**ElveFlow SDK Applied to Python**

Overview

My end point function will 1. Change valve, 2. Change flow rate. That’s it. The SDK interface from Elveflow supplies capabilities that can lead to these functions, but lack any python objects to accomplish it out of the box. Their basis of operation is to:

1. Establish connection to devices
2. Set up things such as calibration array and PI feedback loop (ie. flow rate sensor, pressure feedback to set flow rate)
3. Dictate set points
4. Stop PI loop
5. End communication

As the Elveflow system will be scripted and tested, the script will crash sometimes during testing and make steps 4 and 5 unreliable to get to. For the time being, I plan to skip them unless it ruins operation.

Here is the approach with a python implementation

1. Make init function for class Fluidics does the following:
   1. Establishes connection to both MUX and OB1 units
   2. Adds in calibration array to OB1
   3. Adds in flow sensor
   4. Starts PI loop
2. Have two base methods
   1. Flow method changes flow rate.
   2. Valve change method changes which valve MUX in on
3. Have one higher level method that chains together flow rate changes and valve changes to accomplish exact goals of system.

How was it done in real code?

The init function

def \_\_init\_\_(self, mux\_com\_port, ob1\_com\_port):  
  
 # OB1 initialize  
 ob1\_path = 'ASRL' + str(ob1\_com\_port) + '::INSTR'  
 Instr\_ID = c\_int32()  
 pump = OB1\_Initialization(ob1\_path.encode('ascii'), 0, 0, 0, 0, byref(Instr\_ID))  
 pump = OB1\_Add\_Sens(Instr\_ID, 1, 5, 1, 0, 7,  
 0) # 16bit working range between 0-1000uL/min, also what are CustomSens\_Voltage\_5\_to\_25 and can I really choose any digital range?  
  
 Calib\_path = r'C:\Users\CyCIF PC\Documents\GitHub\AutoCIF\Python\_64\_elveflow\calibration\1\_12\_24\_cal.txt'  
 Calib = (c\_double \* 1000)()  
 Elveflow\_Calibration\_Load(Calib\_path.encode('ascii'), byref(Calib), 1000)  
 #Elveflow\_Calibration\_Default(byref(Calib), 1000)  
 OB1\_Start\_Remote\_Measurement(Instr\_ID.value, byref(Calib), 1000)  
 self.calibration\_array = byref(Calib)  
  
 set\_channel\_regulator = int(1) # convert to int  
 set\_channel\_regulator = c\_int32(set\_channel\_regulator) # convert to c\_int32  
 set\_channel\_sensor = int(1)  
 set\_channel\_sensor = c\_int32(set\_channel\_sensor) # convert to c\_int32  
 PID\_Add\_Remote(Instr\_ID.value, set\_channel\_regulator, Instr\_ID.value, set\_channel\_sensor, 0.9, 0.004, 1)  
  
 # MUX intiialize  
 path = 'ASRL' + str(mux\_com\_port) + '::INSTR'  
 mux\_Instr\_ID = c\_int32()  
 MUX\_DRI\_Initialization(path.encode('ascii'), byref(  
 mux\_Instr\_ID)) # choose the COM port, it can be ASRLXXX::INSTR (where XXX=port number)  
  
 # home  
 # answer = (c\_char \* 40)()  
 self.mux\_ID = mux\_Instr\_ID.value  
 # MUX\_DRI\_Send\_Command(self.mux\_ID, 0, answer, 40)  
  
 self.pump\_ID = Instr\_ID.value  
  
 return

Flow function

def flow(self, flow\_rate):  
  
 set\_channel = int(1) # convert to int  
 set\_channel = c\_int32(set\_channel) # convert to c\_int32  
  
 set\_target = float(flow\_rate) # in uL/min for flow  
 set\_target = c\_double(set\_target) # convert to c\_double  
  
 # OB1\_Start\_Remote\_Measurement(self.pump\_ID, self.calibration\_array, 1000)  
 OB1\_Set\_Remote\_Target(self.pump\_ID, set\_channel, set\_target)  
  
 data\_sens = c\_double()  
 data\_reg = c\_double()  
 set\_channel = int(1) # convert to int  
 set\_channel = c\_int32(set\_channel) # convert to c\_int32  
 time.sleep(3)  
 error = OB1\_Get\_Remote\_Data(self.pump\_ID, set\_channel, byref(data\_reg), byref(data\_sens))  
 current\_flow\_rate = data\_sens.value  
 current\_pressure = int(data\_reg.value)  
 print('current flow rate', int(current\_flow\_rate))  
 print('error: ', error)

Valve change function

def valve\_select(self, valve\_number):  
 *'''  
 Selects valve in mux unit with associated mux\_id to the valve\_number declared.* ***:param*** *c\_int32 mux\_id: mux\_id given from mux\_initialization method* ***:param*** *int valve\_number: number of desired valve to be selected* ***:return****: Nothing  
 '''* desired\_valve = valve\_number  
 valve\_number = c\_int32(valve\_number)  
 MUX\_DRI\_Set\_Valve(self.mux\_ID, valve\_number, 0) # 0 is shortest path. clockwise and cc are also options  
  
 valve = c\_int32(-1)  
 MUX\_DRI\_Get\_Valve(self.mux\_ID, byref(valve))  
 current\_valve = int(valve.value)  
  
 while current\_valve != desired\_valve:  
 MUX\_DRI\_Get\_Valve(self.mux\_ID, byref(valve))  
 current\_valve = int(valve.value)  
 # print('valve', current\_valve, 'deired valve', desired\_valve)  
 time.sleep(1)  
  
 time.sleep(1)