Cavro XP3000 GUI Documentation

Release 1.0

Nikos Koukis

July 16, 2014

Pump3000 is a GUI implementation for communicating with the Cavro XP3000 pump series. It was implemented in Python with the use of PySide. This guide should provide a way of setting up the software on your computer and give you a first taste on how to use it efficiently. For the source code or the .exe file click here

CHAPTER

ONE

CONTENTS:

1.1 Getting Started

There are 2 ways of running the XP3000 GUI:

- Running the *Pump3000.exe* [Windows only]
- Running *from source* [Python required]

Either way the user must first download the necessary files for the software, located at http://www.github.com/bergercookie/Pump3000. This can be done either by downloading the project locally from the github page or by cloning the project

The user can **download the software** by visiting the github page: http://www.github.com/bergercookie/Pump3000 and then pressing the *Download Zip* button

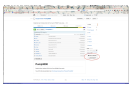

Figure 1.1: Downloading the software

Alternatively the user can **clone** the project. In order to do that he must first make sure that git is installed on the platform. If it isn't then installing it requires issuing:

```
sudo apt-get install git "for Linux users - command-line
sudo port install git "MacOS users - command-line
```

Then in order to clone the project the user can run the git clone command:

```
git clone http://wwww.github.com/bergercookie/Cavro-Pump-XP3000-GUI.git
```

For windows users, the desktop version of Github is suggested: https://help.github.com/articles/set-up-git#platform-windows

1.2 Hardware Configuration

1.2.1 Equipment used

For the hardware communication to be achieved the following equipment is suggested:

- USB -> Serial (RS-232) adapter
- RS-232 -> RS-485 converter

- Power cables (Power + GND)
- Jumper wires (at least 2 Female to Male)
- Breadboard (optional)

1.2.2 Connecting the XP3000

The serial communication with the pump was implemented using the RS-485 serial protocol.

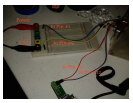

Figure 1.2: Suggested Hardware Communication to the pump

Figure 1.3: Adapters setup

The needed steps to archive a similar connection would be the following:

1. Connect the pump to a power supply. As described in the manual the XP3000 requires a **24VDC power supply** with a current rating of at least **1.5A**. As seen in hardware-conf, during the development part, the power was supplied indirectly to the pump first by running the supply cables into a breadboard and then using extra jumpers to connect to pump pins #1 (power) and #9(GND)

2. For the serial communication, you need to connect the RS-485A, RS-485B signals from the pump PCB into the RS-485+, RS-485- of the data terminal (PC) used for the communication. Take extra notice when it comes to connecting the jumpers from the serial adapter to the pump PCB corresponding pins

3. Also if the computer in use doesn't have a serial port available, the user should use either a USB to RS-485 adapter, or combine a more commonly found *USB to RS-232 with a RS-232 to RS-485* adapters

To sum it up, here is the typical RS-485 pinout for the pump [1], as it is described above

Role	Pin No	Signal to Connect
Power	#1 [Power]	24 VDC
	#9 [GND]	Ground (of the power supply)
Protocol Signals	#11 [RS-485A]	RS-485 (+)
	#12 [RS-385B]	RS-485 (-)

1.3 Software Configuration

By now the user should have a copy of the project on his machine. If not refer to section: *Getting Started*

1.3.1 Windows Installation

There are two ways of running the software on windows:

[1] For more information regarding the pump pinout consult the pump manual

- Running the executable
- Running from source

Running the executable

Running the executable is as simple as running the Pump3000.exe located in the <location_to_project>/python_code/dist folder [2] [3].

Figure 1.4: Finding the .exe file

Running from source

Alternatively, the user can **run the software from source**. This requires the execution of a full python distribution. It is advised that the user uses one of the following distributions due to the simplicity of the installation process

- Anaconda
- Canopy

The user must also have the following packages installed:

- Pyserial
- PySide

For the Anaconda distribution the user can install packages from the Command line using one of the following ways [4]

```
$conda install <package_name>
```

```
$pip install <package_name>
```

Finally, the user should run the software using the command prompt:

```
python <location_to_project>\python_code\Pump3000.py
```

1.3.2 Linux/Mac Installation

In case the user wants to run the software from a *NIX environment, a basic python distribution must be installed on the platform, (already preconfigured on most *NIX systems). The user must also have the following packages installed:

- Pyserial
- PySide

After this configuration, the user can run the software from the command-line [*NIX] command-prompt [Windows]:

[2] You can make a shortcut to Pump.exe file but do not move it outside the dist folder as it depends on the dlls files located there

[3] If the executable doesn't run correctly, try installing the vcredist_XXX file. Replace the XXX with the architecture of your processor. The vcredist files are located in the vcredist directory. After the installation, rerun the executable. If the problem persists, contact me

[4] Consult the instructions on the site of the corresponding distribution for more details on installing packages

```
python Pump3000.py
```

Note that the user should first go to the folder, the Pump3000.py is located.

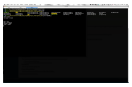

Figure 1.5: Running from source on MacOS

1.3.3 Using the software

The first thing the user should do, if he doesn't know the serial connector to the pump, is figure out the correct port:

- On **Windows** machines this can be done by getting the *'Ports (COM & LPT)'* tab:

  ```
  Start Menu > right-click "My Computer" > select "Manage" >  Click on the "Device Manager"

  (On the "Device Manager") Click "Ports (COM & LPT)" tab > select the port your connector is on
  ```

  See *'Manage' tab*, *Configuring COM ports*

- On ***NIX** machines this can be done the following way:

  ```
  cd /dev

  ls -lart| grep tty
  ```

This will give you a list of the currently available ports. Ejecting and reinserting the connector to the computer and in the meantime running the second command again will help you recognise the port the pump is running on.

Figure 1.6: 'Manage' tab

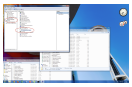

Figure 1.7: Configuring COM ports

> **Warning:** Make sure that you have selected the correct port, otherwise the pump will not respond and will not raise any error Message

After that you are ready to *run the software* (whichever way you want). You should be directed to the New_Device window where the port connected to the pump must be selected

If you have selected the correct port, the connection to the pump is established and the software will automatically initialize the pump, by moving the plunger to the upper position.

You are now in the Main Window. From here you can:

- Command a volume delivery,

Figure 1.8: Configuring correct port for pump

- Change the speed of the plunger,
- Issue a quick command to the pump (halt, push_all, etc)

Figure 1.9: Main window

From the main window you can navigate to a series of **other dialogs:**

- **Editor's Tab**

  The Editor's Tab gives the user the ability to issue a series of commands to the pump. These commands are supplied in the "Pump Commands" page. The user can also issue raw pump commands in the same way.

  A typical example of issued commands would be the following:

```
pump.property_set('speed', '5')
# Python Comments, write as many as you want
# Empty lines don't matter

# Raw commands as well
/1?2R\r

pump.send_Command('A0')
```

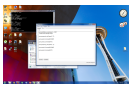

Figure 1.10: The Editor's Tab

- **History**

  From here the user can see all the commands sent to the pump which can be divided to 2 types:

  - Commands issued by the user
  - Commands issued by the software to decide pump status

- **Syringe Size**

  The user can decide the syringe size.

- **Reports**

  Gives the user an overview of the pump currently configured settings

- **Pump Parameters**

  The user can change certain parameters of the plunger movement such as "Top Velocity", "Slope" etc.

  Port

  The user can configure the port that the pump is connected to. This window is also summoned at the start of the Pump30000

Figure 1.11: History window

Figure 1.12: Reports window

1.4 Pump Commands

In this section the basic commands for communication with the pump are introduced. You can issue a series of these commands from the editor's tab. For the full list of available commands see the pump_model.py. You can use most of the methods of the Pump Class in the Editor tab as well.

Note: 'self' is a Python class arguement and shouldn't be issued by the user.

---

pump.connect_new(self, port_name)

Arguements:

```
port_name: string
```

Description

Connect to the given port address.

Example:

```
pump.connect_new('/dev/tty.-SerialPort1-1') -> Configuring a serial connection on OSX
```

pump.initialize_pump(self, output = 'right')

Arguements:

```
output: string [Optional]

    values: 'left', 'right'
```

Description

Initialize the pump. The user can optionally issue the output valve. By default if not given otherwise, the output valve is set to the right.

Examples:

```
pump.initialize_pump() -> Initializing the pump with default output valve (right)
pump.initialize_pump(output = 'left') -> Initializing the pump with output valve set to left
```

pump.valve_command(self, position)

Arguements:

```
position: string

    values: 'I' (Input)
```

```
            'O' (Output)
            'B' (Bypass)
```

Description

Set the valve to certain position.

Examples:

```
pump.valve_command('B') -> Set the valve to Bypass position
```

pump.property_set(self, a_proprty, value)

Arguements:

```
a_property : string
value      : int
```

Description

Set a certain property for the plunger. The following properties are available for modification. These are listed with regards to the command that should be issued and the range of the values permitted:

```
'speed'                 : 'S', [1  , 40]
'backlash'              : 'K', [0  , 31]
'slope                  : 'L', [1  , 20]
'start_velocity'        : 'v', [50 , 1000]
'top_velocity'          : 'V', [5  , 5800]
'cutoff_velocity'       : 'c', [50 , 2700]
'cuttof_velocity_steps' : 'C', [0  , 25]
```

The user is encouraged to consult the manual for an overview on the commands above

Examples:

```
pump.property_set('speed', '10') -> Set the plunger speed to 10
pump.property_set('backlash', '15') -> set the backlash steps to 15
```

pump.volume_command(self, direction = 'P', vol = '0', special=None)

Arguements:

```
direction : string

    Values: 'P'
            'D'

vol       : string [Microliters]

special   : string [Optional]

    Values: 'push_all'
            'pull_all'
```

Description

Volume pushing / drawing mechanism. The user can issue a volume delivery as well as issue a special push / pull all action. In case the special action is given, the 'vol' argument is neglected

Examples:

```
pump.volume_command(direction = 'D', vol = '5') -> Dispense 5 microlitres
pump.volume_command(special = 'push_all') -> Dispense fluid volume
```

pump.send_Command(self, command, bits_on_return = 0)

Arguements:

```
command        : string
bits_on_return : int [Optional]
```

Description

Sending Commands to the pump. Consult the manual for a detailed list of the pump commands for the RS-485 Protocol. Using this method you must have already defined the pump address to send to (Via the main window or the Editor tab) and you should issue neither the Run character [R] nor the terminating character

Examples:

```
pump.send_Command('A3000', 10) -> Move the plunger to abs. position 3000, read back 10 bits
pump.send_Command('T') -> Terminate plunger move in progress
```

pump.ser.write(comand)

Arguements:

```
command: string
```

Description

This command should be used only when a direct serial command has to be sent to the pump. User must issue the pump to which he is addressing to as well as the terminating character It is advisable that the user should prefere higher level commands such as 'send_Command' which doesn't require the prefix & suffix characters, and also check the availability of the pump

Examples:

```
pump.ser.write('/2ZR\r') -> Initialize the pump with the address 1
```

Note: As seen in the Examples section of pump.ser.write, the user should refer to the pump address + 1.

1.5 License

The current software is licensed under LGPLv2.1 as is PySide, the Python Bindings for Qt with which it was developed. The purpose of the license is to encourage the free distribution of the project and to encourage everyone interested to come in and contribute to it. The license is provided in the GNU website https://www.gnu.org/licenses/lgpl-2.1.html

1.6 About

The software is the outcome of my individual project for the Systems Biology & Bioengineering Research Laboratory of the Mech. Engineering Department of NTUA

1.6.1 Tools Used

- The project was written in Python .
- For the GUI development I used the PySide Qt Bindings and the Qt-Designer application.

- The documentation was written in Sphinx
- All the code as well as the documentation was edited using Vim

1.7 Contact Information

For information, bug reports or just to drop a comment about the software, either contact me on github or use one of the methods below:

- Mobile: +30 6978952969
- email: nickkouk@gmail.com

Otherwise you can contact the Systems Biology and Bioengineering Lab:

Department of Mechanical Engineering, Ktirio M
National Technical University of Athens
Heroon Polytechniou 9
15780 Zografou, GREECE
tel: +30 210 7721516

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*