Power Relay Module (PRM3) Software Documentation

Software documentation for the Power Relay Module PRM3

Mask control

Back to Main Power Relay Module (PRM3) Product Page.

The eGauge PRM3 has 3 independently controlled relay contacts. When controlling polyphase loads such as single or three-phase loads, the multiple relay contacts used should be open and closed simultaneously to avoid damaging the equipment. Neutral conductors always remain connected and are not switched by a relay.

To control relay contact simultaneously, the PRM3 interface provides commands to be used with masks. The command used will vary based on the interface used.

Command	**Description**
Mask	Any relay bit set (1) will be CLOSED, any bit unset (0) will be OPENED
Set	Any relay bit set (1) will be CLOSED, any bit unset (0) will be UNAFFECTED
Clear	Any relay bit set (1) will be OPENED, any bit unset (0) will be UNAFFECTED

List of Masks

Mask (Decimal)	**Mask (Binary)**	**Relays SET**
0	000	None
1	001	1
2	010	2
4	100	3
3	011	1 and 2
5	101	1 and 3
6	110	2 and 3

7	111	1, 2, and 3

Understanding the mask

The mask value is a set of 3 bits. The lowest bit (position 0) is for the lowest numbered relay (#1). The middle bit (position 1) is for the middle numbered relay (#2), and the highest bit (position 2) is for the highest numbered relay (#3).

The decimal value is simply the binary value, which can be determined by adding the set bits (1) values.

Depending on the command used, the set and unset bits will affect their respective relays differently.

`MASK` is the binary mask value sent or received
`RELAY` is the relay for the respective column values
`VALUE` is the decimal value for that bit
`BIT POS` is the bit position

MASK	0	0	0
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 0

MASK	0	0	1
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 1

MASK	0	1	0
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 2

MASK	1	0	0
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 4

MASK	0	1	1
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 3

MASK	1	0	1
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 5

MASK	1	1	0
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 6

MASK	1	1	1
RELAY	3	2	1
VALUE	4	2	1
BIT POS	2	1	0

Decimal Value: 7

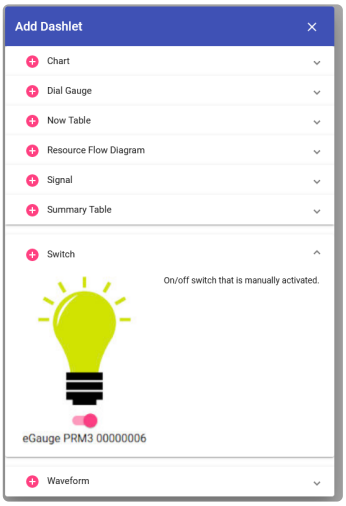
Dashboard control of PRM3

eGauge meter Dashboard Control

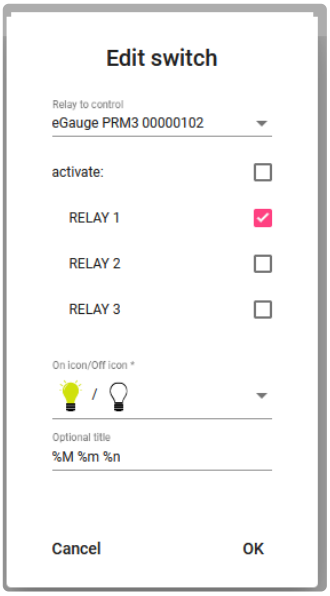
Back to Main Power Relay Module (PRM3) Product Page.

A "Switch" can be toggled to control the relay when connected to the eGauge via USB by using the Mobile-Friendly dashboard interface.

1. From the classic interface, click View -> Mobile Friendly.
2. Use the 3-dot menu, click View -> Dashboard to load the Dashboard interface.
3. Edit the dashboard to add a new dashlet.
4. Choose "Switch":



5. Exit the editing mode on the dashboard and click the newly added dashlet to configure it.
6. Click the drop-down for "Relay to control" and choose the PRM3, which relays the switch should control and the icon. Split-phase loads with a single hot and neutral will use 1 relay inputs, single-phase loads with 2 hots will use 2 relay inputs, and three-phase loads will use all 3 relay inputs.



7. The selected relay(s) can now be toggled by clicking the slider button under the icon:

Modbus Control of PRM3

Back to Main Power Relay Module (PRM3) Product Page.

The Power Relay Module acts as a Modbus server. As such, it responds to requests sent by a client. It never initiates a request on its own.

Connection and Power

Power

The eGauge PRM3 may be powered using the USB-A or the 4-pin port with a 5V/500mA power supply. Only power the PRM3 with one of these methods.

Communication

The PRM3 uses the 4-pin port for Modbus RTU RS-485 serial communication.

Modbus Registers

All registers are holding registers. They can be read with "Read Holding Registers" (function code 0x03) and written with "Write Single Register" (function code 0x06). The PRM3 uses base-0 addressing.

Address	Size	Name	Type	Description
0	1	RS-485 param	Unsigned 16-bit integer	Baud Rate code (bits 0..7) and parity (bits 8..15)*
1	1	Unit number	Unsigned 16-bit integer	Modbus unit number (1–247)
2	1	Relay Mask	Unsigned 16-bit integer	Relay status. On write, if bit (n-1) is set, relay n is closed, opened otherwise. On read, if bit (n-1) is set, relay n is closed, open otherwise.

3	1	Relay Set	Unsigned 16-bit integer	Close relays. On write, if bit (n-1) is set, relay n will be closed, unchanged otherwise. On read, if bit (n-1) is set, relay n is closed, open otherwise.
4	1	Relay Clear	Unsigned 16-bit integer	Open relays. On write, if bit (n-1) is set, relay n will be opened, unchanged otherwise. On read, if bit (n-1) is set, relay n is closed, open otherwise.
100	2	EEPROM writes	Unsigned 32-bit integer	Number of times EEPROM has been written.
102	2	Relay 1 Count	Unsigned 32-bit integer	Relay 1 switch count.
104	2	Relay 2 Count	Unsigned 32-bit integer	Relay 2 switch count.
106	2	Relay 3 Count	Unsigned 32-bit integer	Relay 3 switch count.
108	1	Min. open duration	Unsigned 16-bit integer	Minimum duration (in seconds) for which a relay stays open.
109	1	Min. close duration	Unsigned 16-bit integer	Minimum duration (in seconds) for which a relay stays closed.

RS-485 parameter register*

The “RS-485 param” register provides access to the RS-485 baud rate and parity.

The upper eight bits define the parity as shown below. Note the numerical value is the decimal value of the ASCII code, **not a binary value**.

ASCII code	Value (**decimal**)	Parity Mode
n	110	no parity
e	101	even parity
o	111	odd parity

The lower eight bits define the baud rate as shown below:

Value (decimal)	Baud Rate
1	9600 bps
2	19200 bps
4	38400 bps
6	57600 bps
12	115200 bps

For example, a value of 0x6506 would indicate 57600 bps and even parity.

Unit Number

The “Unit number” register defines the Modbus unit number under which the device responds. By default, this value is 1 but it can be set to any number in the range from 1 to 247.

Relay Commands

> For more information about using masks to control relays see **this article**.

The “Relay mask", “Relay set", and “Relay clear" registers provide access to the relays.

They all return the same value when **read**: in the returned value, if bit ($n - 1$) is set, it means that relay n is closed and if it is cleared, it means that the relay is open.

When **written**, the three registers have different behavior: “Relay mask" sets all the relays as indicated by the written value. That is, if bit ($n - 1$) is set, relay n will be closed and otherwise it will be opened. In contrast, writing “Relay set" will only close the relays for which the corresponding bit is set. Similarly, writing “Relay clear” will only open the relays for which the corresponding bit is set.

SunSpec block

The Power Relay Module also provides an address block to enable device identification according to the SunSpec standard.

Since SunSpec does not have a standardized model for relay controllers, the only model block provided is the Common Model (SunSpec DID 0x0001) as shown below. This block allows identifying the device by manufacturer and model name.

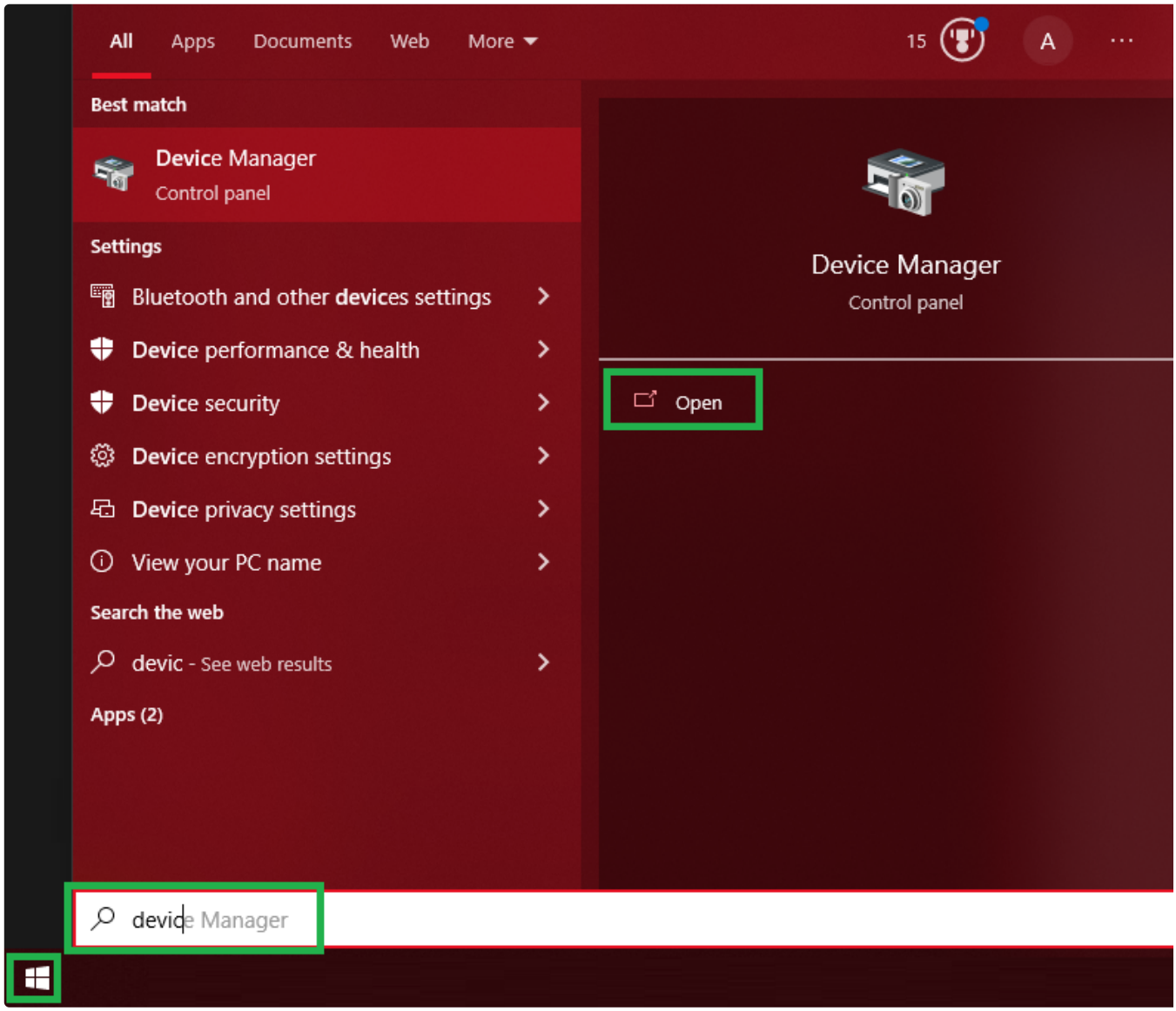
Address	Size	Name	Type	Description
40000	2	SunSpec_ID	Unsigned 32-bit integer	Value = 0x53756e53 ("SunS”).
40002	1	SunSpec_DID	Unsigned 16-bit integer	Value = 0x0001 (Common Model Block)
40003	1	SunSpec_Length	Unsigned 16-bit integer	Value = 65 (Length of block).
40004	16	Manufacturer	32-bit string	Manufacturer "eGauge".
40020	16	Model	32-bit string	Model name (e.g., “PRM3").
40036	8	Options	16-bit string	Installed options.
40044	8	Version	16-bit string	Product version (e.g., "1.00").
40052	16	SerialNumber	32-bit string	Serial number (e.g., "3N013453”).
40068	1	DeviceAddress	Unsigned 16-bit integer	Modbus unit number.
40069	1	SunSpec_DID	Unsigned 16-bit integer	Value = 0xffff (End Marker).
40070	1	SunSpec_Length	Unsigned 16-bit integer	Value = 0x0000.

Accessing the PRM3 from a Windows PC

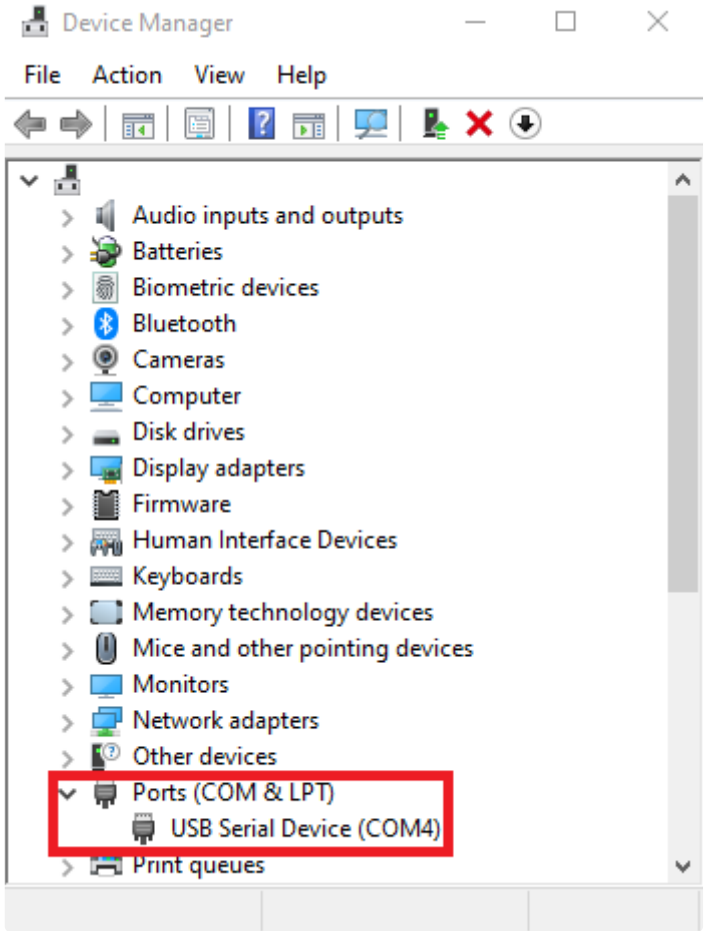
> eGauge Systems does not test or guarantee safety or accuracy of third party software.

Locate the COM port of the PRM3

1. Open the Device Manager, which can be done by opening the Start Menu and typing "Device Manager" and clicking "Open":



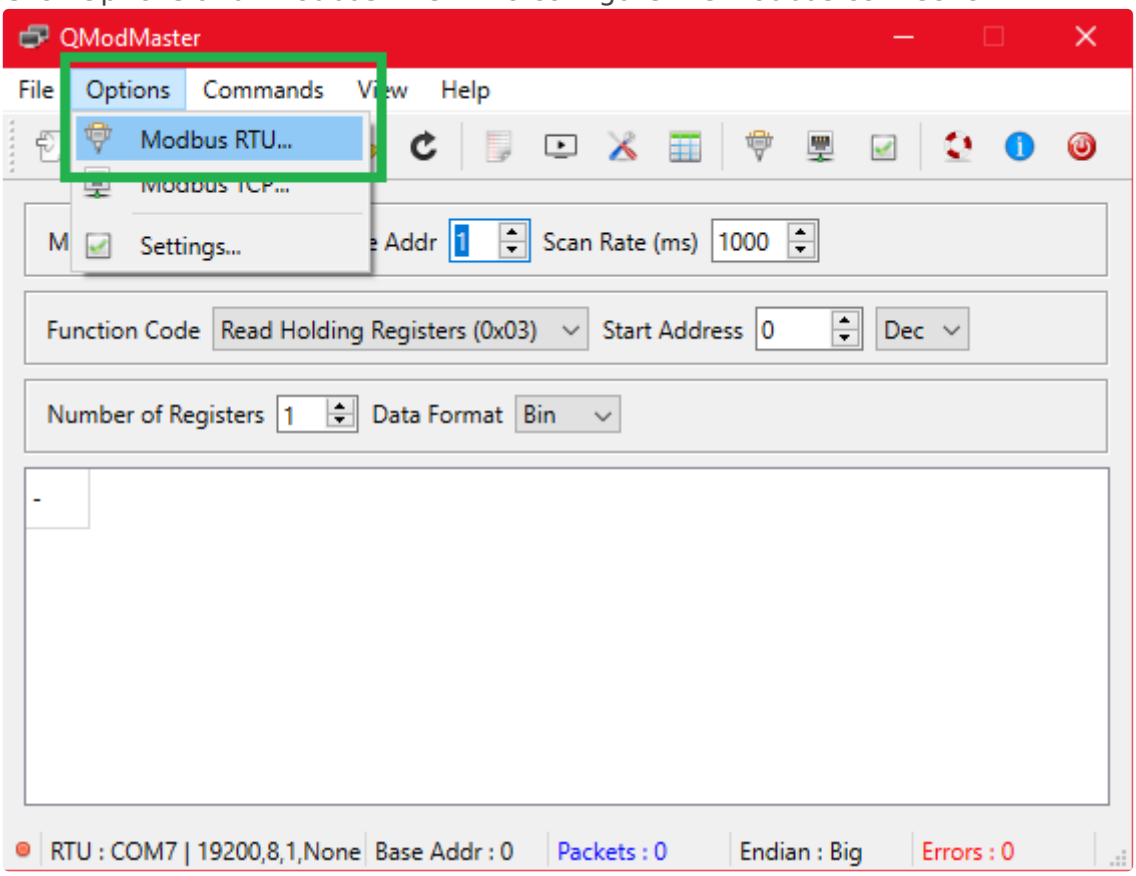
2. Expand the "Ports (COM & LPT)" section:



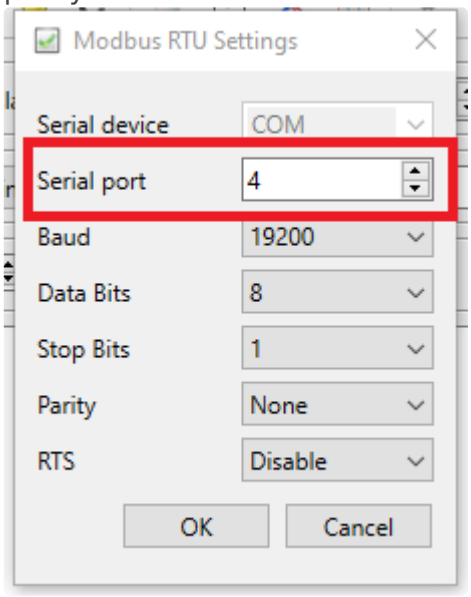
If you have multiple "USB Serial Device" entries, you may unplug the eGauge PRM3, and plug it back in to see which COM port appears when it is connected.

Using QModMaster to communicate with relay

1. Install and open the QModMaster Modbus master simulator.
2. Click Options and "Modbus RTU..." to configure the Modbus connection

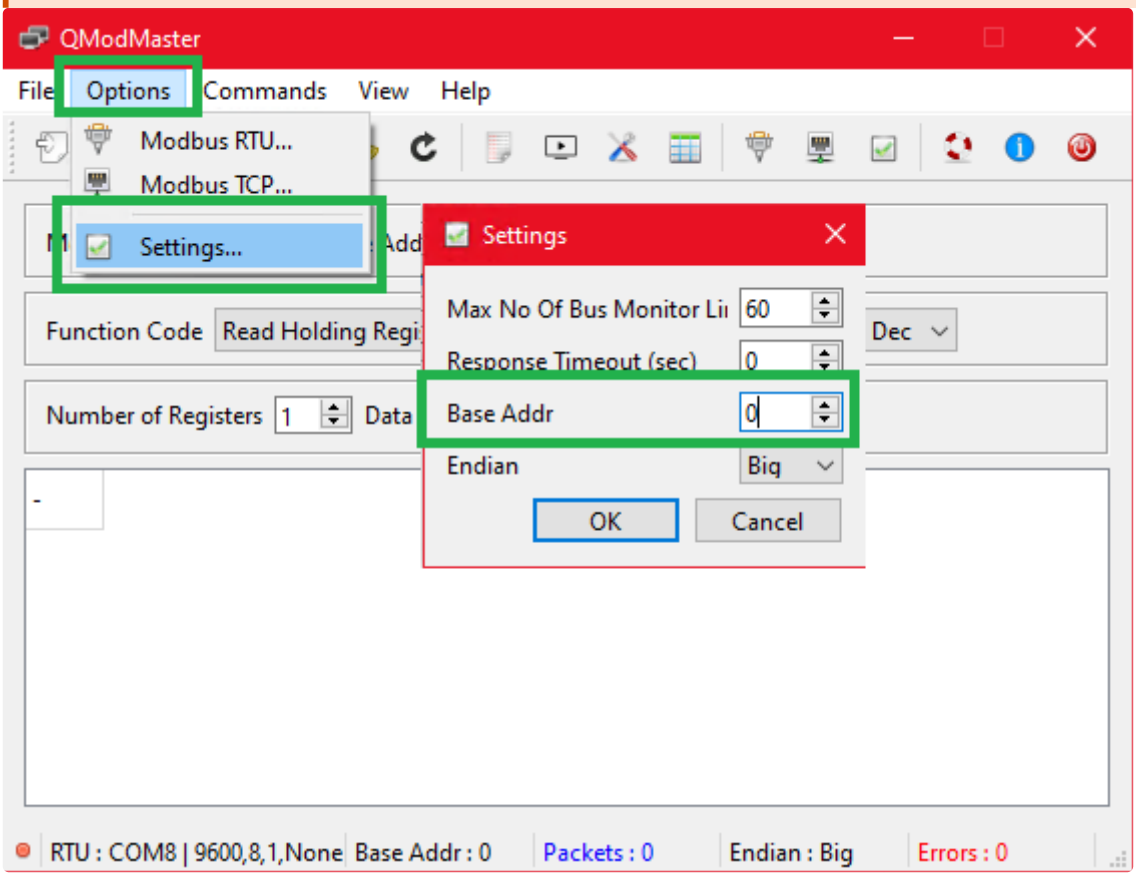


3. Change the COM port number and other parameters if necessary and press OK. The defaults RS-485 settings for the eGauge PRM 3 are 19200 baud, 8 data bits, 1 stop bit, no parity:

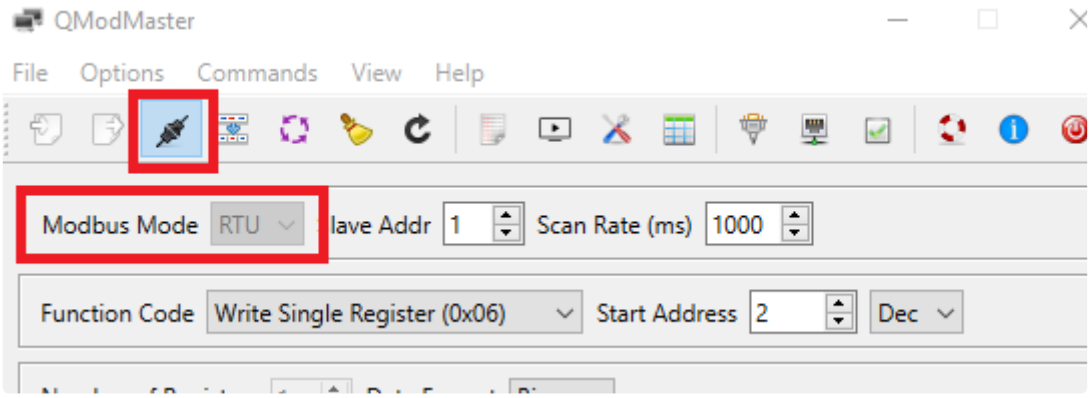


4. Open the Options menu again and go to Settings, and change Base Addr to the number 0:

> Using an incorrect "Base Address" will result in reading or writing to a register one address lower or higher than the intended register address!

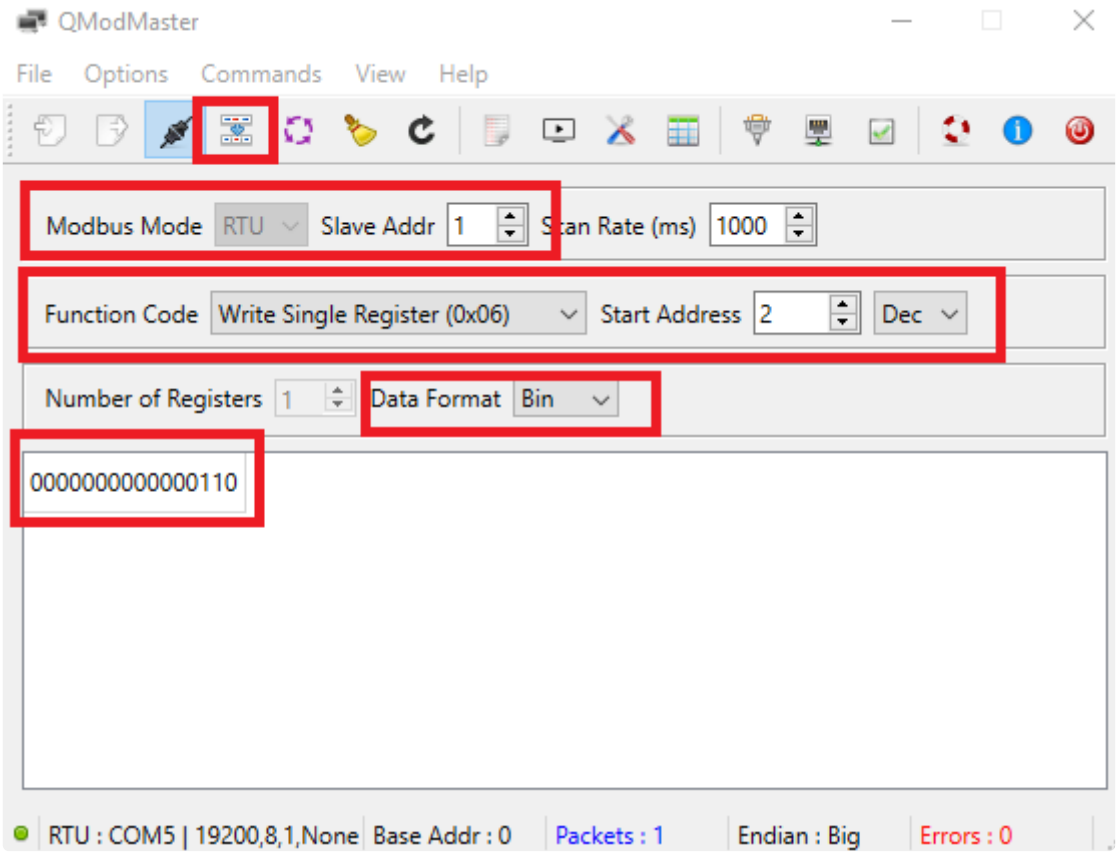


5. Ensure Modbus Mode is set to RTU and click the "connect" button in the toolbar below the "Options" and "Commands" menus:

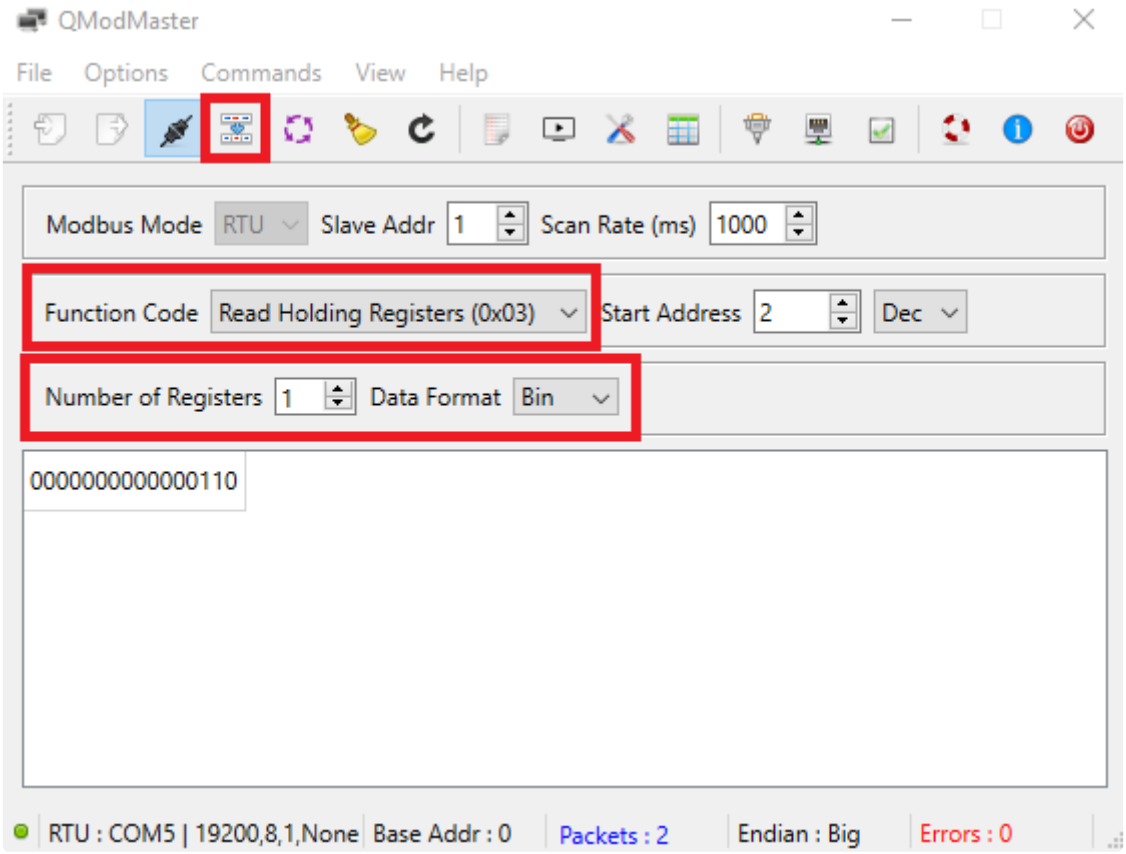


6. First, we will set the mask of the relay to 6 (110) to open relay 1, and close relays 2 and 3.
   - Ensure the Slave Address (Unit Number) is set correctly. The PRM3 by default is set to 1
   - Choose "Write Single Register (0x06)"
   - Set Start Address 2 (decimal format)
   - Choose "Bin" (binary) for Data Format, this is easier to work with masks
   - In the box enter "110" and click outside the number entry box to save it there
   - Click the "Read/Write" button next to the connect button

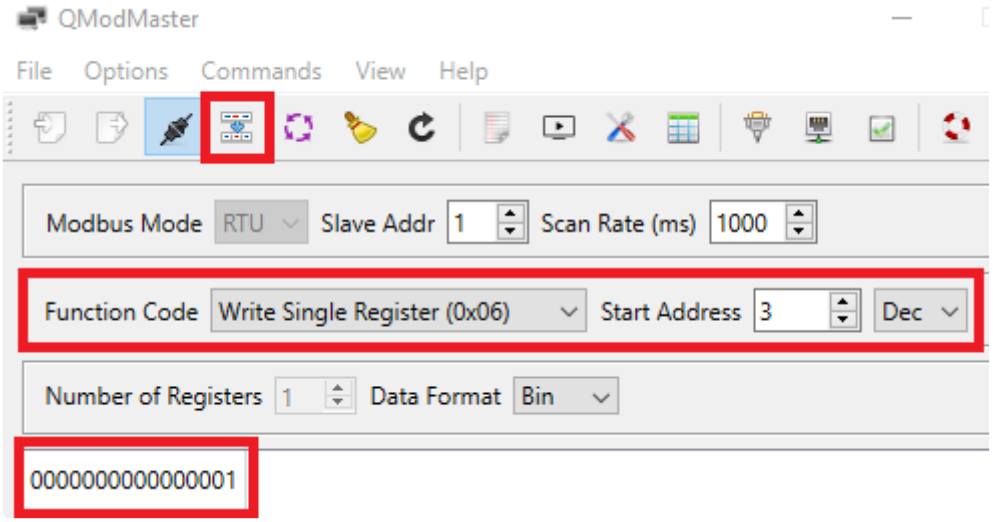
   After sending, the "Packets" (blue text at bottom) number should increment.



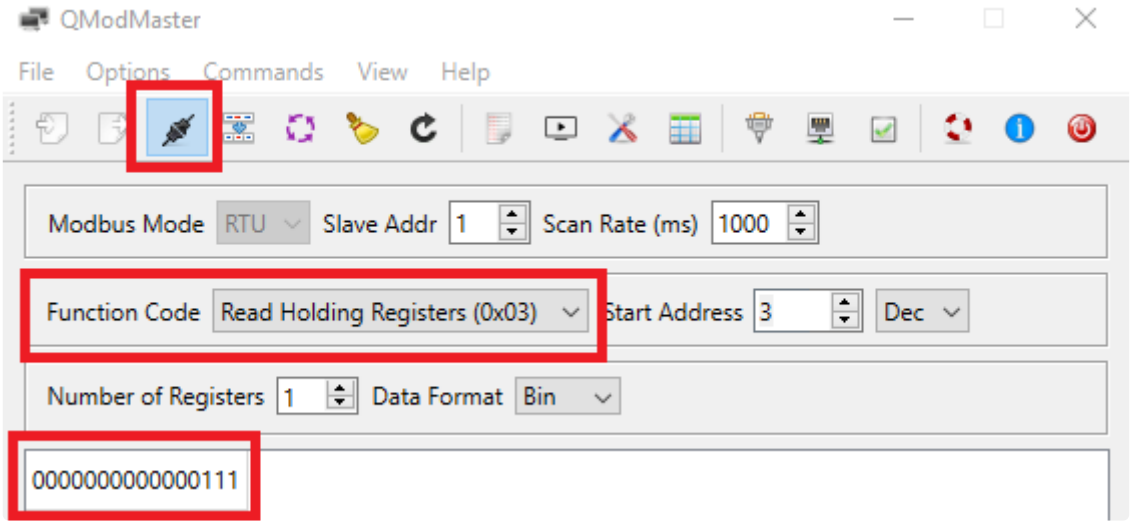
7. Read back the register to confirm the mask was set. Change Function Code to "Read Holding Registers (0x03)", and ensure "Number of Registers" is set to 1 (as it is a 16-bit value) and Data Format as "Bin". Click the Read/Write button again and the Packets should increment. The box should still show "110", meaning relay 1 is open, and relays 2 and 3 are closed.



8. Let's turn relay 1 on and leave 2 and 3 alone, we will use the "Relay Set" register for this (address 3). Change the "Start Address" to 3, the Function Code back to Write Single Register, and the 110 in the text box to 001 (leading zeros are not necessary). This tells the PRM3 to close relay 1, and leave 2 and 3 in their current state. Press Read/Write and note the packet increment:



9. Finally, we'll read current the mask state of the relay. Change the Function Code to Read Holding Registers and click the Read/Write button. We can read from any of the Relay registers to get the current mask from the relay. Note the packet increase. The text area below shows the data received, which is 111 to indicate all 3 relays are closed.



Lua Scripting with PRM3

Back to Main Power Relay Module (PRM3) Product Page.

The eGauge meter provides a Lua scripting environment for interaction and control of the eGauge PRM3 Power Relay Module when connected to the eGauge via USB.

Navigating to the Lua Scripting interface

Control script

> See the main Lua Scripting Overview Control Scripts section for additional Lua Control environment information.

> There is risk of damaging external equipment using control scripts. Only skilled Lua developers familiar with the eGauge meter and software should attempt to use Lua control scripts.

Control scripts can be used to confrol supported equipment such as the eGauge Power Relay Module (PRM3).

For example, the following script reads the instantaneous value of a register called "Temperature" and controls a PRM3 relay contact. If the temperature is lower than 21 C, relay number 0 of the PRM3 is closed (activated), otherwise opens (turns off) relay number 0. It then sleeps for 15 minutes before checking again.

In the real world, the control script should be more advanced

```
dev = ctrl:dev({interface='relay'})
 relay = dev:interface('relay')


 while true do
  print("Temperature is currently: " .. __r("Temperature"))
  if __r("Temperature") < 21 then
     relay:close(0)
  else
     relay:open(0)
  end
  sleep(60*15)
 end
```

SCPI Control of PRM3

Back to Main Power Relay Module (PRM3) Product Page.

SCPI introduction

SCPI (pronounced "skippy") stands for "Standard Commands for Programmable Instruments" and uses ASCII encoded strings. Com mands are entered one line at a time. Each line must end with a carriage-return (ASCII code 13) and/or line-feed character (ASCII code 10). Line length (including line terminators) is limited to at most 64 characters. Commands are caseinsensitive, so `RELAY` has the same meaning as `relay`, for example.

Most SCPI commands may be abbreviated to the first four characters. Required command characters are shown in upper case, optional ones in lower case. For example, `MODBus` indicates that the command may be abbreviated to just `MODB`.

Each SCPI command returns a single response-line which is terminated by a carriage-return line-feed sequence. The response is `INVALID COMMAND` if there was an error processing the command. If the command was processed successfully the response is `OK` or a command-specific response.

> While SCPI convention normally would allow specifying multiple commands in a single line by separating the commands with a semicolon, the Power Relay Module does not support this convention and always expects a single command per line.

Connection and Power

The eGauge PRM3 unit communicates SCPI and gets power over the USB-A connection. The USB connection provides a CDC ACM virtual serial port for the host to communicate with using serial.

Serial Settings

The SCPI interface uses the following serial parameters:

- 19200 baud
- 1 start bit
- 8 data bits, LSB first
- no parity
- 1 stop bit

SCPI commands

he SCPI commands supported by the Power Relay Module are shown below. The first column shows the syntax of the command, the second column the response type, and the third is a description of the command.

For response type devid, the return value consists of a string containing the manufacturer name, model name, product serial-number, and the product version, separated by commas. For example, the returned devid might be `eGauge,PRM3,3N013453,1.00`.

For response type status, the return value consists of either `OK` or `INVALID COMMAND`. For response type decimal, the return value consists of either `INVALID COMMAND` or a decimal integer number string. For response type parity, the return value consists of either `INVALID COMMAND` or a single character, where the character `n` indicates no parity, `e` indicates even parity, and `o` indicates odd parity.

> When controlling polyphase loads, the **mask commands** should be used for simultaneous opening and closing of the multiple relay inputs.

Command	Response	Description
`*IDN?`	devid	Return device identifier
`EPRom?`	decimal	Return number of times the EEPROM has been written.
`RELAy:`*n*`?`	decimal	Query status of relay *n*, where *n* is one of 1, 2, or 3. Returns string `0` if relay is open, `1` if it is closed.
`RELAy:`*n cv*	status	Open or close relay *n*, where *n* is one of 1, 2, or 3. If *cv* is `0` or `OFF`, the relay is opened, if `1` or `ON`, the relay is closed.
`RELAy:`*n*`:COUNt?`	decimal	Return number of times relay *n* has been switched (opened or closed). The value of *n* must be one of 1, 2, or 3.
`RELAy:MASK?`	decimal	Query status of all relays. The returned number has bit (*n*-1) set if relay *n* is closed, cleared otherwise. For example, return value 6 would indicate that relay 1 is open (bit 0 is cleared) and relays 2 and 3 are closed (bits 1 and 2 are set).
`RELAy:MASK` *m*	status	Open or close relays as indicated by mask m. If bit (*n*-1) is set, relay *n* is closed, otherwise it will be opened.

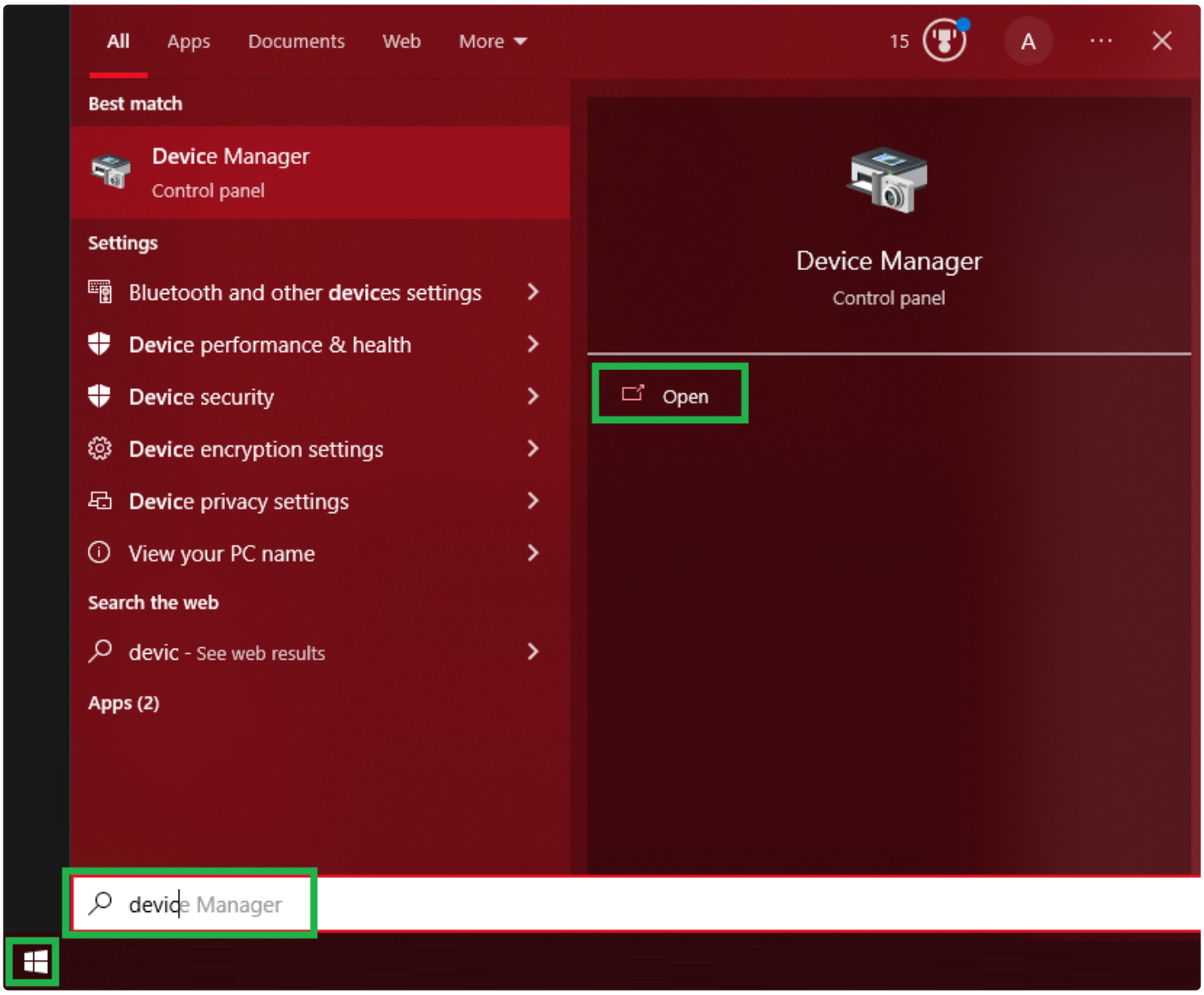
RELAy:MASK:SET *m*	status	Close relays as indicated by mask *m*. If bit (*n*-1) is set, relay *n* is closed, otherwise relay *n* will remain in its current state.
RELAy:MASK:CLR *m*	status	Open relays as indicated by mask *m*. If bit (*n*-1) is set, relay *n* is opened, otherwise relay *n* will remain in its current state.
RELAy:MIN:OFF?	decimal	Query the minimum duration for which a relay remains open. The returned number is the duration in seconds.
RELAy:MIN:OFF *d*	status	Set the minimum duration for which a relay remains open to *d* seconds. The duration must be an integer in the range from 0..255.
RELAy:MIN:ON?	decimal	Query the minimum duration for which a relay remains closed. The returned number is the duration in seconds
RELAy:MIN:ON *d*	status	Set the minimum duration for which a relay remains open to *d* seconds. The duration must be an integer in the range from 0..255.
MODBus:BAUD?	decimal	Returns the baud rate of the RS-485 port.
MODBus:BAUD *n*	status	Sets the RS-485 baud rate to *n* baud. The value of *n* may be one of 9600, 19200, 38400, 57600, or 115200.
MODBus:PARIty?	decimal	Returns the parity used for the RS-485 port.
MODBus:PARIty *p*	status	Sets the RS-485 parity. If *p* is `n`, no parity is selected, if `e`, even parity is selected, and if `o`, odd parity is selected.
MODBus:UNIT?	decimal	Returns the MODBUS unit number of the device.
MODBus:UNIT *n*	status	Sets MODBUS unit number of the device to `n`. The value of `n` may be in the range from 1 through 247.

Accessing the PRM3 from a Windows PC

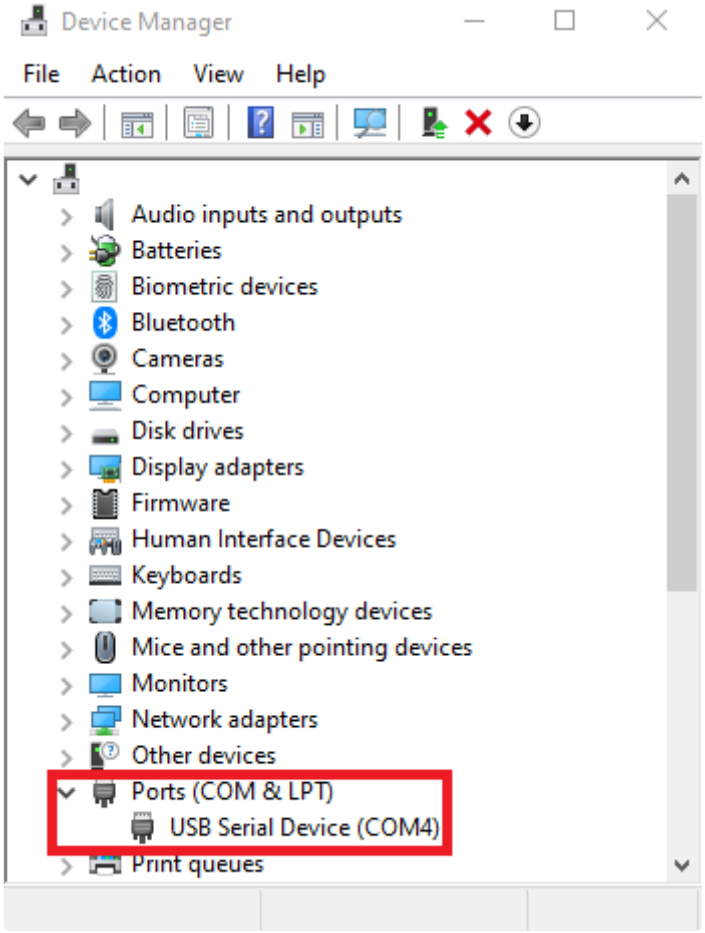
> eGauge Systems does not test or guarantee safety or accuracy of third party software.

Locate the COM port of the PRM3

1. Open the Device Manager, which can be done by opening the Start Menu and typing "Device Manager" and clicking "Open":



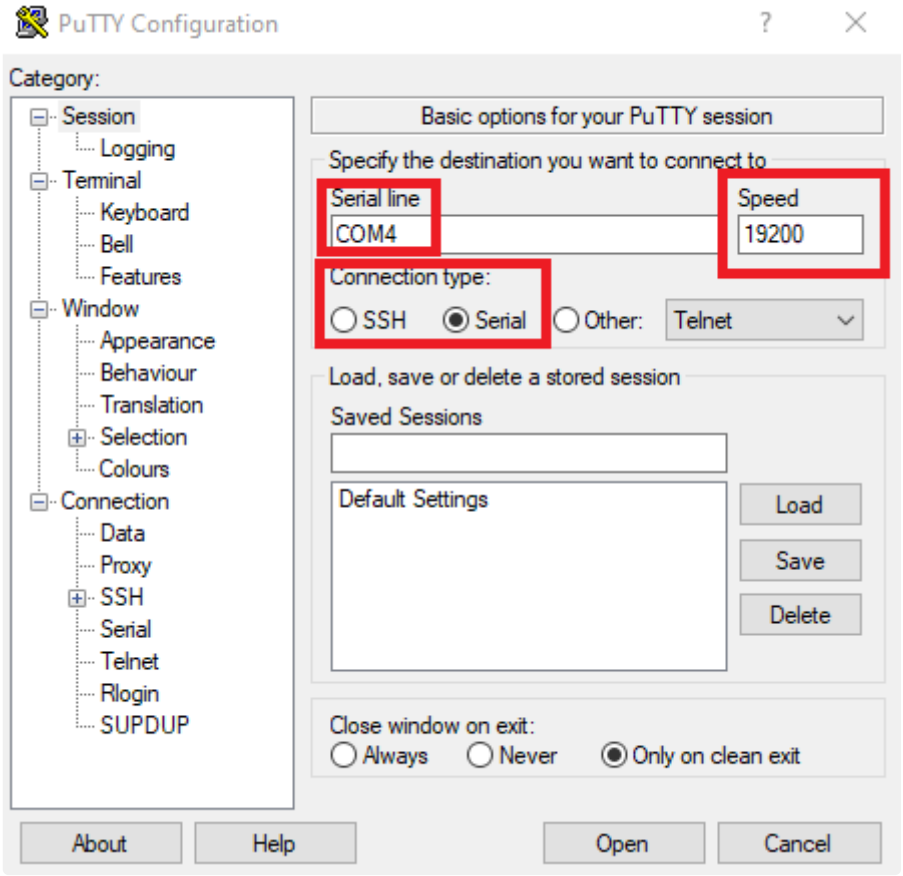
2. Expand the "Ports (COM & LPT)" section:



If you have multiple "USB Serial Device" entries, you may unplug the eGauge PRM3, and plug it back in to see which COM port appears when it is connected.

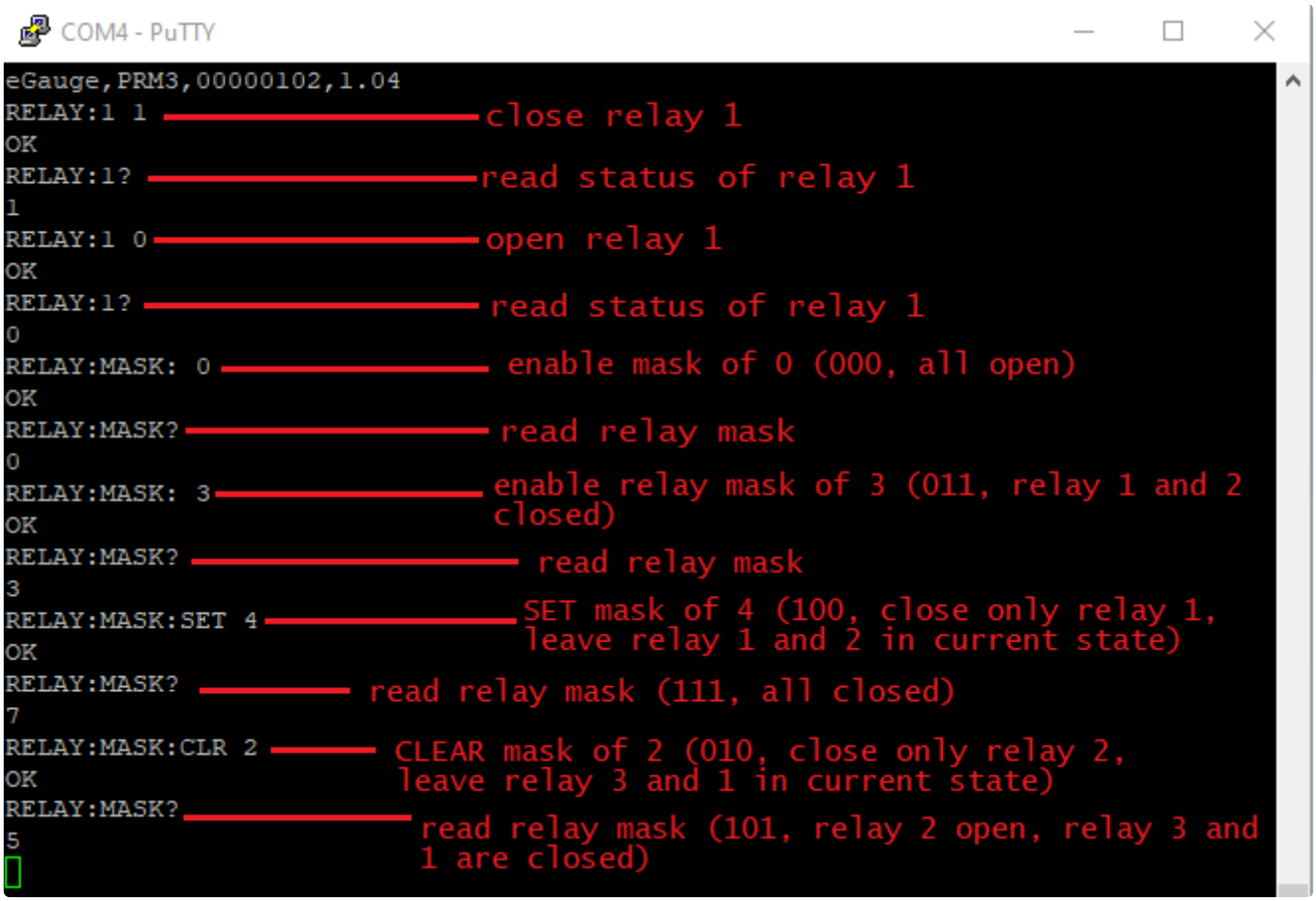
Connect to the COM port using PuTTY

1. Install and open the PuTTY terminal emulator.
2. Change "Serial line" to the COM port found in the device manager, the "Speed" to 19200 and "Connection type" to "Serial" and press Open:



3. You may now enter SCPI commands followed by the "Enter" key.

> Back-spaces may not work correctly and result in an invalid command error. Commands entered using copy and paste may also not work correctly and result in an invalid command error.


COM4 - PuTTY
eGauge,PRM3,00000102,1.04
RELAY:1 1
close relay 1
OK
RELAY:1?
read status of relay 1
1
RELAY:1 0
open relay 1
OK
RELAY:1?
read status of relay 1
0
RELAY:MASK: 0
enable mask of 0 (000, all open)
OK
RELAY:MASK?
read relay mask
0
RELAY:MASK: 3
enable relay mask of 3 (011, relay 1 and 2 closed)
OK
RELAY:MASK?
read relay mask
3
RELAY:MASK:SET 4
SET mask of 4 (100, close only relay 1, leave relay 1 and 2 in current state)
OK
RELAY:MASK?
read relay mask (111, all closed)
7
RELAY:MASK:CLR 2
CLEAR mask of 2 (010, close only relay 2, leave relay 3 and 1 in current state)
OK
RELAY:MASK?
read relay mask (101, relay 2 open, relay 3 and 1 are closed)
5

JSON WebAPI relay control

Back to Main Power Relay Module (PRM3) Product Page.

> WebAPI documentation may be found at https://egauge.net/support/webapi. The `/ctrl` endpoint is used for controlling the PRM3.

> The eGauge Python Library contains helper functions for authentication and requests and is available via Bitbucket and PyPi. Demonstration code provided by eGauge Systems typically requires this library.

> To interact with the WebAPI control service, the authenticated user must have permission "Allowed to view all data, change settings, and control devices from anywhere"

Starting in firmware v4.4, the PRM3 may be controlled with EG4xxx series meters by using the eGauge JSON-based WebAPI /ctrl endpoint. The following is a Python script showing some possible miscellaneous interactions with an eGauge PRM3.

```
#!/usr/bin/env python3

# Example script interacting with an eGauge PRM3 power relay module through a
# meter's WebAPI. The PRM3 must be connected to the eGauge via USB.
#
# This script uses the eGauge Python library, available from bitbucket or pip:
# https://bitbucket.org/egauge/python/src/master/egauge/
# https://pypi.org/project/egauge-python/
# WebAPI documentation: https://egauge.net/support/webapi

# Requires firmware version 4.4 or greater

from egauge import webapi

URI = "https://device-url"
USR = "my_meter_username"
```

```
PWD = "my_meter_password"

dev = webapi.device.Device(URI, webapi.JWTAuth(USR, PWD))

USB_PORT = "USB1" # what USB port the relay is connected to



# get the connected relay(s) information
relays = dev.get("/ctrl/device")
"""e.g.,
{
   'result':
   [
      {
         'path': ['net.egauge.slowd', 'USB1'],
         'mfg': 'eGauge',
         'model': 'PRM3',
         'sn': '00000004',
         'prot': 'SCPI',
         'link': 'serial',
         'quality': '1',
         'interface': ['relay', 'scpi']
      }
   ]
}
"""

# get the serial number of the PRM3 connected to port USB_PORT
# we use the SN to identify which relay to send commands to
for relay in relays["result"]:
   if USB_PORT in relay["path"]:
      relay_sn = relay["sn"]



# get the control interfaces and available methods for the PRM3 relay
# e.g., open_mask, set_mask, get_mask
methods = dev.get("/ctrl/interface")["result"]



# get the relay mask
# https://egauge.net/support/m/prm3/mask
```

```
payload = {
   "attrs": {"sn": relay_sn},
   "method": "relay.get_mask",
   "args": []
}

# store the transaction ID of the request
# e.g., {'result': {'tid': 897412938}}
tid = dev.post("/ctrl/call", payload)["result"]["tid"]

# get the response. e.g., {'result': 3} means relays 0 and 1 are closed, and
# 2 is open https://egauge.net/support/m/prm3/mask
print(dev.get(f"/ctrl/call/{tid}")["result"])



# set the mask to 5 (close relay 0 and 2, open relay 1).
# Use relay.open_mask and relay.close_mask to only open OR close relays
payload = {
   "attrs": {"sn": relay_sn},
   "method": "relay.set_mask",
   "args": [5]
}

# make the request, store the transaction ID
tid = dev.post("/ctrl/call", payload)["result"]["tid"]

# check the transaction response. This should generally be {'result': False}
print(dev.get(f"/ctrl/call/{tid}")["result"])



# we can also send arbitrary SCPI commands, such as configurations.
# here we set the Modbus baud rate to 9600:
payload = {
   "attrs": {"sn": relay_sn},
   "method": "scpi.exec",
   "args": ["MODBus:BAUD 9600"]
}

tid = dev.post("/ctrl/call", payload)["result"]["tid"]

# should be "OK"
```

```
print(dev.get(f"/ctrl/call/{tid}")["result"])



# verify the baud
payload = {
    "attrs": {"sn": relay_sn},
    "method": "scpi.exec",
    "args": ["MODBus:BAUD?"]
}

tid = dev.post("/ctrl/call", payload)["result"]["tid"]

# {'result': '9600\r\n'}
print(dev.get(f"/ctrl/call/{tid}")["result"])
```