GUI Control

You need to have the following installed:

- Python (brew install or apt-get)
    - SIP (pip install …)
    - PyQt5
    - Serial
- Prolific USB to RS232 Drivers (link for OSX)
- pump_control_gui.py, pump_functions.py, set_pump_number.py (repository)

To control the pump, you must:
1) Connect power to the pump.
2) Connect your computer to the pump via the USB-to-RS232 (note, RS232 looks similar to VGA) and CBL-PC-PUMP (RS232 to RJ11, which looks like an old phone line cable/ethernet) cable. Be sure that you have connected the RJ11 plug into “to computer” and not “to network”.
3) Run pump_control_gui.py.

Using the GUI

NOTE: You may have to alt+tab to the GUI window.

You can use this GUI to control one New Era OEM syringe pump. The code can be relatively easily modified to control multiple pumps. After specifying a size and rate (max 4 significant digits, default units are uL/hr), you can start the pump by clicking “Run/Update”. Be sure to double check that you have correctly specified the diameter of your syringe, because it will drastically affect the true pumping rate. Because the direction that the pump is moving is difficult to tell at typical flow rates, be sure to check the “Pump Direction” label. It should say “Infuse” or “Withdraw”. The “Pump Status” label will tell you the output of the last command to the pump. This will probably be hard to interpret in some circumstances. In general, the pump reports its status with an “I” or a “W” (for infuse/withdraw) if it is currently running, or a “P” or an “S” if it is stopped (for paused/stopped). If you change the syringe diameter, it will just respond with an “S”/”P” or “I”/”W”. For a change in pumping rate, it will respond with a directional/pumping status letter, along with the rate you specified.

NOTE: if you are using the dual syringe pump, you cannot adjust the block manually, so you have to switch the direction of the pump to “Withdraw” and then manually change the flow rate to a large number. I suggest temporarily changing the diameter of the syringe to 1mL (regardless of true syringe size), and setting the pumping rate to 10,000 (uL/hr). Be sure to switch the diameter back to the true diameter.

Command Line Control

You need to have the following installed:

- Python (brew install or apt-get)
    - Serial
    - Pump_functions.py (repository)
- Prolific USB to RS232 Drivers (link for OSX)

Connect power to the pump, and connect your computer to the pump via the USB-to-RS232 and CBL-PC-PUMP cable. It may be helpful to read the documentation below before continuing.

Because the pusher block cannot be moved manually on a dual stall pump, you will have to move the block by setting a flow rate, running the pump, and stopping it. You can reverse the direction of the pump. (**pump_rate, pump_dir, pump_run, pump_stop**). **Try not to run the pump into the end of the drive screw. This could strip the screw.** The pump has built in stall detection as a precaution for running the block to the end of the drive screw, but this should not be used as Plan A. I would recommend setting the volume to be dispensed by the pump (**pump_vol**) to be less than the current volume in the syringe. This way, the pump will automatically stop before it reaches the end.

The pump will return its state after every command you give. Most of the responses will just be “S”, which means “stopped”. Other states can be “P” (paused), “I” (running - infusing), “W” (running - withdrawing). If the pump does not understand your command, it will respond with a “?”, followed by an error code. Possible error codes are “NA” (command not applicable), which most often happens after you tell the pump to stop when it is already stopped, and “OOR” (parameter out of range). If there is no error code, the pump did not recognize your command.

Methods of *pump_control* in pump_functions.py

All examples assume you have instantiated an instance of pump_control:

>>> pc = pump_control()

rate(rat=0, adr=0)

This method can be used to query or set the flow rate of the pump. Note that the flow rate will be incorrect if you have not correctly set the diameter/size of the syringe in the pump (see **pump_dia**). Acceptable rate units are uL/mL per hour/minute. If you do not specify a rate, the command will query the pump for the current flow rate setting. You can specify a number as an int/float or as a string to include the units. The rate cannot be changed while the pump is running.

Returns: the response from the pump. This command can only accept a maximum of 4 significant digits, and will return an error if you include more. Include the address of the pump if

you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.rate()
Pump 00 says: S10.00MH
>>> pc.rate(rat=1)
Pump 00 says: S1.00MH
>>> pc.rate(rat='100uh')
Pump 00 says: S100.00UH

dia(adr=0, dia='')

This method can be used to query or set the diameter of the syringe in the pump. If you do not specify a diameter/size, the command will query the pump for the current diameter setting. Acceptable values for the "dia" argument are: '1ml', '3ml', '5ml', '10ml', '20ml', or '60ml'. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address). You cannot change the diameter of the pump while the pump is running.

>>> pc.dia()
('Pump 00 says: S11.99', 5ml')
>>> pc.dia('60ml')
Pump 00 says: S

stop(adr=0)

This method is used to stop the pump from running. Note that telling the pump to stop while it is already stopped will cause the pump to return a "not applicable" error. All three of the below examples have the same functional behavior (the pump stops), but have slightly different return values. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.stop() # initial state of the pump was 'infusing' (running)
Pump 00 says: P
>>> pc.stop() # state is 'paused' before running this command
Pump 00 says: S
>>> pc.stop() # state is 'stopped' before running this command
Pump 00 says: S?NA

run(adr=0)

This method is used to run the pump. The pump will run at the flow rate that is stored in memory, which can be queried via **rate()**. Note that you must stop the pump using **stop()** to change any properties of the pump (address, flow rate, syringe diameter, volume). Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.run()
Pump 00 says: I

dir(dir='', adr=0)

This method can be used to query or set the direction that the pump can run. If you leave the argument ‘dir’ blank, the command will query the current direction setting. Acceptable values for dir are ‘inf’ or ‘wdr’ for infuse or withdraw. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.dir()
Pump 00 says: SINF
>>> pc.dir(dir=’wdr’)
Pump 00 says: S

vol(vol=’’, adr=0)

This method can be used to query volume to be dispensed, set the volume to be dispensed, or set the units used in this method. If you leave the argument ‘vol’ blank, the command will query the current volume to be dispensed. **After dispensing this volume, the pump will stop automatically.** The argument ‘vol’ can be blank, a float, or a string denoting units (‘ul’ or ‘ml’). If the value for ‘vol’ is out of range, the pump will return a ‘?OOR’ error. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.vol()
Pump 00 says: S14.53ML
>>> pc.vol(vol=’ul’) # change units to microliters
Pump 00 says: S
>>> pc.vol(vol=3.14) # change volume to be dispensed to 3.14 UL
Pump 00 says: S

query_dis(adr=0)

This method is used only to query the volume that the pump has dispensed since the volume has last been cleared (**clearvol()**). The pump returns the volumes dispensed in both the infuse and withdraw directions. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.query_dis()
Pump 00 says: SI1.000W0.000ML

reset_vol(dir=’inf’, adr=0)

This method is used to clear the volume dispensed (see **pump_querydis**). The volumes dispensed in the ‘infuse’ and ‘withdraw’ directions are stored separately in memory, and must be cleared individually. The command defaults to clearing the volume dispensed in the ‘infuse’ direction. Include the address of the pump if you have more than one pump connected, otherwise it will default to pump 0 (the default address).

>>> pc.reset_vol()
Pump 00 says: S

adr(adr=0)

This method can be used to change the address of the pump currently connected to the computer. Before trying to change the address of the pump ensure that the intended pump is the only pump connected to your computer. The default address assigned is 0. This function should not be necessary unless you have more than one pump connected to your computer.

>>> pc.adr(adr=0)
Pump 00 says: S
>>> pc.adr(adr=1)
Pump 01 says: S

Relevant Function Documentation for PySerial

Pump_functions.py is essentially a wrapper for these functions. You should not have to use these functions if everything works normally.

serial.Serial(*port=$port*, *baudrate=19200*, *timeout=None)*

Use this command to initialize a serial object which we will use to communicate to the pump. If you are on a *nix machine, the $port is probably '/dev/tty.usbserial' or '/dev/cu.usbserial'. In Windows, you will need to check the device manager to see which COM port your pump was assigned, but usually it is 'COM1'. It may be helpful to set a timeout of <1.

>>> import serial
>>> pump_object = serial.Serial(port='/dev/tty.usbserial', baudrate=19200)

serial.Serial.open()

Used to open a port that has already been initialized using serial.Serial(). After initialization, the port is open, so this function is only useful after the port has been closed.

>>>pump_object.open()

serial.Serial.close()

Used to safely close a port that has been initialized.

>>>pump_object.close()

serial.Serial.write()

This is the primary function we will use to communicate with the pump. The pump will respond to every command you send to it using the following syntax:

(to pump) -> <command data><CR>
(from pump) -> <STX><response data><ETX>

<CR> is a carriage return. Use '\r' or '\x0D'. <STX> and <ETX> are start of text and end of text, respectively. They will be represented by '\x02' and '\x03'.

>>> pump_object.write('0RAT\r')
>>>pump_object.read_all()
'\x0200S10.00MM\x03'

serial.Serial.read_all()

Reads the entire buffer. Use to read the response from the pump. See example from write().