# Maestro servo controller library

***Release 0.1***

**Alex Dutton <maestro@alexdutton.co.uk>**

**Jan 09, 2021**

# CONTENTS:


**1 Installation** **1**
- 1.1 Maestro USB device access on Linux . . . 1

**2 Getting started** **3**
- 2.1 Using a channel as a servo . . . 4

**3 API reference** **5**
- 3.1 Channels . . . 6
- 3.2 Enums . . . 6
- 3.3 Exceptions . . . 8

**4 The Maestro USB control protocol** **9**

**5 Indices and tables** **11**

**Python Module Index** **13**

**Index** **15**

CHAPTER

ONE

# INSTALLATION

You can install this library from PyPI:

```
$ pip install maestro-servo
```

## 1.1 Maestro USB device access on Linux

When you connect a Maestro (or any USB device), udev is used to manage access permissions. There are no rules for Maestro devices distributed with udev, so by default only the root user can access the device. Before you can access your Maestro device as a normal user, you will need to create some udev rules.

You'll need a file in */etc/udev/rules.d/*, e.g. */etc/udev/rules.d/10-pololu.rules* containing the following:

```
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="1ffb", ATTRS{idProduct}=="0089",␣
↪TAG+="uaccess"
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="1ffb", ATTRS{idProduct}=="008a",␣
↪TAG+="uaccess"
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="1ffb", ATTRS{idProduct}=="008b",␣
↪TAG+="uaccess"
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idVendor}=="1ffb", ATTRS{idProduct}=="008c",␣
↪TAG+="uaccess"
```

This ensures that the currently-logged-in user on a desktop machine is able to access the device.

Once this is done, you're all set.

CHAPTER

TWO

# GETTING STARTED

Before you can get started coding, ensure you've performed all the steps in *Installation*, including the step about configuring udev.

You should also read the Pololu Maestro Servo Controller User's Guide, as the concepts documented there correspond to functionality exposed through this Python module.

Assuming you're getting started with a single controller, plug it in and try the following:

```
from maestro import Maestro

maestro = Maestro.get_one()

print("Maestro:", maestro)
print("Channel count:", maestro.channel_count)
```

You should get something like:

```
Maestro: <maestro.device.MiniMaestro "00262773">
Channel count: 12
```

The `maestro.Meastro` instance provides a list-like interface for accessing channels, so we can iterate over all the channels, or get one by index (starting at channel 0):

```
print("All channels:", list(maestro))
print("The third channel:", maestro[2])
```

And the result:

```
All channels: [<maestro.channel.Channel 0: Servo>, <maestro.channel.Channel 1: Servo>,
↪ <maestro.channel.Channel 2: Servo>, <maestro.channel.Channel 3: Servo>, <maestro.
↪channel.Channel 4: Input>, <maestro.channel.Channel 5: Input>, <maestro.channel.
↪Channel 6: Input>, <maestro.channel.Channel 7: Input>, <maestro.channel.Channel 8:␣
↪Input>, <maestro.channel.Channel 9: Input>, <maestro.channel.Channel 10: Input>,
↪<maestro.channel.Channel 11: Input>]
The third channel: <maestro.channel.Channel 2: Servo>
```

## 2.1 Using a channel as a servo

**Note:** To be continued...

CHAPTER

# THREE

# API REFERENCE

**class** maestro.**Maestro**(*dev: usb.core.Device*, *timeout=5000*)

A Maestro servo controller.

You shouldn't instantiate this class directly. Instead you should use one of the following class methods:

- *Maestro.get_all()*
- *Maestro.get_one()*
- *Maestro.get_by_serial_number()*

**property channel_count**

The number of available channels on this servo controller.

**classmethod for_device**(*dev: usb.core.Device*, ***kwargs*) → *maestro.Maestro*

Returns a Maestro instance for the given pyusb Device.

**classmethod get_all**() → Iterable[*maestro.Maestro*]

Returns an iterator over all connected Maestro devices.

**classmethod get_by_serial_number**(*serial_number*)

Get a currently-connected Maestro device by its serial number.

**Raises** ***maestro.exceptions.NoMaestroAvailable*** – if no Maestro is available.

**classmethod get_one**() → *maestro.Maestro*

Get a currently-connected Maestro device.

If more than one is connected, it is undefined as to which device is returned.

**Raises** ***maestro.exceptions.NoMaestroAvailable*** – if no Maestro is available.

**property serial_number**

The self-reported serial number for this device.

You may use this later as an argument to *Maestro.get_by_serial_number()* to ensure you connect to the same device again, in the case where multiple Maestro devices are connected.

## 3.1 Channels

**class** maestro.channel.**Channel**(*maestro:* maestro.Maestro, *index: int*, *mode: ChannelMode*)

A channel on a Maestro servo controller.

**property mode**

The mode of this channel.

**Type** *ChannelMode*

**property neutral**

This option specifies the target value, in microseconds, that corresponds to 127 (neutral) for 8-bit commands.

**property position**

Where the servo controller believes this servo to be currently positioned.

Note that this is where the servo is currently being told to be, which will not necessarily be the target if speed and/or acceleration are non-zero.

The position is specified in milliseconds (ms).

**Type** int

**property target**

The current target position, in ms.

**Type** int

**property value**

The value read by this input, in the range [0, 1023].

The inputs on channels 0–11 are analogue: their values range from 0 to 1023, representing voltages from 0 to Vcc V. The inputs on channels 12–23 are digital: their values are either exactly 0 or exactly 1023.

**Type** int

## 3.2 Enums

These are used as parameters and return values for various methods.

**class** maestro.enums.**ChannelMode**(*value*)

Channel mode.

**Servo = 0**

The channel is a servo.

**ServoMultiplied = 1**

This value is defined within the Maestro Control Centre, but unused.

It is not recommended to use this channel mode, as its purpose is unclear.

**Output = 2**

The channel is an output.

Output channels are controlled with a value in the range [0, 1023], which maps onto the range [0, Vcc] volts.

**Input = 3**
The channel is an input.

The value is in the range [0, 1), corresponding to the input voltage range [0, Vcc] volts.

**class** maestro.enums.**USCParameter**(*value*)
Constants for getting and setting Maestro parameters.

**Initialized = 0**

**ServosAvailable = 1**

**ServoPeriod = 2**

**SerialMode = 3**

**SerialTimeout = 6**

**ChannelModes0To3 = 12**

**ChannelModes4To7 = 13**

**ChannelModes8To11 = 14**

**ChannelModes12To15 = 15**

**ChannelModes16To19 = 16**

**ChannelModes20To23 = 17**

**ServoMultiplier = 26**

**ServoHomeBase = 30**

**ServoMinBase = 32**

**ServoMaxBase = 33**

**ServoNeutralBase = 34**

**ServoRangeBase = 36**

**ServoSpeedBase = 37**

**ServoAccelerationBase = 38**

**class** maestro.enums.**Request**(*value*)
Constants for Maestro USB control requests.

See *The Maestro USB control protocol* for more details about control requests.

**GetRawParameter = 129**

**SetRawParameter = 130**

**GetVariablesMicroMaestro = 131**

**SetServoVariable = 132**

**SetTarget = 133**

**ClearErrors = 134**

**GetVariablesMiniMaestro = 135**

**GetStack = 136**

**GetCallStack = 137**

**SetPWM = 138**

**Reinitialize = 144**

**EraseScript = 160**

**WriteScript = 161**

**SetScriptDone = 162**

**RestartScriptAtSubroutine = 163**

**RestartScriptAtSubroutineWithParameter = 164**

**RestartScript = 165**

**StartBootloader = 255**

## 3.3 Exceptions

**exception** maestro.exceptions.**MaestroException**
Base class for all Maestro-related exceptions.

**exception** maestro.exceptions.**NoMaestroAvailable**
Exception for when a Maestro device was requested, but none is available.

CHAPTER

FOUR

# THE MAESTRO USB CONTROL PROTOCOL

**Note:** To be continued...

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## m

maestro.exceptions, 8

# INDEX


## C

Channel (*class in maestro.channel*), 6
channel_count() (*maestro.Maestro property*), 5
ChannelMode (*class in maestro.enums*), 6
ChannelModes0To3 (*maestro.enums.USCParameter attribute*), 7
ChannelModes12To15 (*maestro.enums.USCParameter attribute*), 7
ChannelModes16To19 (*maestro.enums.USCParameter attribute*), 7
ChannelModes20To23 (*maestro.enums.USCParameter attribute*), 7
ChannelModes4To7 (*maestro.enums.USCParameter attribute*), 7
ChannelModes8To11 (*maestro.enums.USCParameter attribute*), 7
ClearErrors (*maestro.enums.Request attribute*), 7

## E

EraseScript (*maestro.enums.Request attribute*), 8

## F

for_device() (*maestro.Maestro class method*), 5

## G

get_all() (*maestro.Maestro class method*), 5
get_by_serial_number() (*maestro.Maestro class method*), 5
get_one() (*maestro.Maestro class method*), 5
GetCallStack (*maestro.enums.Request attribute*), 7
GetRawParameter (*maestro.enums.Request attribute*), 7
GetStack (*maestro.enums.Request attribute*), 7
GetVariablesMicroMaestro (*maestro.enums.Request attribute*), 7
GetVariablesMiniMaestro (*maestro.enums.Request attribute*), 7

## I

Initialized (*maestro.enums.USCParameter attribute*), 7
Input (*maestro.enums.ChannelMode attribute*), 6

## M

Maestro (*class in maestro*), 5
maestro.exceptions
    module, 8
MaestroException, 8
mode() (*maestro.channel.Channel property*), 6
module
    maestro.exceptions, 8

## N

neutral() (*maestro.channel.Channel property*), 6
NoMaestroAvailable, 8

## O

Output (*maestro.enums.ChannelMode attribute*), 6

## P

position() (*maestro.channel.Channel property*), 6

## R

Reinitialize (*maestro.enums.Request attribute*), 7
Request (*class in maestro.enums*), 7
RestartScript (*maestro.enums.Request attribute*), 8
RestartScriptAtSubroutine (*maestro.enums.Request attribute*), 8
RestartScriptAtSubroutineWithParameter (*maestro.enums.Request attribute*), 8

## S

serial_number() (*maestro.Maestro property*), 5
SerialMode (*maestro.enums.USCParameter attribute*), 7
SerialTimeout (*maestro.enums.USCParameter attribute*), 7
Servo (*maestro.enums.ChannelMode attribute*), 6
ServoAccelerationBase (*maestro.enums.USCParameter attribute*), 7
ServoHomeBase (*maestro.enums.USCParameter attribute*), 7
ServoMaxBase (*maestro.enums.USCParameter attribute*), 7

ServoMinBase (*maestro.enums.USCParameter attribute*), 7
ServoMultiplied (*maestro.enums.ChannelMode attribute*), 6
ServoMultiplier (*maestro.enums.USCParameter attribute*), 7
ServoNeutralBase (*maestro.enums.USCParameter attribute*), 7
ServoPeriod (*maestro.enums.USCParameter attribute*), 7
ServoRangeBase (*maestro.enums.USCParameter attribute*), 7
ServosAvailable (*maestro.enums.USCParameter attribute*), 7
ServoSpeedBase (*maestro.enums.USCParameter attribute*), 7
SetPWM (*maestro.enums.Request attribute*), 7
SetRawParameter (*maestro.enums.Request attribute*), 7
SetScriptDone (*maestro.enums.Request attribute*), 8
SetServoVariable (*maestro.enums.Request attribute*), 7
SetTarget (*maestro.enums.Request attribute*), 7
StartBootloader (*maestro.enums.Request attribute*), 8

## T

target() (*maestro.channel.Channel property*), 6

## U

USCParameter (*class in maestro.enums*), 7

## V

value() (*maestro.channel.Channel property*), 6

## W

WriteScript (*maestro.enums.Request attribute*), 8