Graduate Project: I07 Syringe Pump for Electrochemical Cell

Start Date	Duration	Graduate	Supervisor	Customer	Technical Support
21 Jul 2020	3 months	Dolica Akello-Egwel (STFC)	Gary Yendell	Hadeel Hussain(and other I07 scientists)	None

Project Description

The current streamDevice-based EPICS driver for the Microlab 500/600 syringe pump does not provide an interface for the complex commands required for full control of experiments. This project aims to replace this driver with python API that provides the full interface of the device. The key required feature is the ability to send simultaneous demands to both syringes. This is vital in being able to create continuous flows through the chemical cell. It is important that the library can be imported and used within our existing pythonIOC interpreter to create EPICS IOCs that present PVs to interface with the device. This is currently only supported in Python2.7, so some work will be required to get it working in Python3

Background

EPICS: https://wllrg.rs/2019/06/03/epics.html | https://docs.epics-controls.org/en/latest/guides/EPICS_Intro.html

Python 3 at DLS: https://confluence.diamond.ac.uk/x/GxKRBQ

Microlab 500/600 Manuals

Introduction from Hadeel about the experiments done with the syringe pump...

Current Situation

EPICS driver - Handles simple commands well (e.g. go to position and stop), but does not provide the interface to construct complex commands (e.g. move both syringes simultaneously and continuously)

Python scripts / GUI - Complex commands work reasonably well, but it is unreliable and crashes while pump is moving. Does not use threading to allow commands while control loop is running (e.g. stop!) and does not handle errors / communication properly

Outcomes

Outcome	Target time
Develop a microlab Python3 library - this should include a command line interface providing a limited subset of commands for basic tests	Month 1
Develop a simple simulator for the Microlab 500/600 and use to create system tests	Month 2/3
Create a higher level control layer that provides more complex features, such as continuous flows (Further input from beamline before starting this)	Month 2/3
Use pythonIOC to create an IOC and GUI (EDM screens) mapping PVs to the microlab python library	Month 2/3

Extra goals

The simulator will be be very useful to verify the library works with both the 500 and 600 API and could be used to create system tests. Its complexity will be determined by the viability of remote access to the device at DLS. If we do have any problems, then more time should be spent on the simulator into it to make it as realistic as possible - e.g. a state machine to simulate realistic timescales for commands, readbacks during movements, error states.

If there is interest, some time can be spent on the areaDetector driver for controlling the camera for remote working.

Checklist

Before the placement starts

- [x] Supervisor to fill in the project page with details of the project
- [x] Gary Yendell(probably) to collect equipment from I07 and setup in G07
  - [x] Borrow a suitable camera and create IOC
- [x] Update pythonIOC to Python3 / pipenv

- [x] Supervisor to arrange a kick-off meeting with the Graduate, Andy Wilson, Ulrik Pedersen (plus customer / technical support if appropriate) to present the scope of the project

During the placement

- [x] Supervisor to arrange regular catch-up meetings with the Graduate to check progress
- [x] Supervisor to arrange mid-point review meeting with Graduate, Andy Wilson, Ulrik Pedersen to briefly outline progress
- [ ] Graduate to fill in the project page with documentation on design decisions made, links to documentation, and any other relevant information
- [x] Graduate to arrange a final meeting ~2 weeks before the end of the project with the Supervisor, Andy Wilson, Ulrik Pedersen to present what they have done and what is still outstanding

At the end of the placement

- [ ] Supervisor to arrange handover with the Graduate to make sure documentation is complete and to take back the delivered project

Task List

- [x] 3.1.7 Execution Commands (I don't think we will need R, it will only be used when constructing the complex commands)
- [x] 3.3.1 Instrument Information Requests
- [x] 3.3.2 Instrument Status Requests | The values with multiple values can just return the numerical value it can be looked up in the manual to find out what it means | The timer request can return True of False based on the one bit it uses
- [x] A _send_syringe_command method - This will be used for:
  - [x] Simple syringe setters (3.2.1 and 3.3.3) Syringe speed and valve speed
  - [x] 3.3.3 Syringe readbacks
  - [x] 3.3.4 Valve readbacks | For valve position just return the number
- [x] A send any given command string and get the response method and CLI option
- [x] Split auto address and initialise into two separate methods
- [x] Split socket and send recv methods into a separate class (MicrolabSocket ?)
  - [x] Add a method to this to check if the device is busy via the `Q` command
- [x] Create a method to enable one specific example of a complex command - This will help us figure out the best way to make it completely generic
  - [x] This should take a list of up to five objects: Left Valve & Syringe positions, Right Valve & Syringe positions, and a time delay to insert between one syringe move and the other.
  - [x] It should be able to format the correct command given any subset of these five commands
- [x] Create a startup script for an interactive python shell with the Microlab object created and all useful imports done
- [ ]
- [x] Change Command to SubCommand and create a Command class that can replace usage of List(SubCommand) and do some validation checks too
- [ ]
- [x] Document how to run the iPython prompt and run some commands (do moves, get positions, get error status, other useful things) - for me so I can give it a try, but also it may as well form the actual documentation
- [x] Consider how the library will be used from EPICS
  - [x] Commands and Config Sets should return whether they were successful (True or False)
  - [x] Status and Config gets should return the value if successful or None if not
- [x] Add PVs for library parameters / commands
- [x] Build EDM screens
- [ ] Create internal IOC PVs to control cycling

General code jobs

- [x] Update to use super() consistently in preference to super(<class name>, self)
- [x] Update to use type annotations everywhere

~~Requires further thought~~

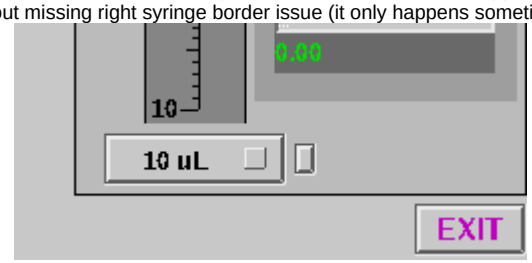
- [ ] ~~Logic to construct generic complex commands - These could contain any(?) combination of: Valve speed / position, syringe speed / position & dispense amounts for either side, timer delays~~
  - [ ] ~~Are there restrictions to what can go into a single commands?~~
    - [ ] ~~Syringe position vs dispense?~~
    - [ ] ~~Multiple syringe positions / dispenses / valve positions / timer delays?~~
  - [ ] ~~SyringeMove, ValveMove, TimerDelay objects for the user API that can be passed into a method to construct the command string (with the order maintained as appropriate) - `complex_command([ValveMove(LEFT, angle=90), SyringeMove(LEFT, pickup=5000), TimerDelay(1), ValveMove(RIGHT, angle=270), SyringeMove(RIGHT, dispense=2500)])`~~

Final Bits

- [x] Fix titles under Display Properties
- [x] Fix firmware version display
- [x] Remove initialise on IOC boot
- [x] Change valve position to a dropdown with the demand and readback
- [x] Verify if device statuses are being parsed correctly and therefore if the device really is always in error state for some reason
- [x] Add pickup / dispense controls (for each side) - PVs:
  - [x] Pickup button
  - [x] Dispense button
  - [x] Increment field in uL
- [x] Add PVs and screen for flow controls - PVs:
  - [x] Fill speed - For the fast move to fill / empty the syringes
  - [x] Flow speed - For the slow move to do the flow
  - [x] Cycle active status LED
  - [x] Run button to start - Run method to get() the above PVs and passes them to internal cycle method (assuming that we want to move between 0 and max)
    - [x] Assume right syringe fills sample and left syringe empties sample (as it is now)
    - [x] This should assume the right syringe is filled and at max, and left syringe is empty
    - [x] Then it should cycle these commands
      - [x] Right Valve = Output, Right Syringe to 0, Left Valve = Input, Left Syringe to MAX, Speed = Flow Speed
      - [x] Right Valve = Input, Right Syringe to MAX, Left Valve = Output, Left Syringe to 0, Speed = Fill Speed
  - [x] Duplicate Stop button on this screen for convenience
  - [x] A nice visualisation of whether it is in fill or flow state?
- [x] Add help overlays where appropriate
- [x] Make led colours consistent - It can be any combination of green/red (good/bad) and dim/lit (inactive/active) e.g.
  - Busy status: Dim green for not busy, lit green for busy (Both states are good, but provide information)
  - Any error: Dim red for no error, lit red for error (active is bad)
  - Monitor Status: Lit green for OK, dim green for not running (active is good)
- [ ] Add API Version parameter for 500/600 to microlab and comms - We can't test this unfortunately as we don't have a 500
  - [ ] Send YSM1 to set full resolution and then set max steps to 2000 for internal calculations (vs 48000 for the 600) - I think this only exists on the 500 https://confluence.diamond.ac.uk/download/attachments/89203680/HamiltonMicrolab500UserManual.pdf?version=1&modificationDate=1570037436000&api=v2 Section F-8)
  - [ ] Check for version in _send and if it is 500 do an extra recv to get the echoed command before the recv to get the actual response
  - [ ]

Things I Didn't Do

- Figure out why the Instrument Error Status always reports right syringe/valve and left syringe/valve errors
- Figure out why the Instrument Busy Status often reports that the left syringe and valve are busy even when they're not
- Make the "Initialised" light appear based on the result of the Instrument Error Request
- Figure out why coverage report includes import statements in wrapper.py
- Implement cleaning button
- Make the syringe volumes update with the mux button

- Figure out what causes the syringe movement arrows to be chopped
- Figure out missing right syringe border issue (it only happens sometimes...)
  - 

- Implement readbacks for the valve position for the cycling screen
- Align text with bytes lights
- Use the more complicated monitor loop in a different file you showed me a few weeks ago (I don't remember what it was)
- Put "hamiltonmicrolab-Bu5xIk7-/" in the right place in the .gitingore file

Documentation

Remote Setup

The syringe pump is set up at Diamond in G07. It can be accessed via a terminal server in the lab at http://172.23.241.5 on port 7020 using TCP.

It is powered by a Networked PDU, so it can be power cycled remotely (this is accessible here: http://172.23.243.206 - Channel 4).

There is a GigE camera providing a live view of the syringe pump (this needs some additions, e.g. a transform plugin to rotate the view). This is running under procServ on pc0118 and exposing an http stream of the camera feed, which can be accessed at http://pc0118.cs.diamond.ac.uk:8094/Microlab.mjpg.mjpg.

I have made a skeleton python3 module at /dls_sw/work/mef65357/python3/RHEL7-x86_64/microlab with a very simple test that it can talk to the device.