

SOFTWARE USER GUIDE

Contents

[Getting Started 4](#_Toc529818892)

[**Requirements** 4](#_Toc529818893)

[**Installation** 4](#_Toc529818894)

[Windows/Mac 4](#_Toc529818895)

[Raspberry-Pi 5](#_Toc529818896)

[**Repository Structure Guide** 6](#_Toc529818897)

[Software 6](#_Toc529818898)

[**Initial Function Testing** 7](#_Toc529818899)

[Testing Light Sources 7](#_Toc529818900)

[Testing Linear Actuators 7](#_Toc529818901)

[Software Structure 8](#_Toc529818902)

[Structure Overview 8](#_Toc529818903)

[Structure Guide 9](#_Toc529818904)

[System.py 9](#_Toc529818905)

[GUI.py 10](#_Toc529818906)

[PlateConfiguration.py 11](#_Toc529818907)

[Protocol.py 11](#_Toc529818908)

[Absorbance.py 11](#_Toc529818909)

[Fluorescence.py 12](#_Toc529818910)

[Auxiliary.py 12](#_Toc529818911)

[Kinetic.py 13](#_Toc529818912)

[Machine.py 13](#_Toc529818913)

[Arduino.py 14](#_Toc529818914)

[Spec.py 14](#_Toc529818915)

[System Level Control 15](#_Toc529818916)

[Importing the module and Initializing a System Instance 15](#_Toc529818917)

[Adding a Plate Configuration 15](#_Toc529818918)

[Selecting Wells 15](#_Toc529818919)

[Adding a protocol to a plate configuration 15](#_Toc529818920)

[Running the Input Protocol Program 16](#_Toc529818921)

[GUI Level Control 17](#_Toc529818922)

[Running the GUI 17](#_Toc529818923)

[The Intro Screen 17](#_Toc529818924)

[The Settings Screen 17](#_Toc529818925)

[The Plate/Well Selection 18](#_Toc529818926)

[The Protocol Selection 19](#_Toc529818927)

[Kinetic Toggling 20](#_Toc529818928)

[Absorbance Settings 21](#_Toc529818929)

[Fluorescence Settings 21](#_Toc529818930)

[Auxiliary Settings 22](#_Toc529818931)

[Review Protocol Sequence 23](#_Toc529818932)

# Getting Started

## **Requirements**

OSP software is implemented using an interface between Python and Arduino. The following are needed to operate the provided software:

* Python 2.x
* Python Modules
  + PyQt 4
  + Seabreeze
  + Pickle
  + Xlswriter
  + Matplotlib
  + Numpy
  + Time
  + Serial
  + Warnings
* Arduino IDE
* Arduino Add-ons
  + Servo

## **Installation**

### Windows/Mac

#### Installing the Requirements:

Installation of OSP software prerequisites on any Windows/Mac device is accomplished utilizing Anaconda (downloadable from [here](https://www.anaconda.com/download/)), which should come prepackaged with all requirements except PyQt4, Seabreeze and Pyserial. When selecting the Anaconda download be sure to choose the version for Python 2.X as well as for your proper operating system (Windows/Mac). PyQt4 and Seabreeze can be installed by running the following commands in Anaconda Prompt (a program which is installed with Anaconda):

**PyQt4** - *conda install pyqt=4*

**Seabreeze** - *conda install -c poehlmann python-seabreeze*

**Pyserial –** *conda install -c anaconda pyserial*

#### Cloning the OSP Repository:

To clone the OSP repository running the following line of code in the Terminal Window on your Raspberry Pi.

*git clone* [*https://github.com/KokoIsLoko/OSP.git*](https://github.com/KokoIsLoko/OSP.git)

#### Running the GUI:

Once the OSP repository has been cloned and all the dependencies installed, the main User Interface can be started by opening Anaconda Prompt and running the following lines of code:

*cd "OSP\Software\Python Files"*

*python GUI.py*

### Raspberry-Pi

#### Installing from OSP.img:

The easiest installation method is to download the OSP.img file (available here) and install it onto an SD card (recommended 16GB) utilizing the *Etcher.io* software (available here). Make sure to download the *Etcher.*io software onto a device which has an SD card slot in it. Alternatively, to add the OSP software to your current Raspberry-Pi system, follow the instructions in the next two sections.

#### Cloning the OSP & Seabreeze Repositories Manually:

To clone the OSP repository running the following line of code in the Terminal Window on your Raspberry Pi.

*git clone* [*https://github.com/KokoIsLoko/OSP.git*](https://github.com/KokoIsLoko/OSP.git)

To clone the Seabreeze (required for STS-VIS) running the following line of code in the Terminal Window on your Raspberry Pi.

*git clone* [*https://github.com/ap--/python-seabreeze.git*](https://github.com/ap--/python-seabreeze.git)

#### Installing Requirements Manually:

Once both the OSP and Seabreeze repositories are cloned on your Raspberry Pi you will need to install all the dependencies by running the following line of code in the Terminal Window.

*sudo bash OSP/install\_reqs.sh*

This installation process can take up to 20 minutes so do not be worried if it is taking long.

#### Running the GUI:

Once all the dependencies installed, the main User Interface can be started by opening the Terminal Window and running the following lines of code:

*cd "OSP\Software\Python Files"*

*sudo python GUI.py*

## **Repository Structure Guide**

Below you will find a detailed explanation of how the repository is structured, including folder and file descriptions.

### Software

#### Arduino Files

The main Arduino script is found in this folder.

#### Graphical Designer Files

All raw GUI Designer files are found in this folder. GUI graphics are derived from these files.

#### Python Files

##### Data

This directory is where all saved data from experiments are saved in the form of .xlsx files.

##### Images

This directory contains all the image files referenced in the GUI.

##### Protocols

This directory is where all protocol sequences are saved from the GUI

##### Screens

This directory contains the Python equivalent of the graphical files found in *Software/Graphical Files*

##### System Settings

This directory contains the setting files such as calibrated well positions, excitation LED wavelengths, and the number of scans to average during a measurement.

##### All Other Files

These rest of the files are Python scripts containing the class objects from which the GUI is accessed. An explanation of each of these classes is provided in the next section.

## **Initial Function Testing**

Once the OSP device is fully built and connected to a computer or Raspberry-Pi, it is useful to test the functionality of the linear actuators and light sources. To do so, open up the Arduino IDE and navigate to the Serial Monitor. If you do not have the Arduino IDE (it comes pre-installed on the Raspberry Pi) you can download it from here ([LINK](https://www.arduino.cc/en/Main/Software)). Follow the instructions below and enter the commands to test the functionality of individual components:

### Testing Light Sources

To test LED 1 type: **L1;**This should result in the LED 1 turning on. To turn it off, type the same command in again.

To test LED 2 type: **L2;**This should result in the LED 2 turning on. To turn it off, type the same command in again.

To test LED 3 type: **L3;**This should result in the LED 1 turning on. To turn it off, type the same command in again.

To test LED 4 type: **L1;**This should result in the LED 4 turning on. To turn it off, type the same command in again.

To test LED 5 type: **L5;**This should result in LED 5 & LED 6 turning on. To turn them off, type the same command in again.

### Testing Linear Actuators

To extend Linear Actuators type: **M1500.001500.00;**This should result in both linear actuators extending to about half their limit.   
  
To contract the linear actuators type: **M1000.001000.00;**This should result in both linear actuators fully contracting.

# Software Structure

The OSP software is object oriented and composed of a total of eleven class objects. The below sections will explain the structure of the software and how information is passed between the objects.

## Structure Overview

The diagram below depicts the general flow of information between the 6 main software objects. The entire structure can be divided into three separate categories based on functionality.



*System.py* and *GUI.py* are the primary classes that allow the user to interface with the plate reader. They track and store user input. *System.py* is the object that contains all the vital functionality of translating the user input into commands to change device settings. This object is responsible for loading-in user settings (such as calibrated well coordinates and excitation led wavelengths) and for preparing the system for execution of input protocols. The user can call this object to set-up and execute protocol sequences from a Python script (more information about this and examples in *System Level Control*). Alternatively, the user can control the device via the provided GUI (GUI.py) object. The GUI.py object (the visual user interface) has access to all the functions and variables of the System.py object.

User input (into either the *GUI.py* or *System.py* constructs) results in the creation of a *PlateConfiguration.py* object containing a list of *Protocol.py* objects. This combination of classes organizes the desired sequence of protocols to be executed. The *PlateConfiguration.py* object contains a list of the selected wells and a separate list of the protocols to be run. As mentioned before, the list of protocols is a list of protocol objects (e.g. *Absorbance.py* is an object for an absorbance protocol and it inherits its attributes from the *Protocol.py* object). Each protocol object contains all the necessary parameters (spectrophotometer settings, excitation LED, acquisition type, etc.) for it to be executed.

The user is not limited to one *PlateConfiguration.py* structure. Performing separate protocols on different sets of wells would simply require multiple *PlateConfiguration.py* objects to be created. For example, to measure fluorescence from the set of wells [A1, A2, A3] using a 470nm LED and then measure fluorescence from the set of wells [D1, D2, D3] using a 610nm LED, the system would create two *PlateConfiguration.py* objects (one for each set of selected wells); the first *PlateConfiguration.py* object would represent the set of wells [A1, A2, A3] and contains a single *Fluorescence.py* object (for the scan with 470nm excitation), and the second *PlateConfiguration.py* object would represent the set of wells [D1, D2, D3] and also contains a single *Fluorescence.py* object (for the scan with 610nm excitation).

Once the *PlateConfiguration.py* object(s) is/are created, a list of these objects is sent to the *Machine.py* construct. This object is the control center of the entire software. *Machine.py* is responsible for parsing through the list of *PlateConfiguration.py* objects and their respective *Protocol.py* objects. As it parses these protocols, it sends the respective commands to the *Arduino.py* and *Spec.py* structures to perform the necessary operations to successfully execute the inputs. In other words, *Machine.py* is the object responsible for sending commands to the Arduino to tell it to move the linear actuators or to turn on/off the LEDs or external devices (except for the detector). In addition, it extracts spectra from the spectrophotometer and outputs the extracted data to a properly formatted Excel sheet. As a result, there is constant bi-directional communication between the *Machine.py*, *Arduino.py & Spec.py* objects.

A more detailed description of the functions and attributes of the involved class objects are below.

## Structure Guide

This section describes each Python object and the functions which are available to call from it. For syntax details and examples of usage please refer to the System Level Control section as well as the commented source code.

### System.py

#### Description:

This class object’s main purpose is to track user inputs and adjust the system wide settings accordingly. It contains all the necessary functions to add plate objects to the protocol sequence, load calibration data, add protocols, and retrieve information about existing protocols.

#### Functions:

**load\_calibration\_data() –** Used to load calibrated well-position data into system.  
**load\_settings() –** Used to load the system settings from the settings file. This includes the   
 labeled wavelengths of the excitation LEDs.  **save\_settings() –** Used to save system settings to the settings file.  **load\_well\_labels() –** Used to make a list of well labels when the object is made.  **set\_plate\_type() –** Creates a new plate object and sets it as either 24 or 96 well. **set\_current\_plate() –** Used to select a plate object so that it can be edited.  **get\_current\_plate\_type() –** Returns whether the current plate is 24 or 96 well. **get\_plate\_count() –** Returns the total number of plate objects. **remove\_plate\_object() ­–** Removes the plate object.  **get\_selected\_wells() –** Returns the labels of the selected wells for a specific plate object. **get\_settings() –** Returns a list containing the system settings.  **select\_wells() –** Select a specified set of wells for a specific plate object. **clear\_wells() –** Unselect any selected wells from a specific plate object.  **select\_all\_wells() –** Select all wells for a specific plate object. **add\_absorbance\_protocol() –** Add an absorbance protocol to a specific plate object.  **add\_fluorescence\_protocol() –** Add a fluorescence protocol to a specific plate object.  **add\_auxiliary\_protocol() –** Add an auxiliary protocol to a specific plate object.  **turn\_kinetic\_tracking\_on() –** Used to mark the start of a kinetic cycle.  **add\_kinetic\_absorbance\_protocol() –** Used to add an absorbance protocol to a kinetic cycle. **add\_kinetic\_fluorescence\_protocol() –** Used to add a fluorescence protocol to a kinetic cycle.  **add\_kinetic\_auxiliary\_protocol() –** Used to add an auxiliary protocol to a kinetic cycle. **get\_all\_protocol\_brief\_descriptions() –** Returns brief descriptions (Labels & Index) of all   
 protocols in a plate object. **get\_all\_protocol\_full\_description() –** Returns detailed descriptions (Labels & Settings) of all   
 protocol in a plate object. **initialize\_machine() –** Opens an instance of the Machine object. **initialize\_datasheet() -** Creates the data output file.  **start\_program() –** Executes the protocol sequence.  **close\_machine() –** Closes the instance of the Machine object.

### GUI.py

#### Description:

This object contains all the code that comes together to create the User Interface. Running this object will bring up an instance of the GUI. For details about how to use the interface see the [GUI Level Control](#_GUI_Level_Control_1) section. This class object has access to all the functions that are in System.py.

#### Functions:

**move\_plate\_out() –** Moves plate holder into position by the front door panel.

**initialize\_calibration() –** Runs the calibration sequence

**change\_settings() –** Used to change system settings such as excitation LED wavelengths.

**update\_settings() –** Used to update the system settings file.

**start\_plate\_selection() –** Opens the plate selection screen of the GUI

**start\_well\_selection() –** Opens the well selection screen of the GUI

**start\_protocol\_selection() –** Opens the protocol selection menu of the GUI

**load\_protocol\_sequence() –** Loads in a saved protocol sequence from a file.

**save\_protocol\_sequence() –** Used to save the currently selected protocol sequence to a file.

**add\_new\_plate\_configuration() –** Adds a new plate object and brings up well selection menu.

**back\_function() –** Used to return to previous screen.

**open\_protocol() –** Opens the menu for a selected protocol.

**add\_protocol() –** Adds the protocol and its settings to the protocol sequence.

**review\_protocols() –** Brings up the review screen of the GUI

**initialize\_program() –** Starts the execution of the protocol sequence

**reset\_system() –** Resets system. Clears the protocol sequence and returns to main menu.

**add\_items() –** Function to add object to the QTreeWidget (PyQt)

**get\_current\_item() -** Function to get selected object from the QTreeWidget (PyQt)

**remove\_plate() –** Deletes a plate object and all its protocols from the protocol sequence.

**remove\_protocol\_selection() –** Deletes a protocol object from the protocol sequence.

**edit\_protocol\_selection() –** Brings up the menu of the chosen protocol and allows editing.

**update\_protocol() –** Sets the edits made to a protocol and updates the protocol sequence

**edit\_knetic\_protocol\_selection() –** Used to edit a protocol part of a kinetic cycle

**update\_kinetic\_protocol() -** Sets into place the edits made to a protocol that is a part of a   
 kinetic cycle and updates the protocol sequence.

**add\_parent() –** Function to add object to the QTreeWidget (PyQt)

**add\_child() –** Function to add object to the QTreeWidget (PyQt)

**add\_subchild() –** Function to add object to the QTreeWidget (PyQt)

### 

### PlateConfiguration.py

#### Description:

This object contains all the information about a specific plate configuration input by the user. A plate configuration is defined as a set of wells to run specific protocols on. Each plate object contains the following information:

1. The plate type (24 or 96)
2. List of protocol of objects (the protocol sequence to perform on the selected wells)
3. List of selected wells.
4. Overall ordering if there is more than one plate configuration input by the user.

#### Functions:

**add\_protocol() –** Used to add a protocol object to the list of protocols.

**get\_protocol\_count() –** Returns the current number of protocols in the protocol list

**initialize\_wells() –** Creates a list of well labels when the plate object is created.

**get\_protocol\_brief\_description() –** Get a short description of the specified protocol.

**get\_protocol\_full\_description() –** Get a detailed description of the specified protocol.

**remove\_protocol() –** Removes a protocol from the protocol list.

**get\_selected\_wells() –** Returns the labels of the wells selected for the plate object.

**get\_well\_status() –** Returns list describing whether a well is selected or not.

**clear\_wells() –** Used to unselect all wells.

**selected\_all\_wells() –** Used to select all wells.

**select\_wells() –** Used to select a specified set of wells.

### Protocol.py

#### Description:

This object is a general parent class for all possible protocols. It contains the following general information about the selected protocol:

1. The protocol label (Absorbance, Fluorescence, Auxiliary, or Kinetic)
2. The plate index to which the protocol is assigned to
3. The order number of the protocol in the protocol list of the assigned plate object.

#### Functions:

**get\_brief\_description() –** Returns a short description (Label & Index in protocol list) of the   
 protocol

### Absorbance.py

#### Description:

This object contains all the system settings and necessary information to perform an absorbance measurement with settings defined by the user:

1. The exposure time.
2. The list of steps that need be run by the machine to perform the measurement.
3. The excel sheet layout of the measurement results.

#### Functions:

**get\_full\_description() –** Returns detailed description (Label & Setting Information) of protocol

**get\_exposure\_time() –** Returns exposure time of protocol

**set\_exposure\_time() –** Sets the exposure time of protocol

**initialize\_excel\_section() –** Creates the header of the protocol in the data output file.

**get\_dark() –** Includes list of Machine.py commands to execute to perform a dark   
 measurement.

**start() –** Includes list of Machine.py commands to execute to perform an Absorbance   
 measurement.

### Fluorescence.py

#### Description:

This object contains all the system settings and necessary information to perform a fluorescence measurement with settings defined by the user:

1. The exposure time.
2. The list of excitation LEDs to trigger.
3. The list of steps that need be run by the machine to perform the measurement.
4. The excel sheet layout of the measurement results.

#### Functions:

**get\_full\_description() -** Returns detailed description (Label & Setting Information) of protocol

**get\_exposure\_time() -** Returns exposure time of protocol

**get\_led\_index() –** Returns a string describing which excitation LED(s) were chosen

**get\_wavelength() –** Returns the wavelength(s) of the selected LED(s)

**set\_exposure\_time() –** Sets the exposure time of the protocol

**set\_excitation\_source() –** Sets which excitation LED(s) will be used.

**initialize\_excel\_section() -** Creates the header of the protocol in the data output file.

**get\_dark() –** Includes list of Machine.py commands to execute to perform a dark   
 measurement.

**start() –** Includes list of Machine.py commands to execute to perform a Fluorescence  
 measurement.

### Auxiliary.py

#### Description:

This object contains all the system settings and necessary information to perform a measurement using the auxiliary port with settings defined by the user:

1. The exposure time.
2. The list of auxiliary ports to trigger.
3. The list of steps that need be run by the machine to perform the measurement.
4. The excel sheet layout of the measurement results.

#### Functions:

**get\_full\_description() -** Returns detailed description (Label & Setting Information) of protocol

**get\_duration() –** Returns the duration in milliseconds

**get\_aux\_index() –** Returns which auxiliary port is being used.

**set\_duration() –** Used to set the duration in milliseconds.

**start() -** Includes list of Machine.py commands to execute Auxiliary functionality.

### Kinetic.py

#### Description:

This object is for time-course measurements. This object represents a set of protocols which need to be performed on each individual well. The following information is contained within this object:

1. The order number of the object in the list of protocols of the assigned plate configuration
2. The type of kinetic method: Interval (regular interval over specified time) or Repeat (repeated set number of time)
3. List of protocol of objects (the protocol sequence to perform in the Kinetic cycle)
4. The layout of the excel sheet of the results of the kinetic cycle

#### Functions:

**get\_full\_description() -** Returns detailed description (Label & Setting Information) of protocol

**add\_protocol() -**  Add protocol to the Kinetic cycle.

**remove\_protocol() –** Remove protocol from kinetic cycle.

**get\_protocol\_count() -** Returns the current number of protocols in the kinetic cycle.

**get\_protocol\_brief\_description() -** Returns a short description (Label & Index in kinetic cycle) of   
 the selected protocol.

**get\_protocol\_full\_description() -** Returns a detailed description (Label & Setting information) of   
 the selected protocol.

**get\_protocol() –** Returns the selected protocol object.

**initialize\_excel\_section\_start() -** Creates the header of the protocol in the data output file that   
 represents the start of the kinetic cycle.  **initialize\_excel\_section\_stop() -** Creates the header of the protocol in the data output file that   
 represents the stop of the kinetic cycle.

### Machine.py

#### Description:

This object contains all the functions to interface with the Arduino, Ocean Optics spectrophotometer, and Excel sheet to which the resulting data is being written to. Each protocol object contains a list of commands that the machine object needs to perform in order to execute the specific protocol.

#### Functions:

**Initialize\_arduino() –** Creates an Arduino object and begins Serial communication. **initialize\_spec() –** Creates a Spec object. **move\_plate() –** Used to move plate to specific position. **turn\_led\_on() –** Used to turn LED on **turn\_led\_off() –** Used to turn LED off **trigger\_aux() –** Used to trigger an auxiliary port. **set\_exposure\_time() –** Used to set the exposure time of the spec.  **set\_scans\_to\_avg() –** Used to set the number of scans to average on the spec. **set\_boxcar\_width() –** Used to set the boxcar width of the spec.  **calibrate() –** Used to run the calibration sequence **start\_program() –** Used to start protocol sequence execution.

### Arduino.py

#### Description:

This object is used to interface with the Arduino. It allows the user to write and read to and from the Arduino’s Serial.

#### Functions

**write()** – Used to write to the Serial port of the Arduino.  **read() –** Used to read from the Serial port of the Arduino.  **close() –** Disconnects system from Arduino.

### Spec.py

#### Description:

This object is used to interface with the Ocean Optics Spectrophotometer through their API. It allows the user to set exposure time, boxcar width, scans to average, and to capture spectra.

#### Functions

For functions, see the linked [Seabreeze](https://github.com/ap--/python-seabreeze) documentation.

# System Level Control

### 

### Importing the module and Initializing a System Instance

At the beginning of any script to interface with the OSP, you first need to import the actual functionality from the System.py class, and then initialize an instance of the System object, by writing:

**from System import \***

**system = System()**

### 

### Adding a Plate Configuration

Before adding any protocols, you must first add a plate configuration (a Plate object) to be able to select wells and protocols by writing:

**system.set\_plate\_type(plate\_type)**

where *plate\_type* is the number 24 (for 24-well plate) or 96 (for 96-well plate).

### 

### Selecting Wells

Once a plate object exists, you can now select the wells to perform specific protocols on, by one of two options.

1. Select all wells, by writing:

**system.select\_all\_wells(plate\_obj\_index)**

where*plate\_obj\_index* is the index of the plate for which wells are being selected. If you only have one plate object, then the index of that plate is zero.

\*\* or \*\*

1. Select individual wells, by writing:

**system.select\_wells(list\_of\_well\_indices)**

where, *list\_of\_well\_indices* is a list of the well indices, where well A1 is indexed at 0 and indices increase along the rows. For example, in a 24-well plate, the indices of the first row wells A1-A6 = 0-5, second row B1-B6 = 6-11, etc.

### Adding a protocol to a plate configuration

The user has four possible options when adding a protocol to a plate configuration: Absorbance, Fluorescence, Auxiliary function, or Toggling Kinetics. Below the addition of an absorbance and fluorescence protocol is demonstrated.

EXAMPLE 1: To add an absorbance protocol, write the following:

**system.add\_absorbance\_protocol(exposure\_time, ‘Absorbance’, plate\_obj\_index, order)**

*exposure\_time**–* the desired exposure time of the measurement in milliseconds.

*plate\_obj\_index* – the index of the plate from which you are selecting wells. If you only have   
 one plate object, then the index of that plate is zero.

*order –* the index of the protocol in the existing list of protocols of the plate configuration object. If this is the first protocol the user is adding to the plate configuration then the *order* is equal to zero. If it is the second, then the order is equal to 1 and so on.

EXAMPLE 2: To add a fluorescence protocol, write the following:  
**system.add\_fluorescence\_protocol(exposure\_time, led\_index, wavelength\_led\_1, wavelength\_led\_2, wavelength\_led\_3, ‘Fluorescence’, plate\_obj\_index, order)**

*exposure\_time –*the desired exposure time of the measurement in milliseconds.

*led\_index ­–* binary string consisting of 4 values, such as ‘1111’. The first three characters of the string represents the on/off status of the excitation sources LED 1, LED 2, and LED 3, respectively. The fourth character represents the on/off status of the external source (for the bottom optical train). A “1” means that the fluorescent measurement will be taken with that specific source on, while a 0 means that light source will be off.

*wavelength\_led\_# -* gives the user the ability to define in the system the wavelength of the LED, which is then output to the excel sheet for reference.

*plate\_obj\_index* – the index of the plate for which you are selecting wells. If you only have one plate object, then the index of that plate is zero.

*order –*  the index of the protocol in the existing list of protocols of the plate configuration object. If this is the first protocol the user is adding to the plate configuration then the *order* is equal to zero. If it is the second, then the order is equal to 1 and so on.

### Running the Input Protocol Program

Once all plate configurations and their respective protocol lists are created, the user can start their script by writing the following:

**system.start\_program(system.plateList, file\_name)**

where,

*system.plateList­ –* the python list containing the list of plate configurations made.

*file\_name –* [this input to the function is optional] optional name of the excel file to which all the resulting data of the program will be written to. If left blank the name of the file will be set to the string containing the date and time of the program of the following form: ‘data\_2018\_08\_12\_18\_57’.

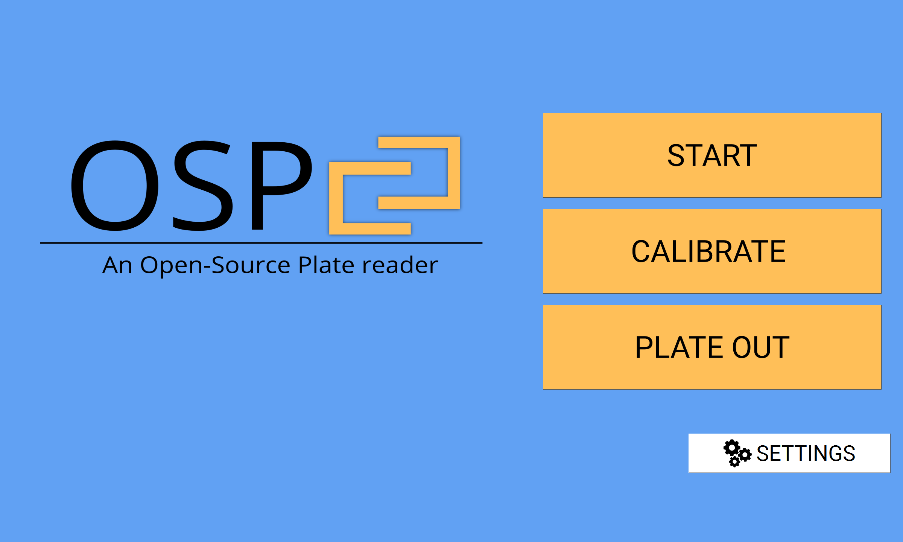
# GUI Level Control

## Running the GUI

If working from a Raspberry-Pi, the best way to initialize OSP is to use the Terminal, direct the Pi to the *‘Python Files’* directory, and run *GUI.py* from there. If not running on a Raspberry-Pi, simply run *GUI.py* either from the Terminal or from any desired IDE.

**!!! Important Note !!!:** The size of the GUI has been optimized for the Raspberry-Pi Touch Screen. If you have a high-resolution display (e.g. 4K), you may need to adjust resolution to display the GUI properly. Users with standard resolution displays (e.g. 1920x1080) can see the GUI in a smaller screen.

## The Intro Screen

****

**4**

**3**

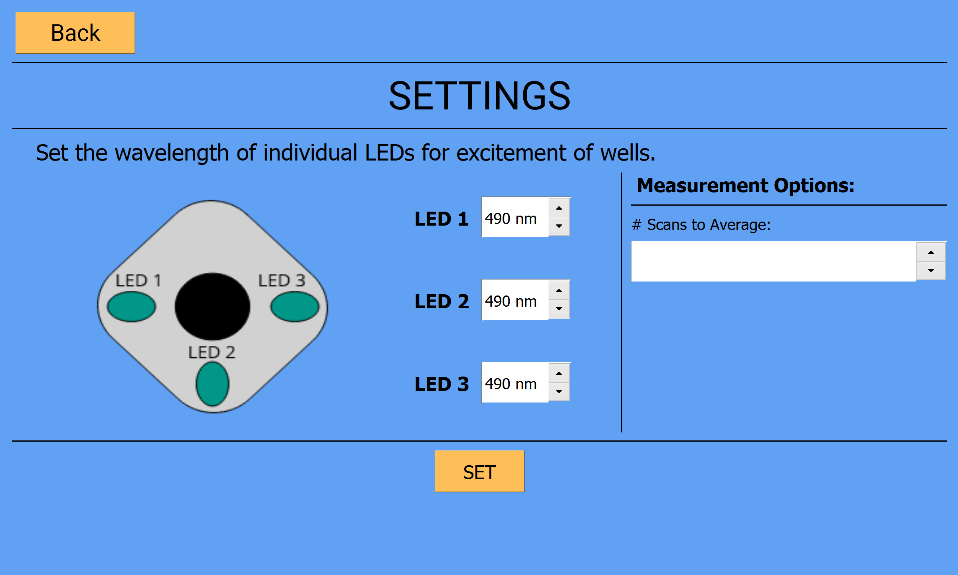
**2**

**1**

1. The **START** button leads to the plate selection screen, where the user can input whether they will be using a 24-well plate or a 96-well plate for the protocol sequence.
2. The **CALIBRATE** button allows the user to re-calibrate the well positions in the system for either a 24-well plate or a 96-well plate. When pressed, a pop-up message shows up asking the user to input the plate-type (24 or 96) that they wish to calibrate the positions for.
3. The **PLATE** **OUT** button allows the user to have the device move the plate holder to the edge of the front door for plate loading and removal.
4. The **SETTINGS** button brings the user to the Settings Window, where they are allowed to change specific settings of the system.

## The Settings Screen

Here, the user has access to change the system settings, specifically, the LED wavelength labels and the number of scans to average.

****

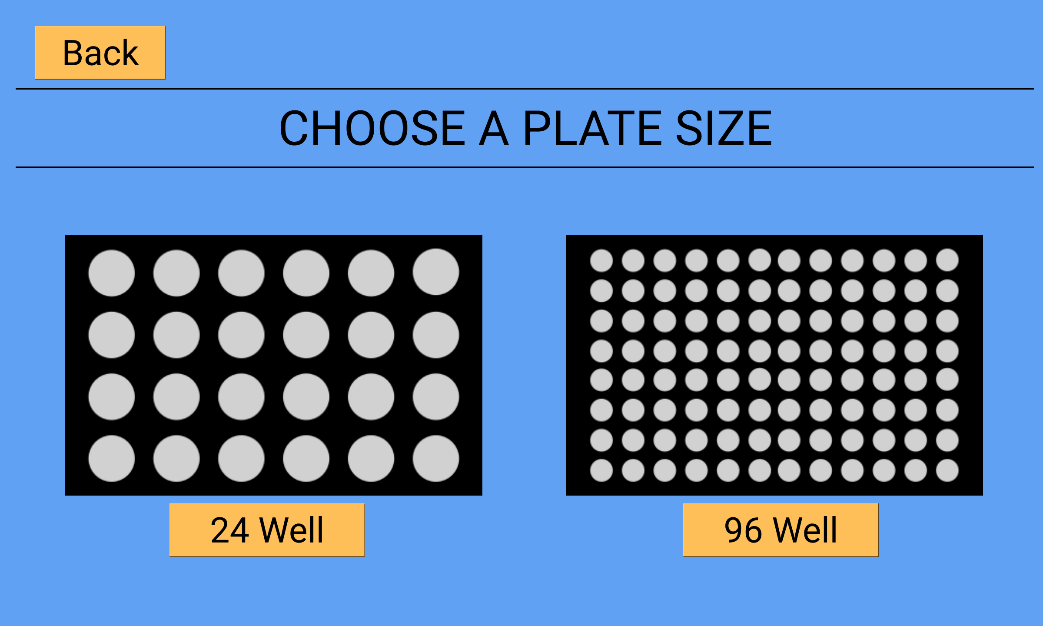
**2**

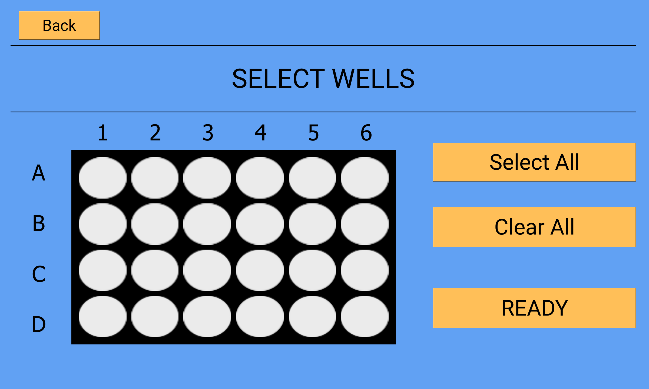
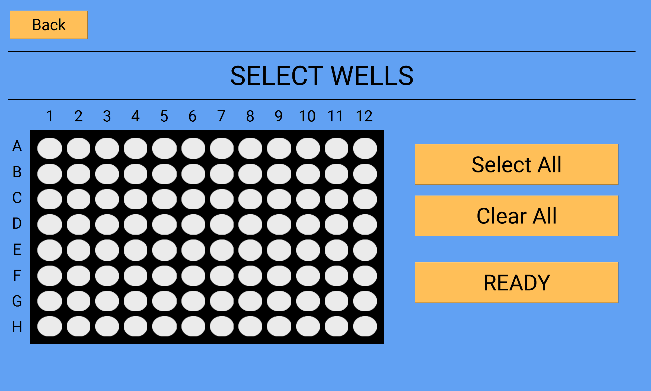
**1**

1. Here, the user can enter an integer to change the number of scans that the spectrophotometer averages when performing a measurement.
2. The three LEDs correspond to the three excitation light sources. Here the user can change the wavelengths to describe the LEDs connected to a specific LED socket position (or digital switch). These wavelengths are then used as reference for data output.
3. Once the user clicks the **SET** button the displayed settings are saved to *‘…/System Settings/savedSettings.txt’* .
4. Hitting the **BACK** button will take the user to the home screen or the protocol selection screen (depending on which screen was up/active before the settings were accessed).

## The Plate/Well Selection

The user has the option to select which type of plate they will perform the protocol sequence on. A protocol sequence can only be run on one type of plate. Plate selection leads the user to the well selection window for the chosen plate type.

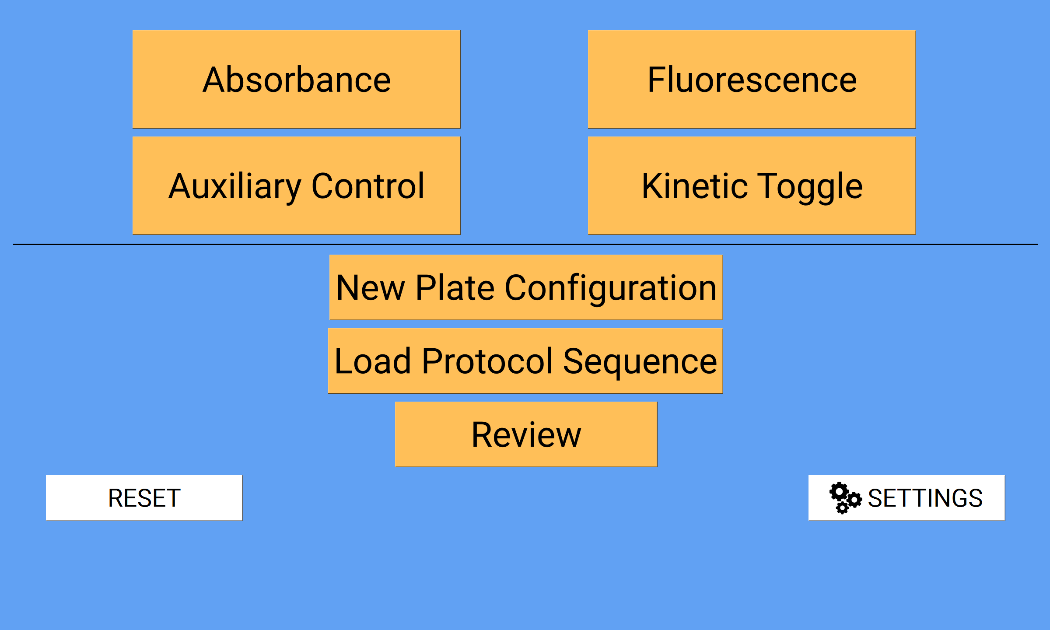
****

** **

*Click & Drag to select wells. Pressing the ‘****BACK’*** *button will return the user to the plate selection window.*

## The Protocol Selection

From this screen, the user can choose to add specific protocols to their protocol sequence. Settings may also be changed from this screen.



**8**

**4**

**7**

**6**

**5**

**3**

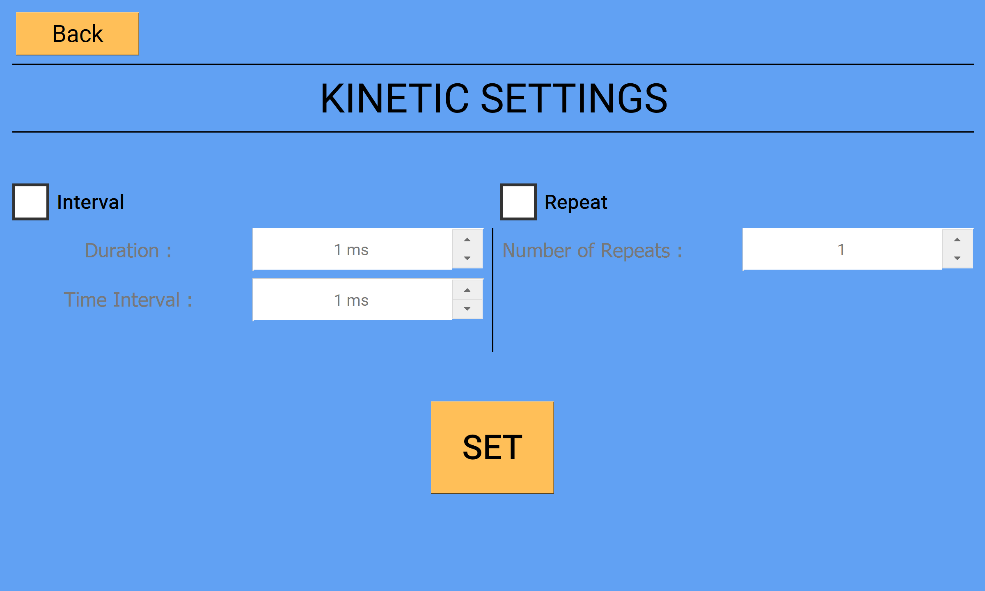
**2**

**1**

1. The **ABSORBANCE** button takes the user to the absorbance protocol settings screen. This allows the user to set the exposure time.
2. The **AUXILIARY CONTROL** button takes the user to the auxiliary control settings screen. Here, the user can choose which auxiliary port to turn on and for how long.
3. The **FLUORESCENCE** button takes the user to the fluorescence protocol settings screen. Here the user chooses which excitation LED to use and sets the exposure time of the measurement.
4. The **KINETIC TOGGLE** switch allows the user to turn on/off a kinetic cycle (time-course).
5. The **NEW PLATE CONFIGURATION** button allows the user to create a new selection of wells for a different protocol sequence. Going down this path creates a new *PlateConfiguration.py* object.
6. The **LOAD PROTOCOL SEQUENCE** button allows the user to load a pre-existing (custom made) protocol sequence that has been saved to the repository. The loaded sequence will replace the current protocol sequence.
7. The **REVIEW** button takes the user to the review screen where they can see the protocol sequence resulting from their inputs, and edit any of the protocols individually or remove them.
8. The **RESET** button will take the user back to the main screen and clear the system of any protocols and well selections.

## Kinetic Toggling

**“**Kinetics” allows the user to iteratively perform a specific protocol for time-course measurements without having to manually repeat the inputs. OSP allows two methods of kinetics: (1) Interval and (2) Repeat. Note that the user can only select one method, described below:

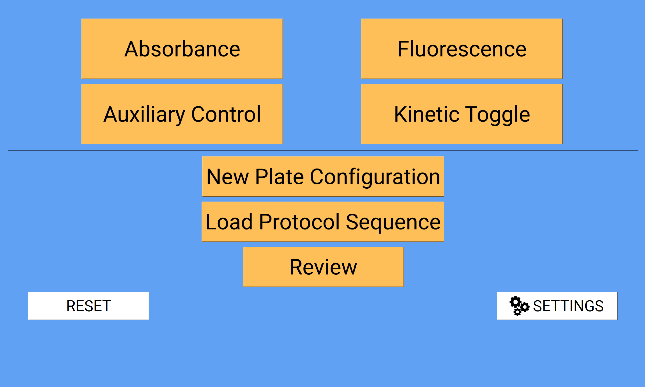
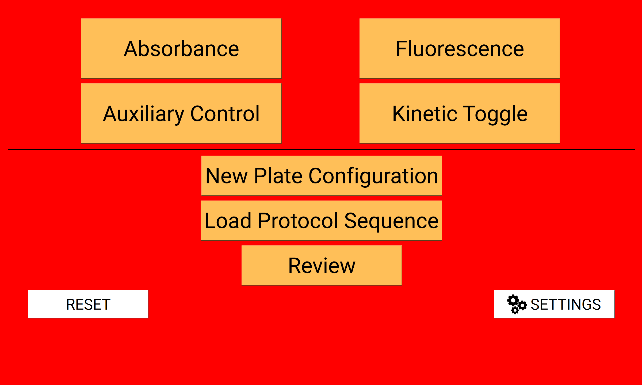
****

#### Interval Kinetics:

This method performs a protocol or a sequence of protocols in intervals (based on the interval specified by the user) for a specific duration (e.g. absorbance of a sample every 10 seconds for 2 minutes). ***The user must ensure that the time it takes to perform the specific protocol or sequence of protocols does not exceed the interval time.***

#### Repeat Kinetics:

This method simply repeats a protocol or a sequence of protocols the specified number of times. Once the SET button is pressed on the ‘Kinetic Settings’ screen, the user is taken back to the protocol selection menu, which will then have a background that is bright red. The kinetic state is set to ON as long as the background is red. To toggle the kinetic cycle off, press the kinetic button again and the screen will return to its original (blue) color. ***The user will not be able to review the protocol or add a new plate configuration until the kinetic cycle is turned off.***

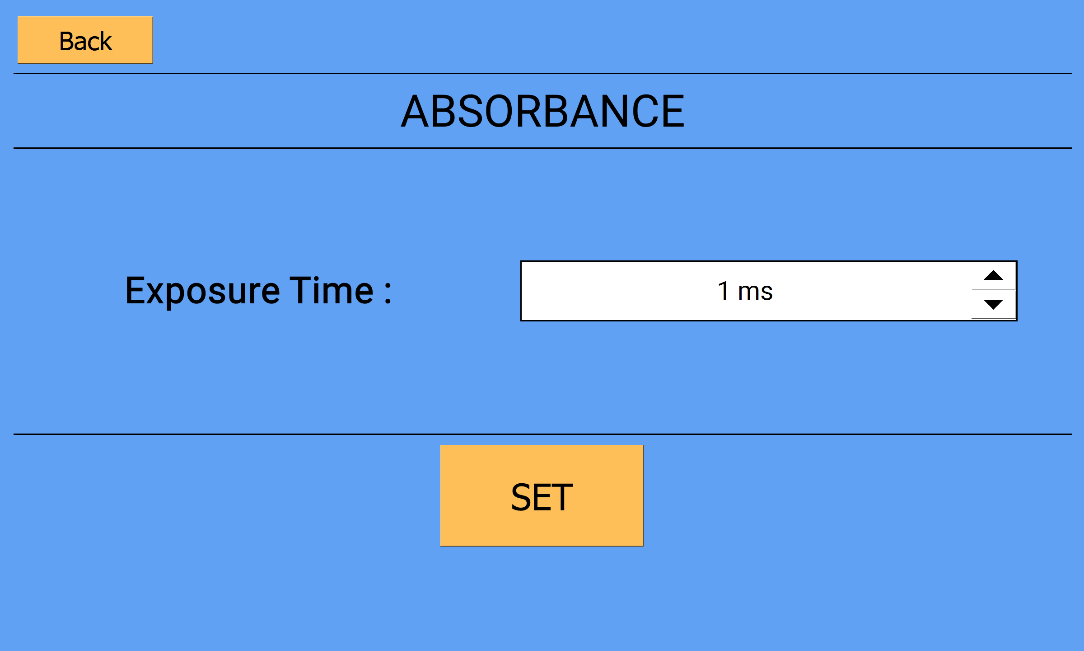
 

*Kinetics ON*

*Kinetics OFF*

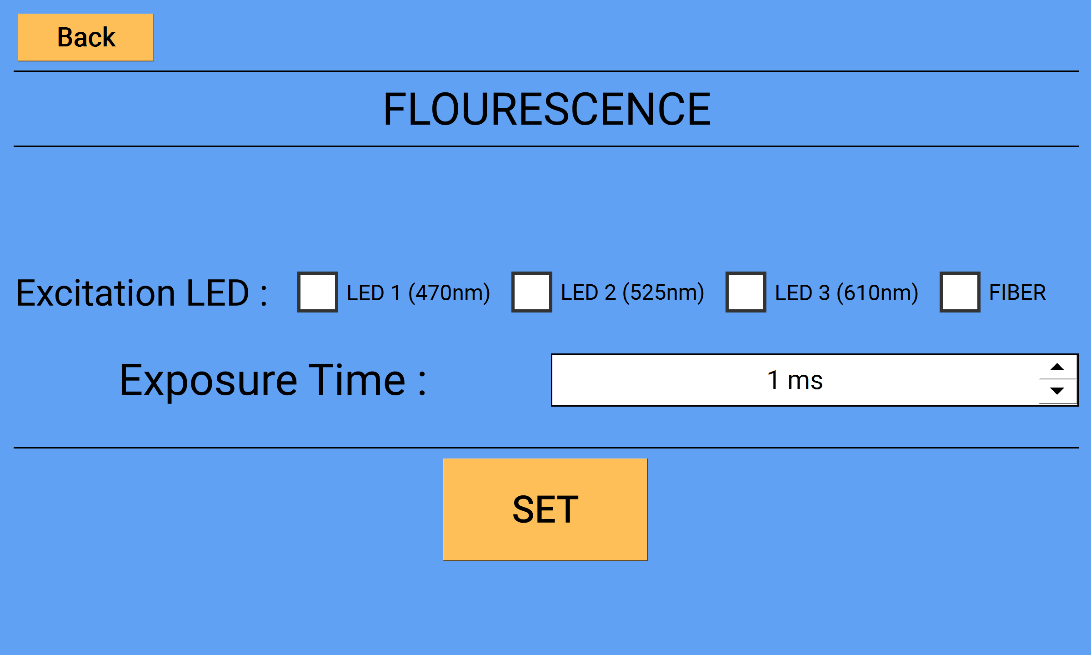
## Absorbance Settings

The only tunable parameter here is exposure time. The light source for absorbance measurements by default is the top LED light source (by default, a 430nm LED + White LED).



## Fluorescence Settings

The user must specify which excitation LED to use. The wavelength of the LED is referenced from the saved settings. The system will display a warning message if the user attempts to press the set button without checking one LED box. Only one LED checkbox can be marked. In other words, only one LED can be turned on at one time.

****

## Auxiliary Settings

Here, the user can select which auxiliary port (see “**Top Optical Train**” Assembly) they want to turn on by digital logic, and for what duration of time. Note that only one port can be turned on at one time. One port is specifically for unpowered LEDs. The other three communication ports are more customizable; to access their trigger ports, see the DB9 port pin-out diagram on the next page. By default, triggering a port also moves the motors to position the selected well under the top auxiliary optics attachment.

****

**2**

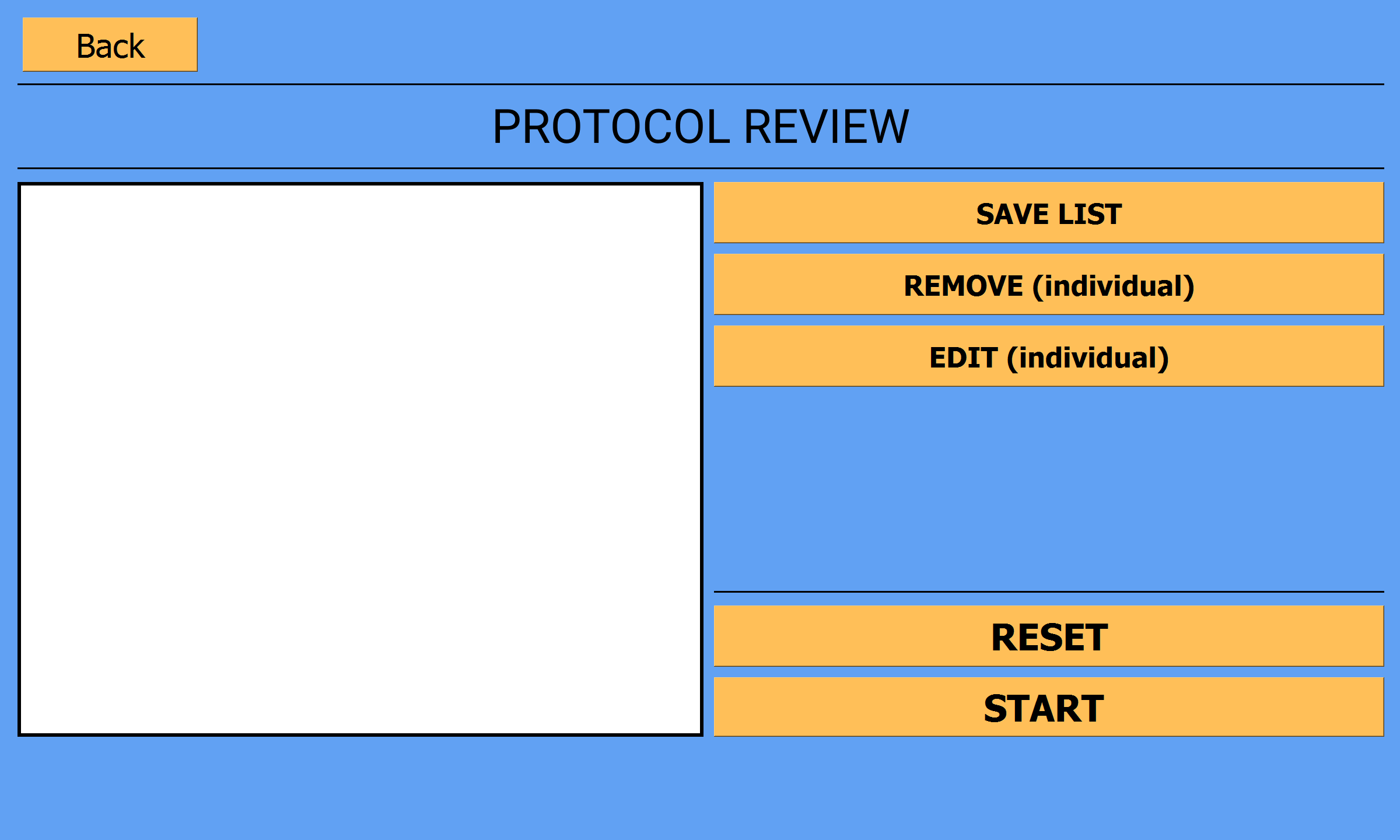
**1**

1. The AUX LED PORT (LED #4) controls an extra unpowered LED of choice, and the default limiting resistor sets the current to ~20mA (assuming a ~3V LED bias voltage). Setting the ‘duration’ sets the amount of time this auxiliary LED is on once the well is positioned under it.
2. Any of the three generic ‘ports’ allows direct control of external electrical connections that can be accessed through the DB9 header (see below diagram for pin-out). Each port contains two data pins (1 IN and 1 OUT) for digital control and inputs to/from the Arduino. The user can customize an Arduino script for automating external devices. For custom series of events where movement under the auxiliary piece is not required, this default can be eliminated by commenting out the movement function (machine.move\_plate()) for the corresponding port number of the Auxiliary.py script.

When accessing the auxiliary port +12V power (pin 3), the total current draw should be less than 3 amps to ensure sufficient power for all internal components. When accessing the auxiliary port +5V power (pin 3), the total current draw should be less than 1.5 amps to prevent damage to the internal switch.

|  |  |  |
| --- | --- | --- |
|  | 1. +5V 2. Port 2: Trigger 3. +12V 4. GND 5. Port 1: Data In | 1. Port 1: Trigger 2. Port 2: Data In 3. Port 3: Trigger 4. Port 3: Data In |

## Review Protocol Sequence

****

**4**

**3**

**2**

**1**

Plate #1

Selected Wells

Protocols

Absorbance

Exposure Time: 1000 (ms)

1. Window displaying the selected plates & protocols in their desired order. Each selection in the window is selectable. By highlighting one of the protocols, the user can edit/remove it.
2. The **SAVE LIST** button allows the user to save the current set of protocols in their order as a ‘protocol’ object. **NOTE:** When saving a sequence of protocols, the system saves the protocols and all their settings in shown order, and the plate type. ***The loaded sequence plate type cannot be changed unless the system is reset.***
3. The **REMOVE** button allows the user to remove one of the protocols OR an entire plate configuration from the protocol sequence. If a plate configuration (e.g. *Plate #1*) is selected, that entire plate object and all of its assigned protocols will be deleted.  **NOTE: *The user can only delete one protocol or plate at a time.***
4. The **EDIT** button allows the user to edit a single protocol or a single selection of wells. In order to edit the selected wells, the user needs to highlight the selected wells tab.