

# Controlling Numato Lab USB GPIO Modules using Python

# Table of Contents

## Contents

1. Introduction .....	3
2. Applicable products .....	3
3. Prerequisites .....	3
4. Fundamentals of communicating to the device through Python.....	3
4.1. Opening the serial port.....	4
4.2. Sending commands to the device.....	5
4.3. Reading a response from the device .....	5
4.4. Closing the port.....	5
5. Python sample code and result .....	6

## 1. Introduction

[Python](#) is a popular programming language. Python allows programmers to write applications effortlessly, to create scripts and applications for automation and other purposes. Python works on multiple operating systems including Windows, Linux, and Mac too. Python can work with Serial Ports when appropriate modules are installed, and this makes Python capable of communicating with Numato Lab's USB devices.

Numato Lab's USB GPIO modules are great products for controlling electrical and electronic devices remotely from a PC or Mobile Device over USB link. Ease of use and wider operating system compatibility are the primary goals behind this product's design. USB GPIO modules (and other GPIO modules as well) are primarily used for some sort of automation such as Industrial Automation and Factory Automation. This article discusses various aspects of using Python for writing scripts/applications to control Numato Lab's USB GPIO modules so that the reader will become familiar with the basic concepts with the help of examples.

## 2. Applicable products

- All Numato Lab USB GPIO Modules

## 3. Prerequisites

The reader is expected to be familiar with Python and the commands supported by the device. More information about the supported commands is available in the product user manual.

Python3 and pySerial module for Python must be installed on the target machine. More information on how to install these applications can be found in the links below.

- [How to install Python](#)
- [How to install pySerial](#)

Once the python3 is installed, the pip library is automatically installed, pySerial can be installed using the pip. Once all the above prerequisites are installed, we can start writing the Python code.

## 4. Fundamentals of communicating to the device through Python

Just like most other USB-based products from [Numato Lab](#), the USB GPIO devices present themselves as a simple Serial Port to the host machine. This serial interface allows the

device to be programmatically controlled by using APIs provided by the Operating System. Be it Windows, Linux, Mac, Android, or any other operating system, the behavior is the same. The only requirement is that the host operating system supports USB CDC devices, which almost all popular operating systems do.

This article uses the 'pySerial' package which is one of the popular packages and it offers some intuitive and easy-to-use APIs.

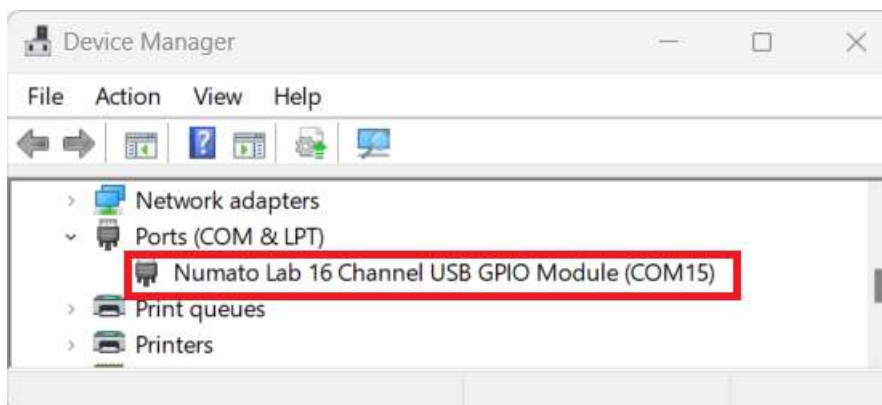
Interacting with a USB GPIO device programmatically can be broken down into the following steps.

1. Open the port corresponding to the device.
2. Send commands to the device. The commands are ASCII human-readable strings.
3. Read back the response from the device where applicable. The responses from the device are also human-readable ASCII strings.
4. Close the port when the operation is complete.

The following sections will discuss in detail each of these steps.

## 4.1. Opening the serial port

Before the port can be opened, the port name for the corresponding device must be located. On Windows, this can be usually done by visiting the Device Manager and looking up the port name manually. Here is the example image below, the port name for the GPIO module would be COM15. On Linux and Mac OSX, the attached devices should be visible in the /dev directory.



Once the port name is figured out, opening the port is very easy. The very first step is to load the Serial Port module and create an object that represents the serial port corresponding to the device. The following code does this. The exact value of the baud rate doesn't matter if it is a legal value. Also, use device names such as *COM15* on Windows and device node names such as */dev/ttyACM0* on Linux/Mac.

```
import serial
```

//On windows use the port name such as COM15 and on Linux/Mac, use the device node name such as /dev/ttyACM0

```
serPort = serial.Serial("COM15",baudrate = 9600, timeout=1)
```

## 4.2. Sending commands to the device

Once the port corresponding to the device is opened successfully, commands can be set to the device using the *write()* method provided by the serial port package. The *write()* method accepts a string contains the command and is sent to the device. Since the device expects the command to end with a Carriage Return (hex value 0x0A), it is important to add a "\r" at the end of all commands. Make sure the *Serial()* method is called on the port object and the port are open before calling the *write()* method. The *flushInput()* method discards all input buffer content.

```
serPort.flushInput ()  
serPort.write (b"ver\r")
```

Depending on the exact command sent, the device may or may not respond with some data. This data can be captured by reading the port immediately after sending the command.

## 4.3. Reading a response from the device

The serialport package provides functions like *read*, *readline* & *readuntil* to receive data over the serial port. The data returned is in bytes, converted to a string by using the *decode()* method. The result is printed using the *print()* method. For more details on how to use *read*, *readline* or *readuntil* refer to the [pySerial API](#).

```
response = serPort.read(20)  
print(response.decode())
```

## 4.4. Closing the port

Finally, a previously opened port can be closed by using the *close()* method.

```
serPort.close ()
```

## 5. Python sample code and result

The following complete Python code will open the port and send commands to the GPIO device to turn on GPIO 0 and turn it off after a few seconds. Any other command described in the product user manual can be sent to the device the same way.

```
#Import required libraries
import serial
import time

# Opening Serial port
serPort = serial.Serial("COM15",baudrate = 9600, timeout=1)

#Set GPIO #0 to High
serPort.flushInput()
serPort.write(b"gpio set 0\r")
response = serPort.read(20)
print(response.decode())

#Read GPIO #0 Status
serPort.flushInput()
serPort.write(b"gpio read 0\r")
response = serPort.read(20)
print(response.decode())

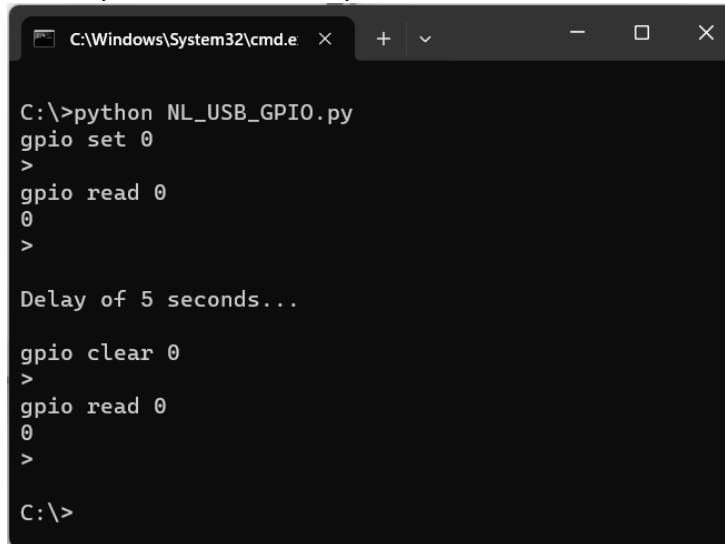
print("\nDelay of 5 seconds...\n")
time.sleep(5)

#Set GPIO #0 to Low
serPort.flushInput()
serPort.write(b"gpio clear 0\r")
response = serPort.read(20)
print(response.decode())

#Read GPIO #0 Status
serPort.flushInput()
serPort.write(b"gpio read 0\r")
response = serPort.read(20)
print(response.decode())

# Closing Serial port
serPort.close()
```

The image below shows the above script running on Windows. Please note the data returned by the device. The device echoes everything that is sent to it in addition to the command prompt and optional result of operation.



```
C:\Windows\System32\cmd.e x + v - □ ×  
  
C:\>python NL_USB_GPIO.py  
gpio set 0  
>  
gpio read 0  
0  
>  
  
Delay of 5 seconds...  
  
gpio clear 0  
>  
gpio read 0  
0  
>  
  
C:\>
```