GUI Control

You need to have the following installed:

- Python (brew install or apt-get)
    - SIP (pip install …)
    - PyQt4
    - Serial
- Prolific USB to RS232 Drivers (link for OSX)
- new_era.py, pump_control.py, set_pump_number.py (repository)

Connect power to the pump, and connect your computer to the pump via the USB-to-RS232 and CBL-PC-PUMP cable. Run set_pump_number.py. Run pump_control.py. Pump_control.py assumes that you are running a *nix system, so if you are running Windows, you will have to manually edit the name of the port on line 6. You may have to alt+tab to the GUI window. Be sure to select the size of your syringe from the drop-down menu. Otherwise, the flow rate will be incorrect. The “Contents” text box is for you to label the contents of the syringe, which is helpful to label if you are connecting more than one pump. Set a flow rate, and click “Run/Update” to change the flow rate/syringe size. You cannot change anything while the pump is running. The “Reverse Direction” button will reverse the direction that the pusher block moves. It will stop all pumps before reversing directions. The “Prime” button starts the pump at a high flow rate to fill the end of your tubing quickly. Because the pusher block cannot be moved manually on the dual stall pump, you will have to adjust it by using the “Prime” button and “Reverse Direction”.

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.

Functions in pump_functions.py

initialize_pump()

This function returns a Serial object that we will pass to all functions used to control the pump. You must run this first to initialize communication with the pump.

>>> import pump_functions as pf
>>> pump_object = pf.initialize_pump()

pump_rate(pump_object, rat=0, adr=0)

This function 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).

>>> pf.pump_rate(pump_object)
Pump 00 says: S10.00MH
>>> pf.pump_rate(pump_object, rat=1)
Pump 00 says: S1.00MH
>>> pf.pump_rate(pump_object, rat=’100uh’)
Pump 00 says: S100.00UH

pump_dia(pump_object, dia=’’, adr=0)

This function 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.

>>> pf.pump_dia(pump_object)

('Pump 00 says: S11.99', 5ml')
>>> pf.pump_dia(pump_object, '60ml')
Pump 00 says: S

pump_stop(pump_object, adr=0)

This function is used to stop the pump from running. I 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).

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

pump_run(pump_object, adr=0)

This function is used to run the pump. The pump will run at the flow rate that is stored in memory, which can be queried via **pump_rate**. Note that you must stop the pump using **pump_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).

>>> pf.pump_run(pump_object)
Pump 00 says: I

pump_dir(pump_object, dir='', adr=0)

This function 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).

>>> pf.pump_dir(pump_object)
Pump 00 says: SINF
>>> pf.pump_dir(pump_object, dir='wdr')
Pump 00 says: S

pump_vol(pump_object, vol='', adr=0)

This function can be used to query volume to be dispensed, set the volume to be dispensed, or set the units used in this function. 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).

>>> pf.pump_vol(pump_object)
Pump 00 says: S14.53ML
>>> pf.pump_vol(pump_object, vol='ul') # change units to microliters
Pump 00 says: S
>>> pf.pump_vol(pump_object, vol=3.14) # change volume to be dispensed to 3.14 UL
Pump 00 says: S

pump_querydis(pump_object, adr=0)

This function is used only to query the volume that the pump has dispensed since the volume has last been cleared (**pump_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).

>>> pf.pump_querydis(pump_object)
Pump 00 says: SI1.000W0.000ML

pump_clearvol(pump_object, dir='inf', adr=0)

This function 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).

>>> pf.pump_clearvol(pump_object)
Pump 00 says: S

pump_adr(pump_object, adr=0)

This function 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.

>>> pf.pump_adr(pump_object, adr=0)
Pump 00 says: S
>>> pf.pump_adr(pump_object, 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().