearranges the resulting lists of pipette volumes in such a way that they cover the entire well plate horizontally, row by row, as depicted previously in Figure 1. This requires some array tiling, reshaping, transposing and flattening operations provided by the `NumPy` package. Then, the diluent volumes are calculated from the difference between the working volume and the sum of stock solution volumes. Negative diluent volumes throw a warning. Eventually, all pipette volumes lists are collected in a dictionary by compound.

---

[1]   Yet, there is an option to bundle the script with csv files and make the Opentrons parse these, but this is not the easiest solution to code in our opinion. Generating a script separately also allows to share it and to interact with an Opentrons using a workstation without a fastKris installation.

4.5 Opentrons operations

Now, it is only a matter of looping over all compounds in this dictionary and all wells to transfer all volumes defined in the lists one by one. Nevertheless, we have included some useful features that ease automation.

The Opentrons allows mounting up to two pipettes. Depending on the volume to be pipetted for a certain well, it will take the most optimal pipette for accuracy reasons. More specifically, it will pick the pipette with an appropriate volumetric range and, in case both are suitable, the one with the closest maximum volumetric capacity. If there is no suitable one, it throws an exception.

The next feature is tip saving. To avoid contamination over wells, every tip that had contact with a liquid mixture should be dropped. You can quickly calculate how many tips it would cost to prepare mixtures of four compounds on a 96 well plate by adding each volume one by one. Test runs also revealed that the process of changing a tip takes much time compared to transferring a volume of liquid. Hence, avoiding contact with the liquid mixture saves many tips and much time. Keeping the tip outlet right at the top of the well while dispensing achieves this while also limiting spilling risks. Picking a new tip is thus only required when the addition of one compound has finished and the iteration of adding a next one is about to begin. Yet, in case of the last component, the contents of the well are mixed and changing tips for each well is still required. As such, a 24 well plate, for example, requires 27 tips and about half an hour.

Sometimes, a droplet sticks at the top of the tip after transferring a volume and might be transferred to the next well or fall off the tip on the deck while moving the instrument. Therefore, the instrument blows out and shakes a little bit to make it fall into the current well before aspirating the next volume from the stock tube. This solution also avoids contact with the well walls at which other droplets might be sticking already.

Furthermore, a basic liquid level tracker avoids submerging instruments. By default, the liquid is aspirated from the bottom of a recipient, but typical stock tubes are higher than tips. The tracker assumes that all tubes are full at the start of the plate preparation. It gradually lowers the aspiration height of the tip by keeping track of the total volume that has been transferred already. To allow some deviation from this assumption, the aspiration height is initially located 4 cm below the top of the tube. This is approximately the length of the smallest tips in the Opentrons product range (10 μL tips: 3.92 cm). Yet, typically, there is a clearance of about 1 cm between the top of a full tube and the liquid level as well. Especially for a diluent, it makes sense to provide a full tube for each experiment as this is the compound type that typically consumes the most liquid.

To finish, the Opentrons will turn on its rail lights during operation and switch them off when ready. As such, the user can quickly verify progress.