



Getting Started with FlexSEA

Here you'll find the documentation related to Dephy's hardware, software, and API. Before continuing, you should first read the [safety](#) information.

- 1 [Datasheets and other Materials](#)
- 2 [Hardware Overview](#)
 - 2.1 [Board Layout](#)
 - 2.2 [Buttons and LEDs](#)
- 3 [GUI](#)
 - 3.1 [GUI Connectivity](#)
 - 3.1.1 [Connecting via USB](#)
 - 3.1.2 [Finding the Device Port on Windows](#)
 - 3.1.3 [Finding the Device Port on Linux](#)
 - 3.1.4 [Connecting via Bluetooth](#)
 - 3.2 [Configuring the Device](#)
 - 3.3 [Controlling the Device](#)
 - 3.4 [Logging Data](#)
 - 3.5 [Visualizing Data](#)
- 4 [Python API](#)
- 5 [Programmable Safety Features](#)
 - 5.1 [Low Voltage Protection \(UVLO\)](#)
 - 5.2 [I_{pt} Current Limit: Programmable Fuse](#)
 - 5.2.1 [Theory of Operation](#)
 - 5.2.2 [Non-linearity](#)
 - 5.2.3 [Calculating and Programming](#)
 - 5.2.4 [How to choose the right values](#)
 - 5.2.5 [Pre-computed values \(\$\pm 30A\$ sensor\)](#)
 - 5.2.6 [Tips and Examples](#)
- 6 [Bootloading](#)
 - 6.1 [Bootloader CLI](#)
 - 6.2 [Programming Directly](#)
 - 6.2.1 [Programming Regulate](#)
 - 6.2.2 [Programming Execute](#)
 - 6.2.3 [Programming Manage](#)
- 7 [Units](#)
- 8 [Gains](#)
 - 8.1 [Current Control Gains](#)
 - 8.2 [Position Control Gains](#)
 - 8.3 [Impedance Control Gains](#)

Datasheets- See Folder

- [ActPack 4.1](#)
- [BA30](#)
- [Smart Dock](#)

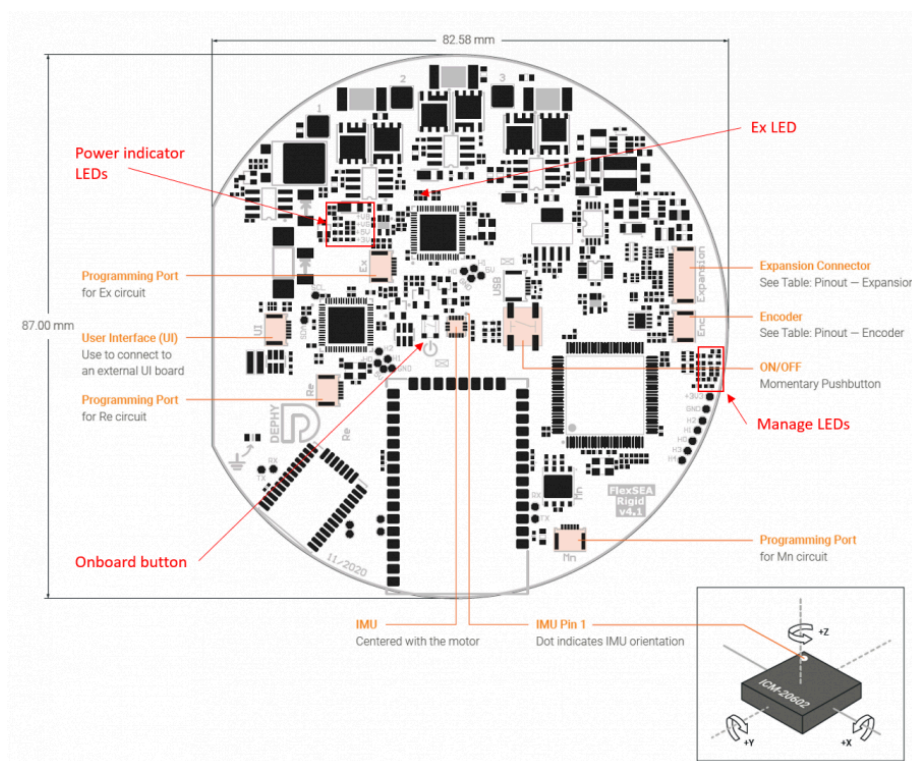
2- Hardware Overview

2.1- Board Layout

The primary PCB (Printed Circuit Board) in Dephy's devices is known as **Rigid**. Rigid contains three microcontrollers:

- **Regulate (Re)** — Contains the on/off logic and is responsible for handling the board's power
- **Execute (Ex)** — Responsible for controlling the motor
- **Manage (Mn)** — Runs code for a controller, sends commands to the other microcontrollers, and sends/receives data to/from the user

An example rigid schematic is shown below:



2.2- Buttons and LEDs

Products can have one of two firmware variants:

- **MultiColor** — Uses an RGB LED (this is the default for ActPack)
 - **Power On:** Press and hold the button for more than 2 seconds. The LED will fade on. When it starts flashing, you can release the button
 - **Normal Operation:** The LED will be aqua (USB power) or green (battery power). The color will not change between fade on and flashing. When the button is released, the LED will enter glowing mode
 - **Fault Detected:** After the fade on, the LED will flash red to indicate that a problem was detected. The board will power off as soon as the button is released
 - **Power Off:** Press and hold the button for more than 2 seconds. The LED will turn red, and fade off. When the LED is off, you can release the button
 - **Brief Click:** The LED will turn purple
- **Monocolor** — Uses a single color LED (green or red, depending on the product generation). This is the default for EB50+ Exos
 - **Power On:** Press the button for a brief moment (around 200ms). If you keep pressing, the LED will flash

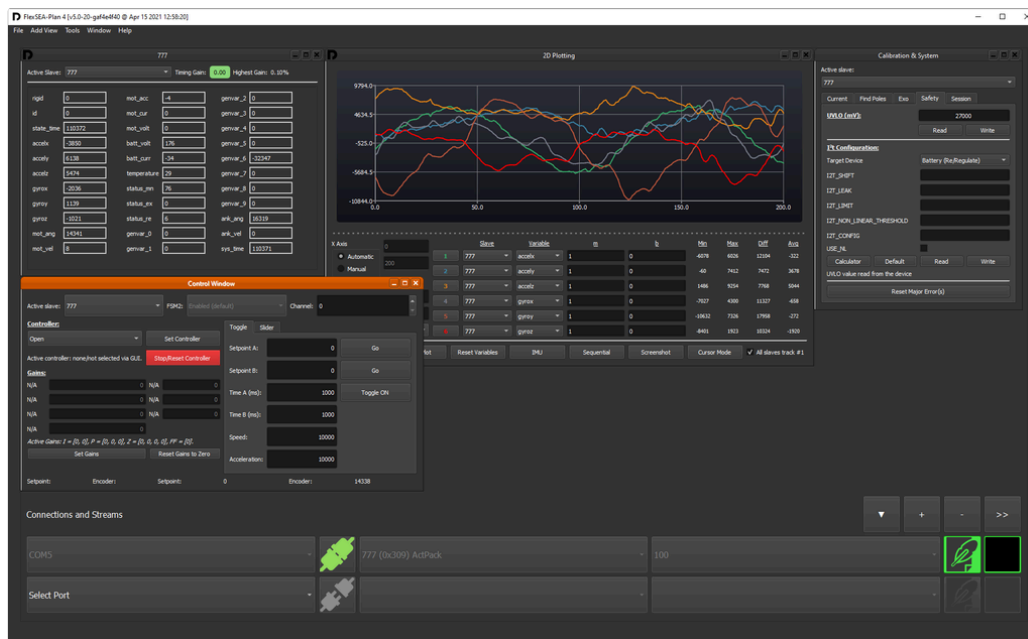
- **Normal Operation:** When the button is released, the LED will enter glowing mode.
- **Fault Detected:** The board will power off as soon as the button is released.
- **Power Off:** Press and hold the button for more than 2 seconds. The LED will fade off. When the LED is off, you can release the button
- **Brief Click:** The LED glowing pattern will be briefly altered

The power indicator LED in the above schematic (the only one visible when the device's cover is on) is controlled by Re.

There are several other LEDs present on rigid:

- **Power Indicator LEDs** — When the board is powered they light up for about 5s.
 - On USB, the 5s counter starts the moment the cable is plugged
 - On battery power, the count starts when the button is pressed
- **Execute (Ex) LEDs** — Has green LEDs. During normal operation, the green LEDs will be flashing

3- GUI

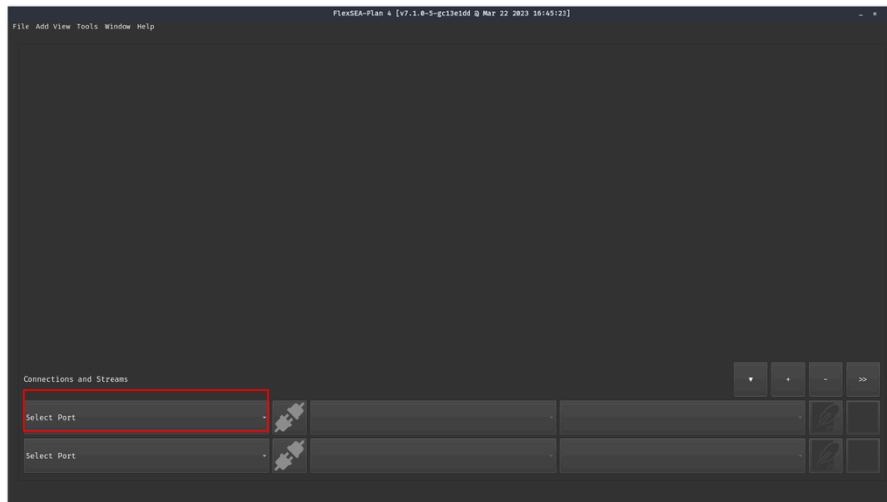


The Dephy GUI (Graphical User Interface) allows data visualization, data logging, and controller tuning.

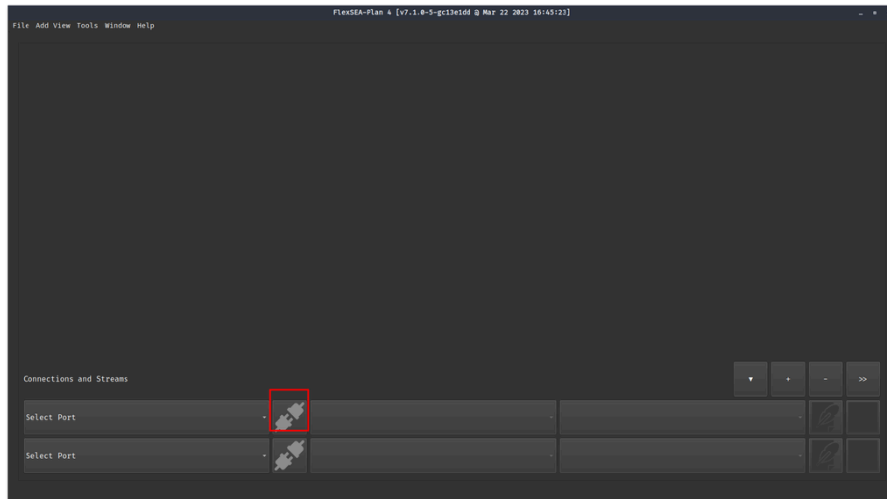
3.1- GUI Connectivity

3.1.1 Connecting via USB

1. Power device on
2. Connect device to computer via USB cable
3. Identify Port (see below)
4. Select the identified port from the dropdown (pictured)



5. Click the connect button (pictured)



3.1.2 Finding the Device Port on Windows

Start → Device Manager → USB and COM Ports. Your device should be listed as a USB Serial Port or STM32 Device.

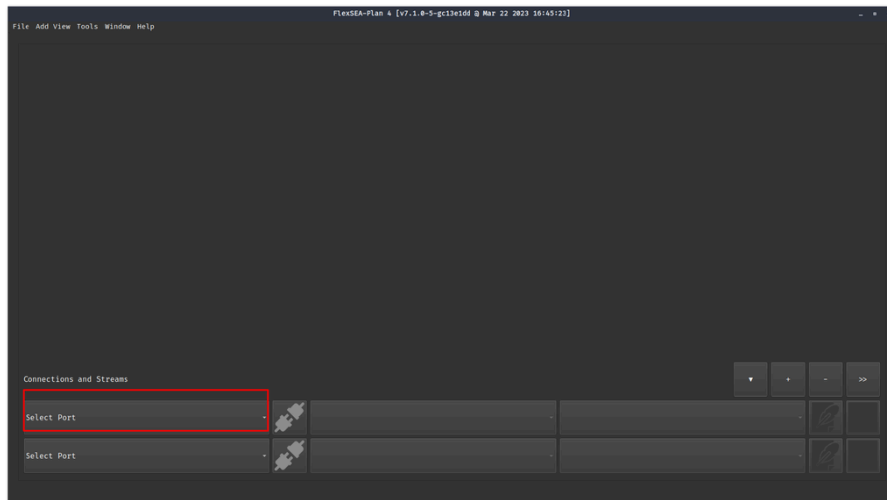
3.1.3 Finding the Device Port on Linux

On Linux, serial devices typically use a file akin to `/dev/ttyACM*`, so you can use the `ls` command with that pattern to find the port.

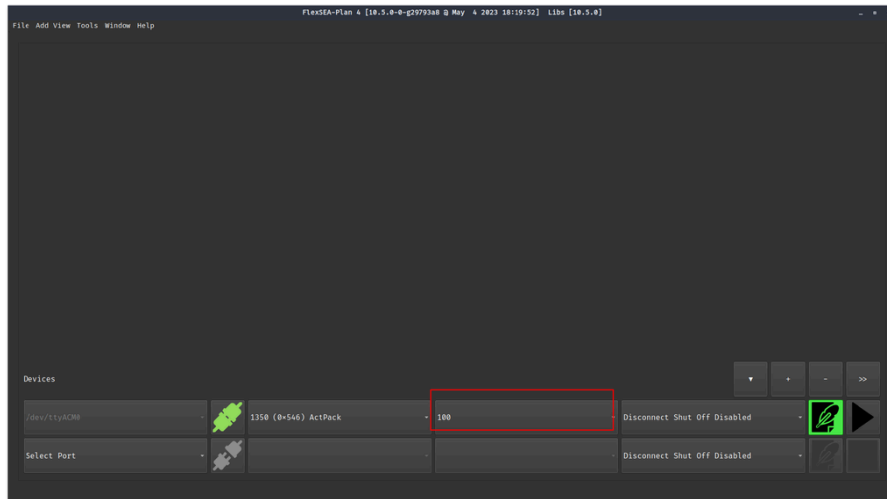
3.1.4 Connecting via Bluetooth

i This feature currently only works on Windows

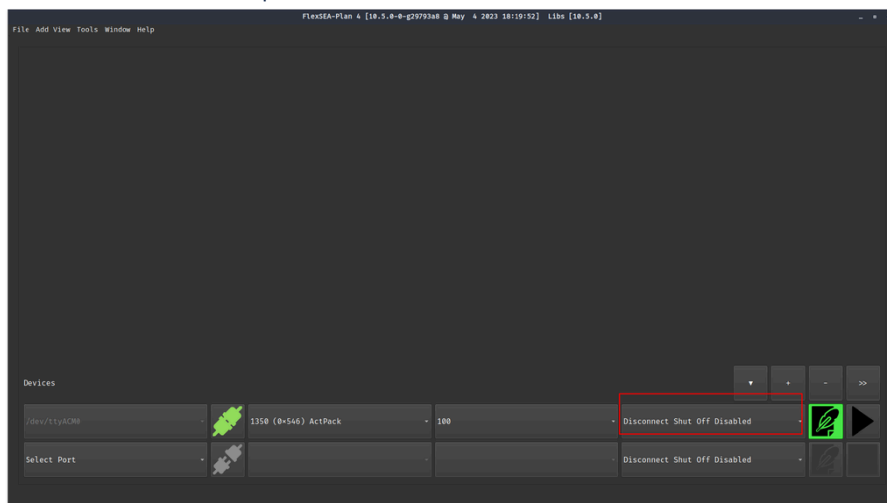
1. Power on the device
2. Find the device's Bluetooth ID (either provided with purchase or affixed as a label to the device)
3. Start → Bluetooth and Other Devices Settings → Add Bluetooth or Other Device → Bluetooth → Exo-SPP-ID, where ID is the Bluetooth ID you found in the previous step
4. Once paired, under Related Settings, select More Bluetooth Options
5. Select the COM Ports tab
6. Find the port name of the *outgoing* port for your device
7. Open the GUI and select the port from the previous step from the Select Port dropdown (pictured)



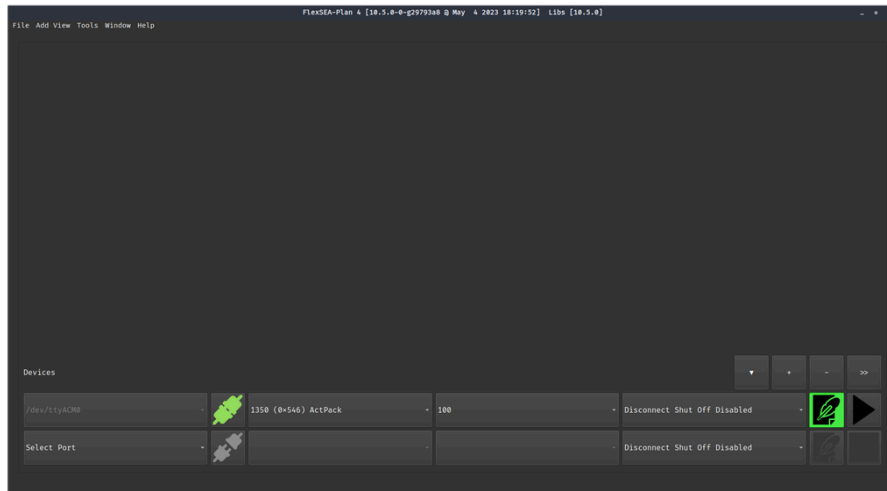
8. Set the frequency at which you would like data to be sent from the device to the computer (pictured)



9. Newer versions of the GUI also have a dropdown menu that allows you to set a safety timer, called the `heartbeat period`. There is a special type of message called a `heartbeat` that the computer periodically sends to the device to let it know that the connection is still active. If the amount of time specified by `heartbeat period` passes without the device receiving a `heartbeat` message, then the device will automatically shut off the motor. If this option is turned off, then the motor will continue to spin even if it becomes disconnected from the computer



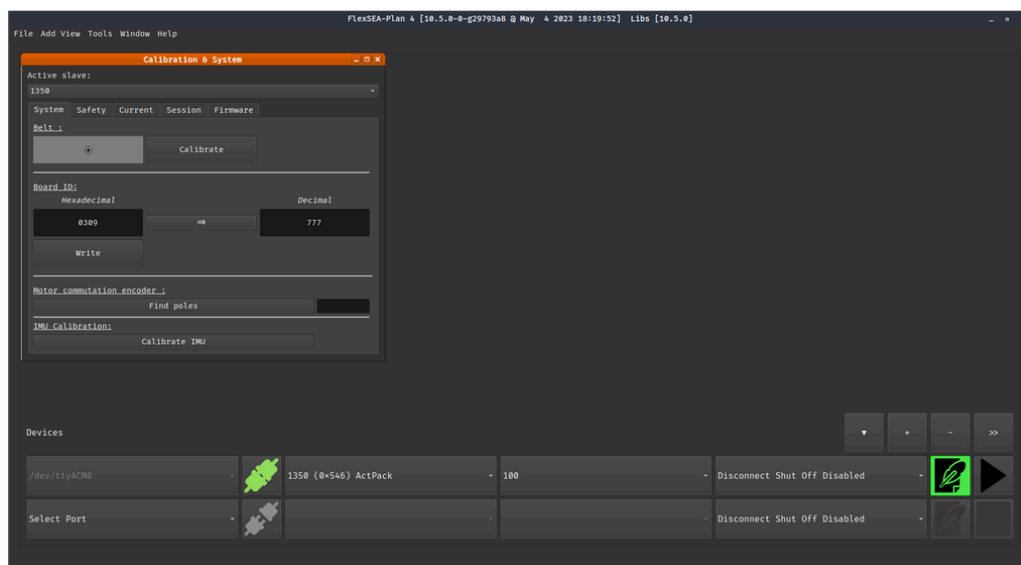
10. Click the connect button (pictured)



11. To disconnect, simply click the **Connect** button again

3.2 Configuring the Device

Once connected, you can calibrate and configure your device by going to **Tools** → **Calibration and System**.



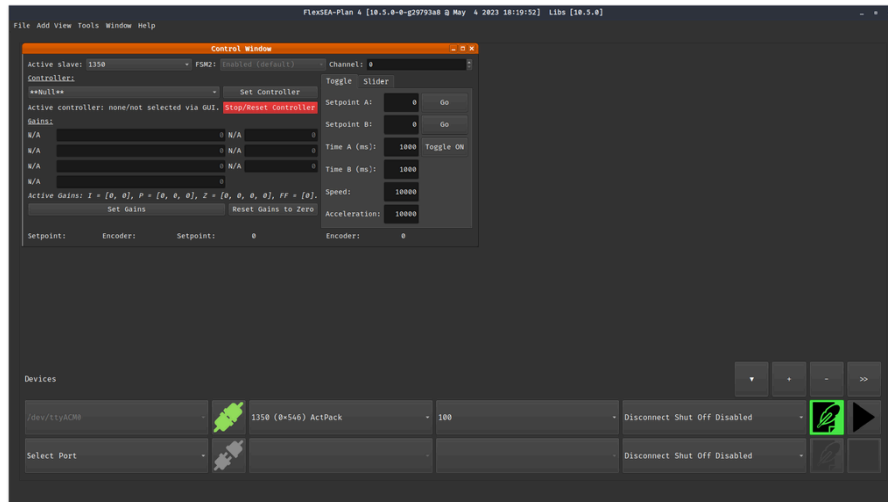
This window has several tabs:

- **System:** Lets you:
 - Calibrate the device belt (exo only)
 - Calibrate the IMU
 - Find poles — should only be needed if you've changed your device firmware. This causes the device to orient itself with the motor poles
 - Set a custom device ID
- **Safety:** From here you can:
 - Get the currently set UVLO
 - Set a new UVLO
 - Get the currently set I²t settings
 - Change your I²t settings
- **Current:** Allows you to set the battery current offset
- **Session:** Gives you statistics about your present run and allows you to save them to disk

- **Firmware:** Allows you to read the current firmware version for each microcontroller

3.3 Controlling the Device

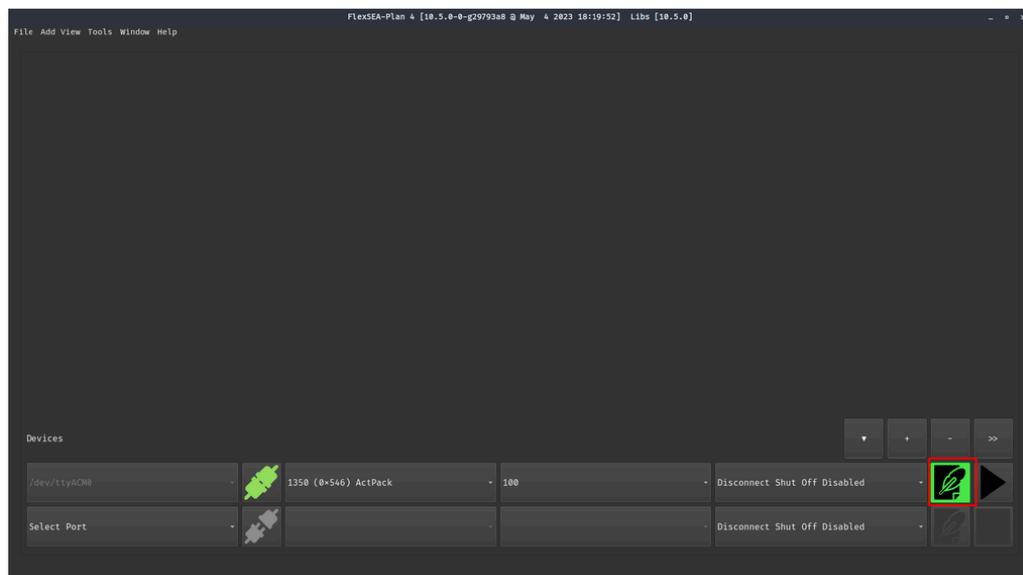
- **Tools** → **Control Loop** brings up the Control Window, from which you can send commands to the motor



- You can now select your desired controller from the **Controller** dropdown menu (then click the **Set Controller** button)
- Stop the controller via the **Stop/Reset Controller** button
- Set the gains via the **Set Gains** button
- Reset the gains to zero via the **Reset Gains to Zero** button
- Use the **Slider** and **Toggle** tabs to set and adjust your desired control values

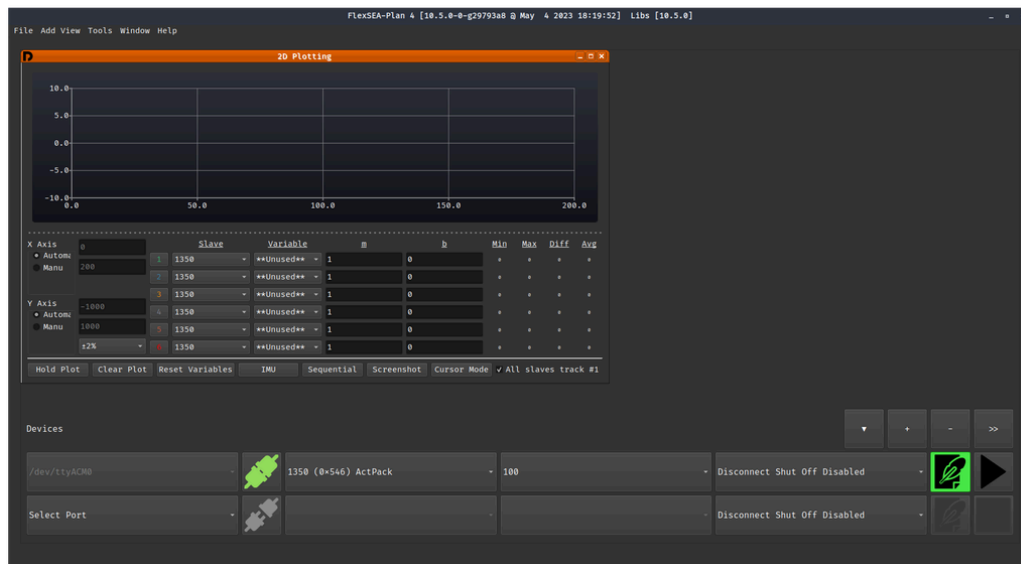
3.4 Logging Data

To log data, you must ensure that the logging button (pictured below) is enabled *before* connecting to your device. Once this is done, you will find two directories: **DataLog** and **DebugLog** inside the directory where your GUI executable is located. The **DataLog** directory is home to all the data streamed by your devices and the **DebugLog** stores all the debugging logs, which will typically only be needed if something goes wrong.



3.5 Visualizing Data

To visualize data: Add View → 2D Plot.



The x-axis of the plot will be a monotonically increasing index value, and you can select multiple fields from the dropdown menus to plot several quantities at once.

4- Python API

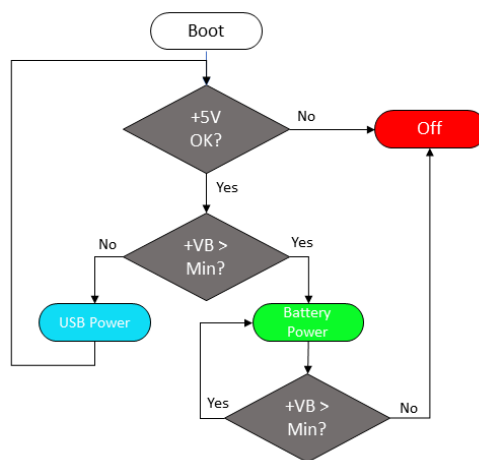
Dephy provides an open-source, publicly available Python API called FlexSEA. The purpose of this API is to allow researchers to write their own controllers, as well as for device users to programmatically access the device data for their own visualization and analysis. The most up-to-date documentation can be found in the [README here](#).

5- Programmable Safety Features

5.1 Low Voltage Protection (UVLO)

Dephy's devices come equipped with undervoltage-lockout (UVLO) protection. This causes the board to turn off if the battery becomes disconnected or discharges below the given minimum voltage. This protects the board from damage. It's important to note that the presence or absence of USB power does not affect whether this protection kicks in or when this protection kicks in. By default, the UVLO protection kicks in after 5000ms for +VB and 1000ms for +5V. You can change the UVLO threshold via the GUI, as described above or, if you have firmware >10, programmatically via the Python API via the `uvlo` property of the `Device` class.

i You should power cycle the device twice after changing the UVLO value to ensure that the threshold stored in non-volatile memory is used.



Not shown on the above diagram: a brief press will turn the LED magenta as a confirmation that the software has registered the user input.

i Known bug (06/2018): the motor current zero will be wrong if you power by USB before powering with a battery or power supply.

5.2 I²t Current Limit: Programmable Fuse

Ex and Re have a software I²t protection. This algorithm allows the user to select limits for average and peak currents (of a certain duration). It is essentially a programmable fuse that can be used to protect the circuits, but also systems built with FlexSEA and the users testing them.

i Only the values associated with Re can be changed. The values for Ex are hard-coded into the firmware for safety.

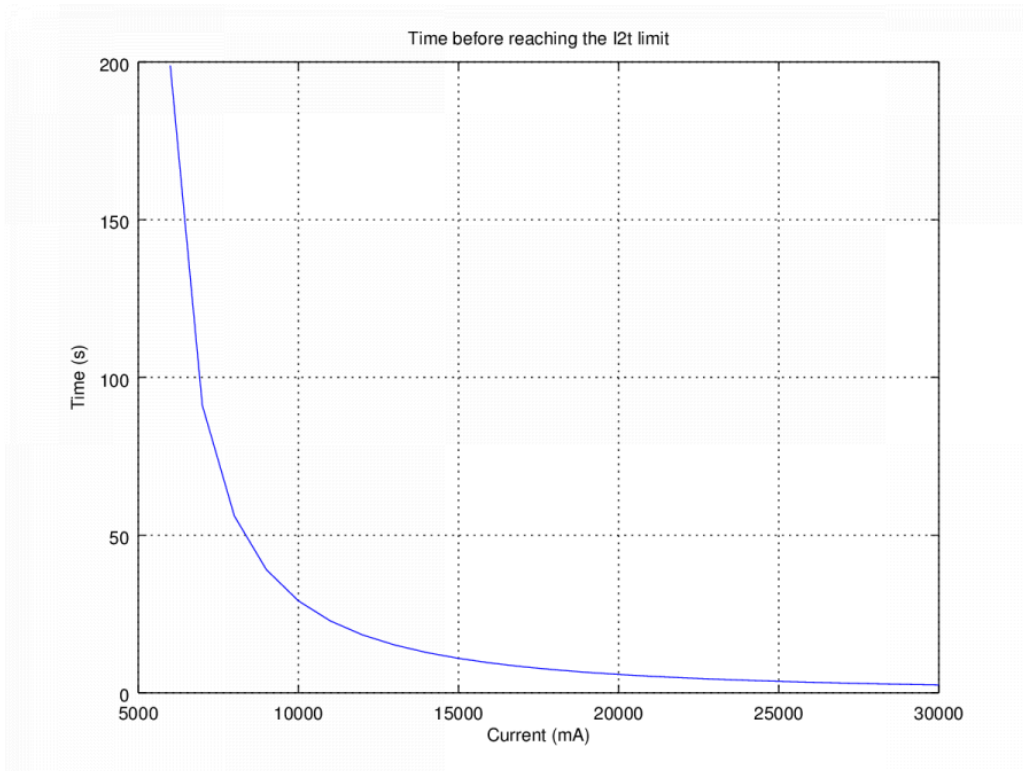
5.2.1 Theory of Operation

The protection is based on a current-time curve that is calculated from three key parameters:

- **Maximum average current:** if the current stays below this value, the protection will never engage
- **Current limit:** combined with the time, this determines the maximum pulse supported
- **Time at current limit:** combined with the current limit, this determines the maximum pulse supported

Similar to the curves found in fuse datasheets, this curve indicates how long a given current will be allowed to flow before the fuse interrupts the flow (blows).

For example, if we program a maximum average current of 5A and a current limit of 30A for 2.5s, we obtain the following curve:

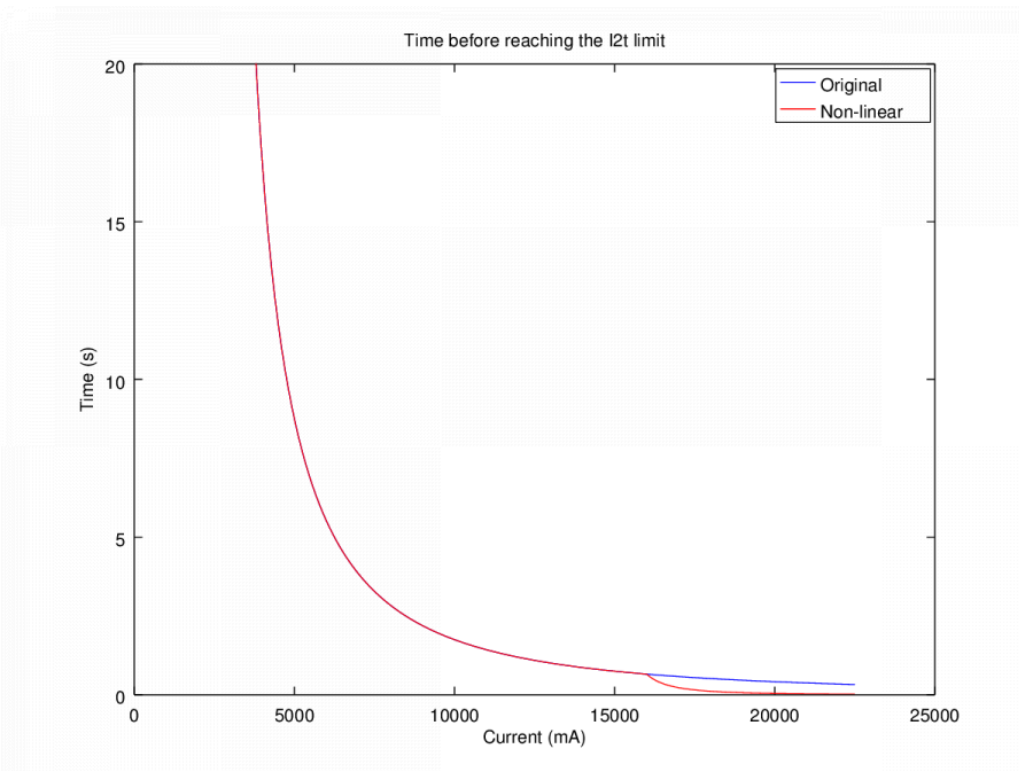


- 5A is on an asymptote: infinite time
- If 30A flows for more than 2.5s, the protection will engage
- If the pulse is 10A, the trip time will be around 30s

5.2.2 Non-linearity

The purpose of the I^2t algorithm is to model thermal dissipation in the device to prevent dangerous conditions. Due to the power of two, peaks can lead to significant heating. The current sensors used by FlexSEA have limits ranging from 5A to 30A, depending on the hardware version. While they will tolerate peaks above their maximum, they will only report up to their max. Using a 30A sensor as an example, if our current is 42A the energy is doubled versus 30A, yet the sensor will report 30A. An optional non-linearity was added to deal with those conditions. In other words, if you get close to the sensor's maximum range, this non-linear tweak to the model allows you to artificially increase the sensor's sensitivity to protect your hardware.

As can be seen in the plot below, below the threshold the algorithm behaves the same as it does without the non-linearity. Above the threshold (set at 16A in this example); however, we can see that the protection trips faster with the non-linearity.



5.2.3 Calculating and Programming

To minimize the processor load, we convert the user parameters to constants on a PC. The embedded firmware uses the pre-calculated constants.

The constants can be obtained from the Plan GUI, as described above, or from this Matlab/Octave script [i2tmatlab.zip](#). The GUI is simpler, but Octave outputs a plot that can help you select the optimal parameters.

The user parameters are:

- **Maximum average current (mA):** if the current stays below this value, the protection will never engage
- **Current limit (mA):** combined with the time, this determines the maximum pulse supported
- **Time at current limit (s):** combined with the current limit, this determines the maximum pulse supported
- **Non-linear threshold (mA):** inflection point for the non-linearity. Must be between the maximum average current and the current limit.
- **Enable non-linearity:** check (set to 1) if you want that feature, uncheck (set to 0) if not
- **Shift:** conversion from mA to 'ticks'. The algorithm uses 8-bit values (0-255) while the current sensor is rated with real current in mA (ex.: 30000mA). We need to scale down the maximum current accordingly. If your hardware is equipped with a $\pm 30A$ sensor ($\pm 30,000mA$) the ratio between 30000 and 256 is 117. For maximum efficiency we use a bit shift instead of a division. The closest (and higher) shift is 7, equivalent to a division by 128. For ActPack users this value will not need to be changed: use the default 7.

5.2.4 How to choose the right values

FlexSEA boards such as FlexSEA-Rigid (used by Exo and ActPack) have a surface-mount fuse to protect the board and the user against extreme events. The maximum current supported by that fuse (typically 15A) is much higher than the maximum current needed for most wearable robotics applications. For a given application, aim to use the lowest current possible.

For example, if you know that your average current is 1A (pro tip: do a quick experiment and look at the average current in the logs) you may decide to pick 1.5A: enough margin to account for errors. If you backtrack your maximum torque to require 10A at the battery for 100ms, use 11A for 150ms.

These values will make the programmable fuse trip as soon as your system goes past those limits (control algorithm bug, stalled motor, etc.) When that happens, the circuit will turn off.

5.2.5 Pre-computed values ($\pm 30A$ sensor)

2.5A continuous, 15A for 1s, non-linearity at 22.5A:

```

1 I2C_SHIFT =      7.0000e+000 (7)
2 I2T_LEAK =      381.4697e+000 (381)
3 I2T_LIMIT =     133.5144e+003 (133514)
4 I2T_NON_LINEAR_THRESHOLD = 175.7812e+000 (176)
5
6 Time at given current (mA):
7 CURR =      25.0000e+003
8 time =     353.5354e-003
9 timeNL =     87.7193e-003

```

10A continuous, 15A for 1s, non-linearity at 20A:

```

1 I2C_SHIFT =      7.0000e+000 (7)
2 I2T_LEAK =      6.1035e+003 (6104)
3 I2T_LIMIT =     76.2939e+003 (76294)
4 I2T_NON_LINEAR_THRESHOLD = 156.2500e+000 (156)
5
6 Time at given current (mA):
7 CURR =      25.0000e+003
8 time =     238.0952e-003
9 timeNL =     22.6244e-003

```

5.2.6 Tips and Examples

The examples below are based on a Pocket board with a $\pm 5A$ sensor, 1.5A continuous and 5A 500ms limit.

```

1 Program these values in i2t-current-limit.h:
2 =====
3 I2C_SCALE_DOWN_SHIFT = 5.0000e+000
4 I2T_LEAK =            2.1973e+003
5 I2T_LIMIT =           111.0840e+003
6 I2T_WARNING =          88.8672e+003
7 Time at 1.6A:
8 time =                36.6935e+000


```

The standby current gets added to the load current. On a 30A board, that's negligible, but on a $\pm 5A$ board with a low limit that can be significant. As you can see, adding 100mA to the 1.5A load will make the protection trip after 37s instead of infinity.

Why does my board turn off with 1A for 900ms and 5A for 100ms? $1^2 \times 0.9 + 5^2 \times 0.1 = 0.9 + 2.5 = 3.4$, $\sqrt{3.4} = 1.84A$ DC. Just the pulse is equivalent to 1.58A DC, enough to kill it.

How long do I need to wait between 5A 500ms pulses (assuming 0A the rest of the time)? $25 \times 0.5 = 12.5$, and our limit is 1.5. $1.5/12.5 = 12\%$ duty cycle, so you need to wait at least 4.17s ($0.5s/0.12$) before the next pulse.

6- Bootloading

 Firmware version 7.2.0 is the latest long-term support release. FASTER customers should not update beyond this version unless given explicit instruction to do so from Dephy.

6.1 Bootloader CLI

Dephy provides an open-source, publicly available bootloading tool written in Python. You can find the bootloader [here](#). To install and use the bootloader, you should follow the instructions provided in the `README` contained in the bootloader repository.

6.2 Programming Directly

Occasionally, it is necessary to program your device directly, such as when something goes wrong using the bootloader CLI and the device becomes bricked.

i You should bootload the microcontrollers in the same order as the sections listed below.

6.2.1 Programming Regulate

i This can only be done on Windows because the PSoC software only works on Windows

You will need:

- A `.hex` file — you will most likely need to reach out to Dephy to obtain this file
- The [PSoC Programmer](#) software. This is also installed when installing PSoC Creator.
- The PSoC programmer hardware:
 - [MiniProg3 \(Digikey #428-2975-ND\)](#)
 - [MiniProg4 \(Digikey #428-4713-ND\)](#)
- A programming adapter:
 - An FFC adapter (pictured) — provided by Dephy
 - If your board is newer (version 4.1+), then it should have pogo-pin slots, which should work with the standard adapter that comes with the PSoC programmer hardware



The procedure:

1. Make sure the device is not connected to a battery, power supply, or USB power
2. Connect the MiniProg device to your computer through an isolator
3. Open the PSoC Programmer software
4. `File` → `File Load` then select the `Regulate .hex` you wish to flash
5. Connect the adapter to the device
 - a. If using the FFC adapter, you will slide it into the FFC port marked `Re` with the gold contacts facing down towards the board
 - b. If using the pogo-pins, they will go in the set of pogo-pin slots marked `Re`
6. Click on the power button in the PSoC Programmer GUI to have the programmer provide power to the board. This is necessary since `Regulate` controls the board's power and gets shut off during the bootloading process, so we need to have the programmer provide power to keep things running
7. Click on the Program button (down arrow) in the PSoC Programmer GUI to start the flash

6.2.2 Programming Execute

i This can only be done on Windows because the PSoC software only works on Windows

You will need:

- A `.hex` file — you will most likely need to reach out to Dephy to obtain this file
- The [PSoC Programmer](#) software. This is also installed when installing PSoC Creator.
- The PSoC programmer hardware:
 - [MiniProg3 \(Digikey #428-2975-ND\)](#)
 - [MiniProg4 \(Digikey #428-4713-ND\)](#)
- A programming adapter:
 - An FFC adapter (pictured) — provided by Dephy (same one used to flash Re)
 - If your board is newer (version 4.1+), then it should have pogo-pin slots, which should work with the standard adapter that comes with the PSoC programmer hardware



The procedure:

1. Make sure the device is not connected to a battery or power supply; only USB power
2. Connect the MiniProg device to your computer through an isolator
3. Open the PSoC Programmer software
4. `File` → `File Load`, then select the Execute `.hex` you wish to flash
5. Turn on the device
6. Connect the adapter to the device
 - a. If using the FFC adapter, you will slide it into the FFC port marked `Ex` with the gold contacts facing down towards the board
 - b. If using the pogo-pins, they will go in the set of pogo-pin slots marked `Ex`
7. Click on the Program button (down arrow) in the PSoC Programmer GUI to start the flash
8. Power cycle the device

6.2.3 Programming Manage You will need:

- A `.hex` file — you will most likely need to reach out to Dephy to obtain this file
- [STM32 ST-LINK utility \(STSW-LINK004\)](#) software
- [ST-Link v3 \(Digikey\)](#).
- A programming adapter:
 - An FFC adapter (pictured) — provided by Dephy
 - If your board is newer (version 4.1+), then it should have pogo-pin slots, which should work with the standard adapter that comes with the ST-Link programmer



The procedure:

1. Make sure the device is not connected to a battery or power supply; only USB power
2. Connect the ST-LINK device to your computer through an isolator
3. Open the STM32-ST-Link Utility Software
4. **File** → **Open File**, then select the Manage **.hex** you wish to flash
5. Turn on the device
6. Connect the adapter to the device
 - a. If using the FFC adapter, you will slide it into the FFC port marked **Mn** with the gold contacts facing down towards the board
 - b. If using the pogo-pins, they will go in the set of pogo-pin slots marked **Mn**
7. Type **CTRL+SHIFT+P** on your computer to open up the flash dialog
8. Click **Start**
9. Power cycle the device

7- Units

The following table is for **ActPack** firmware. ExoBoot walking controller firmware may differ, and documentation will be provided to customers upon shipment.

Variable	Full Name	Units	Notes
Timestamp	Reception time	ms	Not filtered
State_time	Internal time	ms	Not filtered
Accel X, Y, Z	Accelerometer, 3 channels	bits	Conversion factor of 8192 LSB/g, not filtered
Gyro X, Y, Z	Gyroscope, 3 channels	bits	Conversion factor of 32.8 LSB/DPS (DPS = degrees per second), not filtered
Mot_ang	Motor angle	ticks	Not filtered
Mot_vel	Motor velocity	deg/sec	See Note 1
Mot_acc	Motor acceleration	rad/s ²	See Note 1
Mot_curr	Motor current	mA	Internally filtered, high bandwidth
Mot_volt	Motor voltage	mV	Not filtered
Batt_volt	Battery voltage	mV	Analog: 1st order RC, fc = 3.56kHz. Digital: moving average of 32 samples, equivalent to 138Hz.
Batt_curr	Battery current	mA	Analog filter, 884 Hz
Temp	Temperature	°C	Sampled at 10kHz, moving average of 16 (equivalent to 277Hz)
Genvar 0–9	Generic Variable used for Dephy internal development		
Ank_ang	Ankle angle	deg * 100	Not filtered

Ank_vel	Ankle velocity	deg/sec *10	Sampled at 10kHz, moving average of 16 (equivalent to 277Hz)
Step_energy	Step energy	mJ	Integrated positive power, not filtered

⚠ *Mot_vel* and *Mot_acc* use custom C filters optimized for speed. We suggest calculating motor velocity and motor acceleration in post-processing if you need them and would like to avoid lag.

8- Gains

i The gains listed here are specific to an ActPack with rigid 4.1B being used in table-top experiments. They are provided as examples only to give you an idea of the scale of values used. The correct gains depend on your application and your system, and will most likely require experimentation on your part to determine.

The feed-forward (FF) gain ranges from 0 (0%) to 128 (100%). Setting FF to 128 may cause instabilities in certain scenarios. Only increase it if the performance is really needed

The current units reflect the Q-axis current in Field Oriented Control

The feed-forward gain only applies to the current and impedance controllers, the differential gain only applies to the position controller, the stiffness gain only applies to the impedance controller, and the damping gain only applies to the impedance controller

8.1 Current Control Gains

- Proportional gain (k_p): [0, 80], 40 recommended
- Integral gain (k_i): [0, 800], 400 recommended
- Feed Forward gain (ff): [0, 128], 128 recommended

8.2 Position Control Gains

- Proportional gain (k_p): [0 1000]
- Integral gain (k_i): [0 1000]
- Differential gain: [0 1000], 0 recommended

8.3 Impedance Control Gains

Proportional gain (k_p): 40 Integral gain (k_i): 400 Stiffness (K): [0 2000] Damping (B): [0 20000], begins to vibrate around 15000 Feed Forward gain (ff): 128

The following pairs represent K and B pairs that result in 0 overshoot:

Commanded motor current = $A \times vel \times B + C \times K \times (\text{delta})$

- vel = motor velocity [degrees per second]
- delta = (setpoint - motor position) [motor clicks → 16384 clicks per rotation]
- A = 0.00028444
- C = 0.0007812

K	B
100	650
200	1200

300	1600
500	2500
1000	3500
2000	5500

Motor velocity units: degrees per second

Impedance Setpoint: motor clicks

Motor Position: motor clicks

Motor click: 16383 clicks per full rotation

Motor Torque (mNm) = Commanded current * 0.146

B & K were sized to offer sufficient precision within an integer-based embedded system. Please refer to the units above to convert them to physical units if necessary.

Date	November 21, 2024
Revision	C_0001_UG_0004_V01_FLEXSEASTART
Created by	Carlos Asmat
Reviewed by	Sarah Gardner  <small>Sarah Gardner (Nov 21, 2024 11:41 EST)</small>
Approved by	Matt Mooney 
Purpose	Start Up Guide for Dephy Flexsea that has been reviewed and approved for release