8 Channel USB Relay Module

2732 views February 18, 2016 admin 0

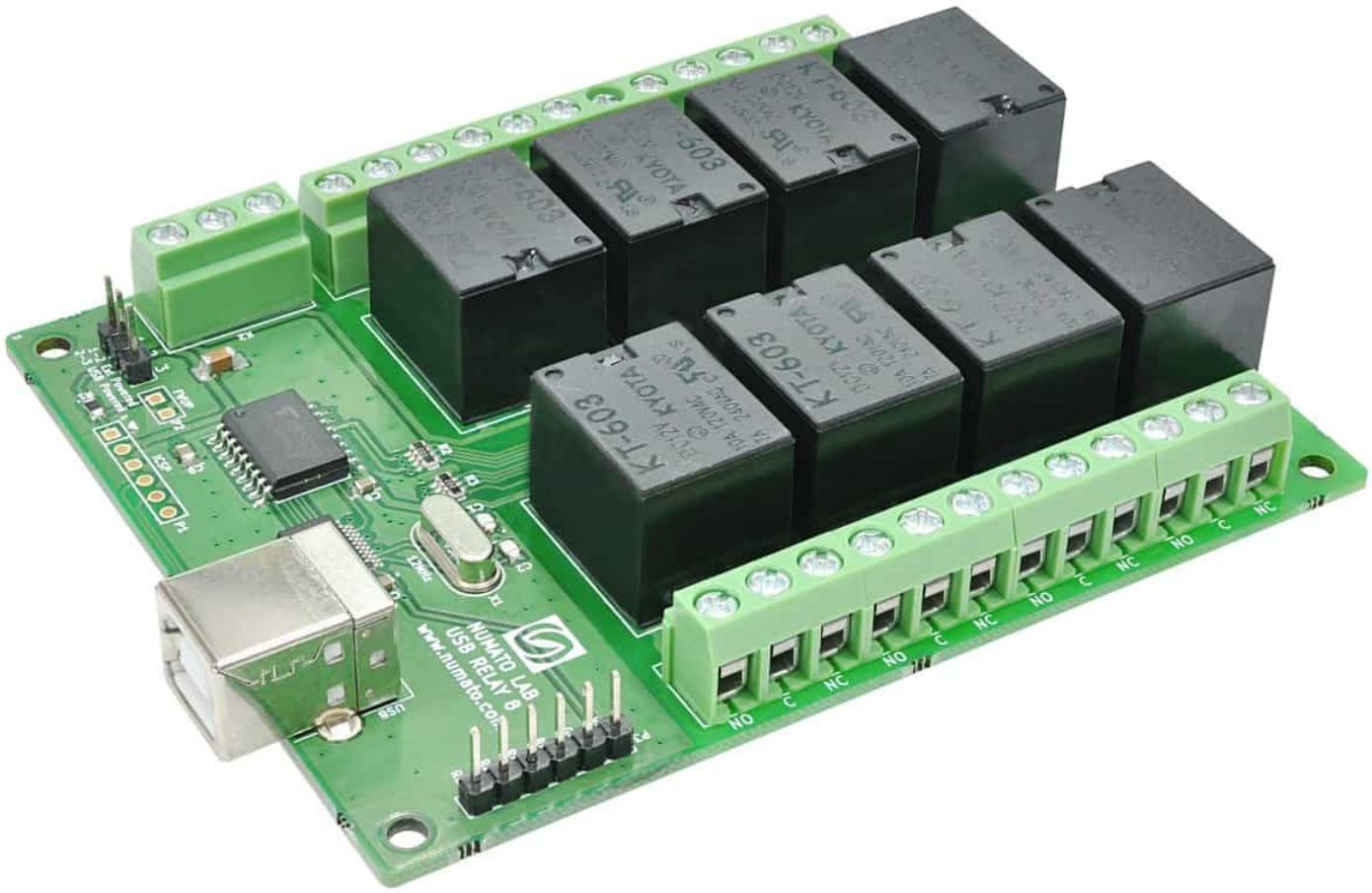
Search

- Sending Commands
  - The Command Set
  - Using the Serial Terminal Emulator Software On Windows
    - HyperTerminal
    - TeraTerm
  - Using Serial Terminal Software on Linux
    - GNU Screen
    - PuTTY
  - Using Serial Terminal Software on Mac OS X
    - Using Screen Command To Control Devices On Mac OSX
    - Using CoolTerm To Control Devices On Mac OSX
    - Using goSerial To Control Devices On Mac OSX
  - Using Serial Terminal Software on Android
    - USB Serial Terminal Lite
  - Using Custom Program and Script
    - C/C++ on Windows
      - Visual C++
    - JavaScript
      - Node.JS
- Additional Information
  - Using relay modules with inductive

loads

- + Technical Specifications
- + Frequently Asked Questions (FAQs)
- + Mechanical Dimensions

Introduction



(https://docs.numato.com/wp-content/uploads/2016/02/usbrelay8.jpg)

Numato Lab's 8 Channel USB Relay Module is a great product 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. Built in USB to serial conversion allows the module to be used without any USB specific knowledge. This simplicity allows use of off-the-shelf Terminal Emulation programs such as Hyper Terminal and PUTTY for controlling the module with a simple set of human readable commands. For power users, this module can be controlled by writing programs in various programming languages.

Features:

- 8 Mechanical Relay with contact rating up to 10A
- USB interface with CDC support. As easy as using a serial port, no USB knowledge required
- Relay contacts available on easy to access screw terminals

- Digital circuitry can be powered from USB or external power supply
- Can be controlled by using standard serial console applications or custom applications
- No vendor specific libraries or APIs required

Some of the possible uses of this module include

- Home Automation
- Lighting Control
- Garden Equipment Control
- Industrial Automation
- Test Fixtures
- DIY and Hobby

This product is compatible with the following operating systems:

- Windows XP and later versions (Windows 7, 8/8.1, 10 and future versions)
- Windows 7 Embedded and later
- Linux
- Mac OS X
- Android
- Or any other operating system that supports USB CDC devices.

And these are some of the languages that can be used for programming:

- C/C++
- Visual Basic (VB6, VB2008, VB2010 express and other editions)
- Visual Basic for Applications (Microsoft Office VBA)
- Perl
- Python
- JAVA
- Android
- Javascript (Node.js)
- And many more…

A complete list of sample code is available here (https://github.com/numato/samplecode/tree/master/RelayAndGPIOModules/USBRelayAndGPIOModules).

This product has eight on board relays and associated drivers capable of controlling variety of devices including lamps, motors, locks etc… (Please see recommendations for using this product with inductive loads elsewhere in this document). The module communicates with host PC over full speed USB link. When connected to PC, the module will appear as a serial port in Windows Device Manager (or a serial tty device in Linux and Mac).

How to Use 8 Channel USB Relay Module

Using this product is very easy, thanks to the human readable and easy to use command set and the USB CDC interface that allows the device to be used with most readily available serial terminal software such as Tera Term or HyperTerminal. This document has more information about using this device with the following Serial terminal software. But in no way limited to this software though.

- Windows
    - HyperTerminal
    - Tera Term
- Linux
    - GNU Screen
    - PuTTY
- Mac OS X
    - Screen Command
    - CoolTerm
    - goSerial

Using this product involves the following simple steps.

1. Connect the device to a USB port on the host system
2. Install driver (Only needed for Windows. Driver available for download on the product page.)
3. Open the COM port corresponding to the device using a Serial Terminal software
4. Enter commands (Very similar to entering commands at DOS prompt or Bash prompt)
5. Optionally write a script or application to automate your task (More info on custom programming/scripting is available in this document. Also we have a large set of sample code (https://github.com/numato/samplecode/tree/master/RelayAndGPIOModules/USBRelayAndGPIOModules) to help you with)

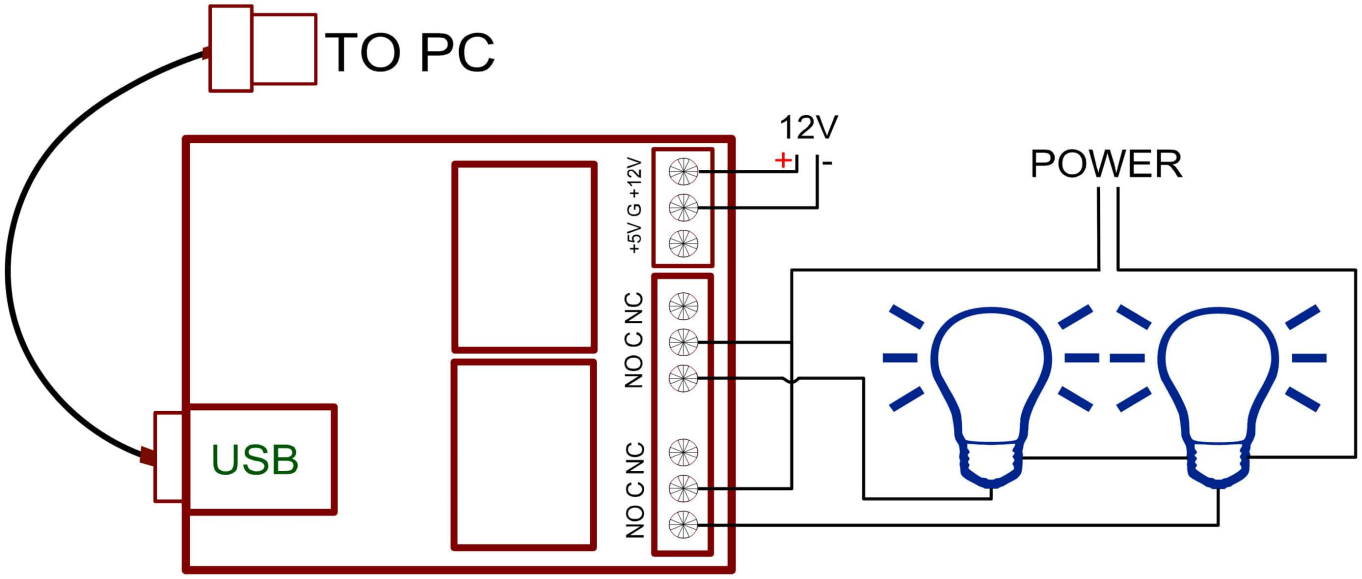
All aspects of the above steps are covered in the following sections including step by step demonstration.

Components/Tools Required

Along with the module, you may need the items in the list below for easy and fast installation.
1. USB A to Mini B cable.
2. Medium size Philips screw driver.
3. 12V/ 24V Power Supply.

Connection Details

> IMPORTANT! Please exercise utmost caution while working with electrical mains or other high voltages. Failure to comply with safety regulations may result in injury and or death.



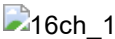
(https://numato.com/blog/wp-content/uploads/2016/02/2ChannelUSBRelayConnectionDiagram.jpg)

Connection Diagram

The picture above shows basic connection diagram that can be used in most of the situations. The connection diagram is same for both AC and DC loads. Please make sure to use a freewheeling diode or snubber circuit if the load is inductive. More details about using inductive loads is available elsewhere in this document. Use a USB A to Mini B cable to connect with a PC. It is important to make sure that the wires used to connect loads are sufficiently rated to handle expected load current. Exercise caution while working with high voltages. Short circuits can cause damage to the module and the PC. The following sections identify individual connections in detail.

Suggest edit

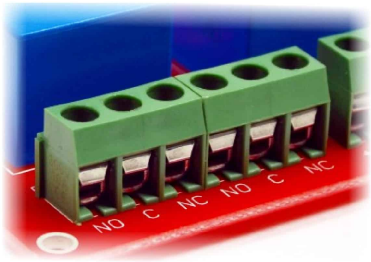
USB Interface



The on board full speed USB controller helps a PC/Linux/computer to communicate and control this module seamlessly. Use a USB A to Mini B cable to connect with a PC. Please visit http://www.numato.com (http://www.numato.com) to buy cables and accessories for this product. By default, the logic section of the module is powered from USB so make sure not to overcrowd unpowered USB hubs (the picture on the right shows USB Mini connector).

Suggest edit

Relay Contacts

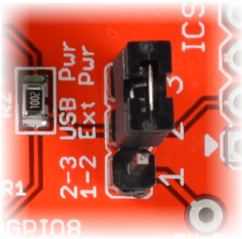


(https://numato.com/blog/wp-content/uploads/2016/02/2ChannelUSBRelayModuleScrewTerminals.jpg)This module has two mechanical relays that can switch up to 10A of current. All contacts on the relay is available externally on screw terminals for easy user access. The relays are rated for AC and DC supply voltages. Please see the electrical parameter table for more details. Each relay has three contacts(C, NO and NC). C is the common terminal and is used in both normally open and normally closed positions. The contacts NC and C will be connected when the relay is turned off and will be disconnected when relay is turned on. And vice versa, the contacts C and NO will be disconnected when relay is turned off and will be connected when the relay is turned on. Table below summarizes possible relay contact positions.

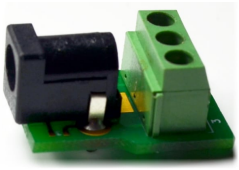
Relay State	Connection between NC and C	Connection between NO and C
OFF	Close	Open
ON	Open	Close

Suggest edit

DC Power Supply



This module uses +5V power supply to function properly. **By default the board is configured to use +5V supply from USB**. So an external +5V power is not required unless USB port is unable to supply enough current. In most cases USB ports are capable of providing enough current for the module. If for any reason, an external 5V power supply needs to be used for the logic section of the module, the power select jumper should be configured properly before connecting the power supply. Please refer to the marking on the board for more details. Make sure to connect the power supply in correct polarity. Connect the positive terminal of the power supply to the +5V terminal on the module.



This product also would require a +12V/ +24V power supply for relays. This is separate from the +5V power used for the digital circuitry including the microcontroller on board. Each 12V relay coil will consume approximately 30mA and 24V relay coil will consume approximately 15mA during ON state. So the minimum amount of current required for the 12V/ 24V rail can be calculated by using the following equation.

Minimum current required for 12V/ 24V power supply = Number of relays x Relay coil current(30mA/ 15mA)

The resultant number will give minimum current required to keep the relays in ON state. But the important thing to keep in mind is that the relays can draw more than 30mA/15mA briefly for 12V/ 24V relay coils respectively during OFF > ON transition time due to the relay coil inductance. So it is recommended that the +12V/ +24V power supply is rated 2 to 3

times the calculated minimum current. It is possible to manage with a smaller power supply if relays are turned on a few at a time compared to turning on all relays at once or by connecting a large capacitor (4000uF or larger recommended) across the +12V/ +24V rail.

Using a product similar to Numato's DC Barrel Jack Adapter is recommended if the power supply has a Barrel Jack connector (See the image on right).

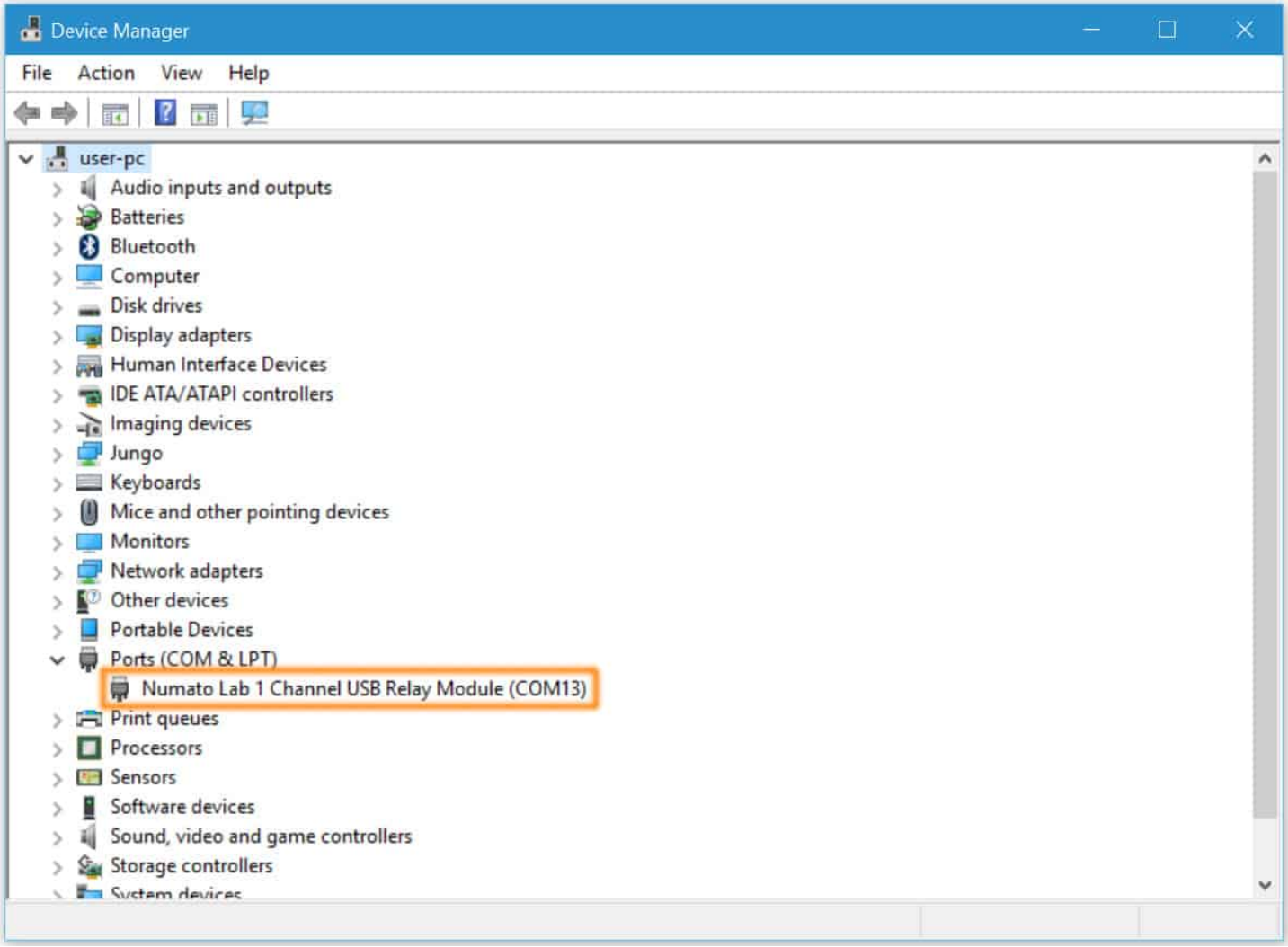
> Connecting power supply incorrectly can cause damage to the module and/or other devices.

Suggest edit

Driver Installation

Installing Numato Lab CDC Driver - Windows Desktop and Server Editions

The driver package for Numato Lab's products can be downloaded from the product page at ttp://numato.com (https://numato.com). To install the driver, unzip the contents of the downloaded driver package to a folder. Attach USB cable to the PC and when asked by Windows device installation wizard, point to the folder where driver files are present. When driver installation is complete, the module should appear in Windows Device Manager as a serial port. The picture below shows a 1 Channel USB Relay Module (https://numato.com/1-channel-usb-powered-relay-module/) (https://numato.com/32-channel-usb-gpio-module-with-analog-inputs/)visible in Windows Device Manager. For other devices (USB GPIO and USB Relay modules), the name will be different but how the device is displayed and used is exactly same.



(https://numato.com/blog/wp-content/uploads/2016/02/UsbpoweredrelayDeviceManager.jpg)

Note down the name of the serial port (COM1, COM2 etc..). This information is required to control the module from the PC.

You may notice that the driver package does not come with a .sys or .exe file as most driver packages do and is expected to be that way. The driver binary necessary in this case is shipped with all copies of windows Desktop/Server editions and gets installed automatically while Windows is installed for the first time. The .inf and .cat files present in the driver package downloaded from http://numato.com (https://numato.com) merely associate this pre-existing driver with the attached Numato Lab device .

The following video demonstrates how to install the driver on Windows 10.

Suggest edit

Installing on Windows Embedded Editions

Windows Embedded editions do not install the infrastructure necessary for USB CDC by default in favor of a smaller footprint. This will cause the driver install to fail unless the necessary files are manually installed prior to installing the driver. Please follow the steps below to install the prerequisites and driver correctly. These steps are tested on Windows 7 Embedded Edition.

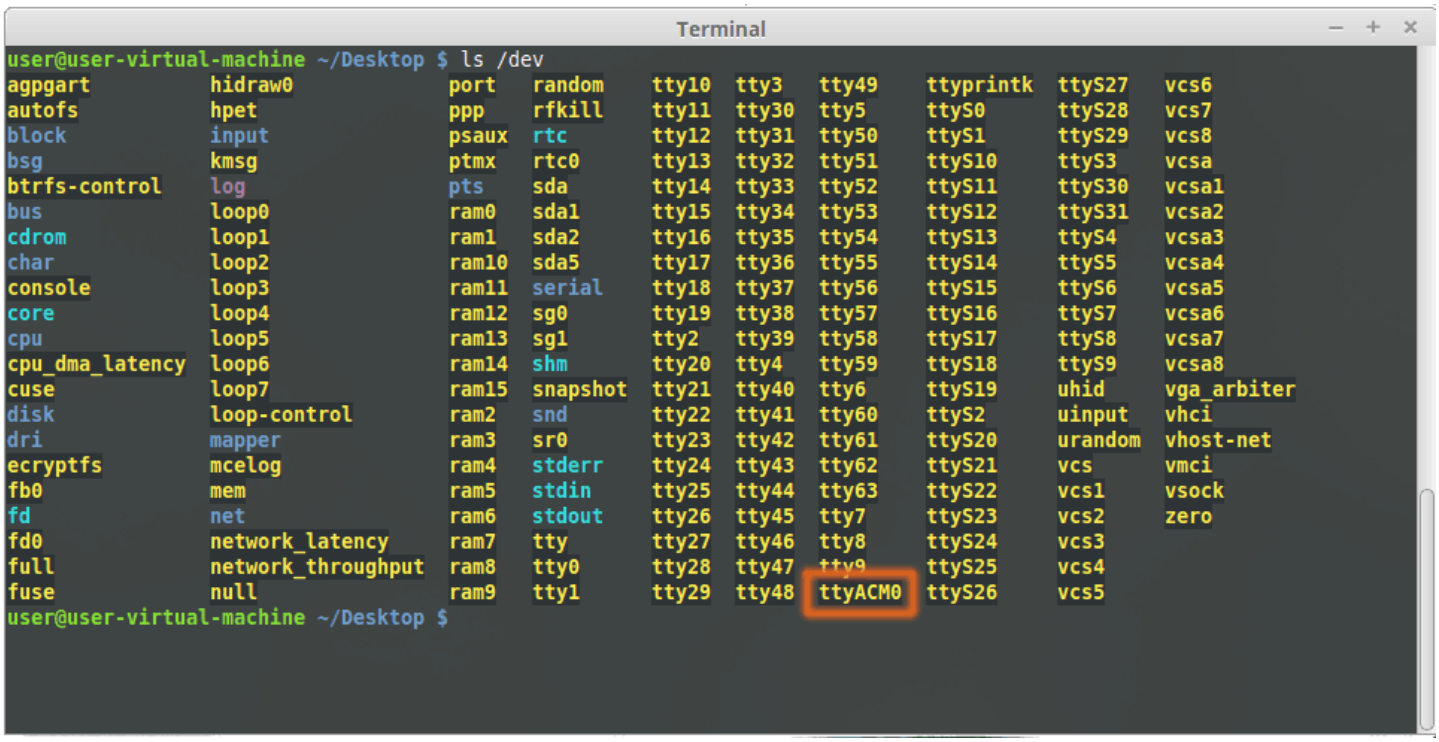
1. Locate *winemb-inf-mdmcpq.cab* on Win 7 Embedded DVD/ISO image
2. Copy *winemb-inf-mdmcpq.cab* to a folder Ex: *C:Temp*
3. Run command *DISM.exe /online /Add-Package /PackagePath:C:Temp*
4. Wait for Windows to restart (Restart machine manually if DISM does not restart the machine automatically)
5. After reboot is complete, plug the device to a USB port and install driver normally (Driver is available for download at the product page)

Suggest edit

Installing on Linux

To use any device that uses USB CDC protocol with Linux, USB CDC driver needs to be compiled in to the kernel. **Fortunately, most Linux distributions (Ubuntu, Redhat, Debian etc..) has this driver pre-installed**. The chances of you requiring to rebuild the kernel to include the USB CDC driver is very slim. When connected to a Linux machine, this

product should appear as a serial port under */dev* directory. Usually the name of the device will be *ttyACMx* or similar. The name may be different depending on the Linux distribution you have. The image below shows the result of *ls /dev* command on a Linux Mint system with a USB GPIO/Relay device attached.



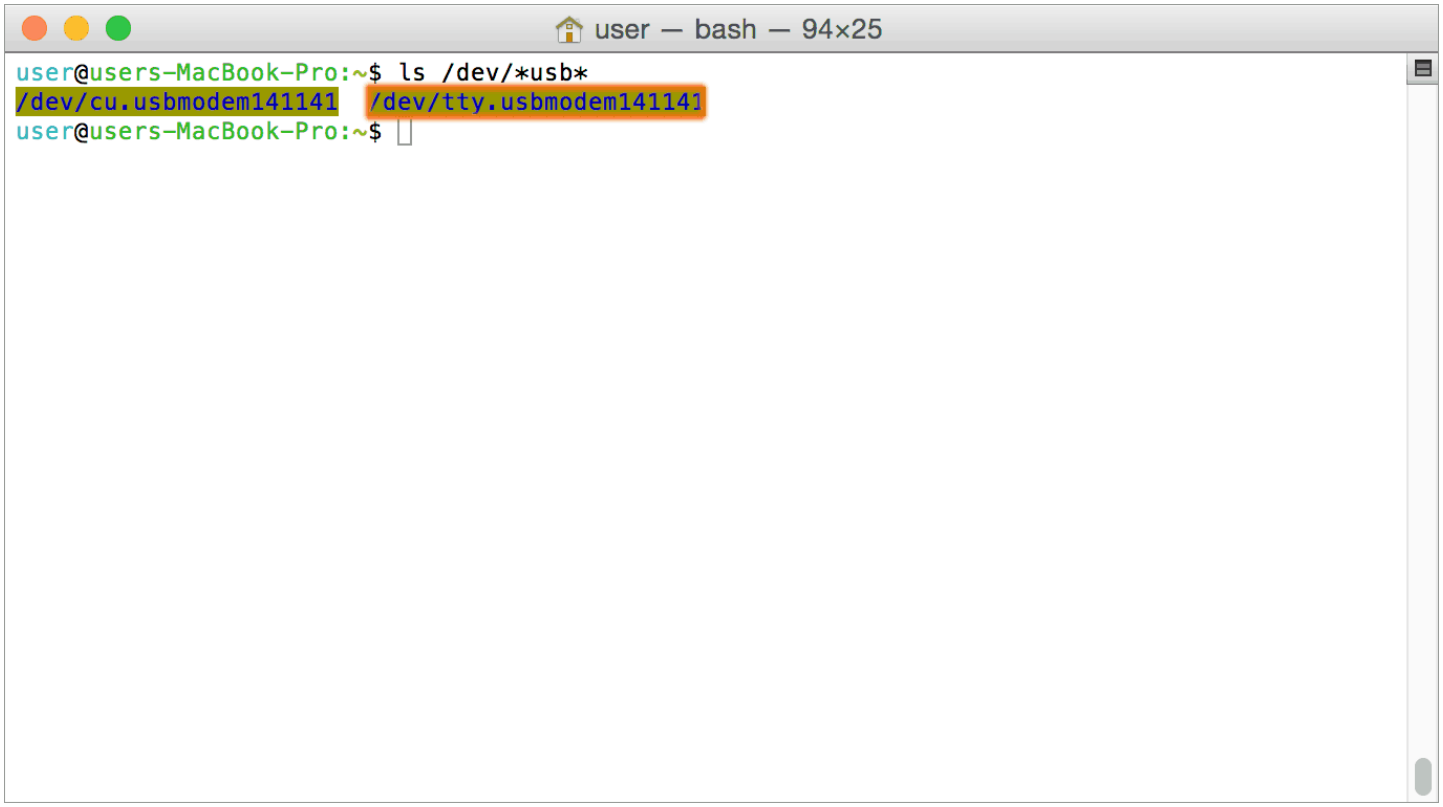
(https://docs.numato.com/wp-content/uploads/2016/01/LinuxxDeviceListing.png)

In this particular case, the device shows up as *ttyACM0* (highlighted in orange color) but it could be *ttyACM1* or *ttyACM2* etc… depending on the specific system and other connected devices. Once the device is visible under */dev* directory, it can be treated just like any other serial device. Commands can be sent to the device using any mechanism that is valid for regular serial ports such as *screen* command or Serial Terminal Emulation applications. If there are more than one devices connected to the same host computer, each device will be displayed as separate serial devices with unique names. These separate serial devices can be used to control individual devices attached.

Installing on Mac OSX

Mac OSX is usually shipped with USB CDC driver pre-installed. When connected to a Mac computer, this product should appear as a serial port under */dev* directory. Usually the name of the device will be *tty.usbserialportx* or similar. The name may be different depending on the Mac OSX version you have. The image below shows the result of *ls*

*/dev/*usb** command on a Mac OSX Yosemite system with a USB GPIO/Relay device attached.



(https://docs.numato.com/wp-content/uploads/2016/01/ListingUSBSerialDevices.png)

In this particular case, the device shows up as *tty.usbmodem141141* (highlighted on orange color) but it could be any name starting *tty.usbmodem* or even a completely different name depending on the exact version of operating system and other connected devices. Once the device is visible under */dev* directory, it can be treated just like any other serial device. Commands can be sent to the device using any mechanism that is valid for regular serial ports such as *screen* command or Serial Terminal Emulation applications. If there are more than one devices connected to the same host computer, each device will be displayed as separate serial devices with unique names. These separate serial devices can be used to control individual devices attached.

Installing CDC Devices On Android

Most Android versions has built in driver that supports external USB Devices. Due to the presence of built-in driver any external USB device including Numato Lab's USB GPIO/Relay products connected to Android based gadget will be enumerated by Android OS. Such enumerated devices can be listed/viewed by using apps such as USB Device Info (https://play.google.com/store/apps/details?id=aws.apps.usbDeviceEnumerator&hl=en). The image below shows info about a Numato Lab USB device printed by USB Device Info (https://play.google.com/store/apps/details?id=aws.apps.usbDeviceEnumerator&hl=en) app. All Numato Lab's USB GPIO and USB Relay modules will be displayed the same way.

(https://docs.numato.com/wp-content/uploads/2016/01/USBGPIOListedOnAndroid.png)

Devices detected by Android can be controlled by using any off the shelf Serial Terminal App such as USB Serial Terminal Lite (https://play.google.com/store/apps/details?id=com.oneman.freeusbtools&hl=en).

Suggest edit

Sending Commands

One of the most powerful features of this module is the simple easy to use command set it supports. This command set hides the complex USB protocol and gives a very simple interface to access the features of the module. The following sections give details of the command set and how to use the command set.

> The sections below talks in detail about how to send commands to the device using different tools on different operating systems. Since GPIOs are a common feature on all devices including relay modules, most sections here will use them to explain and demonstrate the process of sending commands. The process of sending other commands such such as id get/set or relay on/off/read are exactly same, in fact in most cases simply replacing the existing command in the example code with the new one should work right away.

Suggest edit

The Command Set

This product supports a very simple command set that is designed to be less cryptic and easy to use manually (using serial terminal emulation programs) or through a program written in many supported languages.

List of currently supported commands.

No.	Command	Parameters	Example	Description
1	ver	none	ver	Returns firmware Version.
2	id	get/set xxxxxxxx	Id get, id set 12345678	Reads/Writes ID of the module
3	relay	on/off/read, relay number readall/writeall	relay on 0, relay off 0, relay read 0, relay readall, relay writeall FF	Control the relays

The table below has more detailed information about available commands.

No.	Command	Example	Description
1	ver	ver	Returns current firmware version.
2	id	id get id set xxxxxxxx	Id get reads the module ID. Id set will assign a new ID to the module. “x” stands for alphanumeric characters including symbols. The new ID must be exactly 8 characters in length.
3	relay	relay on x	Turns a particular relay on. The parameter “x“ stands for the relay number. The relay number starts from zero. See some examples below. relay on 0 – Turns on relay 0
		relay off x	Turns a particular relay off. The parameter “x“ stands for the relay number. The relay number starts from zero. See some examples below. relay off 0 – Turns off relay 0

No.	Command	Example	Description
		relay read x	Returns the status of a particular relay. The parameter “x“ stands for the relay number. The relay number starts from zero. See some examples below. relay read 0 – Returns status of relay 0 The data returned in response to this command will be either “on” or “off” depending on the current status of the relay.
		relay readall	Reads the status of all relays in a single operation. The return value will a hexadecimal number with binary value 1 at bit positions for relays in ON state and 0 for relays in OFF state. Eg: a return value 0000 (binary 0000 0000 0000 0000) means all relays are OFF. A value FF (binary 1111 1111) means all relays are ON.
			Control all relays in a single operation. A hexadecimal value must be specified with desired bit positions set to 0 or 1. A value 0 at a bit position will turn off the corresponding relay. A value 1 at a bit position will turn on the corresponding relay. For example, relay writeall FF – Turns on all relays
4	reset	reset	Resets all relays to off state which is the default state.

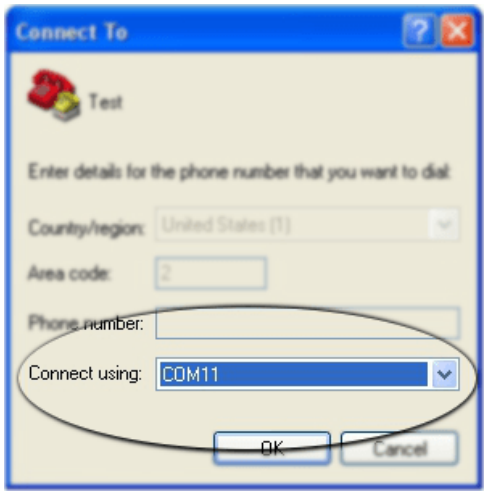
Suggest edit

Using the Serial Terminal Emulator Software On Windows

HyperTerminal

Using this module with Hyper Terminal is very easy. Please follow the steps below.

- Connect the module to the computer, install driver and note down the name of the new serial port that appears in the device manager.
- Open Hyper Terminal and select the serial port corresponding to the GPIO module. Click OK.

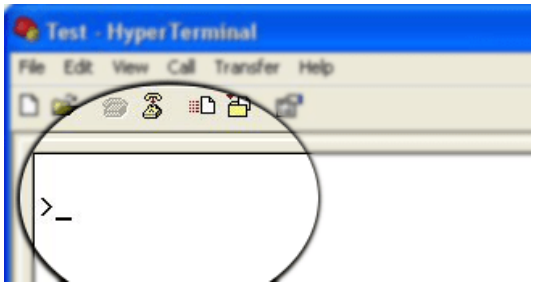


(https://numato.com/help/wp-content/uploads/2016/01/HyperTerminal_USBGPIO_SelectCOMPort.png)

- A new window will pop up where the serial port settings can be changed. In this window, leave all settings to defaults except Flow Control which needs to be set to “None” and click OK.

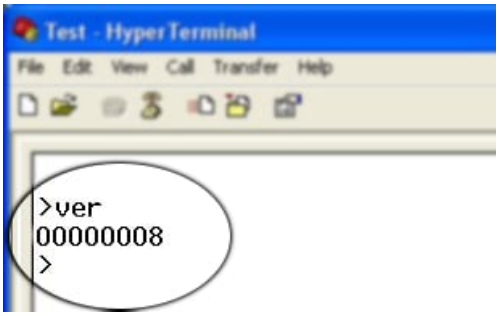
(https://numato.com/help/wp-content/uploads/2016/01/HyperTerminal_USBGPIO_COMPortSettings.png)

- If everything goes well, you should be presented with a blank screen. Press ENTER key and the command prompt should appear. Commands listed in the table above can be entered here now.



(https://numato.com/help/wp-content/uploads/2016/01/HyperTerminal_USBGPIO_CommandPrompt.png)

For example, here is the response for “ver” command.



(https://numato.com/help/wp-content/uploads/2016/01/HyperTerminal_USBGPIO_VersionCommand.jpg)

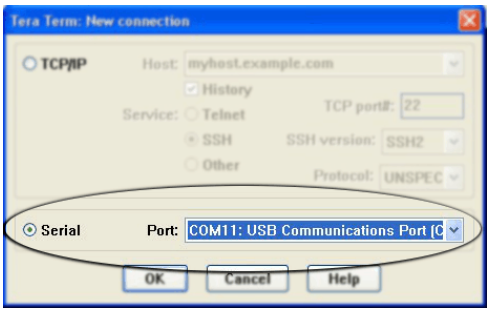
Suggest edit

TeraTerm

Using the GPIO module with Teraterm is just as easy. Please follow the steps below.

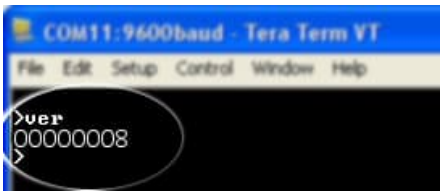
> Teraterm is an open source software. A free copy can be downloaded from http://en.sourceforge.jp/projects/ttssh2/releases/ (http://en.sourceforge.jp/projects/ttssh2/releases/)

- Run Teraterm application and select the port corresponding to the GPIO module in the “New connection” dialog and click OK.



- Press ENTER key on the main window and a command prompt should appear as in the image below.

- Enter the command at the command prompt. Example “ver” command and response is in the image below.

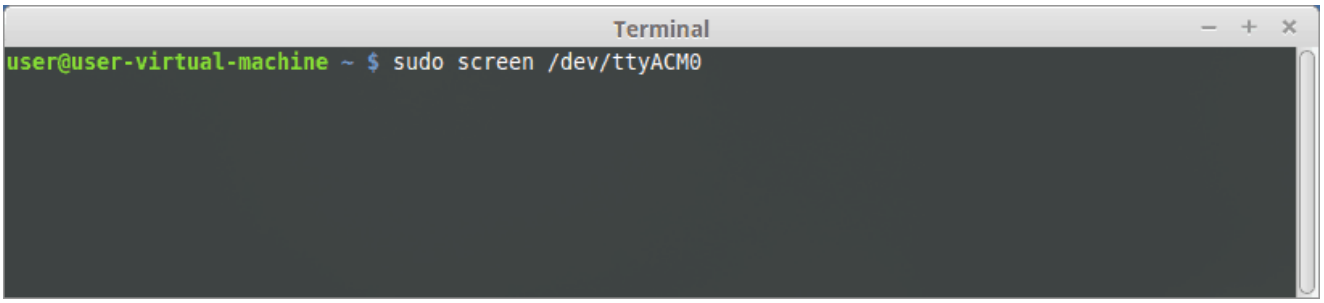


Suggest edit

Using Serial Terminal Software on Linux

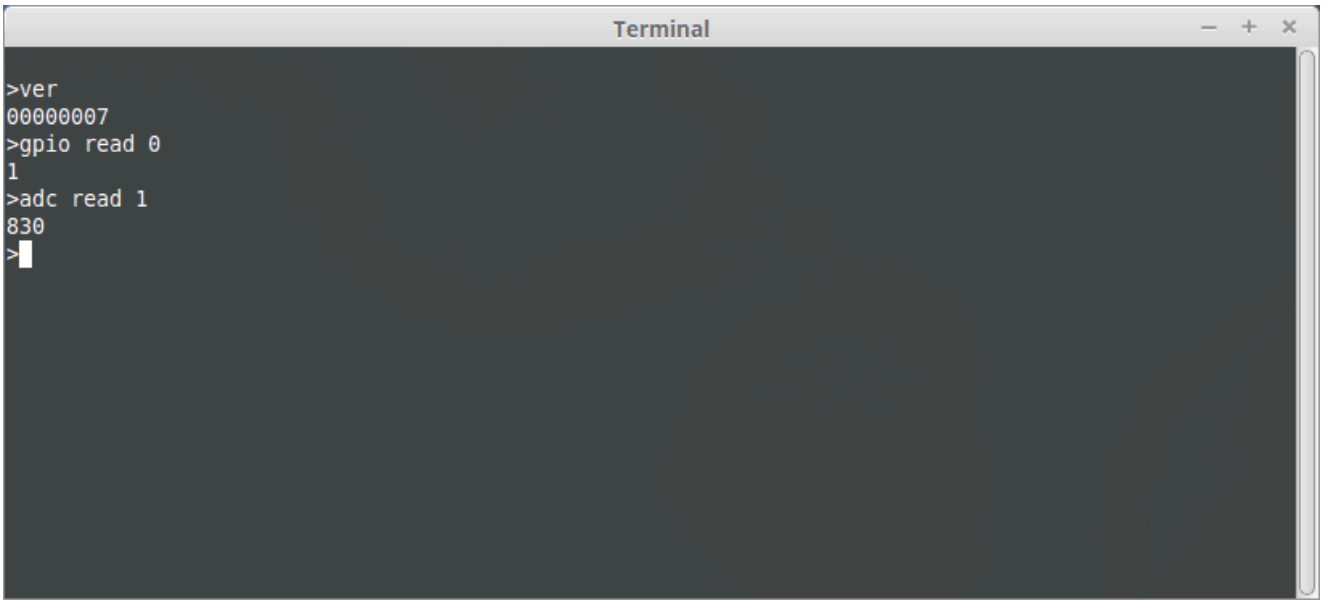
GNU Screen

GNU Screen is a full-screen Window Manager that works with Serial Terminals as well. This makes GNU Screen a very handy tool to communicate with Numato Lab's USB GPIO (https://numato.com/gpio-modules/usb-gpio/)/Relay (https://numato.com/relay-modules/usb-relay/) modules and other products. Please visit this document (http://docs.numato.com/how-to-install-gnu-screen-terminal-program-on-linux/) for details on how to install GNU Screen on your Linux machine. Run the command screen /dev/ttyACM0 as shown in the image below. Please don't forget to replace the device name with device name retrieved from your system. Use *sudo* or set appropriate permissions for the device where necessary.



(https://docs.numato.com/wp-content/uploads/2016/01/ScreenCommandOnLinux.png)

When the *screen* command is executed, you will be presented with an empty screen. Press ENTER key to see the command prompt. At the command prompt enter any command supported by the device. All commands must be completed by ENTER key press. Please see the list of available commands in the product documentation. The image below shows a few sample commands executed through the *screen* program and corresponding results.

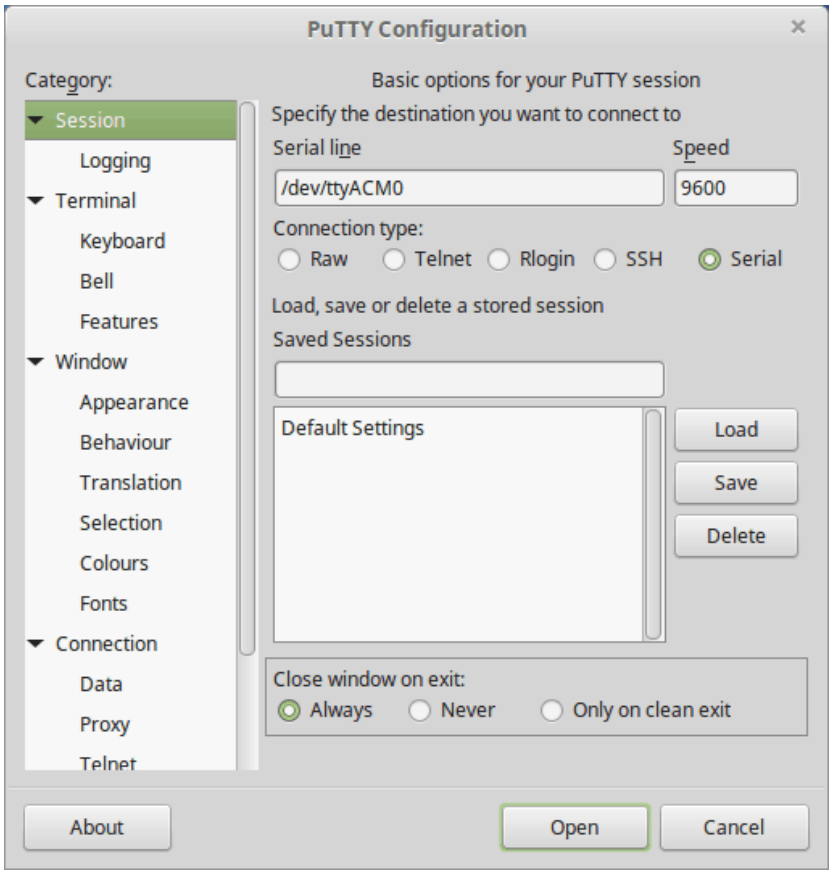


(https://docs.numato.com/wp-content/uploads/2016/01/ScreenCommandExamplesOnLinux.png)

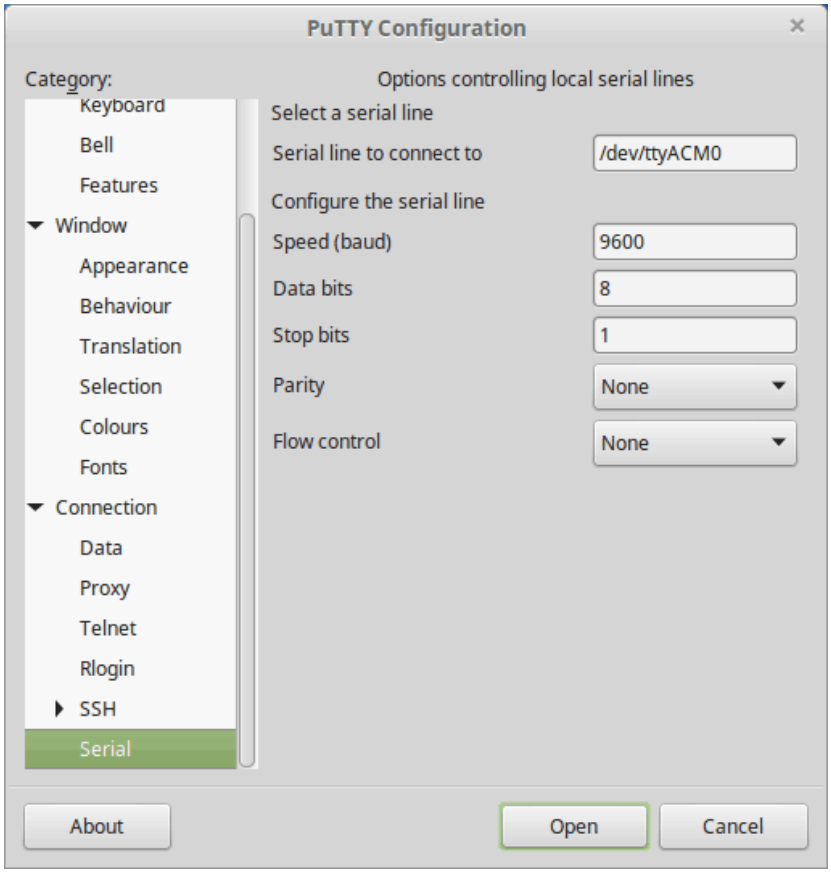
Suggest edit

PuTTY

PuTTY is a free SSH/TELNET client that works with Serial Terminals as well. This makes PuTTY a very handy tool to communicate with Numato Lab's USB GPIO (https://numato.com/gpio-modules/usb-gpio/)/Relay (https://numato.com/relay-modules/usb-relay/) modules and other products. Please visit this document (http://docs.numato.com/how-to-install-gnu-screen-terminal-program-on-linux/) for details on how to install PuTTY on your Linux machine. Start PuTTY by running the command putty at the terminal or double clicking the PuTTY icon. Use *sudo* or set appropriate permissions for the device where necessary. You will see a screen similar to the below image when *PuTTY* is started.



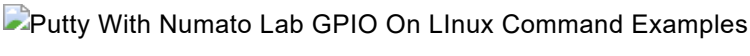
(https://docs.numato.com/wp-content/uploads/2016/01/PuttyWithNumatoLabGPIOOnLInux_Settings1.png)



(https://docs.numato.com/wp-content/uploads/2016/01/PuttyWithNumatoLabGPIOOnLInux_Settings2.png)

Leave settings as shown in the images above. Please don't forget to replace the device name with device name retrieved from your system. Click Open button to start PuTTY session. PuTTY will start with an empty screen. Press ENTER key to show the command prompt. At the command prompt enter any command supported by the device. All

commands must be completed by ENTER key press. Please see the list of available commands in the product documentation. The image below shows a few sample commands executed through PuTTY and corresponding results.



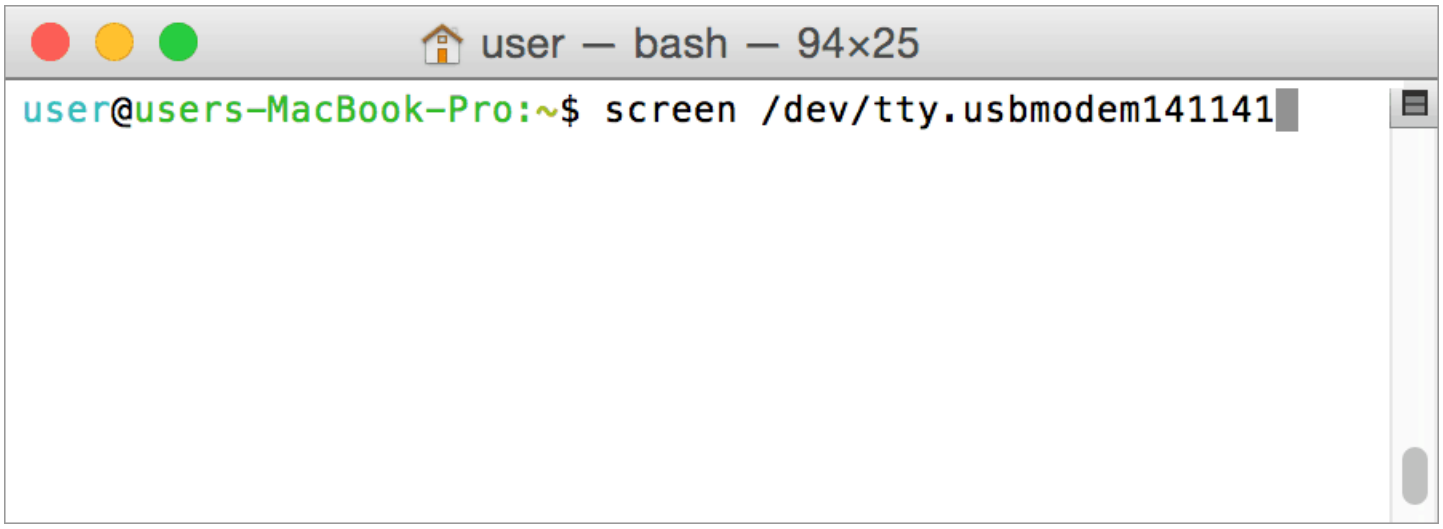
(https://docs.numato.com/wp-content/uploads/2016/01/PuttyWithNumatoLabGPIOOnLInux_CommandExamples.png)

Suggest edit

Using Serial Terminal Software on Mac OS X

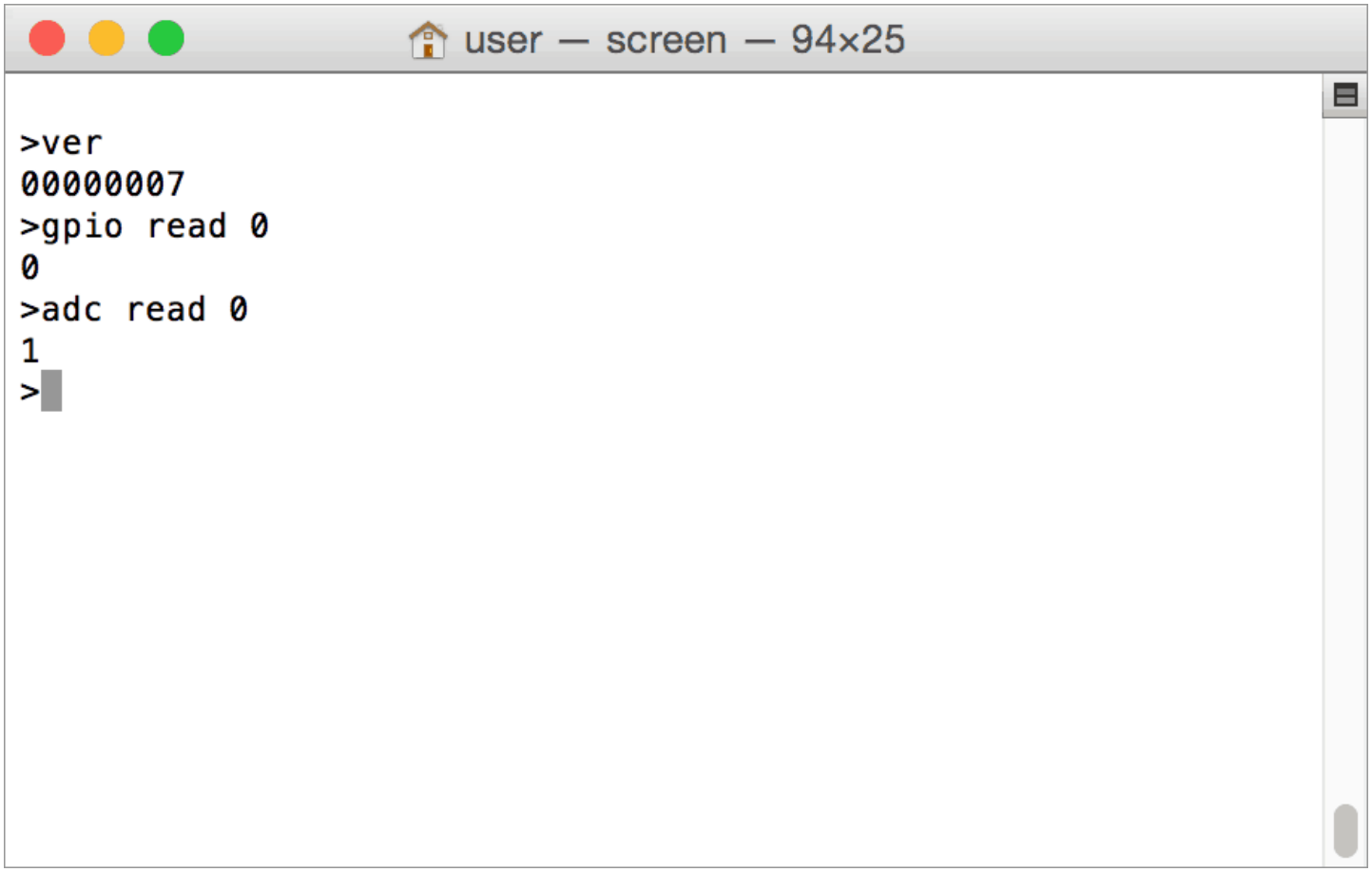
Using Screen Command To Control Devices On Mac OSX

Mac OSX provides a built in command/program called *screen* that can be used to communicate with any serial device attached to a Mac device. The *screen* program is a light weight Terminal Emulator that is very easy to use. Run the command screen */dev/tty.usbmodem141141* as shown in the image below. Please don't forget to replace the device name with device name retrieved from your system.



(https://docs.numato.com/wp-content/uploads/2016/01/ScreenCommandOnMac.png)

When the *screen* command is executed, you will be presented with an empty screen. Press ENTER key to see the command prompt. At the command prompt enter any command supported by the device. All commands must be completed by ENTER key press. Please see the list of available commands in the product documentation. The image below shows a few sample commands executed through the *screen* program and corresponding results.



(https://docs.numato.com/wp-content/uploads/2016/01/ScrenCommandExamplesOnMac.png)

Suggest edit

Using CoolTerm To Control Devices On Mac OSX

CoolTerm is a free Serial Port Terminal application available for Mac OSX . Download and install CoolTerm on your Mac and run.

> CoolTerm is a free software and can be downloaded from http://freeware.the-meiers.org/ (http://freeware.the-meiers.org/)

Select the appropriate serial port corresponding to your attached device. In the image below */dev/tty.usbmodem141141* is selected. Please don't forget to select the correct port name corresponds to your device and leave all other settings as in the image below. Once all correct settings are selected, click the OK button to open the port.

(https://docs.numato.com/wp-content/uploads/2016/01/CoolTerm_DeviceSettings.png)

If the port opened successfully, you will be presented with an empty screen. Press ENTER key to see the command prompt. At the command prompt enter any command supported by the device. All commands must be completed by ENTER key press. Please see the list of available commands in the product documentation. The image below shows a

few sample commands executed using the *CoolTerm* and corresponding results.


(https://docs.numato.com/wp-content/uploads/2016/01/CoolTerm_CommandExamples.png)

Suggest edit

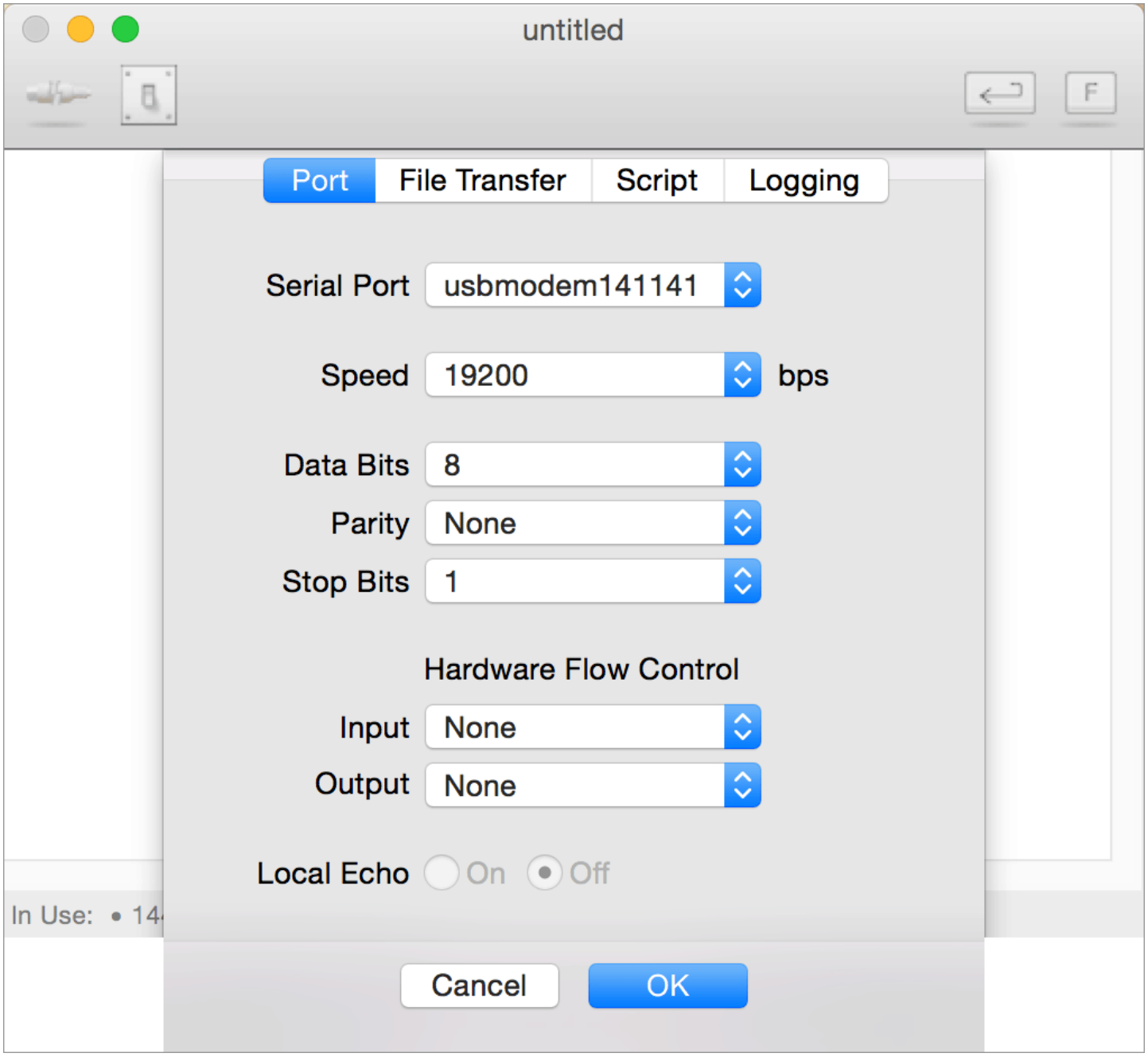
Using goSerial To Control Devices On Mac OSX

goSerial is a free Serial Port Terminal application available for Mac OSX . Download and install goSerial on your Mac and run.

> goSerial is a free software and can be downloaded from http://www.furrysoft.de/?page=goserial (http://www.furrysoft.de/?page=goserial)

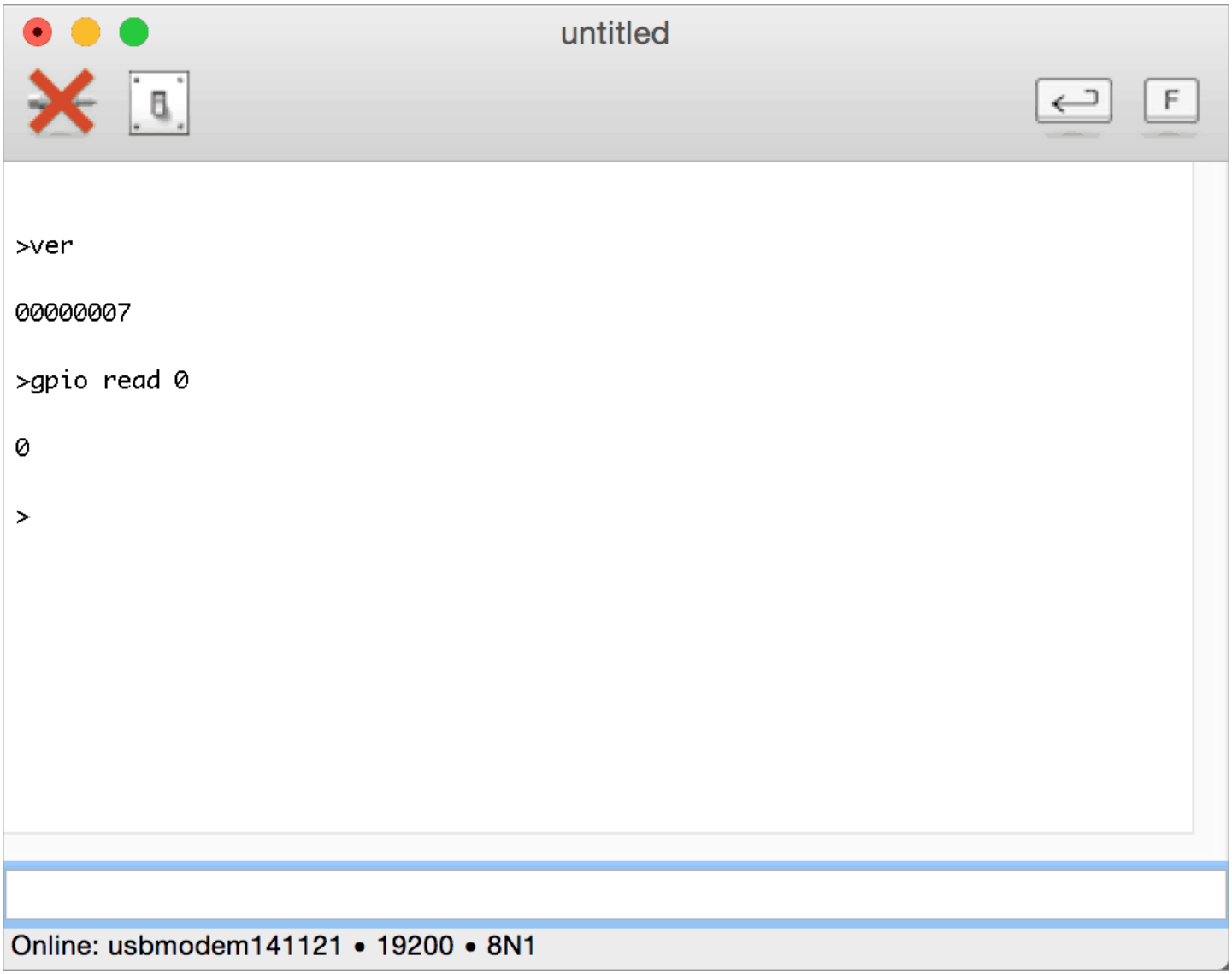
Select the appropriate serial port corresponding to your attached device. In the image below */dev/tty.usbmodem141141* is selected. Please don't forget to select the correct port name corresponds to your device and leave all other settings as in the image below. Once all correct settings are selected, click the OK button to

open the port.



(https://docs.numato.com/wp-content/uploads/2016/01/GoSerial_DeviceSettings.png.png)

If the port opened successfully, you will be presented with an empty screen. Press ENTER key to see the command prompt. At the command prompt enter any command supported by the device. All commands must be completed by ENTER key press. Please see the list of available commands in the product documentation. The image below shows a few sample commands executed using the goSerial and corresponding results.



(https://docs.numato.com/wp-content/uploads/2016/01/GoSerial_CommandExamples.png)

Suggest edit

Using Serial Terminal Software on Android

USB Serial Terminal Lite

Device detected by Android can be controlled by sending commands using USB Serial Terminal Lite (https://play.google.com/store/apps/details?id=com.oneman.freeusbtools&hl=en). Once connected to the device, enter commands at the command window and press the send button to execute. The commands and the results will be printed in the log window.

(https://docs.numato.com/wp-content/uploads/2016/01/NumatoUSBGPIOCOntrolOnAndroid.png)

Suggest edit

Using Custom Program and Script

This GPIO module can be controlled using custom programs written in many languages. Almost any language can be used as long as it supports some sort of serial communication method. Some of the supported languages include

- C/C++
- Visual Basic
- Visual Basic for Applications (Microsoft Office VBA)
- Perl
- Python
- JAVA
- And a lot more…

The APIs need to be used may be different depending on the target operating system even when the same language is used. For example when using C/C++ on Windows, Win32 Serial Communication APIs along with File IO APIs (CreateFile, ReadFile, WriteFile, etc…) needs to be used (see Microsoft Developer Page on serial communications (https://msdn.microsoft.com/en-us/library/ff802693.aspx)). But when C/C++ is used on Linux operating system “termios”, APIs can be used for serial communication. Please refer to your compiler/language documentation for more details about serial port communication.

Specific details of programming may vary depending on the language and operating system of choice. But the fundamental steps for writing a program to control the GPIO module can be more or less the same. Here is the list of steps that you may need to follow while writing your own program.

1. Open the serial port for communication.
2. Set port parameters. Most of the parameters can be left to defaults except Flow Control, which needs to be set to “none”.
3. To send a command to the module, use an API equivalent to write/writefile and pass the buffer/string containing the command. It is important to append Carriage Return (ASCII 13) to emulate the ENTER key.
4. If return data is expected (Eg: “ver” command), try to read the characters from the serial port input buffer. APIs equivalent to Read/ReadFile can be used to read data from the module. Please note that the return data will include the command itself (since the module echoes everything back), the result, carriage return and the “>” symbol.

Please visit the product page (https://numato.com/products/) for available sample programs.

C/C++ on Windows

Visual C++

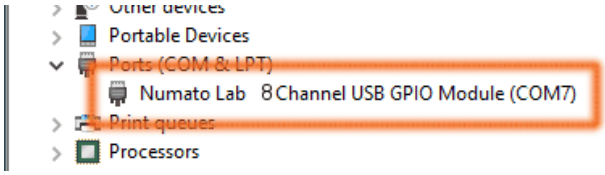
Microsoft Visual C++ is a C++ compiler/IDE from Microsoft. Express editions of Visual C++ are free to download and use for commercial and non-commercial purposes. Visual C++ Express is perfectly suitable for writing full fledged applications for Windows and certainly can be used for writing custom applications for Numato Lab’s GPIO/Relay

modules. This section talks about C implementation but using the device with C++ is very similar except that you may use object oriented programming techniques where necessary, but APIs and their usage will be same.

Accessing Numato Lab's GPIO/Relay modules using Visual C++ 2010 Express will require the following implemented.

1. Open the serial port representing the device and obtain a handle to the port using *CreateFile()* Windows API. This handle can be used to communicate with the device.
2. Use *Writefile()* Windows API to write commands to the device
3. Use *ReadFile()* Windows API to read response from the device

The reader is encouraged to become familiar with the Windows APIs CreateFile() (https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx), WriteFile() (https://msdn.microsoft.com/en-us/library/windows/desktop/aa365747(v=vs.85).aspx) and ReadFile() (https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467(v=vs.85).aspx) to make it easier to understand the code snippets presented here.



(https://docs.numato.com/wp-content/uploads/2016/01/PortNameInDeviceManager.png)To open the port representing the attached device, we need to look up the port name from Windows Device Manager (Please see image to the right). The complete name of the device is not required, only the *COMn* part is required. If the port number is 9 or below (for example, COM9, COM2 etc..) the port name can be used as is in the program. If the port number is 10 or above (for example, COM10, COM35 etc..) , a different syntax needs to be used. The required syntax is *.COMn* (for example, COM35 cannot be used as is, rather *.COM35* must be used). This syntax works with all port numbers including 9 and below. Microsoft has an informative KnowledgeBase article on this subject (https://support.microsoft.com/en-us/kb/115831). The developer is recommended to visit this article before proceeding.

Once the port name is discovered, the *CreateFile()* Windows API can be used to open the port.

```
wchar_t PortName[] = L"\.COM35";

hComPort = CreateFile(PortName, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0);

if (hComPort == INVALID_HANDLE_VALUE)
{
    printf("Error: Unable to open the specified portn");
    return 1;
}
```

If you are wondering why there are more back slashes in the code as part of the port name, please read through http://en.wikipedia.org/wiki/Escape_sequences_in_C (https://en.wikipedia.org/wiki/Escape_sequences_in_C)for more information.

If the above code is executed successfully, a handle to the port will be available in the variable *hComPort*. This handle can be used to further communicate with the port/device.

Communication with the device usually falls in to two categories.

1. Writing to the device (For example, sending a command)
2. Reading from the device (Fr example, reading the result of a previously sent command)

Setting/Clearing a GPIO is an example of Write operation. To set a GPIO (GPIO0 in this example), we will need to send the command g*pio set 0*. It is not sufficient to send the command alone, but the command must be terminated using Carriage Return (https://en.wikipedia.org/wiki/Carriage_return) symbol so the device can recognize the end of the command and execute it. The following code writes *gpio set 0* command to the device. Please take special note of the line where *CR* character (0x0D) is added.

```
/* Copy the command to the command buffer */
strcpy(cmdBuffer, "gpio set 0");

/* Append 0x0D to emulate ENTER key */
cmdBuffer[10] = 0x0D;

/* Write the command to the GPIO module. Total 11 bytes including 0x0D  */

printf("Info: Writing command  to the GPIO modulen");

if(!WriteFile(hComPort, cmdBuffer, 11, &numBytesWritten, NULL))
{
    CloseHandle(hComPort);
    printf("Error: Unable to write to the specified portn");
    return 1;
}

printf("Info:  Command sent successfulyn");
```

Reading from the device is very similar. One can use the Windows API ReadFile() (https://msdn.microsoft.com/en-us/library/windows/desktop/aa365467(v=vs.85).aspx) to read results back from the device. The following code snippet attempts to read data from the device. Please note the third parameter which is the number of bytes to be read. This parameter must be set to a number larger than the number of bytes expected from the device. Please note that the bytes returned from the device is not merely the result rather it includes the original command itself, command prompt and newline character etc.. This is because the device will echo all the data sent to it along with the command prompt character (>) along with the result to provide a seamless experience when used with a serial terminal software. So the number of bytes to read (third parameter) must be calculated accordingly. The easiest way to do this is to send the command manually using a Serial Terminal software and count the number of resulting data. These additional bytes can be filtered out in the software by using string manipulation functions or Regex.

The following code snippet shows how to read bytes from the device.

```
/*Read back the response*/
if(!ReadFile(hComPort, responseBuffer, 16, &numBytesRead, NULL))
{
    CloseHandle(hComPort);
    printf("Error: Unable to write to the specified portn");
    return 1;
}
```

Since the device always responds to a command initiated by the host and does not send any data on its own, for all practical purposes, a read operation is usually preceded by a write operation where a command is sent to the device. The code snippet below shows a complete Write/Read sequence. This code attempts to send *gpio read 1* command to the device and read the result back.

```
/* Copy the command to the command buffer */
strcpy(cmdBuffer, "gpio read 1");

/* Append 0x0D to emulate ENTER key */
cmdBuffer[11] = 0x0D;

/* Write the command to the GPIO module. Total 12 bytes including 0x0D  */

printf("Info: Writing command  to the GPIO module\n");

if(!WriteFile(hComPort, cmdBuffer, 12, &numBytesWritten, NULL))
{
    CloseHandle(hComPort);
    printf("Error: Unable to write to the specified port\n");
    return 1;
}

printf("Info:  Command sent successfuly\n");

/*Read back the response*/
if(!ReadFile(hComPort, responseBuffer, 14, &numBytesRead, NULL))
{
    CloseHandle(hComPort);
    printf("Error: Unable to write to the specified port\n");
    return 1;
}
```

The data returned will always have the original command itself, command prompt and newline character etc.. in addition to the result. This additional data can be filtered out using string manipulation techniques or Regex. The following code snippet extracts the actual analog data returned from the device.

```
/* Add a null character at the end of the response so we can use the buffer
   with string manipulation functions.
 */
responseBuffer[numBytesRead] = '
```


JavaScript

Node.JS

Node.JS has become one of the most popular framework for network and web based applications. Node.JS has the ability to work with Serial Ports when appropriate modules are installed and this makes Node capable of communicating with Numato Lab (https://numato.com/)‘s USB devices. **Node.JS works on Windows, Linux and Mac OS X**. Install Node.JS, NPM and the SerialPort module for Node. More information on how to install these application can be found on the links below.

- How to install Node.JS (https://nodejs.org/en/download/package-manager/)
- How to install NPM (http://blog.npmjs.org/post/85484771375/how-to-install-npm)

- How to install Serial Port module for Node (https://www.npmjs.com/package/serialport)

Once all the above prerequisites are installed, we can start writing the JavaScript code.

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 as long as it is a legal value. Also use device names such as *COM4* on Windows and device node name such as */dev/ttyACM0* on Linux/Mac.

```
var SerialPort = require("serialport").SerialPort

//On Windows use the port name such as COM4 and on Linux/Mac, use the device node name such as /dev/ttyACM0
var port = "com4";

var portObj = new SerialPort(port,{
 baudrate: 19200
}, false);
```

This object can be used to open the port and communicate with the device. The objects *Open()* method opens the port for communication.

```
portObj.open(function (error){
  if ( error ) {
              console.log('Failed to open port: ' + error);
  } else {
   //Communicate with the device
}
```

The *write()* method can be used to write commands to the device if the port is already opened. The following code snippet shows how.

```
portObj.write("gpio set 0r", function(err, results){
    if(error){
        console.log('Failed to write to port: '+ error);
    }
});
```

The Serial Port modules *data* event can be used to receive response from the device. The following code snippet prints all data received from the device. The data received can be further processed to retrieve the required information such as analog value or GPIO status.

```
portObj.on('data', function(data){
        console.log('Data Returned by the device');
        console.log('--------------------');
        console.log(String(data));
        console.log('--------------------');
});
```

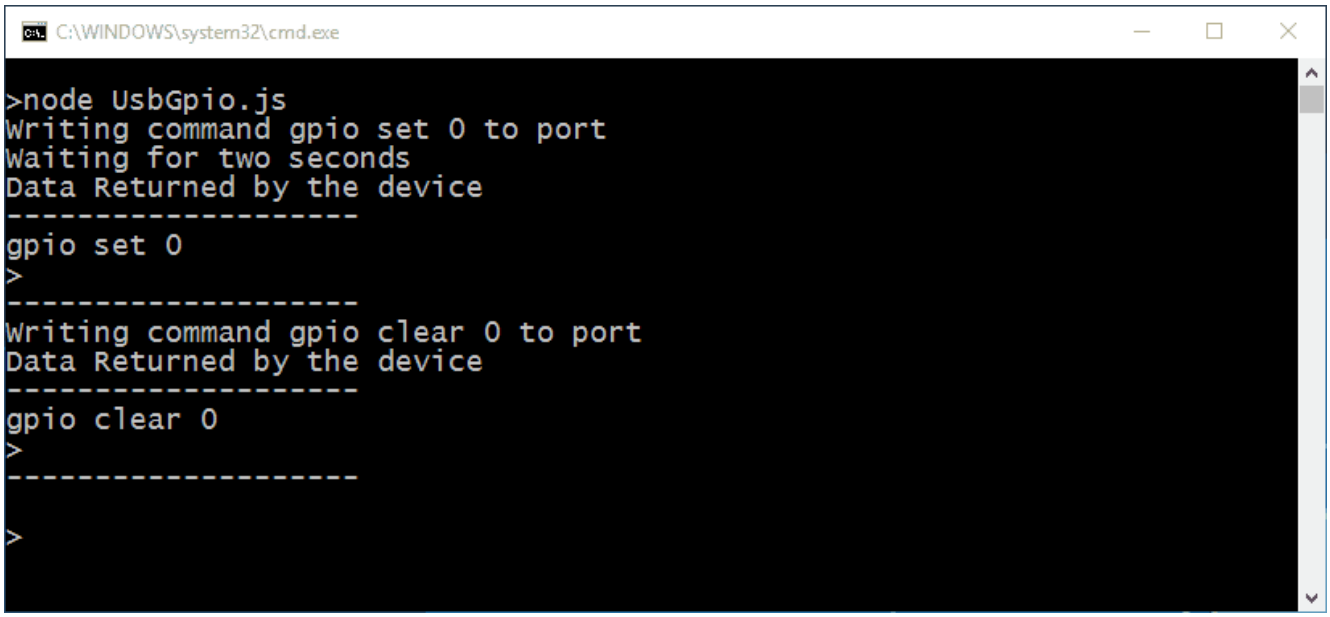
Finally a previously opened port can be closed by using the *close()* method.

```
portObj.close();
```

The following complete Node.JS 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.

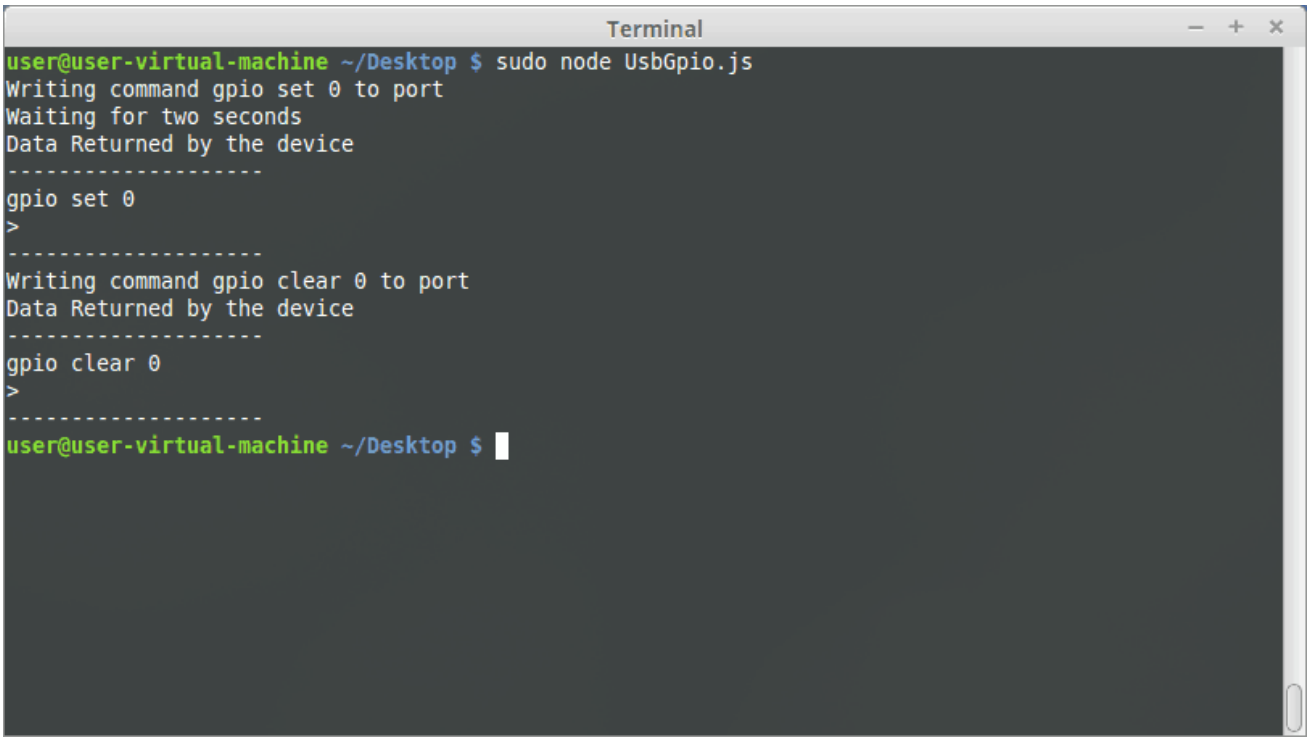
```
var SerialPort = require("serialport").SerialPort
var port = "COM11";

var portObj = new SerialPort(port,{
  baudrate: 19200
}, false);

portObj.on('data', function(data){
        console.log('Data Returned by the device');
        console.log('--------------------');
        console.log(String(data));
        console.log('--------------------');
        portObj.close();
});

portObj.open(function (error){
  if ( error ) {
                console.log('Failed to open port: ' + error);
  } else {
                console.log('Writing command gpio set 0 to port');
                portObj.write("gpio set 0r", function(err, results){
                        if(error){
                                console.log('Failed to write to port: '+ error);
                        }
                });

                console.log('Waiting for two seconds');
                setTimeout(
                        function(){
                                console.log('Writing command gpio clear 0 to port');
                                portObj.write("gpio clear 0r", function(err, results){
                                        if(error){
                                                console.log('Failed to write to port: '+ error);
                                        }
                                });

                                setTimeout( function(){process.exit(code=0);}, 1000);
                        }
                ,2000);
  }
});
```

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 command prompt and optional result of operation.



(https://docs.numato.com/wp-content/uploads/2016/01/Node.JS_GPIOOnWindows.png)

The image below shows the same script running on Linux Mint.



(https://docs.numato.com/wp-content/uploads/2016/01/Node.JS_GPIOOnLinux.png)

And Mac OS X (This screenshot is taken on Yosemite)

Node.JS GPIO On Mac OSX

(https://docs.numato.com/wp-content/uploads/2016/01/Node.JS_GPIOOnMacOSX.png)

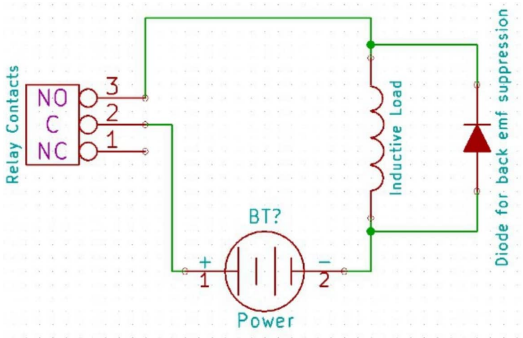
Latest files can be downloaded from our GitHub repository (https://github.com/numato/samplecode/tree/master/RelayAndGPIOModules/USBRelayAndGPIOModules/node.js/usbgpio).

Suggest edit

Additional Information

Using relay modules with inductive loads

It is important to take additional care when using relays with inductive loads. An inductive load can be pretty much anything that has a coil and works based on magnetic principles like Motors, Solenoids and transformers. But in practice, even a wire longer than a few feet can display substantial inductance. Inductive loads produce back emf when the magnitude of the load current changes. The back emf can be in the order of tens or even hundreds of voltage (See this Wikipedia article http://en.wikipedia.org/wiki/Counter-electromotive_force (https://en.wikipedia.org/wiki/Counter-electromotive_force)). This effect is most severe when power is disconnected from inductive load because the rate of change of current is maximum at that point. Even though the back emf lives only for a very short time (a



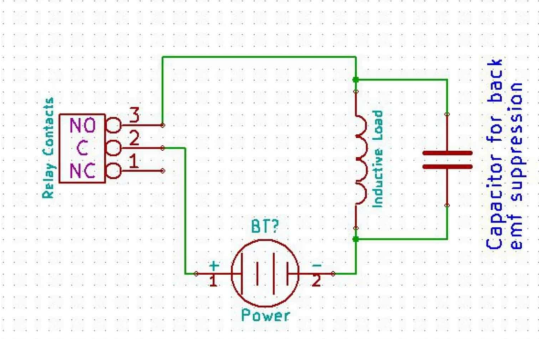
(https://numato.com/blog/wp-content/uploads/2016/02/SnubberDiode.jpg)

Diode As Snubber

few milliseconds) it can cause sparks between the relay contacts and can deteriorate the contact quality over time and reduce the life span for the relays considerably.

So it is important to take countermeasures to suppress the back emf to acceptable levels to protect relay contacts. Usually this requires connecting electronic devices in parallel with the load such that they absorb the high voltage components generated by the load. For solenoids, connecting a diode (fast switching diode is recommended) in parallel to the load (in reverse direction to the load current) is very effective. A diode used for this purpose is usually called a freewheeling diode. Please see the diagram on the right for connection details.

A capacitor with proper rating is recommended for protecting the relay contacts when a motor is used as load. The capacitor should be rated enough to withstand the back emf that is generated by the motor. Please see the diagram below for connection details.



(https://numato.com/blog/wp-content/uploads/2016/02/CapacitiveSnubber.jpg)

Capacitor As Snubber

Please note that the relay modules are NOT shipped with back emf suppression devices pre-installed. The exact kind of suppression device and the parameters of the selected device can vary depending on the load itself. Some of the parameters that affects the suppression device selection are the inductance of the load, power supply voltage, load current, physical size/structure of the load etc.. It is obvious that it is impossible for us to predict these parameters and design required back emf suppression device and incorporate that on the board. So we believe this is a task best left to the module user. There is an excellent article on designing back emf suppression on Wikipedia at http://en.wikipedia.org/wiki/Flyback_diode (https://en.wikipedia.org/wiki/Flyback_diode)

Suggest edit

Technical Specifications

Parameter	Value	Unit
Number of relays	8	
Digital circuit power supply voltage (USB or external)	5	V
Maximum current drawn by digital circuitry	95	mA
Relay Specifications		
Nominal relay coil voltage	12/ 24	V
Nominal coil power consumption (per relay)	360	mW

Parameter	Value	Unit
Relay contact material	Silver Alloy	
Contact rating	7A 240VAC/ 10A 120VAC	
Maximum switching voltage	240VAC/ 30VDC	
Maximum switching current	10	A
Maximum switching power	270VA/ 240W	
Contact resistance (initial)	100 Max at 6VDC 1A	mΩ
Life expectancy (Electrical)	100,000	Operations
Life expectancy (Mechanical)	10,000,000	Operations
Nominal insulation resistance	100 Min at 500VDC	MΩ
Nominal switching on response time	10	mS
Other Information		
USB Vendor ID	0x2A19	
USB Product ID	0x0C02	

Suggest edit

Frequently Asked Questions (FAQs)

Q. What are the serial parameters I need to use when communicating with this board?
A. Since this module uses USB as the underlying transport mechanism, most of the serial parameters do not affect the communication. You can leave all parameters to any legal value (Eg: 2400, 4800, 9600 etc… for baud rate) except Flow control. Flow control needs to be set to “None”.

Q. Where do I find driver for this product?
A. Visit http://numato.com (https://numato.com) and navigate to the product page. There will be a link to download windows driver. Linux does not require driver installation since in most cases they are shipped with the driver pre-installed.

Q. Why there is no *.sys* or *.exe* file in the Windows driver package I downloaded?
A. This product uses USB CDC driver binary which is already present on Windows. All Windows versions (with the exception of Embedded Editions) has this driver binary installed by default. The *.inf* and *.cat* files present in the zip file

helps Windows identify the device properly and associate appropriate driver (*.sys*) to the device

Q. Does this product work with Linux?
A. Yes, this product works with Linux. Please see more details on how to use this product with Linux elsewhere in this document.

Q. Does this product work with Mac OSX?
A. Yes, this product works with Mac OSX. Please see more details on how to use this product with Mac elsewhere in this document.

Q. What are the serial terminal software that this product work with?
A. This product works with a lot of different Serial Terminal software. Some examples can be found elsewhere in this document. Different Serial Terminal software are written by different developers with different purposes in mind. So you may encounter some software that may not work with this product. But usually alternatives are available in most if not all cases.

Q. The GPIO looses its previously set value when trying to read the status. Why it is so?
A. When a gpio is to output a value (high/low), that particular GPIO is put to output mode. When you are trying to read the GPIO, it needs to be put in input mode. In input mode, the GPIO will go to high impedance state and thus looses the previously set value.

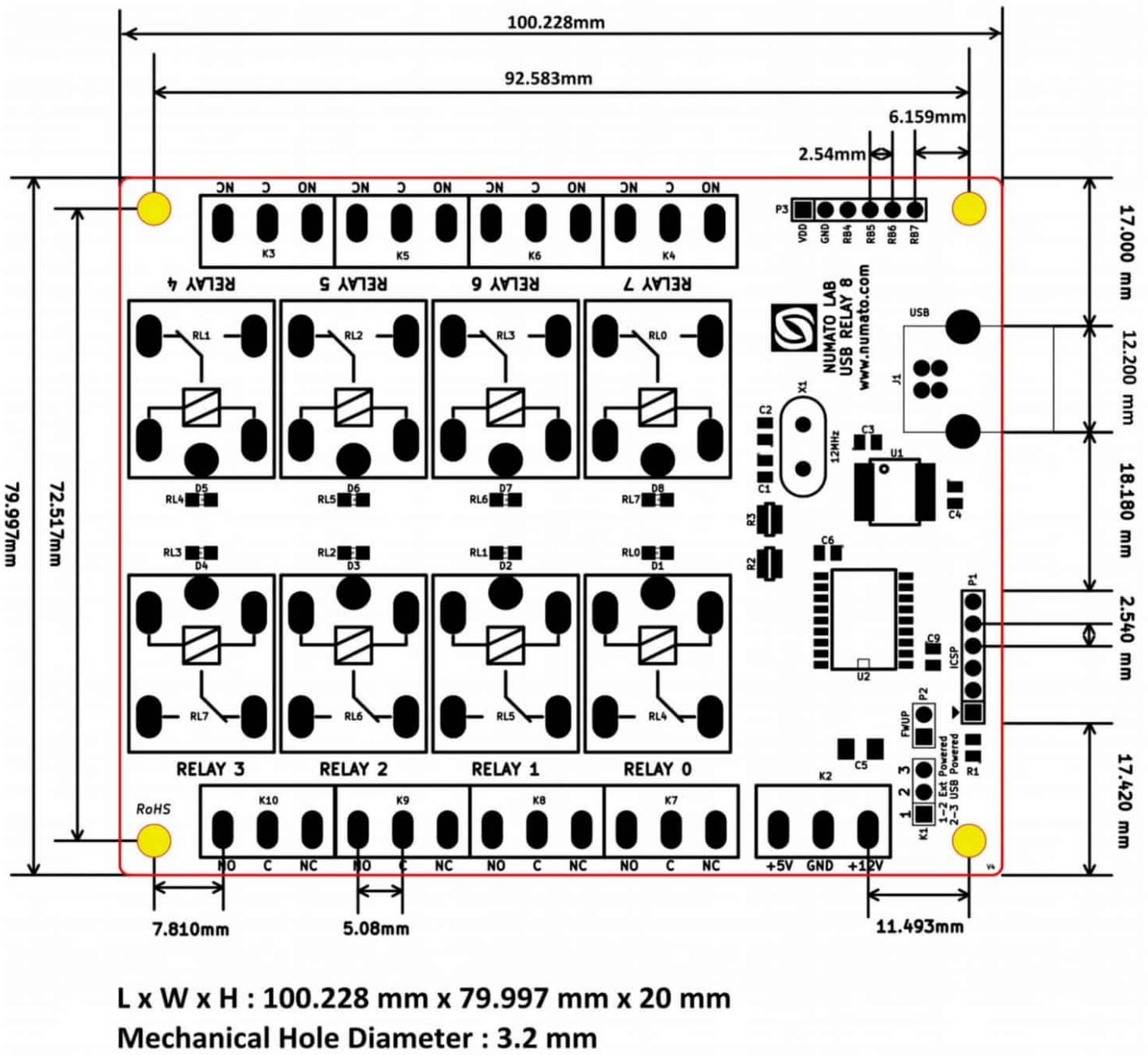
Q. I'm using x language for programming. How do I find out if this language can be used to program and control the GPIO module?
A. Find out if the language of interest supports some kind of APIs/Functions/Components for serial communication. If it does, most likely you should be able to use that language with this module.

Q. What is the connector marked as ICSP on this module?
A. This connector is used to program the on-board microcontroller. This connector is primarily intended for factory use.

Q. I need a customized version of this product, can Numato do the customization for me?
A. Yes, we can definitely do customization but there may be minimum order requirements depending on the level of customization required. Please write to sales@numato.com for a quote.

Q. Where can I buy this product?
A. All Numato products can be ordered directly from our web store http://www.numato.com (http://www.numato.com). We accept major credit cards and Paypal and ship to almost all countries with a few exceptions. We do have distributors in many countries where you can place your order. Please find the current list of distributors at http://numato.com/distrib (https://numato.com/where-to-buy/).

Mechanical Dimensions



(https://numato.com/blog/wp-content/uploads/2016/02/8ChannelUSBRelayMechanicalDimensions.jpg)

Suggest edit

(https://twitter.com/home?status=https://numato.com/docs/8-channel-usb-relay-module/)

(https://www.facebook.com/sharer/sharer.php?u=https://numato.com/docs/8-channel-usb-relay-module/)

(https://pinterest.com/pin/create/button/?url=https://numato.com/docs/8-channel-usb-relay-

module/&media=&description=) (https://plus.google.com/share?url=https://numato.com/docs/8-channel-usb-relay-module/) (http://www.linkedin.com/shareArticle?mini=true&url=https://numato.com/docs/8-channel-usb-relay-module/)

Make the best product decisions

Stay in the loop with new products, software, and tools!

email address

Subscribe

(https://twitter.com/numatolab) (https://www.facebook.com/numato/)

(https://www.youtube.com/user/NumatoLab) (https://plus.google.com/+Numatosystems)

Privacy Policy Terms of Use