ICOP Technology Inc.

Start Guide

QEC-M-02: MQTT Communication to Node-RED UI with 86EVA

86Duino Coding IDE 501

EtherCAT Library

(Version 1.0)

Revision

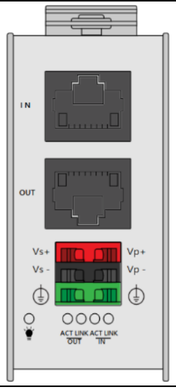
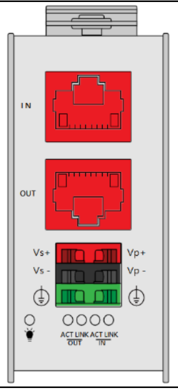
Date	Version	Description
2026/3/3	Version1.0	New release.

Preface

In this guide, we will show you how to use the EtherCAT MDevice QEC-M-02 and the QEC-R11MP3S series (EtherCAT 3-axis Stepper Motor Controller). Data is transmitted via MQTT to a Mosquitto broker and visualized in real time using Node-RED on a PC dashboard.

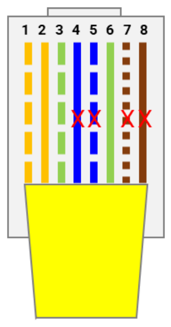
Notes QEC's PoE (Power over Ethernet)

In QEC product installations, users can easily distinguish between PoE and non-PoE: if the RJ45 house is red, it is PoE type, and if the RJ45 house is black, it is non-PoE type.

 IN OUT Vs+ Vs - Vp+ Vp - ACT LINK ACT LINK OUT IN	 IN OUT Vs+ Vs - Vp+ Vp - ACT LINK ACT LINK OUT IN
Non-PoE type	PoE type

PoE (Power over Ethernet) is a function that delivers power over the network. QEC can be equipped with an optional PoE function to reduce cabling. In practice, PoE is selected based on system equipment, so please pay attention to the following points while evaluating and testing:

1. The PoE function of QEC is different and incompatible with EtherCAT P, and the PoE function of QEC is based on PoE Type B, and the pin functions are as follows:



T568B Color

PINS	10/100 DC on Spares (Mode B)
1	Rx+
2	Rx-
3	Tx+
4	DC+
5	DC+
6	Tx-
7	DC-
8	DC-

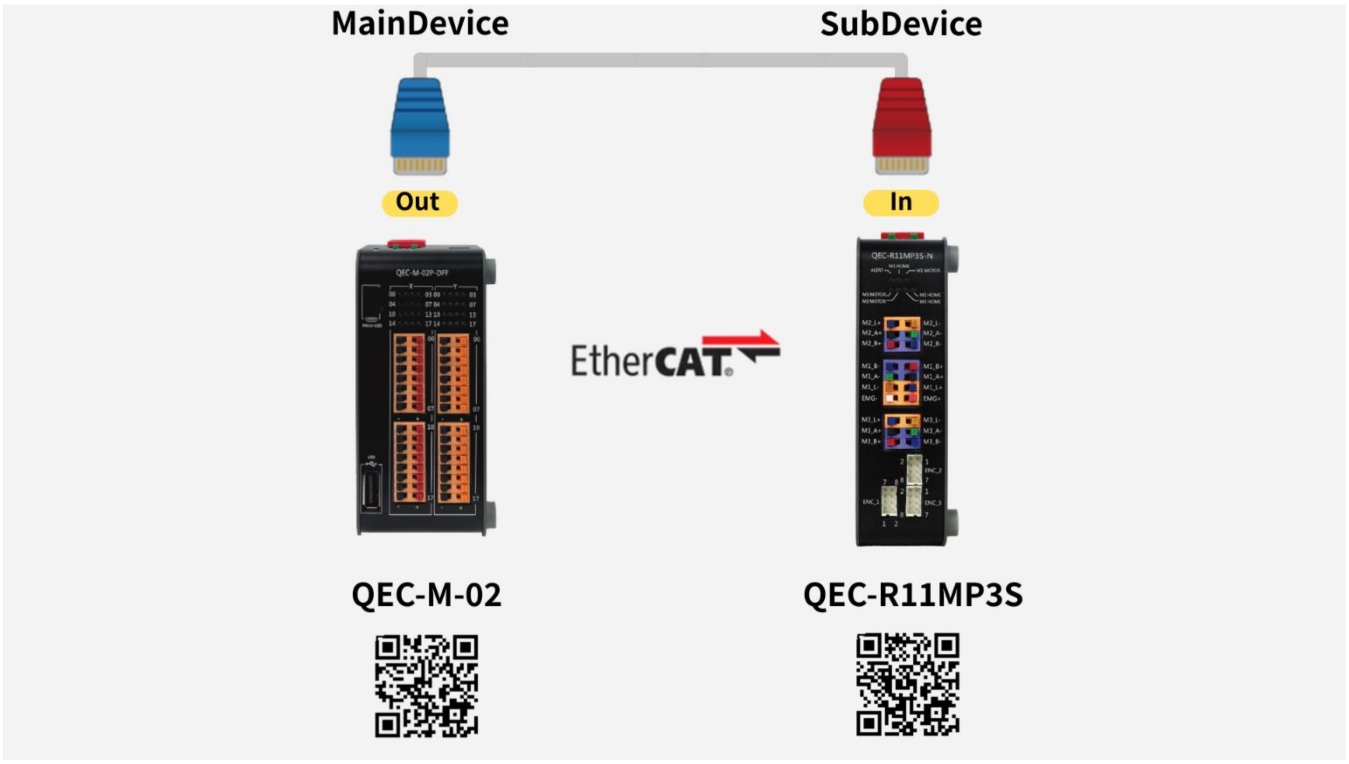
2. When connecting PoE and non-PoE devices, make sure to disconnect Ethernet cables at pins 4, 5, 7, and 8 (e.g., when a PoE-supported QEC EtherCAT MDevice connects with a third-party EtherCAT SubDevice).

3. QEC's PoE power supply is up to 24V/3A.

1. Connection and wiring hardware

The following devices are used here:

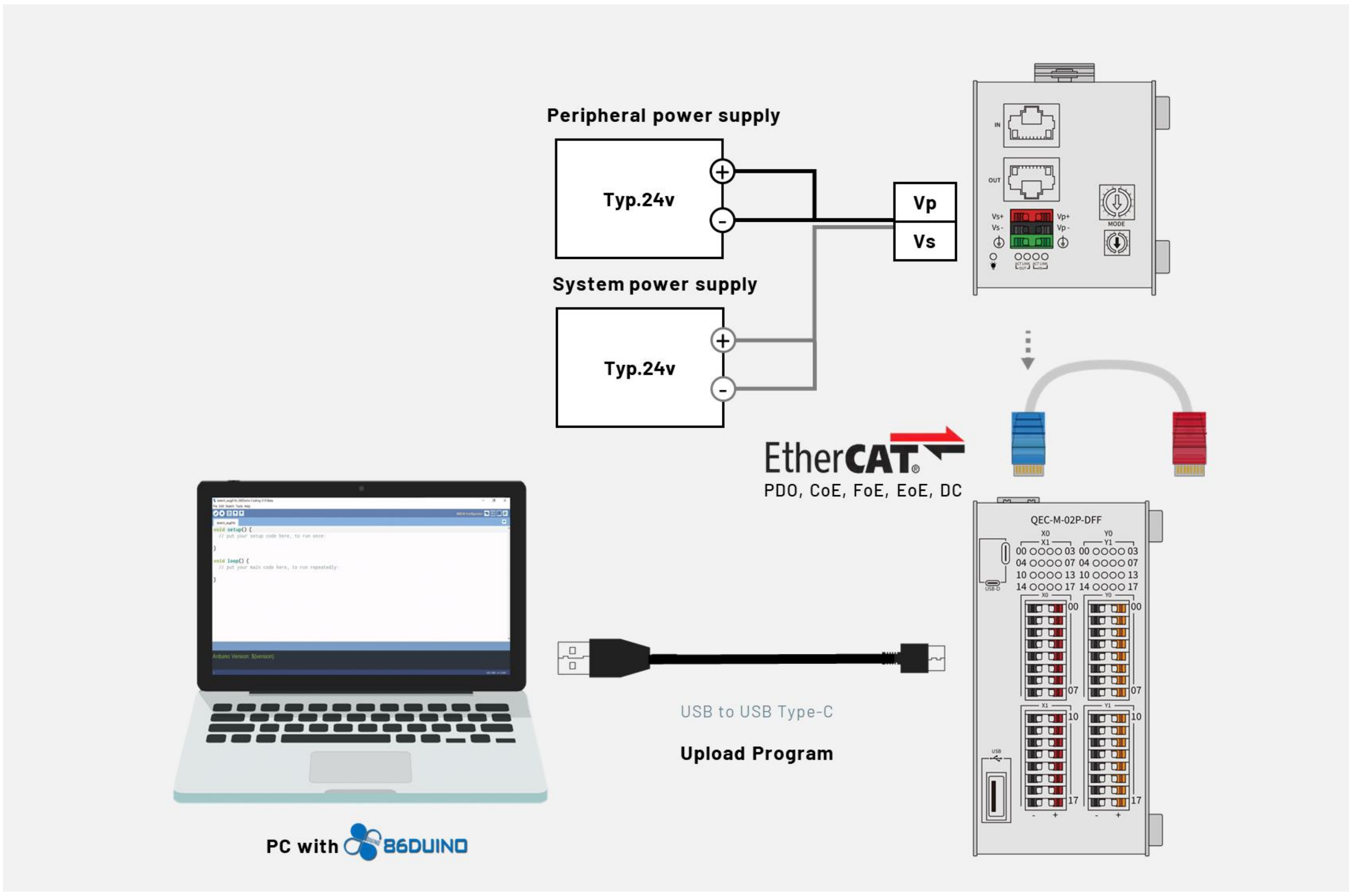
1. QEC-M-02 (EtherCAT MDevice)
2. QEC-R11MP3S series (EtherCAT 3-axis Stepper Motor Controller).
3. A 4-wire two-phase bipolar 42 stepper motor (refer to 86STEP | 86Duino)
4. 24VDC power supply & EU-type terminal cable & LAN cable



1.1 QEC-M-02

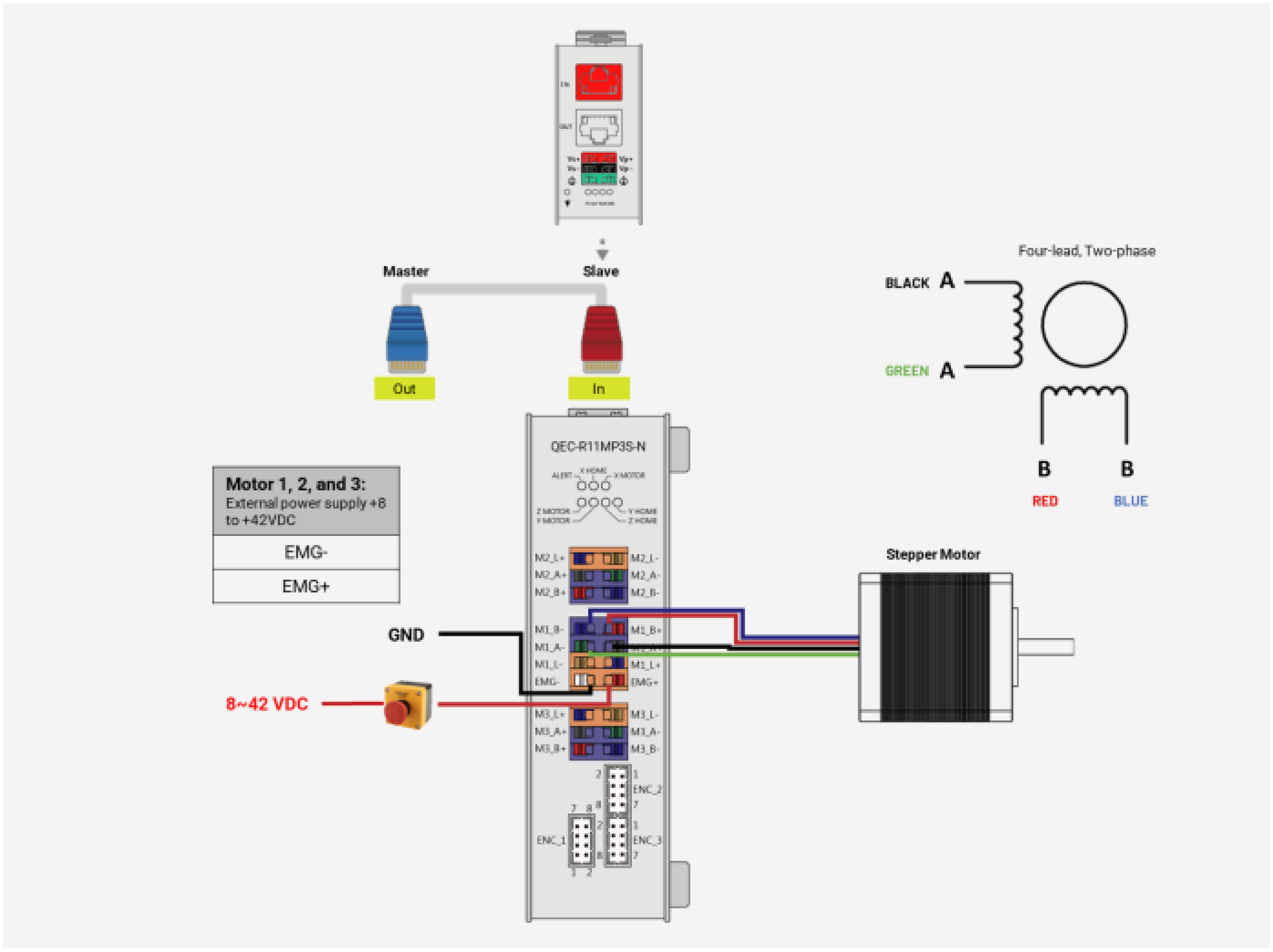
QEC EtherCAT MDevice.

1. Power Supply: Connect to Vs+/Vs- and Vp+/Vp- power supplies via EU terminals for 24V power.
2. EtherCAT Connection: Using the EtherCAT Out port (On the top side) connected to the EtherCAT In port of EtherCAT SubDevice via RJ45 cable.



1.2 QEC-R11MP3S-N

The QEC-R11MP3S-N with PoE function.



1. Connecting an 8–42V power supply to EMG+/- is required.

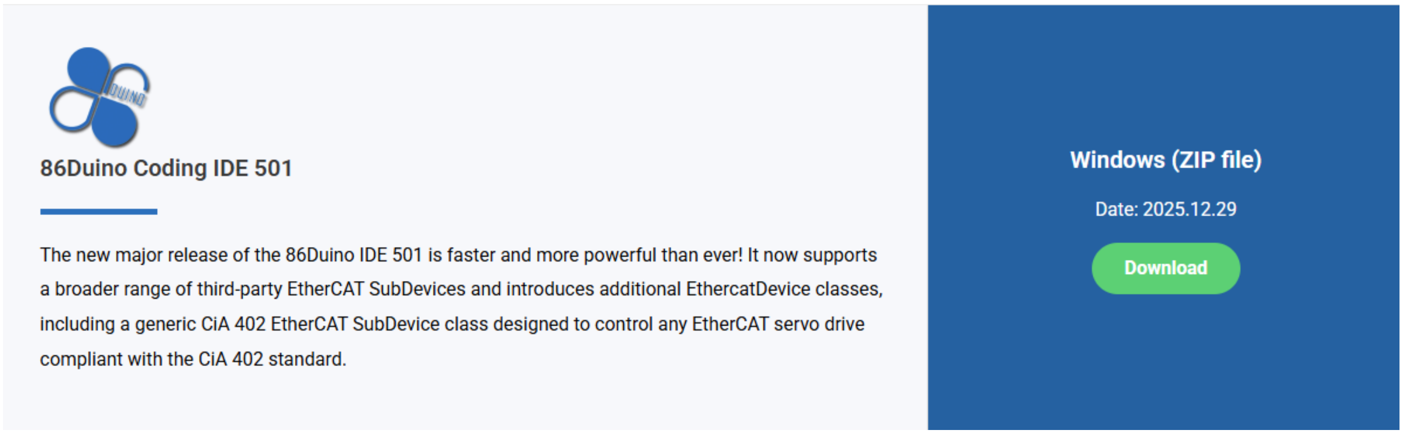
Note: The Emergency Stop is a safety mechanism for quickly shutting down machinery in emergencies via a hardware switch.

2. Connect the 4-wire bidirectional 42-stepper motor to the M1 axis (middle) of the QEC-R11MP3S-N as follows:
   - A+ (black) connects to M1_A+; A- (green) connects to M1_A-.
   - B+ (red) connects to M1_B+; B- (blue) connects to M1_B-.

Note: The wiring colors of the motor are the same as the terminal colors of the QEC-R11MP3S-N, users can follow the color for wiring reference.

2. Software/Development Environment

Download 86duino IDE from https://www.qec.tw/software/.



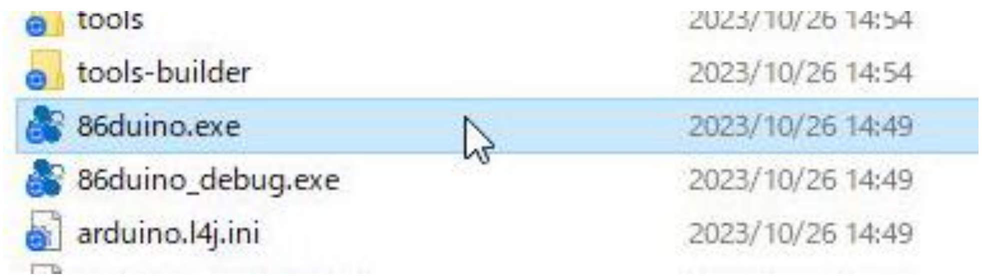
86Duino Coding IDE 501

The new major release of the 86Duino IDE 501 is faster and more powerful than ever! It now supports a broader range of third-party EtherCAT SubDevices and introduces additional EthercatDevice classes, including a generic CiA 402 EtherCAT SubDevice class designed to control any EtherCAT servo drive compliant with the CiA 402 standard.

Windows (ZIP file)

Date: 2025.12.29

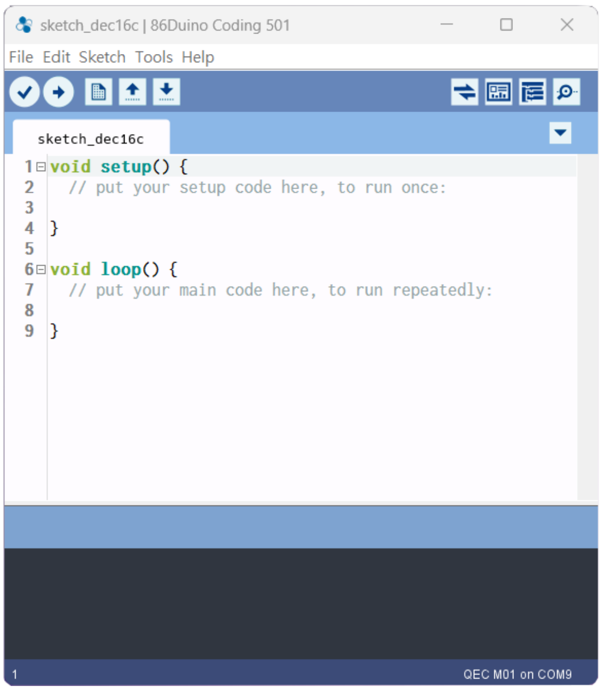
After downloading, please unzip the downloaded zip file, no additional software installation is required, just double-click 86duino.exe to start the IDE.



tools	2023/10/26 14:54
tools-builder	2023/10/26 14:54
86duino.exe	2023/10/26 14:49
86duino_debug.exe	2023/10/26 14:49
arduino.l4j.ini	2023/10/26 14:49

***Note**: If Windows displays a warning, click Details once and then click the Continue Run button once.

86Duino Coding IDE 501+ looks like below.



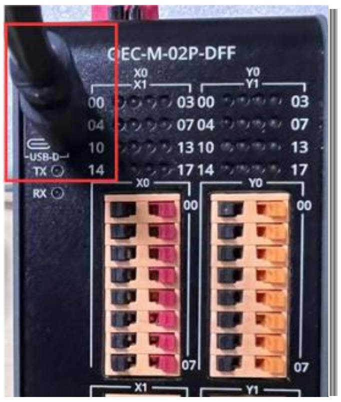
sketch_dec16c | 86Duino Coding 501
File Edit Sketch Tools Help
sketch_dec16c
QEC M01 on COM9

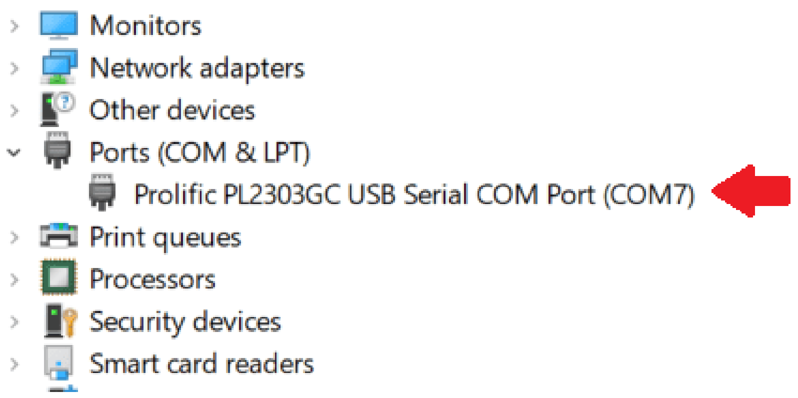
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

3. Connect to PC and set up the environment

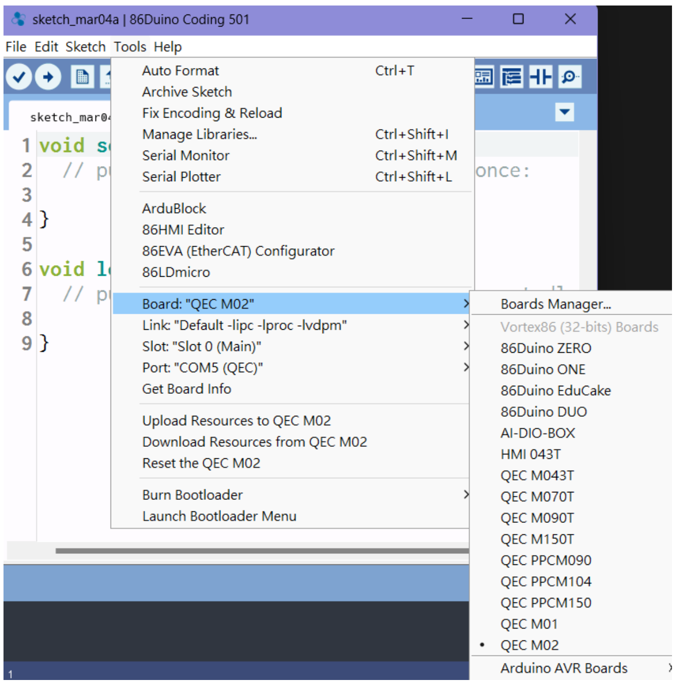
Follow the steps below to set up the environment:

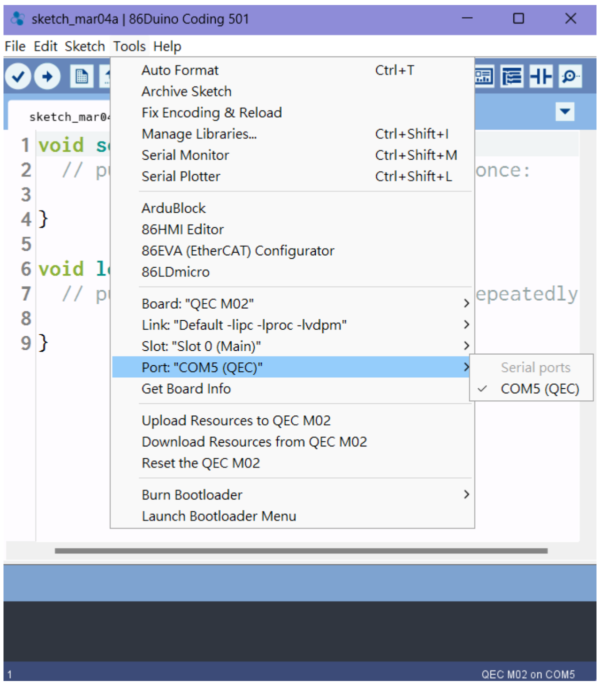
1. Connect the QEC-M-02 to your PC via a Type-C to USB cable (86Duino IDE installed).
2. Turn on the QEC power.
3. Open “**Device Manager**” (select in the menu after pressing Win+X) ->” **Ports (COM & LPT)**” in your PC and expand the ports; you should see that the “**Prolific PL2303GC USB Serial COM Port (COMx)**” is detected; if not, you will need to install the required drivers.
   (For Windows PL2303 driver, you can download here)





4. Open the 86Duino IDE.
5. Select the correct board: In the IDE's menu, select “**Tools**” > “**Board**” > “**QEC-M02**” (or the QEC MDevice model you use).
6. Select Port: In the IDE's menu, select “**Tools**” > “**Port**” and select the USB port to connect to the QEC MDevice (in this case, COM5 (QEC)).




sketch_mar04a | 86Duino Coding 501
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ArduBlock
86HMI Editor
86EVA (EtherCAT) Configurator
86LDmicro
Board: "QEC M02"
Link: "Default -lipc -lproc -lvdpm"
Slot: "Slot 0 (Main)"
Port: "COM5 (QEC)"
Get Board Info
Upload Resources to QEC M02
Download Resources from QEC M02
Reset the QEC M02
Burn Bootloader
Launch Bootloader Menu
Serial ports
✓ COM5 (QEC)
QEC M02 on COM5

4. Use 86EVA

This example shows how to operate the EtherCAT MDevice (QEC-M-02) and the QEC-R11MP3S (EtherCAT 3-axis Stepper Motor Controller) through the 86Duino IDE's graphical low-code programming tool, **86EVA**.

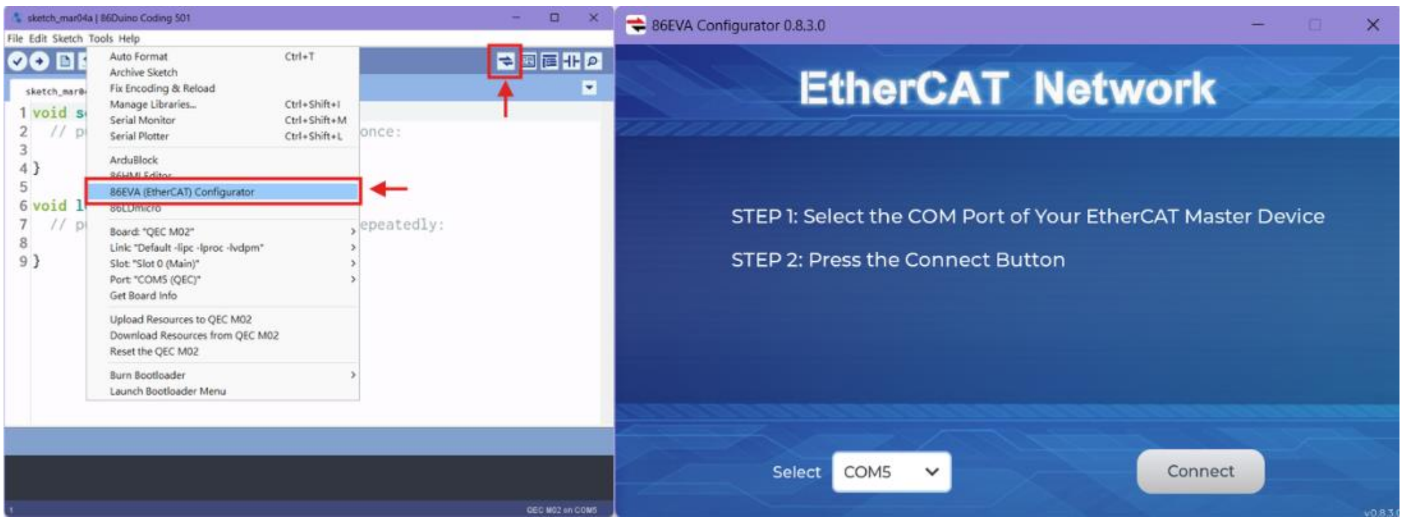
Software Tools Description:

- **86EVA (EVA, EtherCAT-Based Virtual Arduino):**
  is a graphical EtherCAT configuration tool based on the EtherCAT Library in the 86Duino IDE and is one of the development kits for 86Duino.

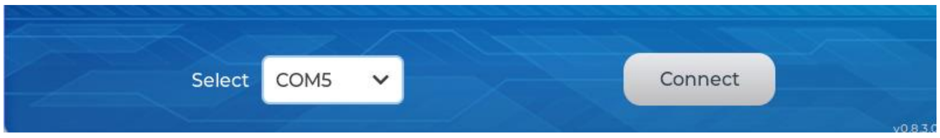
We will present the 86EVA usage in the following section, step by step.

Step 1: Turn on 86EVA and scan

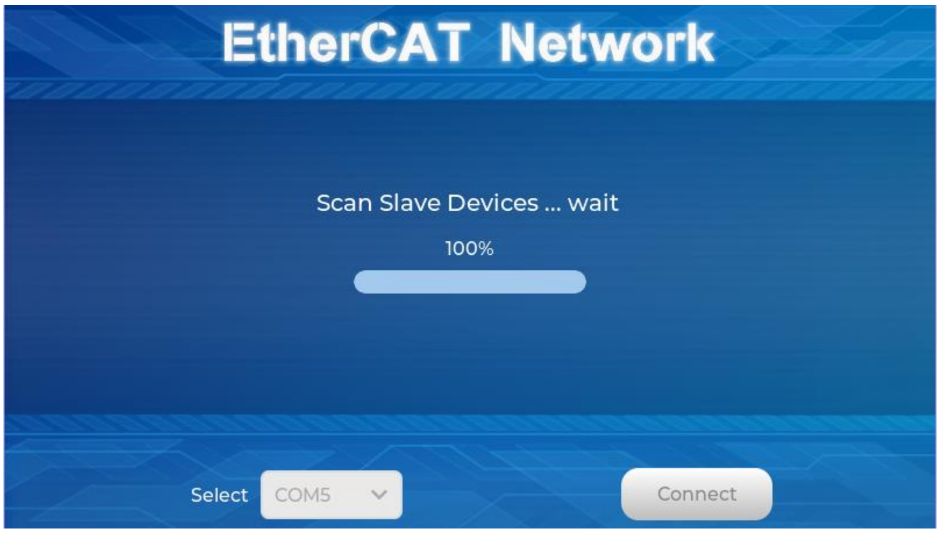
The 86EVA tool can be opened via the following buttons.


sketch_mar04a | 86Duino Coding 501
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ArduBlock
86HMI Editor
86EVA (EtherCAT) Configurator
86LDmicro
Board: "QEC M02"
Link: "Default -lipc -lproc -lvdpm"
Slot: "Slot 0 (Main)"
Port: "COM5 (QEC)"
Get Board Info
Upload Resources to QEC M02
Download Resources from QEC M02
Reset the QEC M02
Burn Bootloader
Launch Bootloader Menu
QEC M02 on COM5
86EVA Configurator 0.8.3.0
EtherCAT Network
STEP 1: Select the COM Port of Your EtherCAT Master Device
STEP 2: Press the Connect Button
Select
COM5
Connect

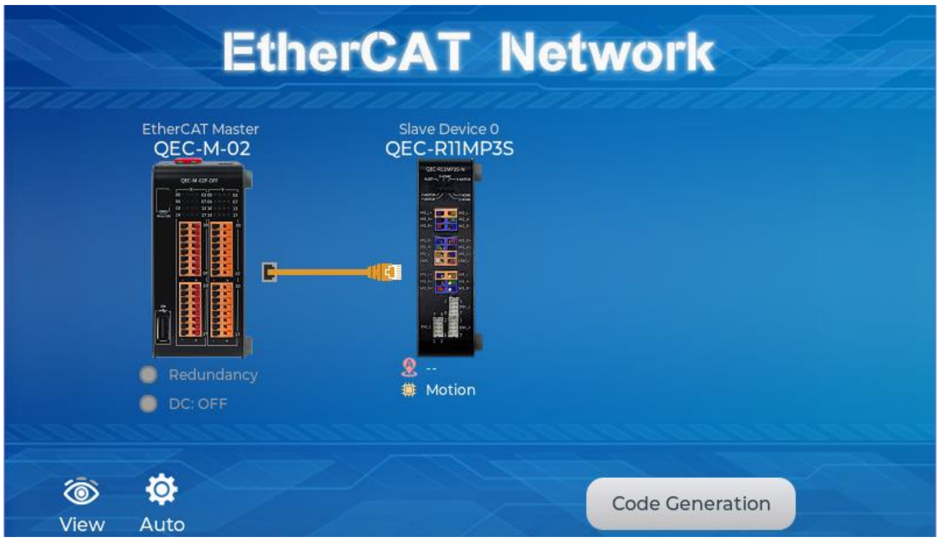
Please select the correct COM port and then click the "**Connect**" button.



Once you have confirmed that the correct COM port has been selected of QEC-M-02, press the Connect button to start scanning the EtherCAT network.



The connected devices will be displayed after the EtherCAT network has been scanned.

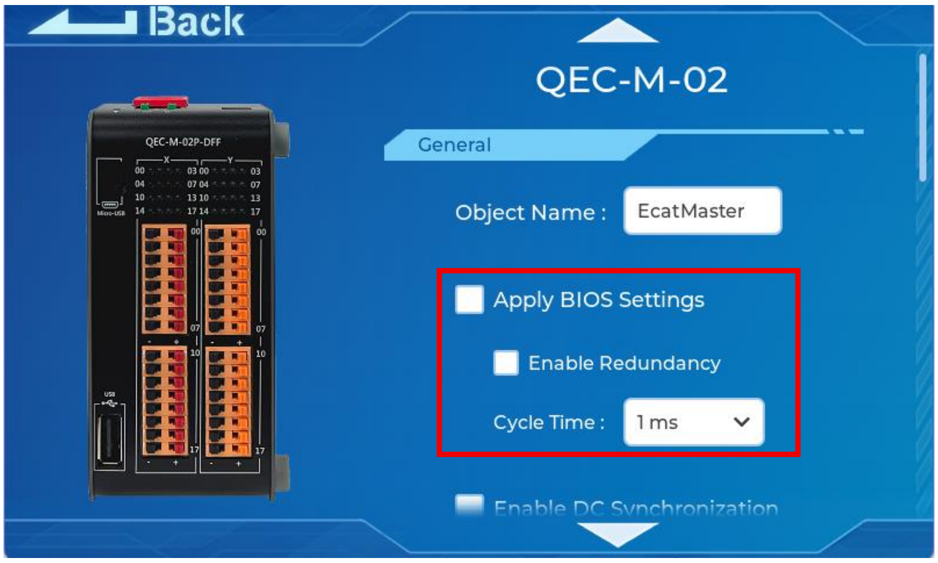


Step 2: Set the parameters

Press twice on the scanned device image to enter the corresponding parameter setting screen.

Step 2.1: QEC-M-02

Press twice on the image of the QEC-M-01 to see the parameter settings.



Object Name : EcatMaster

Apply BIOS Settings [ ]

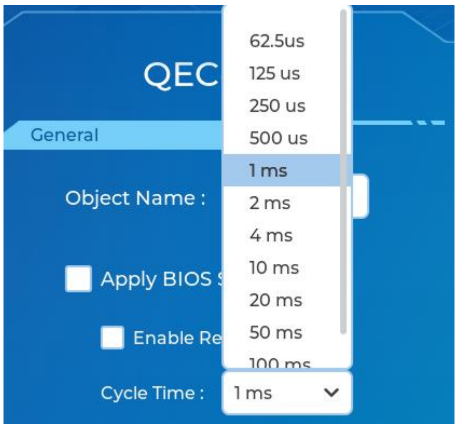
Enable Redundancy [ ]

Cycle Time : 1 ms

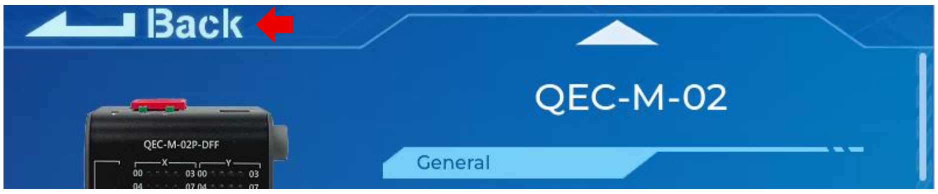
Enable DC Synchronization [ ]

Please check the following configures.

1. Turn off the "**Apply BIOS Settings**".
2. Select "**1ms**" to the Cycle Time.

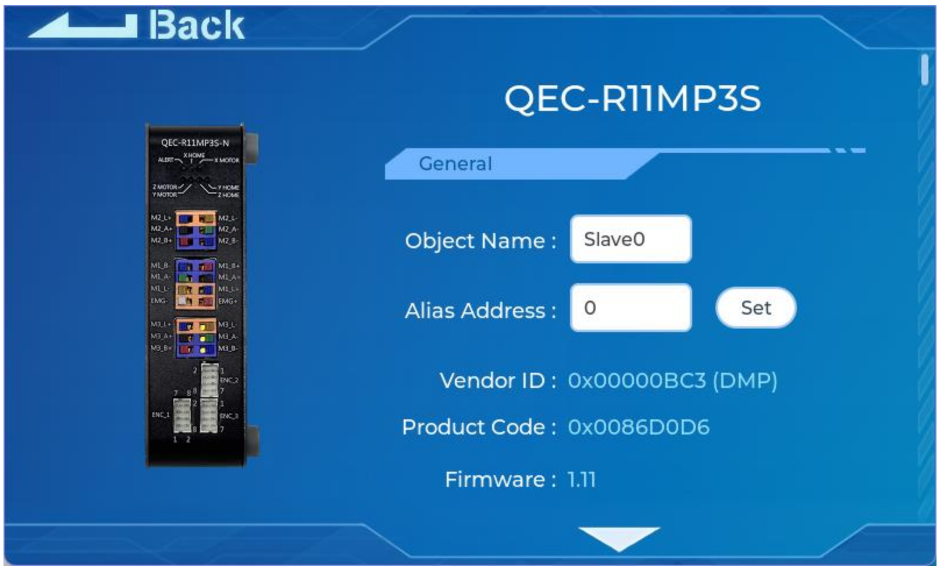


Click "**Back**" in the upper left corner to return.



Step 2.2: QEC-R11MP3S

Press twice on the image of the QEC-R11MP3S to see the parameter settings.



QEC-R11MP3S

Object Name : Slave0

Alias Address : 0

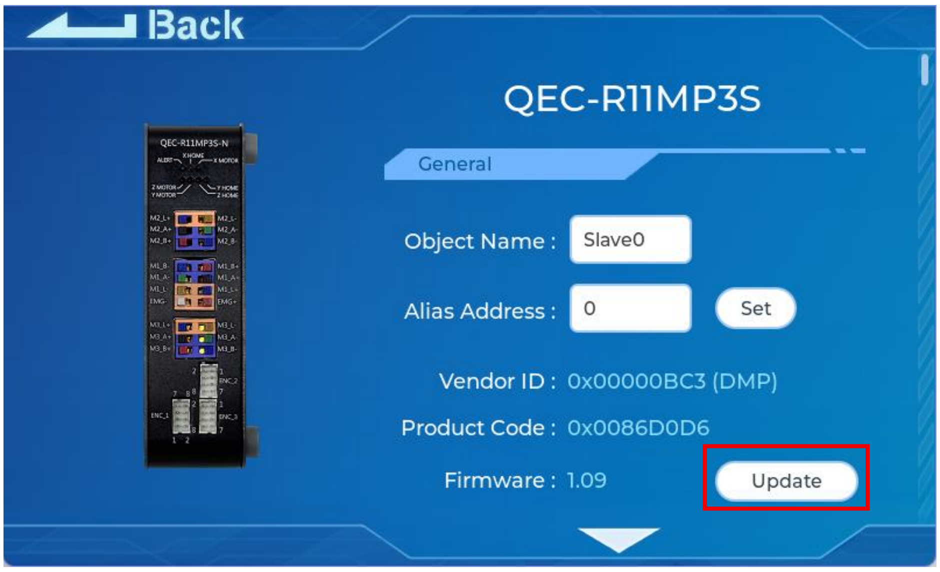
Vendor ID : 0x00000BC3 (DMP)

Product Code : 0x0086D0D6

Firmware : 1.11

The page will show the Object Name, Alias Address, Vendor ID, Product Code, Virtual Arduino Mapping, and Virtual Servo Configuration parameters.

Note:
If you see an **Update/Downgrade** button next to the firmware (as shown in the figure below), please click it. For detailed instructions, refer to the **Troubleshooting** section.



QEC-R11MP3S

Object Name : Slave0

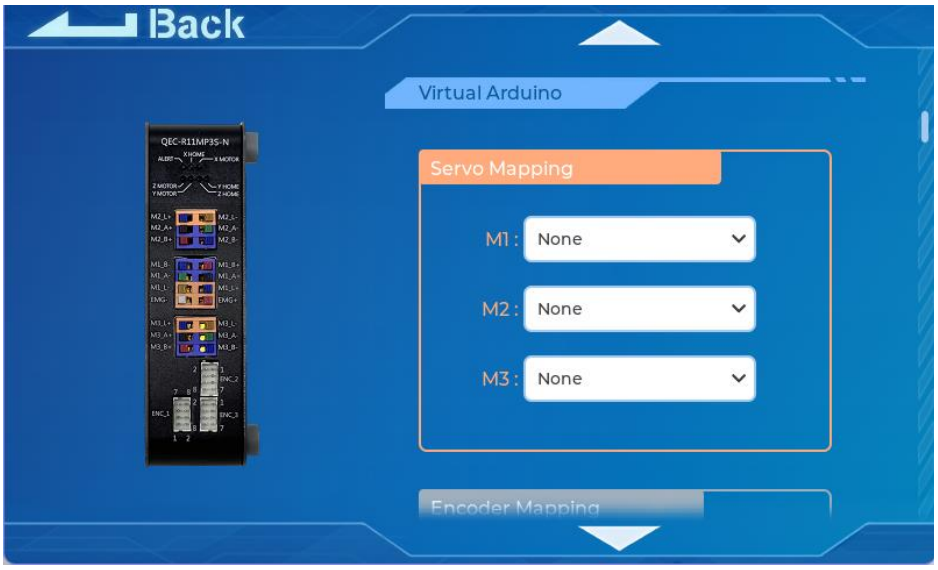
Alias Address : 0

Vendor ID : 0x00000BC3 (DMP)

Product Code : 0x0086D0D6

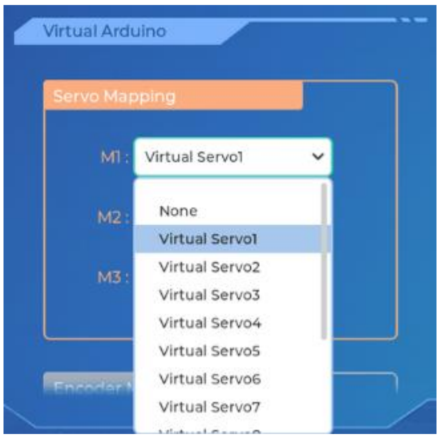
Firmware : 1.09

Continue down to the "**Virtual Arduino**" area.
First, we're in the Servo mapping area.



M1 : None
M2 : None
M3 : None

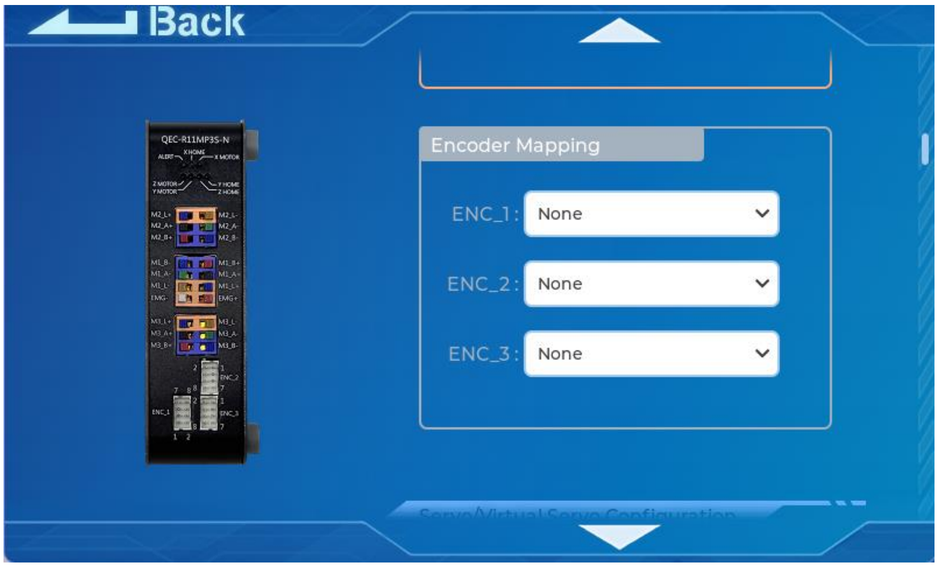
We select "**Virtual Servo1**" from the "**M1**" drop-down in "**Servo Mapping**".



M1 : Virtual Servo1

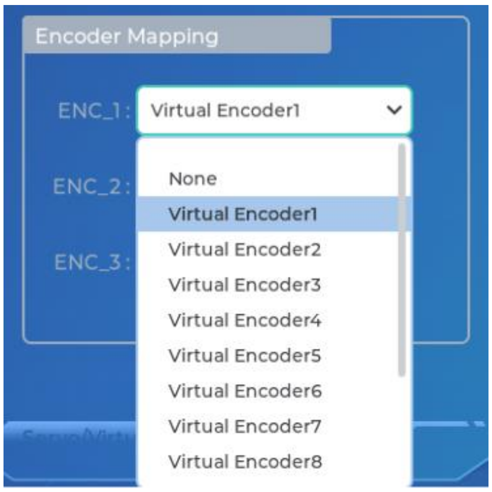
None
Virtual Servo1
Virtual Servo2
Virtual Servo3
Virtual Servo4
Virtual Servo5
Virtual Servo6
Virtual Servo7

Next, we go down to the digital output pin mapping area.

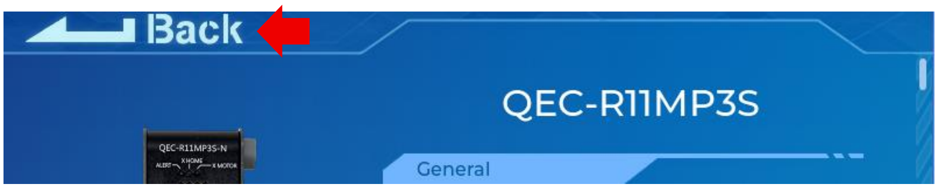


ENC_1 : None
ENC_2 : None
ENC_3 : None

We select “**Virtual Encoder1**” from the “**ENC_1**” drop-down in “**Encoder Mapping**”.

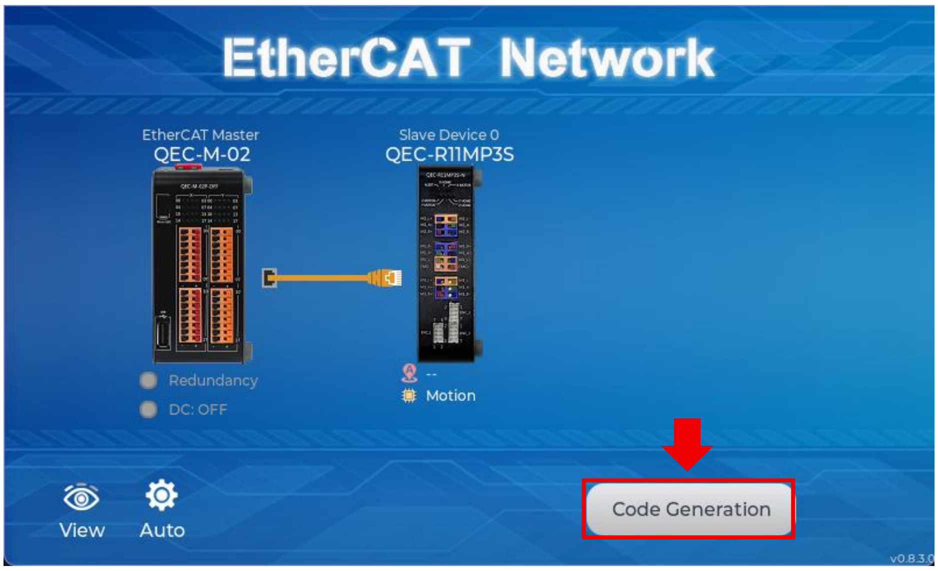


Click "**Back**" in the upper left corner to return.

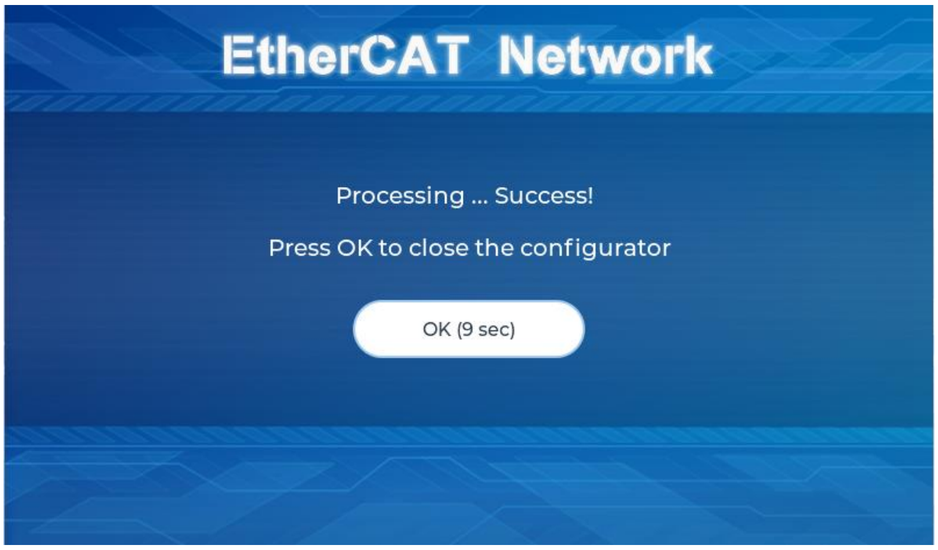


Step 3: Generate the code

Once you've set your device's parameters, go back to the home screen and press the "**Code Generation**" button in the bottom right corner.

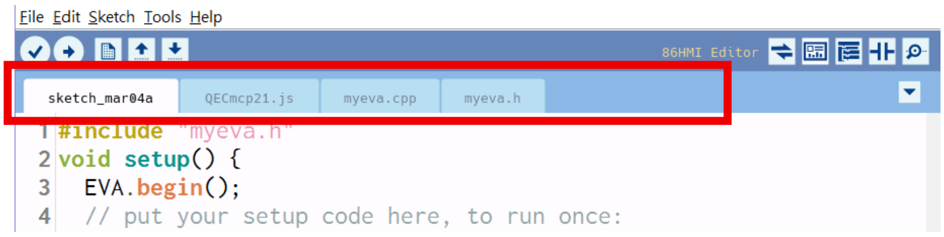


When you're done, double-click the OK button to turn off 86EVA, or it will close in 10 seconds.



The generated code and files are as follows:

- sketch_oct20c: Main Project (.ino, depending on your project name)
- myeva.cpp: C++ program code of 86EVA
- myeva.h: Header file of 86EVA



```
#include "myeva.h"
void setup() {
  EVA.begin();
  // put your setup code here, to run once:
```

***Additional note:**

After 86EVA generates code, the following code will be automatically generated in the main program (.ino), and any of them missing will cause 86EVA not to work.

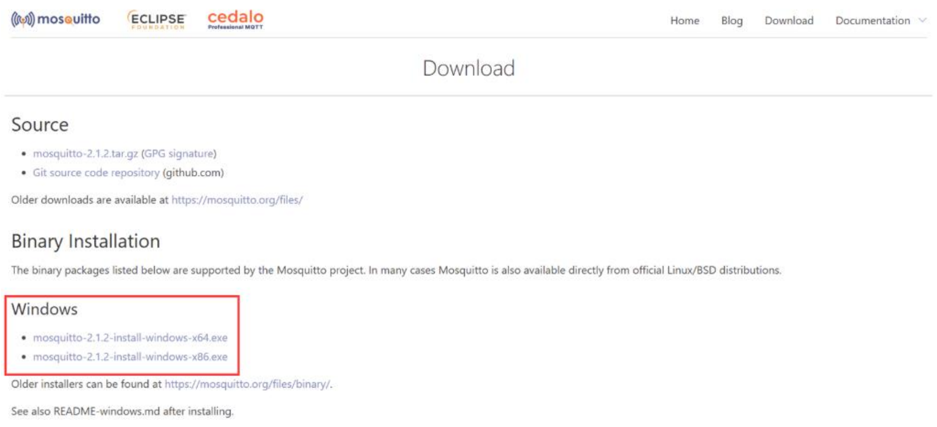
1. `#include “myeva.h”` : Include EVA Header file
2. `EVA.begin();` in `setup()` : Initialize the EVA function

5. MQTT Communication Setup

This section describes the steps required to install the **Mosquitto MQTT broker** on **Windows 11**. The broker provides the messaging service required for MQTT communication between devices, Node-RED, and other MQTT clients.
If MQTT broker is already installed and running in your environment, this section may be skipped.

Step 1: Download Mosquitto

Download Mosquitto from https://mosquitto.org/download/.



Download

Source

- mosquitto-2.1.2.tar.gz (GPG signature)
- Git source code repository (github.com)

Older downloads are available at https://mosquitto.org/files/

Binary Installation

The binary packages listed below are supported by the Mosquitto project. In many cases Mosquitto is also available directly from official Linux/BSD distributions.

Windows

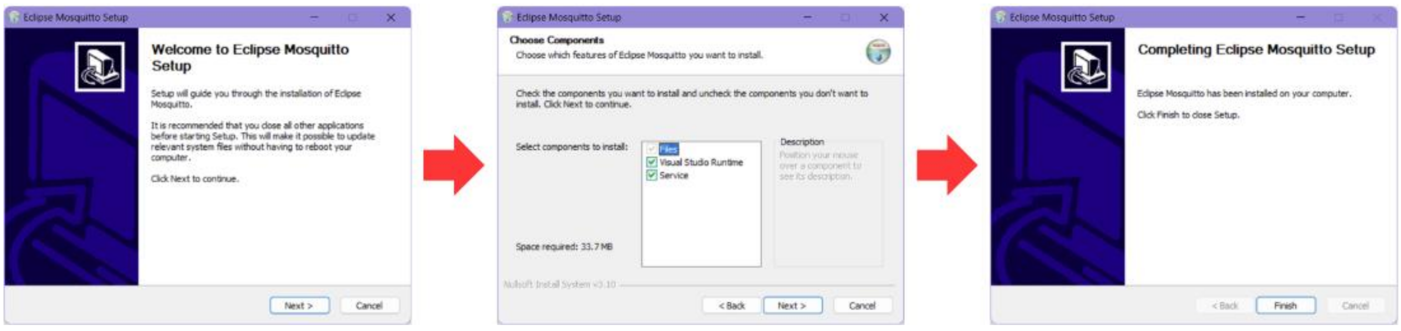
- mosquitto-2.1.2-install-windows-x64.exe
- mosquitto-2.1.2-install-windows-x86.exe

Older installers can be found at https://mosquitto.org/files/binary/.

See also README-windows.md after installing.

Step 2: Run Mosquitto install

Run the installer and follow the installation wizard.
The default installation path is usually: C:\Program Files\mosquitto



Welcome to Eclipse Mosquitto Setup

Setup will guide you through the installation of Eclipse Mosquitto.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Choose Components

Choose which features of Eclipse Mosquitto you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install: ✓ Files, ✓ Visual Studio Runtime, ✓ Service

Description: Position your mouse over a component to see its description.

Space required: 33.7 MB

Nullsoft Install System v3.10

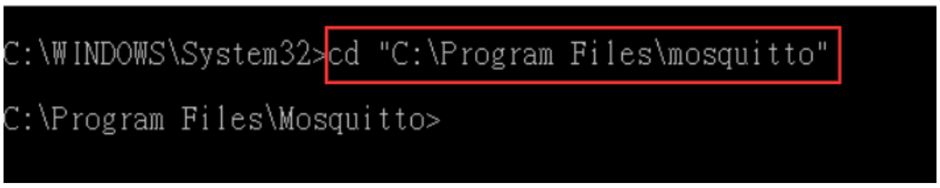
Completing Eclipse Mosquitto Setup

Eclipse Mosquitto has been installed on your computer.

Click Finish to close Setup.

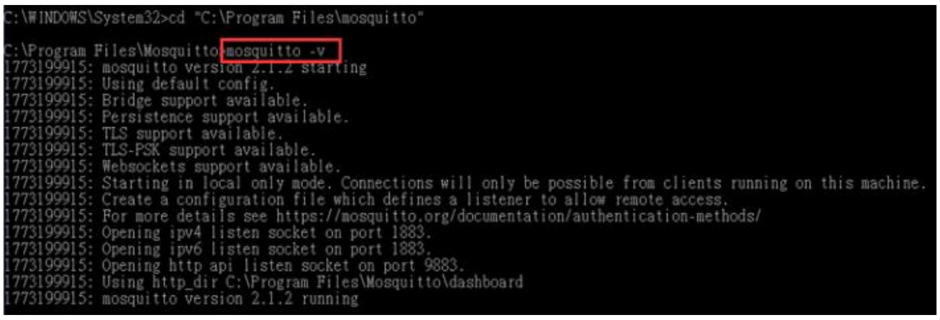
Step 3: Start Mosquitto Broker

Open Command Prompt as an administrator and run: `cd "C:\Program Files\mosquitto"`



```
C:\WINDOWS\System32>cd "C:\Program Files\mosquitto"

C:\Program Files\Mosquitto>
```

When the terminal is in the Mosquitto directory, run: `mosquitto -v`
If you see the following output in the terminal, Mosquitto has started successfully.



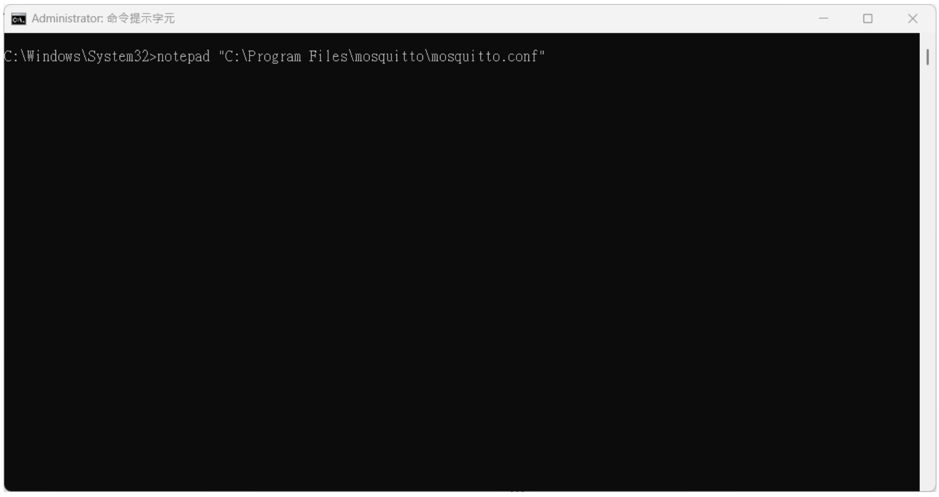
```
C:\WINDOWS\System32>cd "C:\Program Files\mosquitto"

C:\Program Files\Mosquitto>mosquitto -v
1773199915: mosquitto version 2.1.2 starting
1773199915: Using default config.
1773199915: Bridge support available.
1773199915: Persistence support available.
1773199915: TLS support available.
1773199915: TLS-PSK support available.
1773199915: Websockets support available.
1773199915: Starting in local only mode. Connections will only be possible from clients running on this machine.
1773199915: Create a configuration file which defines a listener to allow remote access.
1773199915: For more details see https://mosquitto.org/documentation/authentication-methods/
1773199915: Opening ipv4 listen socket on port 1883.
1773199915: Opening ipv6 listen socket on port 1883.
1773199915: Opening http api listen socket on port 9883.
1773199915: Using http_dir C:\Program Files\Mosquitto\dashboard
1773199915: mosquitto version 2.1.2 running
```

Step 4: Edit mosquito.conf

When you have successfully installed and started the Mosquitto broker, you need to modify the **mosquitto.conf** file to allow external network connections to access the broker.

First, open **Command Prompt** as an administrator, then enter:

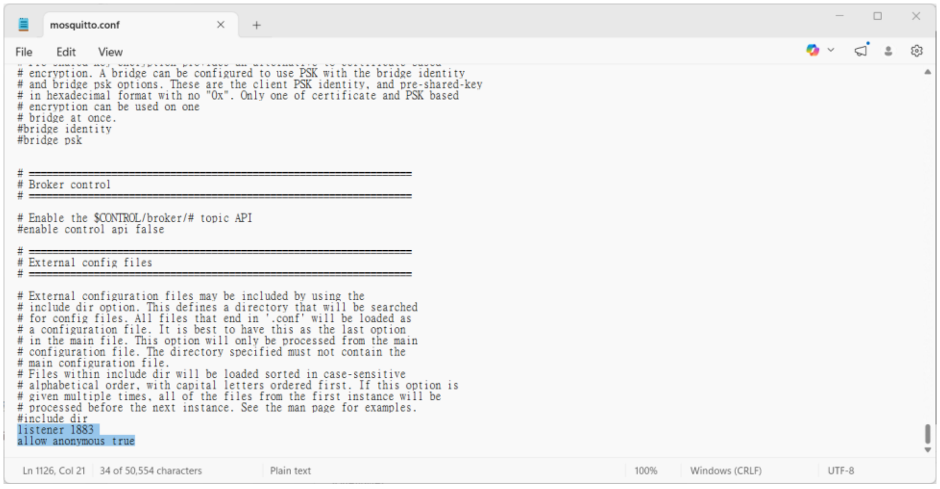
```
notepad "C:\Program Files\mosquitto\mosquitto.conf"
```



```
C:\Windows\System32>notepad "C:\Program Files\mosquitto\mosquitto.conf"
```

After entering the command, the **mosquitto.conf** file will open in Notepad. Scroll to the bottom and add the following:

```
listener 1883
allow_anonymous true
```



```
# encryption. A bridge can be configured to use PSK with the bridge identity
# and bridge psk options. These are the client PSK identity, and pre-shared-key
# in hexadecimal format with no "0x". Only one of certificate and PSK based
# encryption can be used on one
# bridge at once.
#bridge identity
#bridge psk


# =================================================================
# Broker control
# =================================================================

# Enable the $CONTROL/broker/# topic API
#enable control api false

# =================================================================
# External config files
# =================================================================

# External configuration files may be included by using the
# include dir option. This defines a directory that will be searched
# for config files. All files that end in '.conf' will be loaded as
# a configuration file. It is best to have this as the last option
# in the main file. This option will only be processed from the main
# configuration file. The directory specified must not contain the
# main configuration file.
# Files within include dir will be loaded sorted in case-sensitive
# alphabetical order, with capital letters ordered first. If this option is
# given multiple times, all of the files from the first instance will be
# processed before the next instance. See the man page for examples.
#include dir
listener 1883
allow anonymous true
```

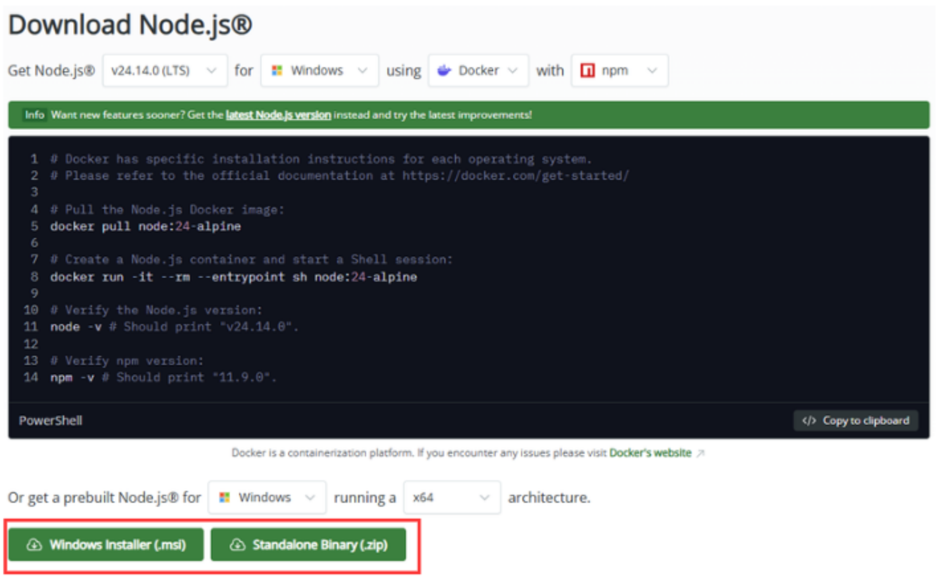
After completing the edits, close the file and **restart Mosquitto Broker**.

6. Node-RED Setup

This section explains how to install Node-RED on Windows 11 and how to install the required node packages.
If Node-RED is already installed in your environment, you may skip this section.

Step 1: Download Node.js

Download Node.js from https://nodejs.org/en/download.
Select the installer that matches your system configuration.



Download Node.js®

Get Node.js® v24.14.0 (LTS) for Windows using Docker with npm

Info Want new features sooner? Get the latest Node.js version instead and try the latest improvements!

```
# Docker has specific installation instructions for each operating system.
# Please refer to the official documentation at https://docker.com/get-started/

# Pull the Node.js Docker image:
docker pull node:24-alpine

# Create a Node.js container and start a Shell session:
docker run -it --rm --entrypoint sh node:24-alpine

# Verify the Node.js version:
node -v # Should print "v24.14.0".

# Verify npm version:
npm -v # Should print "11.9.0".
```

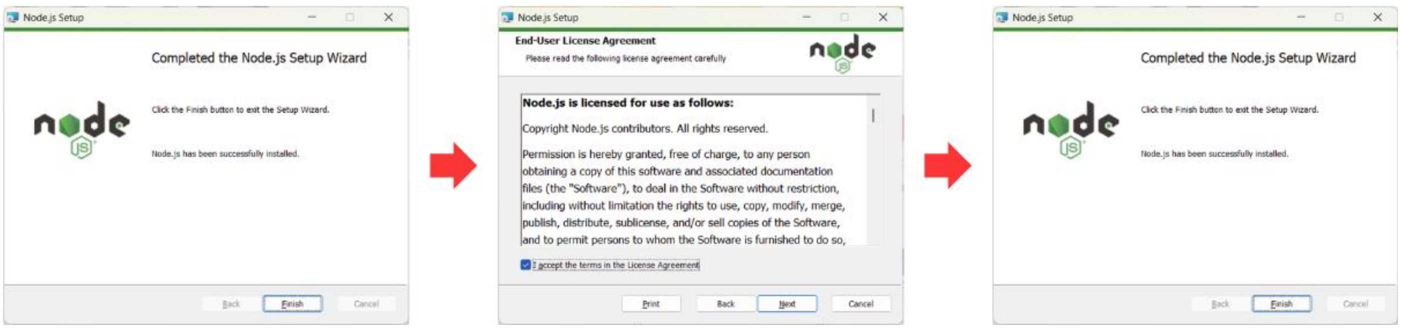
PowerShell

Docker is a containerization platform. If you encounter any issues please visit Docker's website

Or get a prebuilt Node.js® for Windows running a x64 architecture.

Step 2: Run Node.js install

Run the installer and follow the installation wizard.



Node.js Setup

Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

Node.js Setup

End-User License Agreement

Please read the following license agreement carefully

Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,

[✓] I accept the terms in the License Agreement

Node.js Setup

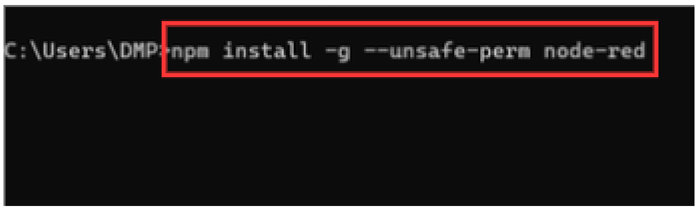
Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

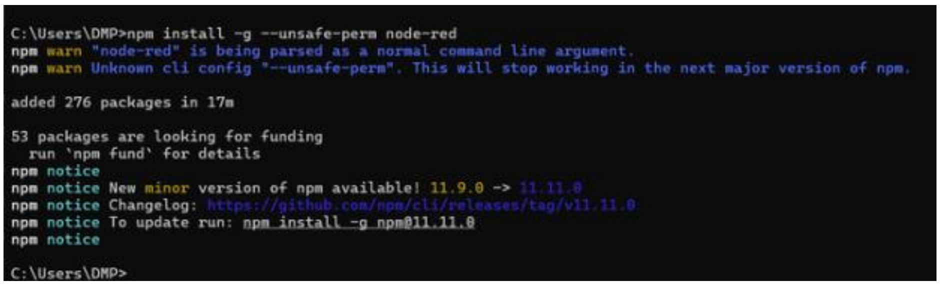
Step 3: Install Node-RED

Open **Command Prompt** and run: `npm install -g --unsafe-perm node-red`



```
C:\Users\DMP>npm install -g --unsafe-perm node-red
```

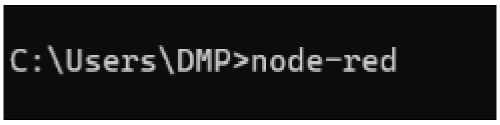
After the installation is complete, you will see a page similar to the one below.



```
C:\Users\DMP>npm install -g --unsafe-perm node-red
npm warn "node-red" is being parsed as a normal command line argument.
npm warn Unknown cli config "--unsafe-perm". This will stop working in the next major version of npm.

added 276 packages in 17m

53 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 11.9.0 -> 11.11.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.11.0
npm notice To update run: npm install -g npm@11.11.0
npm notice

C:\Users\DMP>
```

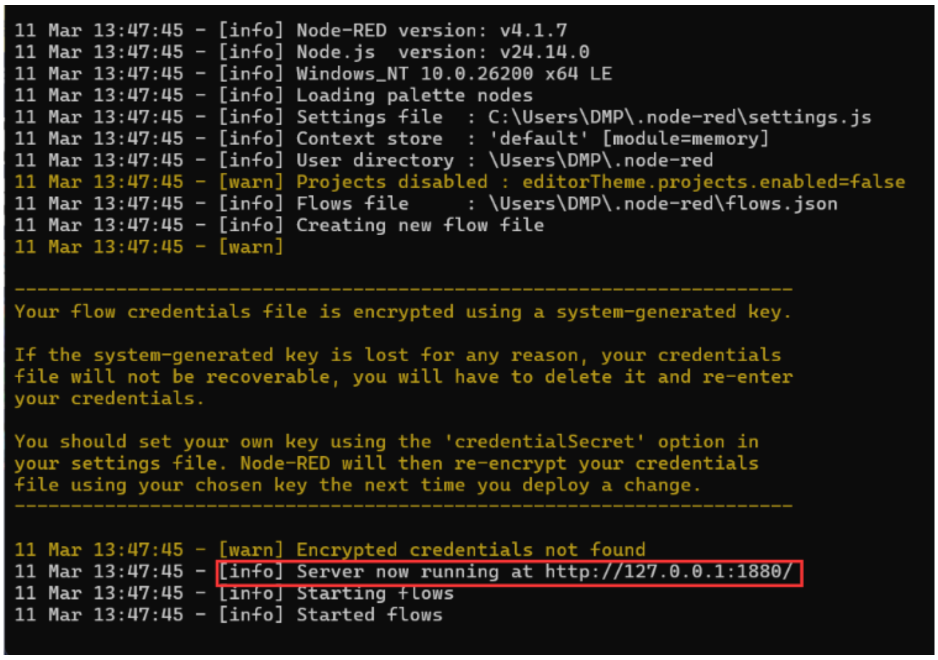
Step 4: Start Node-RED

Open **Command Prompt** and run: `node-red`



```
C:\Users\DMP>node-red
```

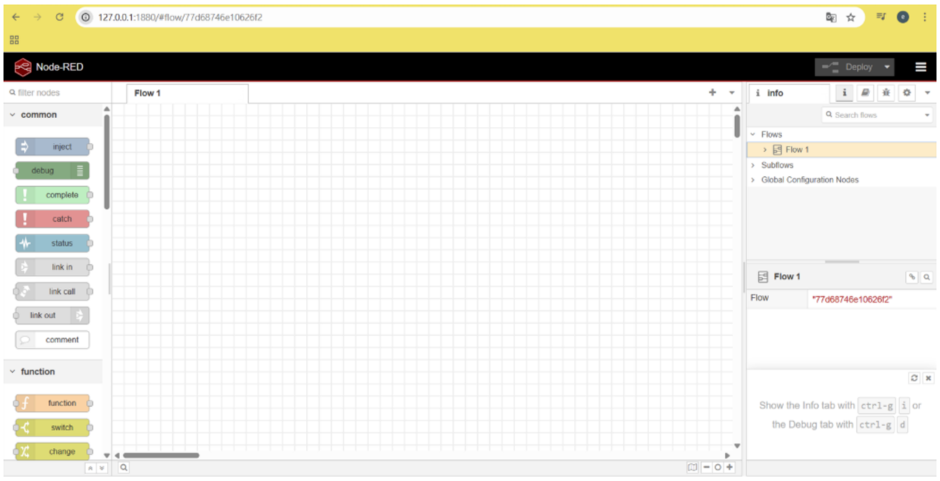
When Node-RED starts successfully, you will see a page like the one below.
You should see: **Server now running at http://127.0.0.1:1880/**



```
11 Mar 13:47:45 - [info] Node-RED version: v4.1.7
11 Mar 13:47:45 - [info] Node.js  version: v24.14.0
11 Mar 13:47:45 - [info] Windows_NT 10.0.26200 x64 LE
11 Mar 13:47:45 - [info] Loading palette nodes
11 Mar 13:47:45 - [info] Settings file  : C:\Users\DMP\.node-red\settings.js
11 Mar 13:47:45 - [info] Context store  : 'default' [module=memory]
11 Mar 13:47:45 - [info] User directory : \Users\DMP\.node-red
11 Mar 13:47:45 - [warn] Projects disabled : editorTheme.projects.enabled=false
11 Mar 13:47:45 - [info] Flows file     : \Users\DMP\.node-red\flows.json
11 Mar 13:47:45 - [info] Creating new flow file
11 Mar 13:47:45 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

11 Mar 13:47:45 - [warn] Encrypted credentials not found
11 Mar 13:47:45 - [info] Server now running at http://127.0.0.1:1880/
11 Mar 13:47:45 - [info] Starting flows
11 Mar 13:47:45 - [info] Started flows
```

Step 5: Open Node-RED Editor

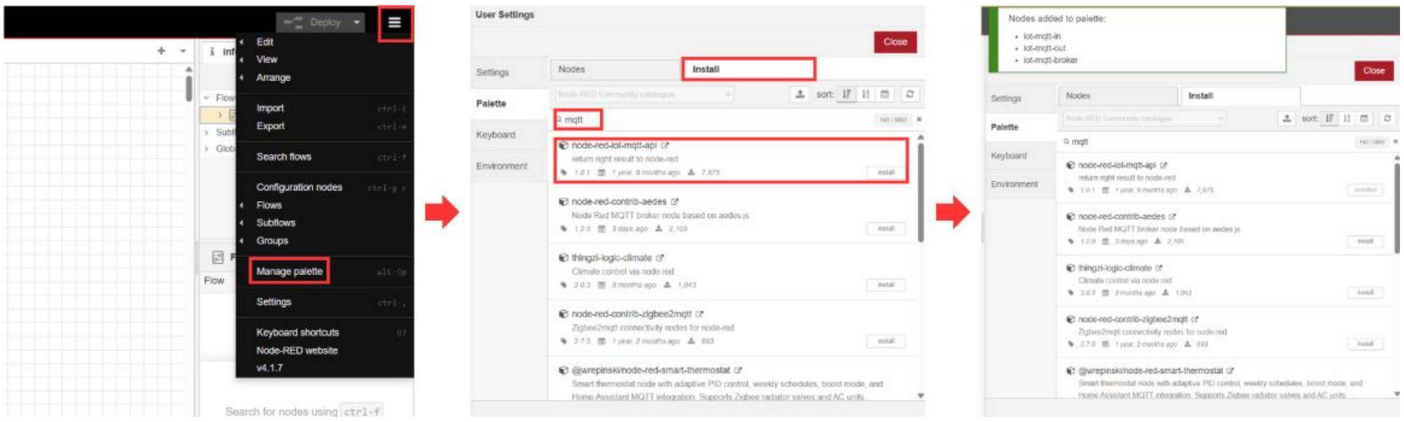
Open your browser and go to: http://127.0.0.1:1880/
The Node-RED editor interface will appear.



Step 6: Install Nodes in Node-RED

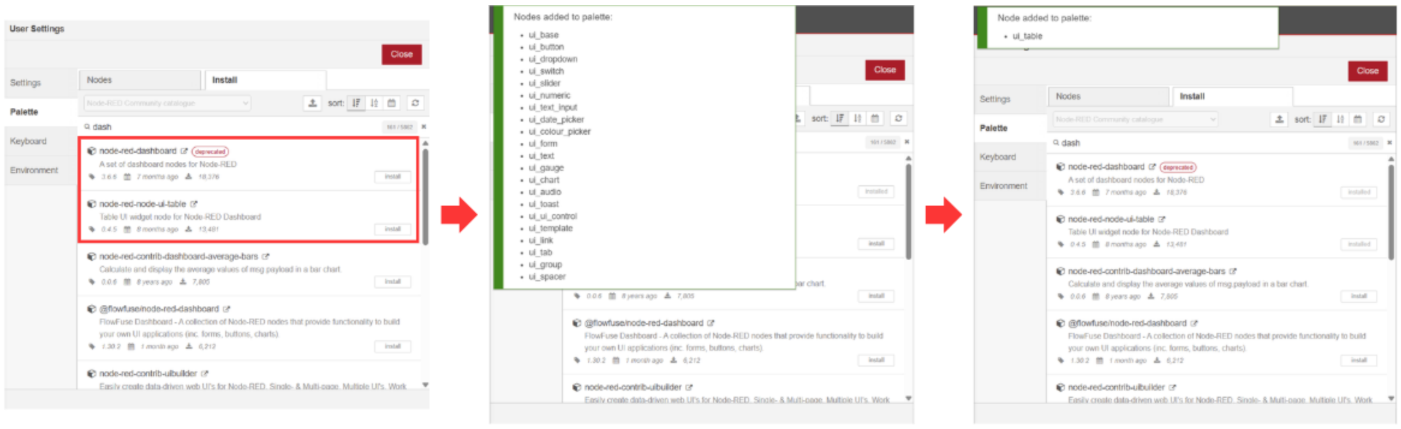
After successfully opening the Node-RED Editor in your browser, you need to install the MQTT and Dashboard packages.

1. Click the **Menu (☰)** in the top-right corner of the Node-RED Editor.
2. Select **Manage palette**.
3. Go to the **Install** tab.
4. Search for: **mqtt**
5. Find the package **node-red-iot-mqtt-api** and click **Install**.



Nodes added to palette:
- iot-mqtt-in
- iot-mqtt-out
- iot-mqtt-broker

After installing the MQTT package, if you want to use the Node-RED UI, you need to install the **Dashboard** and **UI** packages.



User Settings
Close
Settings
Palette
Keyboard
Environment
Nodes
Install
dash
node-red-dashboard
deprecated
A set of dashboard nodes for Node-RED
node-red-node-ui-table
Table UI widget node for Node-RED Dashboard
node-red-contrib-dashboard-average-bars
Calculate and display the average values of msg payload in a bar chart
@flowfuse/node-red-dashboard
FlowFuse Dashboard - A collection of Node-RED nodes that provide functionality to build your own UI applications (inc. forms, buttons, charts)
node-red-contrib-uibuilder
install
installed

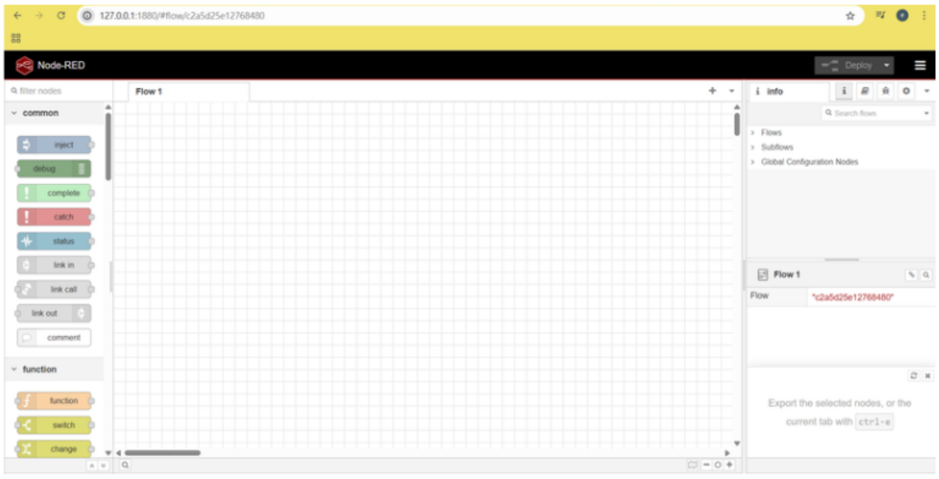
Nodes added to palette:
- ui_base
- ui_button
- ui_dropdown
- ui_switch
- ui_slider
- ui_numeric
- ui_text_input
- ui_date_picker
- ui_colour_picker
- ui_form
- ui_text
- ui_gauge
- ui_chart
- ui_audio
- ui_toast
- ui_ui_control
- ui_template
- ui_link
- ui_tab
- ui_group
- ui_spacer

Node added to palette:
- ui_table

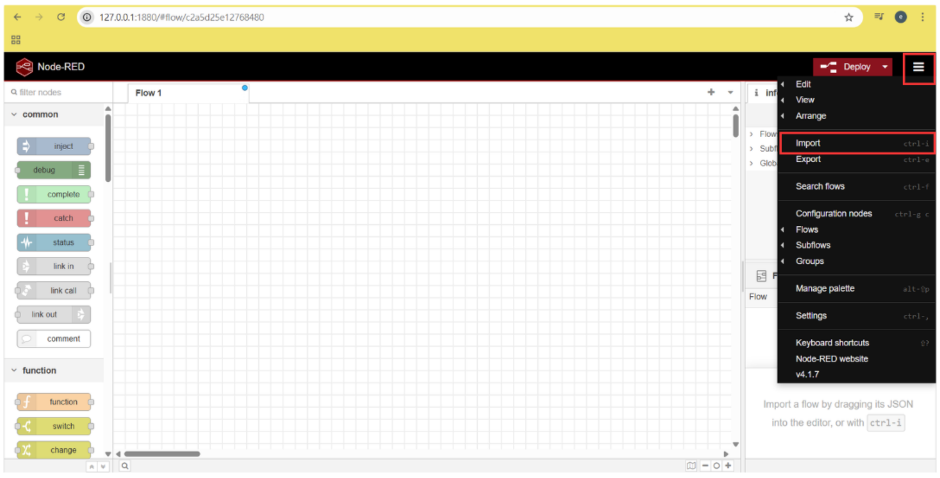
Step7: Import Node-RED Flow (JSON)

When you have a Node-RED UI flow JSON file (such as a sample flow generated by AI), you can import it into Node-RED by following the steps below.

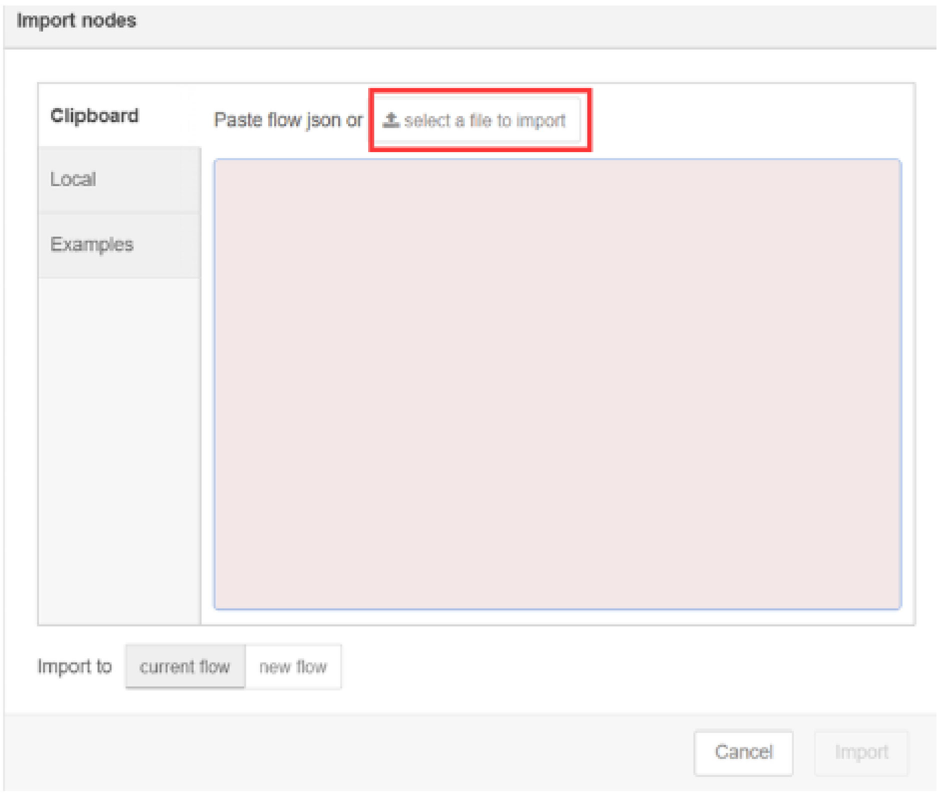
1.Open the **Node-RED editor** in your browser.



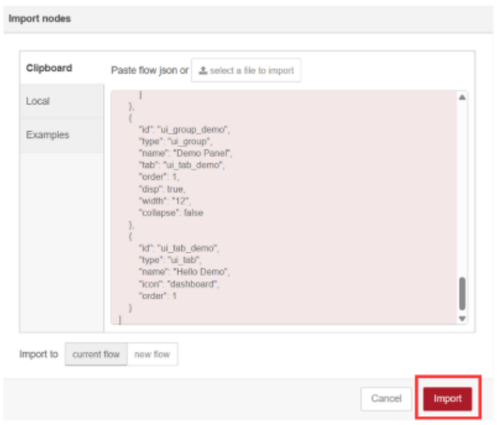
2. Click the **menu button (☰)** in the top-right corner and Select **Import** from the menu.



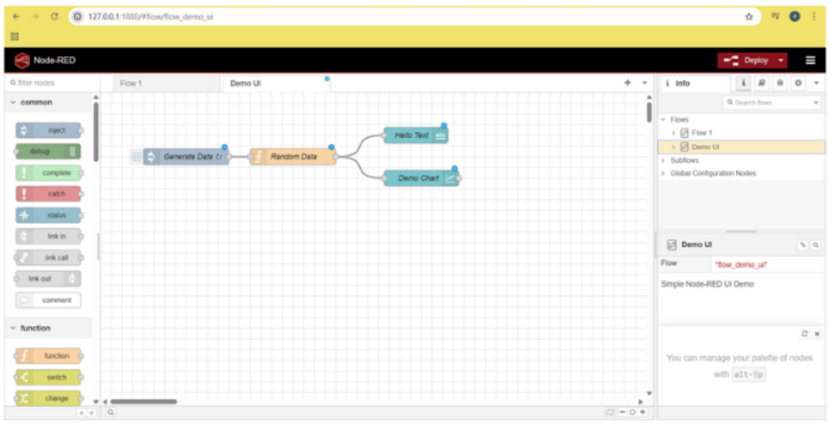
3. Paste the **JSON flow content** into the import window or upload the JSON file.



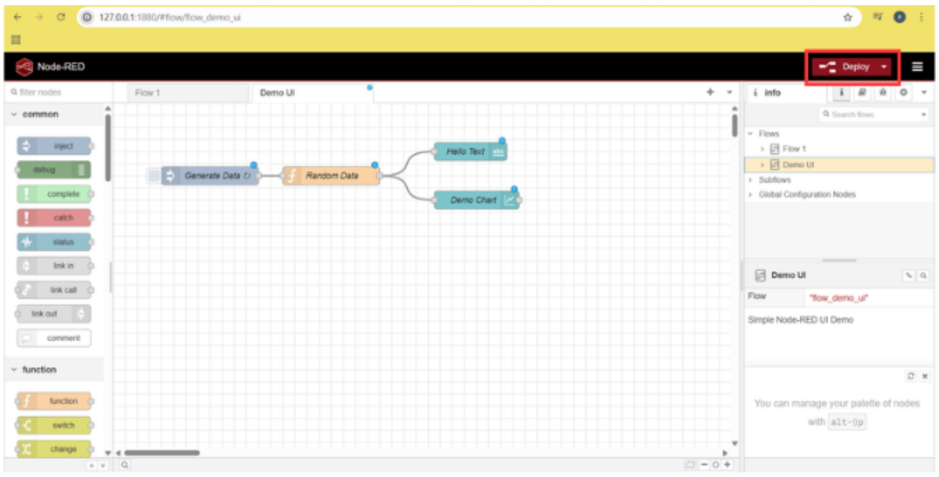
4. Click **Import** to add the flow to your workspace.



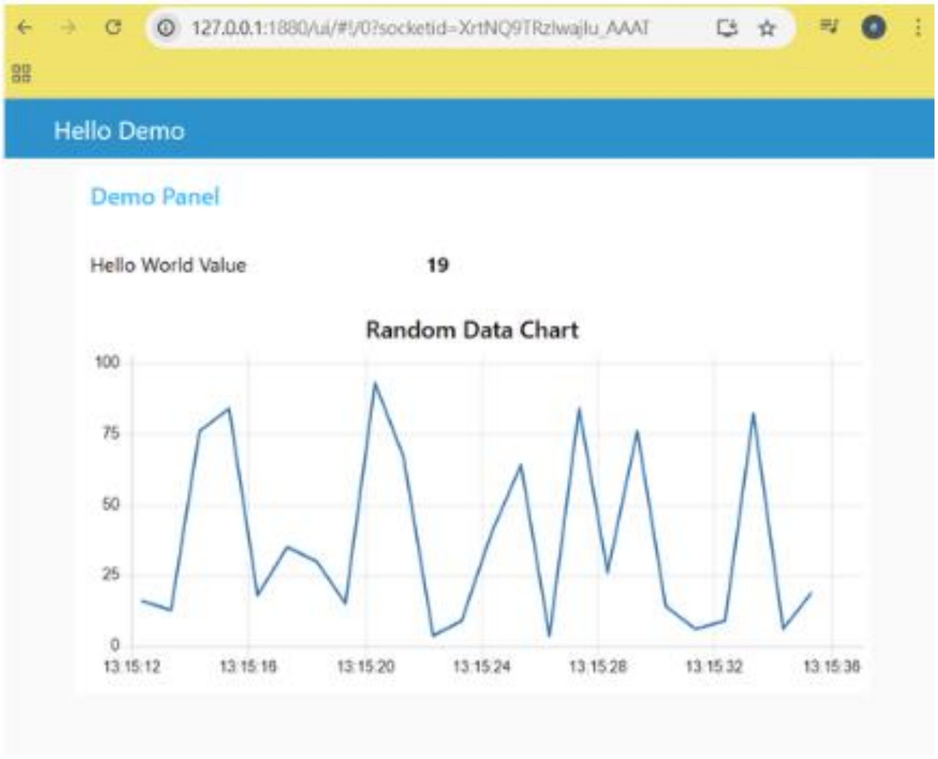
```
        ]
    },
    {
        "id": "ui_group_demo",
        "type": "ui_group",
        "name": "Demo Panel",
        "tab": "ui_tab_demo",
        "order": 1,
        "disp": true,
        "width": "12",
        "collapse": false
    },
    {
        "id": "ui_tab_demo",
        "type": "ui_tab",
        "name": "Hello Demo",
        "icon": "dashboard",
        "order": 1
    }
]
```



5. Click **Deploy** to apply the changes.



6. Once deployed, the dashboard will be available at http://127.0.0.1:1880/ui.



7. Example Code

QEC-M-02 supports remote monitoring and control through the MQTT communication protocol. It enables functions such as device data monitoring, digital I/O control, and motor operation control. The development is performed using the 86Duino IDE, together with Ethernet and PubSubClient libraries to implement MQTT communication.

In the following sections, we will provide four example programs to help you quickly build an MQTT-based remote monitoring and control system using QEC-M-02:

- **MQTT Connection** – Establish MQTT connection between QEC-M-02 and the broker.
- **Motor Control** – Control motor enable/disable and motion via MQTT.
- **Digital I/O Control** – Control digital outputs and monitor I/O status via MQTT.
- **Device Status** – Publish device data for Node-RED visualization.

Hardware Configuration

- MQTT Client Device: QEC-M-02
- EtherCAT Slave Device: QEC-R11MP3S-N
- Communication Protocol: MQTT (Publish/Subscribe)
- MQTT Broker: Mosquitto running on NAS
- Ethernet MAC/IP: Assigned to QEC-M-02 for network access

Software Configuration

- Development IDE: 86Duino IDE 501
- Network Library: Ethernet Library
- MQTT library: PubSubClient
- Visualization Platform: Node-RED Dashboard

Before running the following examples, please make sure that MQTT (Mosquitto) and Node-RED have been successfully installed as described in Sections 5 and 6.

Example 1 – MQTT Connection

This example demonstrates how to establish an MQTT connection between the QEC-M-02 and the MQTT broker. The device publishes connection information and system status for monitoring.

A. In Setup Function

The setup() function initializes the system environment and establishes the MQTT connection.

1. Initialize the 86EVA EtherCAT environment.
2. Begin serial communication for debugging.
3. Initialize the Ethernet interface with the predefined MAC and IP address.
4. Configure the MQTT broker address and port.
5. Attempt to establish the MQTT connection.

B. In Loop Function

The loop() function maintains the MQTT connection and periodically publishes status information.

1. Check whether the MQTT client is connected.
2. Reconnect to the broker if the connection is lost.
3. Execute client.loop() to process MQTT communication.
4. Periodically print the MQTT connection status to the Serial Monitor.
5. Publish connection and system information every 5 seconds.

C. MQTT Reconnect

The MQTT functions manage the MQTT connection and publish status data.

1. mqttReconnect() attempts to connect to the MQTT broker and records the connection time.
2. publishStatusSplit() publishes connection information such as local IP, broker IP, port, and uptime.
3. The device also publishes the MQTT connection state and success count for monitoring.

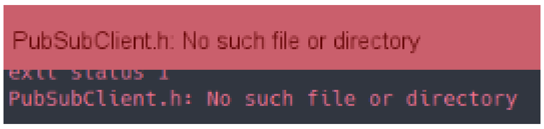
The example code is as follows:
You can download code from here.

```
#include "myeva.h"
#include "Arduino.h"
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

// --------------- Ethernet ---------------
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress localIp(192, 168, 2, 200);      // QEC IP

// --------------- MQTT ---------------
const char* mqtt_server     = "192.168.2.103";
const int      mqtt_port          = 1883;
const char* mqtt_user         = "";
const char* mqtt_pass         = "";
const char* mqtt_clientId = "QEC_MQTT_CONNECT_TEST";

// --------------- Topic ---------------
const char* TOPIC_LOCAL_IP              = "qec/mqtt_test/local_ip";
const char* TOPIC_BROKER_IP            = "qec/mqtt_test/broker_ip";
const char* TOPIC_BROKER_PORT         = "qec/mqtt_test/broker_port";
const char* TOPIC_CONNECT_TIME_MS = "qec/mqtt_test/connect_time_ms";
const char* TOPIC_MQTT_STATE          = "qec/mqtt_test/mqtt_state";
const char* TOPIC_MQTT_CONNECTED    = "qec/mqtt_test/mqtt_connected";
const char* TOPIC_MESSAGE               = "qec/mqtt_test/message";
const char* TOPIC_UPTIME_MS            = "qec/mqtt_test/uptime_ms";
const char* TOPIC_SUCCESS_COUNT     = "qec/mqtt_test/success_count";

EthernetClient ethClient;
PubSubClient client(ethClient);

// --------------- Global ---------------
static unsigned long g_lastPrintMs = 0;
static unsigned long g_lastPublishMs = 0;
static unsigned long g_lastConnectCostMs = 0;
static unsigned long g_connectSuccessCount = 0;

// --------------- Utility ---------------
void ipToString(IPAddress ip, char* out, size_t outSize) {
   snprintf(out, outSize, "%u.%u.%u.%u", ip[0], ip[1], ip[2], ip[3]);
```

```
}

void publishSingleTopic(const char* topic, const char* payload, bool retained = true) {
  client.publish(topic, payload, retained);

  Serial.print("[PUB] ");
  Serial.print(topic);
  Serial.print(" = ");
  Serial.println(payload);
}

void publishStatusSplit(bool mqttOk, int mqttState) {
  char localIpStr[20];
  char brokerIpStr[20];
  char buf[32];

  ipToString(Ethernet.localIP(), localIpStr, sizeof(localIpStr));
  snprintf(brokerIpStr, sizeof(brokerIpStr), "%s", mqtt_server);

  publishSingleTopic(TOPIC_LOCAL_IP, localIpStr);
  publishSingleTopic(TOPIC_BROKER_IP, brokerIpStr);

  snprintf(buf, sizeof(buf), "%d", mqtt_port);
  publishSingleTopic(TOPIC_BROKER_PORT, buf);

  snprintf(buf, sizeof(buf), "%lu", g_lastConnectCostMs);
  publishSingleTopic(TOPIC_CONNECT_TIME_MS, buf);

  snprintf(buf, sizeof(buf), "%d", mqttState);
  publishSingleTopic(TOPIC_MQTT_STATE, buf);

  publishSingleTopic(TOPIC_MQTT_CONNECTED, mqttOk ? "1" : "0");

  if (mqttOk) {
    publishSingleTopic(TOPIC_MESSAGE, "congratulation connect successfully");
  } else {
    publishSingleTopic(TOPIC_MESSAGE, "mqtt connect failed");
  }

  snprintf(buf, sizeof(buf), "%lu", millis());
  publishSingleTopic(TOPIC_UPTIME_MS, buf);
  snprintf(buf, sizeof(buf), "%lu", g_connectSuccessCount);
  publishSingleTopic(TOPIC_SUCCESS_COUNT, buf);
}

static void mqttReconnect() {
  while (!client.connected()) {
    unsigned long t0 = millis();

    Serial.print("Attempting MQTT connection... ");

    bool ok;
    if (mqtt_user[0] == '\0') {
      ok = client.connect(mqtt_clientId);
    } else {
      ok = client.connect(mqtt_clientId, mqtt_user, mqtt_pass);
    }

    g_lastConnectCostMs = millis() - t0;

    if (ok) {
      g_connectSuccessCount++;

      Serial.println("connected!");
      Serial.print("Connect cost(ms): ");
      Serial.println(g_lastConnectCostMs);

      publishStatusSplit(true, client.state());
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.print(", cost(ms)=");
      Serial.print(g_lastConnectCostMs);
      Serial.println(" -> retry in 2s");

      delay(2000);
    }
  }
}

void setup() {
  EVA.begin();
  Serial.begin(3000000);
  delay(200);

  Serial.println("=== Ethernet begin ===");
  Ethernet.begin(mac, localIp);
  delay(300);

  Serial.print("Local IP: ");
  Serial.println(Ethernet.localIP());

  client.setServer(mqtt_server, mqtt_port);

  mqttReconnect();
}

void loop() {
  if (!client.connected()) {
    mqttReconnect();
  }

  client.loop();

  unsigned long now = millis();

  if (now - g_lastPrintMs >= 5000) {
    g_lastPrintMs = now;

    Serial.print("[MQTT] connected=");
    Serial.print(client.connected() ? "YES" : "NO");
    Serial.print(", state=");
    Serial.print(client.state());
    Serial.print(", last_connect_cost_ms=");
    Serial.println(g_lastConnectCostMs);
  }

  if (client.connected() && (now - g_lastPublishMs >= 5000)) {
    g_lastPublishMs = now;
    publishStatusSplit(true, client.state());
  }
}
```

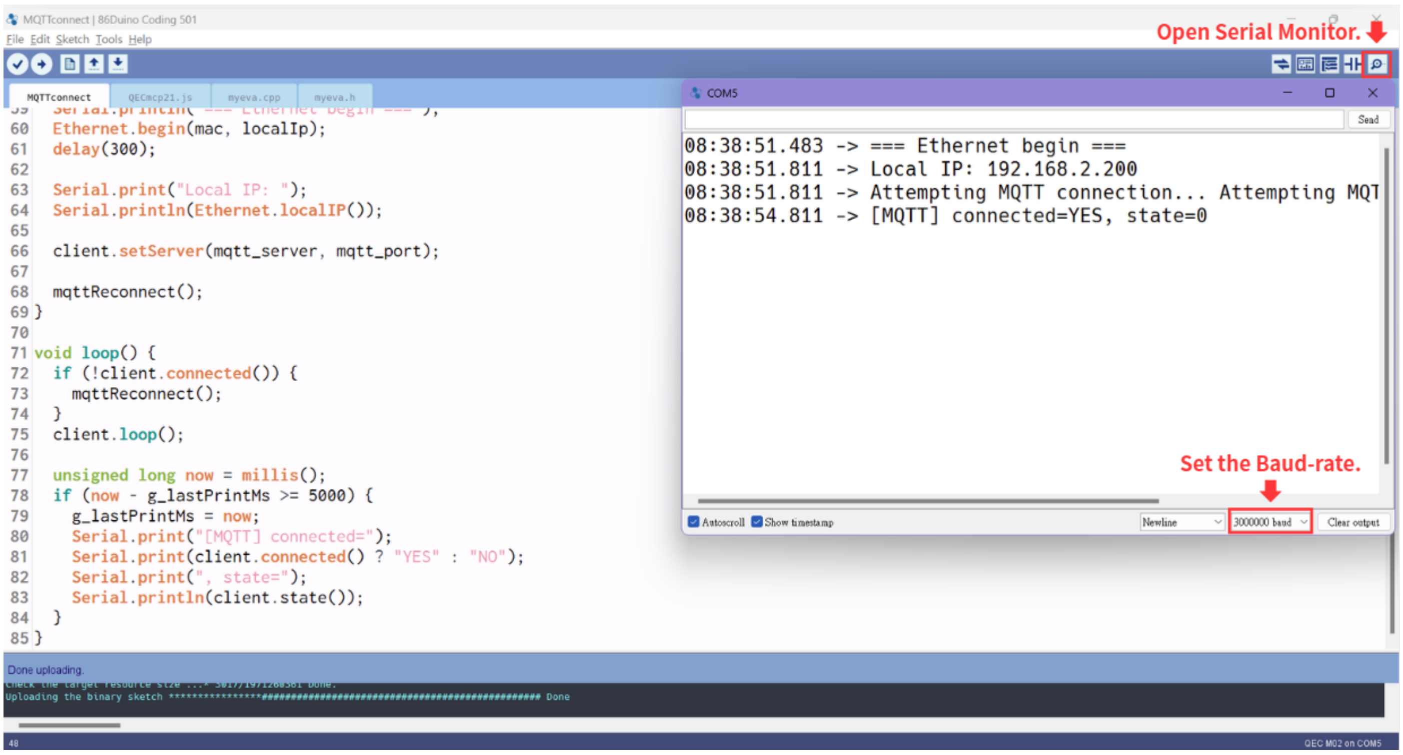
Note:
If you see the error message “PubSubClient.h: No such file or directory” in the window below,

please install the PubSubClient library first. For detailed installation steps, please refer to the Troubleshooting section.



```
PubSubClient.h: No such file or directory
exit status 1
PubSubClient.h: No such file or directory
```

After you successfully upload the program to the QEC-M-02, you can open the Serial Monitor on 86Duino IDE. Please check that the Serial baud rate is the same as your setting.



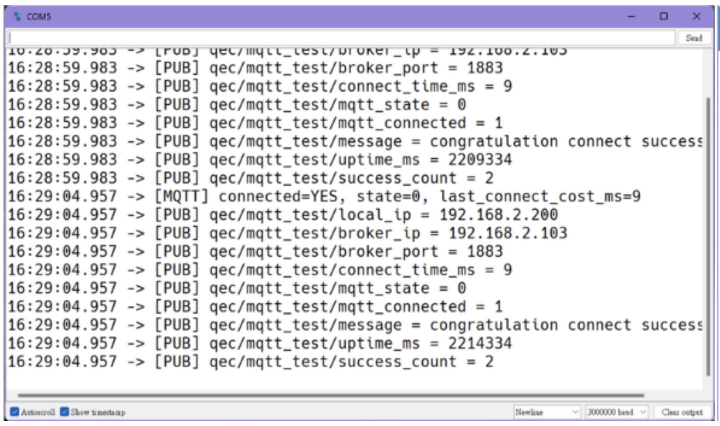
```
  Ethernet.begin(mac, localIp);
  delay(300);

  Serial.print("Local IP: ");
  Serial.println(Ethernet.localIP());

  client.setServer(mqtt_server, mqtt_port);

  mqttReconnect();
}

void loop() {
  if (!client.connected()) {
    mqttReconnect();
  }
  client.loop();

  unsigned long now = millis();
  if (now - g_lastPrintMs >= 5000) {
    g_lastPrintMs = now;
    Serial.print("[MQTT] connected=");
    Serial.print(client.connected() ? "YES" : "NO");
    Serial.print(", state=");
    Serial.println(client.state());
  }
}
```

```
08:38:51.483 -> === Ethernet begin ===
08:38:51.811 -> Local IP: 192.168.2.200
08:38:51.811 -> Attempting MQTT connection... Attempting MQT
08:38:54.811 -> [MQTT] connected=YES, state=0
```

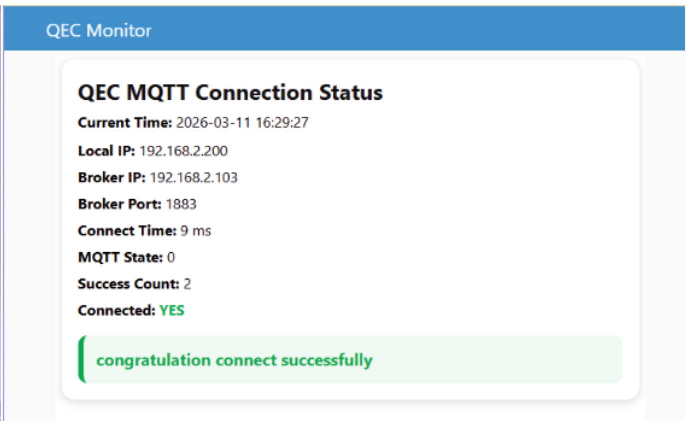
Done uploading.

```
Uploading the binary sketch ***************############################################################ Done
```

If the MQTT connection is successful, a connection success message will appear in the Serial Monitor, and the result will also be displayed on the Node-RED UI, as shown in the figures below.



```
16:28:59.983 -> [PUB] qec/mqtt_test/broker_ip = 192.168.2.103
16:28:59.983 -> [PUB] qec/mqtt_test/broker_port = 1883
16:28:59.983 -> [PUB] qec/mqtt_test/connect_time_ms = 9
16:28:59.983 -> [PUB] qec/mqtt_test/mqtt_state = 0
16:28:59.983 -> [PUB] qec/mqtt_test/mqtt_connected = 1
16:28:59.983 -> [PUB] qec/mqtt_test/message = congratulation connect success
16:28:59.983 -> [PUB] qec/mqtt_test/uptime_ms = 2209334
16:28:59.983 -> [PUB] qec/mqtt_test/success_count = 2
16:29:04.957 -> [MQTT] connected=YES, state=0, last_connect_cost_ms=9
16:29:04.957 -> [PUB] qec/mqtt_test/local_ip = 192.168.2.200
16:29:04.957 -> [PUB] qec/mqtt_test/broker_ip = 192.168.2.103
16:29:04.957 -> [PUB] qec/mqtt_test/broker_port = 1883
16:29:04.957 -> [PUB] qec/mqtt_test/connect_time_ms = 9
16:29:04.957 -> [PUB] qec/mqtt_test/mqtt_state = 0
16:29:04.957 -> [PUB] qec/mqtt_test/mqtt_connected = 1
16:29:04.957 -> [PUB] qec/mqtt_test/message = congratulation connect success
16:29:04.957 -> [PUB] qec/mqtt_test/uptime_ms = 2214334
16:29:04.957 -> [PUB] qec/mqtt_test/success_count = 2
```



QEC Monitor

QEC MQTT Connection Status

Current Time: 2026-03-11 16:29:27
Local IP: 192.168.2.200
Broker IP: 192.168.2.103
Broker Port: 1883
Connect Time: 9 ms
MQTT State: 0
Success Count: 2
Connected: YES

congratulation connect successfully

Example 2 – Motor Control

This example shows how to control a motor via MQTT. The QEC-M-02 receives commands from the broker and publishes motor position and status.

A. In Setup Function

The setup() function initializes the system environment, including EtherCAT, Ethernet communication, MQTT configuration, and motor parameters.

1. Begin serial communication at 3000000 baud for debugging.
2. Initialize the RTC module and 86EVA EtherCAT environment.
3. Configure the motor parameters of VirtualServo1 and set the initial position.
4. Initialize the Ethernet interface with the predefined MAC address and local IP address.
5. Configure the MQTT broker address, port, and callback function.
6. Initialize the watchdog timer and start the SCoop scheduler.

B. In Loop Function

The loop() function maintains the watchdog timer while the main logic runs in SCoop tasks.

1. Continuously reset the watchdog timer using TimerWDT.reset().
2. MQTT communication, motor control, and data publishing are handled by SCoop tasks.

C. In MQTT Function

The MQTT functions handle communication between QEC-M-02 and the MQTT broker.

1. mqttCallback() receives motor control commands.
2. The motor switch topic enables or disables the motor.
3. Target topics set the motor position.
4. publishAck() publishes command acknowledgment messages.
5. mqttEnsureConnected() maintains the MQTT connection.

D. In SCoop Task Functions

The SCoop tasks handle MQTT communication, motor control, and position publishing.

Task 1 – MQTT Service Loop

- Maintain the MQTT connection.
- Execute mqtt.loop() to process incoming messages.

Task 2 – Motor Motion Control

- Check whether the motor is enabled.
- Move the motor to the requested target position.
- Publish the motor motion state through MQTT.

Task 3 – Position Publishing

- Read the current motor position periodically.
- Publish the position to the MQTT topic for Node-RED dashboard monitoring.

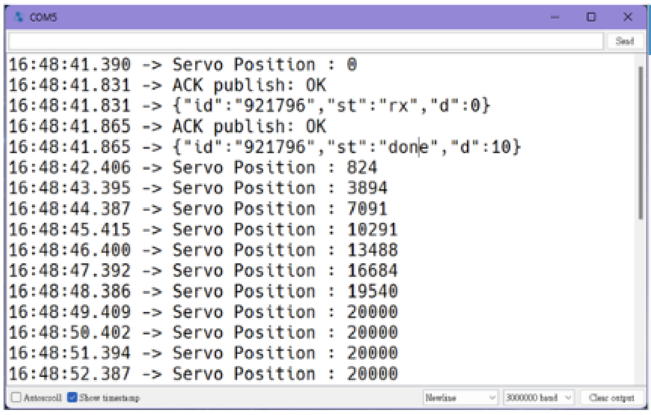
The example code is as follows:
You can download code from here.

```
#include "myeva.h"
#include <Arduino.h>
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>
#include "TimerWDT.h"
#include "RTCZero.h"
#include "SCoop.h"

RTCZero rtc;

// ============================================================================
// Ethernet / MQTT configuration
// ============================================================================
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress localIp(192, 168, 2, 200);

const char* mqtt_server     = "192.168.2.103";
const int      mqtt_port        = 1883;
const char* mqtt_user         = "";
const char* mqtt_pass         = "";
const char* mqtt_clientId = "QEC_EVA_MQTT_MOTOR";

EthernetClient ethClient;
PubSubClient mqtt(ethClient);

// ============================================================================
// MQTT topics
// ============================================================================
static const char* TOPIC_MOTOR_SWITCH_SET = "QEC/Servo/switch/set";
static const char* TOPIC_TARGET0_SET          = "QEC/Servo/target/0/set";
static const char* TOPIC_TARGET10K_SET       = "QEC/Servo/target/10000/set";
static const char* TOPIC_TARGET20K_SET       = "QEC/Servo/target/20000/set";

static const char* TOPIC_POS_PUB      = "QEC/Servo/pos";
static const char* TOPIC_STATE_PUB = "QEC/Servo/state";
static const char* TOPIC_ACK_PUB      = "QEC/Servo/ack";

// ============================================================================
// Control state
// ==============================================================================
static volatile bool g_motorEnabled    = false;
static volatile bool g_targetRequest = false;
static volatile long g_targetPos         = 0;

// ==============================================================================
// Motion state machine
// ==============================================================================
enum MotionState : uint8_t {
   MS_IDLE = 0,
   MS_MOVING_TARGET
};

static MotionState g_motionState = MS_IDLE;

// ==============================================================================
// Pending command metadata
// Store cmd_id and receive time
// ==============================================================================
static char g_pendingCmdId[16] = "";
static unsigned long g_pendingRecvMs = 0;
static bool g_hasPendingMeta = false;

// ==============================================================================
// Stop motor immediately
// ==============================================================================
static void stopMotorNow() {
   long cur = VirtualServo1.read();
   VirtualServo1.write(cur);
}

// ==============================================================================
// Publish device state
// ==============================================================================
static void publishState(const char* s) {
   mqtt.publish(TOPIC_STATE_PUB, s, true);
}

// ==============================================================================
// Extract string field from JSON
// Example: {"cmd_id":"24864"}
// ==============================================================================
static bool extractJsonString(const char* json, const char* key, char* out, size_t outSize) {
   if (!json || !key || !out || outSize == 0) return false;

   char pattern[24];
   snprintf(pattern, sizeof(pattern), "\"%s\":\"", key);

   const char* p = strstr(json, pattern);
   if (!p) return false;

   p += strlen(pattern);
   const char* end = strchr(p, '"');
   if (!end) return false;

   size_t len = (size_t)(end - p);
   if (len >= outSize) len = outSize - 1;

   memcpy(out, p, len);
   out[len] = '\0';
   return true;
}

// ============================================================================
// Extract long value from JSON
// Example: {"value":1}
// ============================================================================
static bool extractJsonLong(const char* json, const char* key, long* outValue) {
   if (!json || !key || !outValue) return false;

   char pattern[24];
   snprintf(pattern, sizeof(pattern), "\"%s\":", key);

   const char* p = strstr(json, pattern);
   if (!p) return false;

   p += strlen(pattern);
   *outValue = atol(p);
   return true;
}

// ============================================================================
// Parse MQTT payload
// Supports:
```

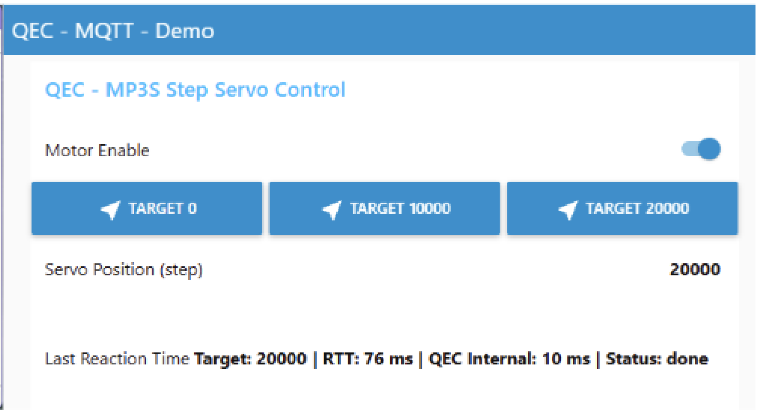
```
// 1) plain number "1"
// 2) JSON {"value":1,"cmd_id":"24864"}
// ==========================================================================
static void parsePayload(
  const char* msg,
  long* outValue,
  char* outCmdId,
  size_t cmdIdSize
) {
  if (!msg || !outValue || !outCmdId) return;

  *outValue = 0;
  outCmdId[0] = '\0';

  if (msg[0] == '{') {
    long value = 0;
    if (extractJsonLong(msg, "value", &value)) {
      *outValue = value;
    }
    extractJsonString(msg, "cmd_id", outCmdId, cmdIdSize);
  } else {
    *outValue = atol(msg);
  }
}

// ==========================================================================
// Send simplified ACK
// Format: {"id":"24864","st":"done","d":21}
// st:
//     sw_on / sw_off / rx / done / ign
// ==========================================================================
static void publishAck(const char* cmdId, const char* status, unsigned long delayMs) {
  char payload[64];

  snprintf(
    payload,
    sizeof(payload),
    "{\"id\":\"%s\",\"st\":\"%s\",\"d\":%lu}",
    cmdId ? cmdId : "",
    status ? status : "",
    delayMs
  );
```

```
    bool ok = mqtt.publish(TOPIC_ACK_PUB, payload, false);

    Serial.print("ACK publish: ");
    Serial.println(ok ? "OK" : "FAIL");
    Serial.println(payload);
}

// ================================================================================
// MQTT callback handler
// ================================================================================
static void mqttCallback(char* topic, byte* payload, unsigned int length) {

    if (length >= 95) length = 95;

    char msg[96];
    memcpy(msg, payload, length);
    msg[length] = '\0';

    long value = 0;
    char cmdId[16];
    parsePayload(msg, &value, cmdId, sizeof(cmdId));

    unsigned long recvMs = millis();

    // ------------------------------------------------------------------------
    // Motor switch control
    // ------------------------------------------------------------------------
    if (strcmp(topic, TOPIC_MOTOR_SWITCH_SET) == 0) {

        g_motorEnabled = (value != 0);

        if (!g_motorEnabled) {
            stopMotorNow();
            g_motionState = MS_IDLE;
            g_targetRequest = false;
            g_hasPendingMeta = false;

            publishState("{\"switch\":0}");
            publishAck(cmdId, "sw_off", 0);
        } else {
            publishState("{\"switch\":1}");
            publishAck(cmdId, "sw_on", 0);
        }
        return;
    }

    // ------------------------------------------------------------------
    // Ignore target commands if switch is OFF
    // ------------------------------------------------------------------
    if (!g_motorEnabled) {
        publishAck(cmdId, "ign", 0);
        return;
    }

    // ------------------------------------------------------------------
    // Target position commands
    // ------------------------------------------------------------------
    if (strcmp(topic, TOPIC_TARGET0_SET) == 0 && value != 0) {
        g_targetPos = 0;
        g_targetRequest = true;

        strncpy(g_pendingCmdId, cmdId, sizeof(g_pendingCmdId) - 1);
        g_pendingCmdId[sizeof(g_pendingCmdId) - 1] = '\0';

        g_pendingRecvMs = recvMs;
        g_hasPendingMeta = true;

        publishAck(g_pendingCmdId, "rx", 0);
        return;
    }

    if (strcmp(topic, TOPIC_TARGET10K_SET) == 0 && value != 0) {
        g_targetPos = 10000;
        g_targetRequest = true;

        strncpy(g_pendingCmdId, cmdId, sizeof(g_pendingCmdId) - 1);
        g_pendingCmdId[sizeof(g_pendingCmdId) - 1] = '\0';

        g_pendingRecvMs = recvMs;
        g_hasPendingMeta = true;

        publishAck(g_pendingCmdId, "rx", 0);
        return;
    }

    if (strcmp(topic, TOPIC_TARGET20K_SET) == 0 && value != 0) {
        g_targetPos = 20000;
        g_targetRequest = true;

        strncpy(g_pendingCmdId, cmdId, sizeof(g_pendingCmdId) - 1);
        g_pendingCmdId[sizeof(g_pendingCmdId) - 1] = '\0';

        g_pendingRecvMs = recvMs;
        g_hasPendingMeta = true;

        publishAck(g_pendingCmdId, "rx", 0);
        return;
    }
}

// ========================================================================
// Ensure MQTT connection and subscribe topics
// ========================================================================
static bool mqttEnsureConnected() {

    if (mqtt.connected()) return true;

    if (mqtt.connect(mqtt_clientId, mqtt_user, mqtt_pass)) {

        mqtt.subscribe(TOPIC_MOTOR_SWITCH_SET);
        mqtt.subscribe(TOPIC_TARGET0_SET);
        mqtt.subscribe(TOPIC_TARGET10K_SET);
        mqtt.subscribe(TOPIC_TARGET20K_SET);

        publishState("{\"mqtt\":1}");
        return true;
    }

    return false;
}

// ========================================================================
// Setup
// ========================================================================
void setup() {
    Serial.begin(3000000);
    rtc.begin();
    EVA.begin();

    VirtualServo1.setVelocity(3200);
    VirtualServo1.setAcceleration(3200);
    VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);
    VirtualServo1.write(0);

    Ethernet.begin(mac, localIp);

    mqtt.setServer(mqtt_server, mqtt_port);
    mqtt.setCallback(mqttCallback);

    TimerWDT.initialize(6000000, true);

    mySCoop.start(0);
}

// ============================================================================
// Main loop
// ============================================================================
void loop() {
    TimerWDT.reset();
}

// ============================================================================
// Task 1: MQTT service loop
// ============================================================================
defineTaskLoop(scoopTask1) {

    TimerWDT.reset();

    mqttEnsureConnected();
    mqtt.loop();

    delay(10);
}

// ============================================================================
// Task 2: Motion state machine
// ===========================================================================
defineTaskLoop(scoopTask2) {

    TimerWDT.reset();

    if (!g_motorEnabled) {
      g_motionState = MS_IDLE;
      delay(20);
      return;
    }

    if (g_targetRequest && g_motionState == MS_IDLE) {

      g_targetRequest = false;
      g_motionState = MS_MOVING_TARGET;

      unsigned long execMs = millis();
      VirtualServo1.write((int)g_targetPos);
      publishState("{\"moving\":1}");

      if (g_hasPendingMeta) {
        unsigned long delayMs = 0;
        if (execMs >= g_pendingRecvMs) {
          delayMs = execMs - g_pendingRecvMs;
        }

        publishAck(g_pendingCmdId, "done", delayMs);
        g_hasPendingMeta = false;
      }
    }

    if (g_motionState == MS_MOVING_TARGET) {
      if (!VirtualServo1.isMoving()) {
        g_motionState = MS_IDLE;
        publishState("{\"moving\":0}");
      }
    }

    delay(20);
}

// ===========================================================================
// Task 3: Publish position every 1 second
// ==================================================================
defineTaskLoop(scoopTask3) {

    TimerWDT.reset();

    long pos = VirtualServo1.read();
    Serial.print("Servo Position : ");
    Serial.println(pos);

    char buf[24];
    ltoa(pos, buf, 10);
    mqtt.publish(TOPIC_POS_PUB, buf, true);

    delay(1000);
}
```

After successfully uploading the program to the QEC-M-02, you can check the servo position in the Serial Monitor (default position = 0). You can then assign the target position using the Node-RED UI or any MQTT publish tool.



```
16:48:41.390 -> Servo Position : 0
16:48:41.831 -> ACK publish: OK
16:48:41.831 -> {"id":"921796","st":"rx","d":0}
16:48:41.865 -> ACK publish: OK
16:48:41.865 -> {"id":"921796","st":"done","d":10}
16:48:42.406 -> Servo Position : 824
16:48:43.395 -> Servo Position : 3894
16:48:44.387 -> Servo Position : 7091
16:48:45.415 -> Servo Position : 10291
16:48:46.400 -> Servo Position : 13488
16:48:47.392 -> Servo Position : 16684
16:48:48.386 -> Servo Position : 19540
16:48:49.409 -> Servo Position : 20000
16:48:50.402 -> Servo Position : 20000
16:48:51.394 -> Servo Position : 20000
16:48:52.387 -> Servo Position : 20000
```



Motor Enable

Servo Position (step) 20000

Last Reaction Time **Target: 20000 | RTT: 76 ms | QEC Internal: 10 ms | Status: done**

Example 3 – Digital I/O Control

This example demonstrates how to control digital outputs via MQTT. The QEC-M-02 receives control commands from the MQTT broker and updates the output states while publishing the current status.

A. In Setup Function

The setup() function initializes the system environment, including EtherCAT, Ethernet communication, MQTT settings, and digital outputs.

1. Begin serial communication at 3000000 baud for debugging.
2. Initialize the 86EVA EtherCAT environment.
3. Configure output pins 0–7 and set the initial state to OFF.
4. Initialize the watchdog timer.
5. Configure the Ethernet connection and MQTT broker settings.

B. In Loop Function

The loop() function maintains the MQTT connection and processes incoming messages.

1. Reset the watchdog timer.
2. Reconnect to the MQTT broker if the connection is lost.
3. Execute client.loop() to process incoming MQTT messages.
4. Process pending I/O commands and update the output state.
5. Publish the updated channel status.

C. In MQTT Function

The MQTT functions handle control messages and update the I/O status.

1. mqttCallback() receives control messages from topics in the format QEC/IO/bit/<ch>/set.
2. The payload determines the ON/OFF state of the channel.
3. Command metadata such as cmd_id may be included in the payload.
4. publishAck() publishes command acknowledgment messages for monitoring.
5. mqttReconnect() maintains the MQTT connection and subscribes to all control topics.

The example code is as follows:
You can download code from here.

```
#include "myeva.h"
#include <Arduino.h>
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>
#include "TimerWDT.h"

// ==========================================================================
// I/O configuration
// ==========================================================================
const int ledCount = 8;
int ledPins[ledCount] = {0, 1, 2, 3, 4, 5, 6, 7};
static bool ioState[ledCount] = {0};

// Interval for printing I/O status to the serial monitor
static unsigned long lastPrintTime = 0;
const unsigned long PRINT_INTERVAL = 1000;

// ==========================================================================
// Ethernet / MQTT configuration
// ==========================================================================
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress localIp(192, 168, 2, 200);

const char* mqtt_server    = "192.168.2.103";
const int    mqtt_port     = 1883;
const char* mqtt_user      = "";
const char* mqtt_pass      = "";
const char* mqtt_clientId = "QEC_EVA_MQTT_IO";

EthernetClient ethClient;
PubSubClient client(ethClient);

// ==========================================================================
// MQTT Topics
//
// Control topic : QEC/IO/bit/<ch>/set
// Status topic   : QEC/IO/bit/<ch>
// ACK topic       : QEC/IO/ack
// ==========================================================================
static const char* TOPIC_BIT_PREFIX = "QEC/IO/bit/";
static const char* TOPIC_ACK_PUB       = "QEC/IO/ack";
// ========================================================================
// Per-channel pending command metadata
// Each channel has its own pending command state
// ========================================================================
static volatile bool g_hasPendingCmd[ledCount] = {false};
static volatile bool g_pendingOn[ledCount] = {false};
static unsigned long g_pendingRecvMs[ledCount] = {0};
static char g_pendingCmdId[ledCount][16];

// ========================================================================
// I/O helper
// Set the output state for a single channel
// ========================================================================
static void setOneIO(int ch, bool on) {
   if (ch < 0 || ch >= ledCount) return;

   ioState[ch] = on;
   digitalWrite(ledPins[ch], on ? HIGH : LOW);
}

// ========================================================================
// Publish one channel state
// ========================================================================
static void publishOne(int ch) {
   if (ch < 0 || ch >= ledCount) return;

   char topic[48];
   char payload[2];

   snprintf(topic, sizeof(topic), "%s%d", TOPIC_BIT_PREFIX, ch);
   payload[0] = ioState[ch] ? '1' : '0';
   payload[1] = '\0';

   client.publish(topic, payload, true);
}

// ========================================================================
// Publish all channel states
// ========================================================================
static void publishAll() {
    for (int i = 0; i < ledCount; i++) {
        publishOne(i);
    }
}

// ============================================================================
// Print IO state to Serial
// Format example: [IO] CH0:1 CH1:0 CH2:1 ...
// ============================================================================
static void printIOState() {
    Serial.print("[IO] ");
    for (int i = 0; i < ledCount; i++) {
        Serial.print("CH");
        Serial.print(i);
        Serial.print(":");
        Serial.print(ioState[i] ? "1" : "0");
        Serial.print(" ");
    }
    Serial.println();
}

// ============================================================================
// Parse ON/OFF string values
// ============================================================================
static bool parseOnOff(const String& s) {
    String t = s;
    t.trim();
    t.toLowerCase();
    return (t == "1" || t == "on" || t == "true" || t == "high");
}

// ============================================================================
// Extract channel index from MQTT topic
// ============================================================================
static bool extractIndexFromTopic(const char* topic, int& outIndex) {
    const size_t prefixLen = strlen(TOPIC_BIT_PREFIX);

    if (strncmp(topic, TOPIC_BIT_PREFIX, prefixLen) != 0) return false;

    const char* p = topic + prefixLen;
    const char* slash = strchr(p, '/');
    if (!slash) return false;
```

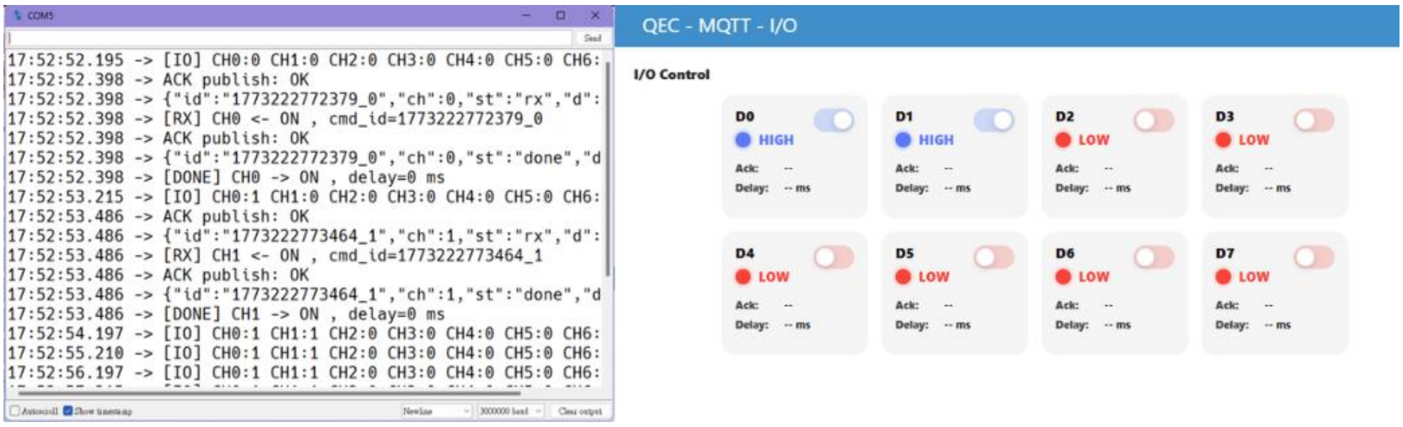
```
  String idxStr = String(p).substring(0, (int)(slash - p));
  idxStr.trim();
  int idx = idxStr.toInt();

  String suffix = String(slash);
  suffix.toLowerCase();
  if (suffix != "/set") return false;

  if (idx < 0 || idx >= ledCount) return false;

  outIndex = idx;
  return true;
}

// =============================================================================
// Extract JSON string field
// =============================================================================
static bool extractJsonString(const char* json, const char* key, char* out, size_t outSize) {
  if (!json || !key || !out || outSize == 0) return false;

  char pattern[24];
  snprintf(pattern, sizeof(pattern), "\"%s\":\"", key);

  const char* p = strstr(json, pattern);
  if (!p) return false;

  p += strlen(pattern);
  const char* end = strchr(p, '"');
  if (!end) return false;

  size_t len = (size_t)(end - p);
  if (len >= outSize) len = outSize - 1;

  memcpy(out, p, len);
  out[len] = '\0';
  return true;
}

// =============================================================================
// Extract JSON long field
// =============================================================================
static bool extractJsonLong(const char* json, const char* key, long* outValue) {
    if (!json || !key || !outValue) return false;

    char pattern[24];
    snprintf(pattern, sizeof(pattern), "\"%s\":", key);

    const char* p = strstr(json, pattern);
    if (!p) return false;

    p += strlen(pattern);
    *outValue = atol(p);
    return true;
}

// ==========================================================================
// Parse payload
// ==========================================================================
static void parsePayload(
    const char* msg,
    long* outValue,
    char* outCmdId,
    size_t cmdIdSize
) {
    if (!msg || !outValue || !outCmdId) return;

    *outValue = 0;
    outCmdId[0] = '\0';

    if (msg[0] == '{') {
        long value = 0;
        if (extractJsonLong(msg, "value", &value)) {
            *outValue = value;
        }
        extractJsonString(msg, "cmd_id", outCmdId, cmdIdSize);
    } else {
        *outValue = atol(msg);
    }
}

// ==========================================================================
// Publish simplified ACK message
// ==========================================================================
static void publishAck(const char* cmdId, int ch, const char* status, unsigned long delayMs) {
  char payload[96];

  snprintf(
    payload,
    sizeof(payload),
    "{\"id\":\"%s\",\"ch\":%d,\"st\":\"%s\",\"d\":%lu}",
    cmdId ? cmdId : "",
    ch,
    status ? status : "",
    delayMs
  );

  bool ok = client.publish(TOPIC_ACK_PUB, payload, false);

  Serial.print("ACK publish: ");
  Serial.println(ok ? "OK" : "FAIL");
  Serial.println(payload);
}

// ============================================================================
// MQTT callback
// ============================================================================
static void mqttCallback(char* topic, byte* payload, unsigned int length) {
  if (length >= 95) length = 95;

  char msg[96];
  memcpy(msg, payload, length);
  msg[length] = '\0';

  int ch = -1;
  if (!extractIndexFromTopic(topic, ch)) return;

  long value = 0;
  char cmdId[16];
  parsePayload(msg, &value, cmdId, sizeof(cmdId));

  bool on = (value != 0);
  unsigned long recvMs = millis();

  // Update pending state for this channel
  g_pendingOn[ch] = on;
```

```
    g_pendingRecvMs[ch] = recvMs;
    g_hasPendingCmd[ch] = true;

    strncpy(g_pendingCmdId[ch], cmdId, sizeof(g_pendingCmdId[ch]) - 1);
    g_pendingCmdId[ch][sizeof(g_pendingCmdId[ch]) - 1] = '\0';

    publishAck(g_pendingCmdId[ch], ch, "rx", 0);

    Serial.print("[RX] CH");
    Serial.print(ch);
    Serial.print(" <- ");
    Serial.print(on ? "ON" : "OFF");
    Serial.print(" , cmd_id=");
    Serial.println(g_pendingCmdId[ch]);
}

// ================================================================
// Subscribe to all control topics
// ================================================================
static void subscribeAll() {
    char topic[48];
    for (int i = 0; i < ledCount; i++) {
        snprintf(topic, sizeof(topic), "%s%d/set", TOPIC_BIT_PREFIX, i);
        client.subscribe(topic);
    }
}

// ================================================================
// Ensure MQTT connection
// ================================================================
static void mqttReconnect() {
    static uint32_t lastAttempt = 0;
    const uint32_t now = millis();

    if (now - lastAttempt < 2000) return;
    lastAttempt = now;

    if (client.connected()) return;

    bool ok = false;

    if (mqtt_user && mqtt_user[0] != '\0') {
        ok = client.connect(mqtt_clientId, mqtt_user, mqtt_pass);
    } else {
        ok = client.connect(mqtt_clientId);
    }

    if (ok) {
        subscribeAll();
        publishAll();

        Serial.println("[MQTT] connected");
    } else {
        Serial.print("[MQTT] reconnect failed, rc=");
        Serial.println(client.state());
    }
}

// ==========================================================================
// setup
// ==========================================================================
void setup() {
    Serial.begin(3000000);
    EVA.begin();

    for (int i = 0; i < ledCount; i++) {
        pinMode(ledPins[i], OUTPUT);
        digitalWrite(ledPins[i], LOW);
        ioState[i] = false;

        g_hasPendingCmd[i] = false;
        g_pendingOn[i] = false;
        g_pendingRecvMs[i] = 0;
        g_pendingCmdId[i][0] = '\0';
    }

    TimerWDT.initialize(6000000, true);

    Ethernet.begin(mac, localIp);
    delay(200);

    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(mqttCallback);
```

```
  Serial.println("=== IO MQTT ACK Mode Start (Per-Channel Pending) ===");
  Serial.print("Local IP: ");
  Serial.println(Ethernet.localIP());
}

// ===========================================================================
// loop
// ===========================================================================
void loop() {
  TimerWDT.reset();

  if (!client.connected()) {
    mqttReconnect();
  }

  client.loop();

  // Process pending command per channel
  for (int ch = 0; ch < ledCount; ch++) {
    if (!g_hasPendingCmd[ch]) continue;

    bool on = g_pendingOn[ch];
    unsigned long recvMs = g_pendingRecvMs[ch];

    char cmdId[16];
    strncpy(cmdId, g_pendingCmdId[ch], sizeof(cmdId) - 1);
    cmdId[sizeof(cmdId) - 1] = '\0';

    g_hasPendingCmd[ch] = false;

    // Execute actual I/O operation
    setOneIO(ch, on);
    publishOne(ch);

    unsigned long doneMs = millis();
    unsigned long delayMs = 0;
    if (doneMs >= recvMs) {
      delayMs = doneMs - recvMs;
    }

    publishAck(cmdId, ch, "done", delayMs);
```

```
        Serial.print("[DONE] CH");
        Serial.print(ch);
        Serial.print(" -> ");
        Serial.print(on ? "ON" : "OFF");
        Serial.print(" , delay=");
        Serial.print(delayMs);
        Serial.println(" ms");
    }

    if (millis() - lastPrintTime >= PRINT_INTERVAL) {
        lastPrintTime = millis();
        printIOState();
    }
}
```

After successfully uploading the program to the QEC-M-02, you can check the I/O status in the Serial Monitor (default = 0). The I/O can then be turned ON or OFF using the Node-RED UI or any MQTT publishing tool.



```
17:52:52.195 -> [IO] CH0:0 CH1:0 CH2:0 CH3:0 CH4:0 CH5:0 CH6:
17:52:52.398 -> ACK publish: OK
17:52:52.398 -> {"id":"1773222772379_0","ch":0,"st":"rx","d":
17:52:52.398 -> [RX] CH0 <- ON , cmd_id=1773222772379_0
17:52:52.398 -> ACK publish: OK
17:52:52.398 -> {"id":"1773222772379_0","ch":0,"st":"done","d
17:52:52.398 -> [DONE] CH0 -> ON , delay=0 ms
17:52:53.215 -> [IO] CH0:1 CH1:0 CH2:0 CH3:0 CH4:0 CH5:0 CH6:
17:52:53.486 -> ACK publish: OK
17:52:53.486 -> {"id":"1773222773464_1","ch":1,"st":"rx","d":
17:52:53.486 -> [RX] CH1 <- ON , cmd_id=1773222773464_1
17:52:53.486 -> ACK publish: OK
17:52:53.486 -> {"id":"1773222773464_1","ch":1,"st":"done","d
17:52:53.486 -> [DONE] CH1 -> ON , delay=0 ms
17:52:54.197 -> [IO] CH0:1 CH1:1 CH2:0 CH3:0 CH4:0 CH5:0 CH6:
17:52:55.210 -> [IO] CH0:1 CH1:1 CH2:0 CH3:0 CH4:0 CH5:0 CH6:
17:52:56.197 -> [IO] CH0:1 CH1:1 CH2:0 CH3:0 CH4:0 CH5:0 CH6:
```

QEC - MQTT - I/O

I/O Control

D0 ● HIGH Ack: -- Delay: -- ms	D1 ● HIGH Ack: -- Delay: -- ms	D2 ● LOW Ack: -- Delay: -- ms	D3 ● LOW Ack: -- Delay: -- ms
D4 ● LOW Ack: -- Delay: -- ms	D5 ● LOW Ack: -- Delay: -- ms	D6 ● LOW Ack: -- Delay: -- ms	D7 ● LOW Ack: -- Delay: -- ms

Example 4 - Device Status

This example demonstrates how to monitor device status using MQTT. The QEC-M-02 reads power and status data from the EVA Master and EtherCAT Slaves and publishes the information for dashboard monitoring.

A. In Setup Function

The setup() function initializes the system environment, including EtherCAT, RTC, Ethernet communication, MQTT settings, and the watchdog timer.

1. Begin serial communication at 3000000 baud.
2. Initialize the RTC module and 86EVA EtherCAT environment.
3. Initialize the watchdog timer.
4. Configure the Ethernet connection and MQTT broker settings.

B. In Loop Function

The loop() function periodically collects device data and publishes it via MQTT.

1. Reset the watchdog timer.
2. Ensure the MQTT client remains connected.
3. Read the latest data from the EVA Master and EtherCAT Slaves.
4. Publish Master data as a JSON message and Slave data to individual topics.
5. Device data is published every 1 second, while device names are published every 30 seconds.

C. In MQTT Function

The MQTT functions manage the MQTT connection and data publishing.

1. mqttReconnect() ensures the MQTT client reconnects if the connection is lost.
2. publishMasterData() publishes Master device information.
3. publishSlaveData() publishes each Slave device status to MQTT topics.

The example code is as follows:
You can download code from here.

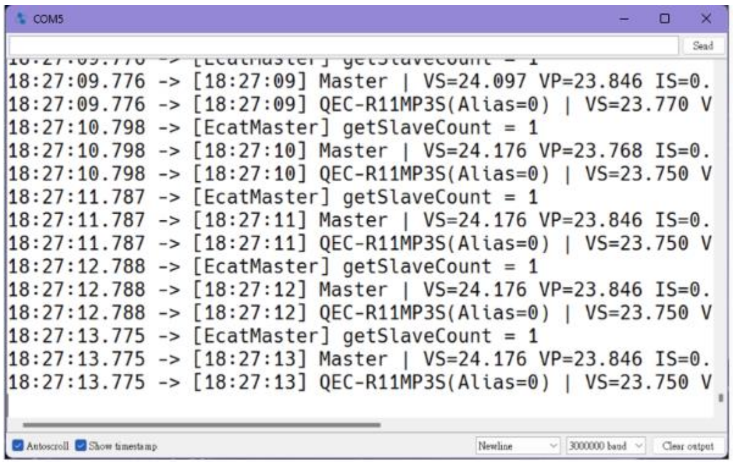
```
#include "myeva.h"
#include "TimerWDT.h"
#include "RTCZero.h"

#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

RTCZero rtc;

// ===========================================================================
// Publish timing control
//============================================================================
static unsigned long g_lastTickMs = 0;
const unsigned long PUBLISH_INTERVAL_MS = 1000;      // publish every 1 second

// ===========================================================================
// Master (local EVA) variables
// ===========================================================================
double g_masterVs    = 0.0;
double g_masterVp    = 0.0;
double g_masterIs    = 0.0;
double g_masterIp    = 0.0;
double g_masterTemp = 0.0;

// ===========================================================================
// RTC time variables
// ===========================================================================
int g_year = 0, g_month = 0, g_day = 0;
int g_hour = 0, g_minute = 0, g_second = 0;

// ===========================================================================
// Slave variables
// ===========================================================================
const int MAX_SLAVES = 16;

int      g_slaveCount = 0;
int      g_slaveAlias[MAX_SLAVES] = {0};

// Publish slave names every 30 seconds
```

```
static unsigned long g_lastNamePubMs = 0;
const unsigned long NAME_PUBLISH_INTERVAL_MS = 30000;

// Slave name table
const char* g_slaveName[MAX_SLAVES] = {nullptr};

// Slave data arrays
double g_slaveVs[MAX_SLAVES]           = {0.0};
double g_slaveVp[MAX_SLAVES]           = {0.0};
double g_slaveIs[MAX_SLAVES]           = {0.0};
double g_slaveIp[MAX_SLAVES]           = {0.0};
double g_slaveTemp[MAX_SLAVES]         = {0.0};
double g_slaveWorkingHours[MAX_SLAVES] = {0.0};
double g_slaveBootTimes[MAX_SLAVES]    = {0.0};

// ============================================================================
// Ethernet / MQTT configuration
// ============================================================================
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress localIp(192, 168, 2, 200);

const char* mqtt_server   = "192.168.2.103";
const int   mqtt_port     = 1883;
const char* mqtt_user     = "";
const char* mqtt_pass     = "";
const char* mqtt_clientId = "QEC_EVA_MQTT_DEVICE";

// Master topic
const char* TOPIC_MASTER  = "QEC/Master";

EthernetClient ethClient;
PubSubClient client(ethClient);

// ============================================================================
// Update RTC variables
// ============================================================================
static void updateRtcVariables() {
  g_year   = rtc.getYear() + 2000;
  g_month  = rtc.getMonth();
  g_day    = rtc.getDay();
  g_hour   = rtc.getHours();
  g_minute = rtc.getMinutes();
```

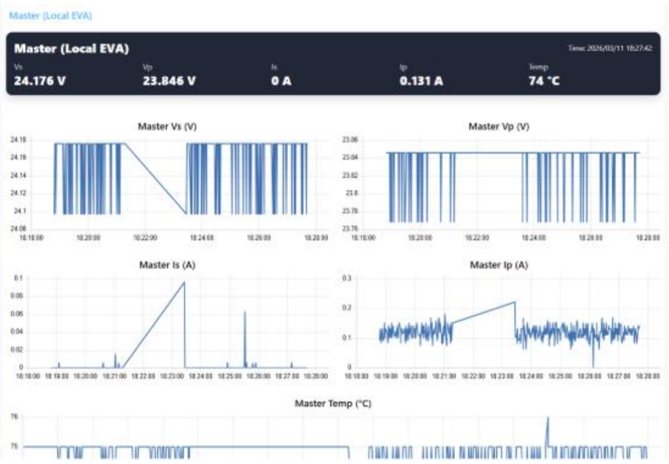
```
    g_second = rtc.getSeconds();
}

// ===========================================================================
// Update Master data
// ===========================================================================
static void updateMasterVariables() {
    g_masterVs     = EVA.getUsVoltage(-1);
    g_masterVp     = EVA.getUpVoltage(-1);
    g_masterIs     = EVA.getIsCurrent(-1);
    g_masterIp     = EVA.getIpCurrent(-1);
    g_masterTemp = EVA.getTemperature(-1);
}

// ===========================================================================
// Update all Slave data
// ===========================================================================
static void updateSlaveVariables() {

    int count = EcatMaster.getSlaveCount();
    if (count < 0) count = 0;
    if (count > MAX_SLAVES) count = MAX_SLAVES;

    g_slaveCount = count;

    for (int i = 0; i < g_slaveCount; i++) {

        int alias = EcatMaster.getAliasAddress(i);
        g_slaveAlias[i] = alias;

        const char* name = getMachineName(alias);
        g_slaveName[i] = (name && name[0] != '\0') ? name : "unknown";

        g_slaveVs[i] = EVA.getUsVoltage(alias);
        g_slaveVp[i] = EVA.getUpVoltage(alias);
        g_slaveIs[i] = EVA.getIsCurrent(alias);
        g_slaveIp[i] = EVA.getIpCurrent(alias);

        g_slaveTemp[i]              = EVA.getTemperature(alias);
        g_slaveWorkingHours[i] = EVA.getWorkingHours(alias);
        g_slaveBootTimes[i]       = EVA.getBootTimes(alias);
    }
    Serial.print("[EcatMaster] getSlaveCount = ");
    Serial.println(g_slaveCount);
}

// ============================================================================
// MQTT reconnect handler
// ============================================================================
static void mqttReconnect() {

    while (!client.connected()) {

        Serial.print("Attempting MQTT connection...");

        if (client.connect(mqtt_clientId, mqtt_user, mqtt_pass)) {

            Serial.println("connected");
            client.loop();
            delay(50);

        } else {

            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" retry in 2s");
            delay(2000);
        }
    }
}

// ============================================================================
// Print device data (single-line log format)
// ============================================================================
static void printDeviceData() {

    char timeStr[16];
    snprintf(timeStr, sizeof(timeStr), "%02d:%02d:%02d", g_hour, g_minute, g_second);

    // Master
    Serial.print("[");
    Serial.print(timeStr);
    Serial.print("] Master | VS=");
    Serial.print(g_masterVs,3);
    Serial.print(" VP=");
    Serial.print(g_masterVp,3);
    Serial.print(" IS=");
    Serial.print(g_masterIs,3);
    Serial.print(" IP=");
    Serial.print(g_masterIp,3);
    Serial.print(" Temp=");
    Serial.println(g_masterTemp,2);

    // Slaves
    for (int i = 0; i < g_slaveCount; i++) {

        const char* name = (g_slaveName[i] && g_slaveName[i][0]) ? g_slaveName[i] : "unknown";

        Serial.print("[");
        Serial.print(timeStr);
        Serial.print("] ");

        Serial.print(name);
        Serial.print("(Alias=");
        Serial.print(g_slaveAlias[i]);
        Serial.print(") | VS=");
        Serial.print(g_slaveVs[i],3);

        Serial.print(" VP=");
        Serial.print(g_slaveVp[i],3);

        Serial.print(" IS=");
        Serial.print(g_slaveIs[i],3);

        Serial.print(" IP=");
        Serial.print(g_slaveIp[i],3);

        Serial.print(" Temp=");
        Serial.print(g_slaveTemp[i],2);

        Serial.print(" WH=");
        Serial.print(g_slaveWorkingHours[i],2);

        Serial.print(" BT=");
        Serial.println((long)g_slaveBootTimes[i]);
```

```
    }
}

// ==============================================================================
// Publish Master JSON
// ==============================================================================
static bool publishMasterData() {

    char payload[320];

    int n = snprintf(payload, sizeof(payload),
        "{"
            "\"time\":\"%04d/%02d/%02d %02d:%02d:%02d\","
            "\"vs\":%.3f,"
            "\"vp\":%.3f,"
            "\"is\":%.3f,"
            "\"ip\":%.3f,"
            "\"temp\":%.2f"
        "}",
        g_year, g_month, g_day, g_hour, g_minute, g_second,
        g_masterVs, g_masterVp, g_masterIs, g_masterIp, g_masterTemp
    );

    if (n <= 0 || n >= (int)sizeof(payload)) {
        Serial.println("[ERR] master payload overflow");
        return false;
    }

    return client.publish(TOPIC_MASTER, payload);
}

// ==============================================================================
// Publish Slave data by split topics
// ==============================================================================
static bool publishSlaveData(bool publishNameNow) {

    char topic[128];
    bool allOk = true;

    for (int i = 0; i < g_slaveCount; i++) {

        int alias = g_slaveAlias[i];
        snprintf(topic, sizeof(topic), "QEC/Slave/%d/vs", alias);
        if (!client.publish(topic, String(g_slaveVs[i],3).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/vp", alias);
        if (!client.publish(topic, String(g_slaveVp[i],3).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/is", alias);
        if (!client.publish(topic, String(g_slaveIs[i],3).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/ip", alias);
        if (!client.publish(topic, String(g_slaveIp[i],3).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/tp", alias);
        if (!client.publish(topic, String(g_slaveTemp[i],2).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/wh", alias);
        if (!client.publish(topic, String(g_slaveWorkingHours[i],2).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/bt", alias);
        if (!client.publish(topic, String((long)g_slaveBootTimes[i]).c_str())) allOk = false;

        snprintf(topic, sizeof(topic), "QEC/Slave/%d/ts", alias);
        char ts[16];
        snprintf(ts, sizeof(ts), "%02d:%02d:%02d", g_hour, g_minute, g_second);
        if (!client.publish(topic, ts)) allOk = false;

        if (publishNameNow) {
            snprintf(topic, sizeof(topic), "QEC/Slave/%d/name", alias);
            const char* name = (g_slaveName[i] && g_slaveName[i][0]) ? g_slaveName[i] : "unknown";
            if (!client.publish(topic, name)) allOk = false;
        }
    }

    return allOk;
}

// ============================================================================
// Setup
// ============================================================================
void setup() {
    Serial.begin(3000000);
    rtc.begin();
    EVA.begin();

    TimerWDT.initialize(6000000, true);

    Ethernet.begin(mac, localIp);
    client.setServer(mqtt_server, mqtt_port);

    g_lastTickMs = millis();
}

// ============================================================================
// Main loop
// ============================================================================
void loop() {

    TimerWDT.reset();

    if (!client.connected()) {
        mqttReconnect();
        return;
    }

    client.loop();

    unsigned long now = millis();

    if (now - g_lastTickMs >= PUBLISH_INTERVAL_MS) {

        g_lastTickMs = now;

        updateRtcVariables();
        updateMasterVariables();
        updateSlaveVariables();

        printDeviceData();

        publishMasterData();

        bool publishNameNow = false;
        if (now - g_lastNamePubMs >= NAME_PUBLISH_INTERVAL_MS) {
            g_lastNamePubMs = now;
            publishNameNow = true;
        }

        publishSlaveData(publishNameNow);
    }
}
```

After successfully uploading the program to the QEC-M-02, you can check the device status in the Serial Monitor. You can also use the Node-RED UI or any MQTT publishing tool to monitor the device data.



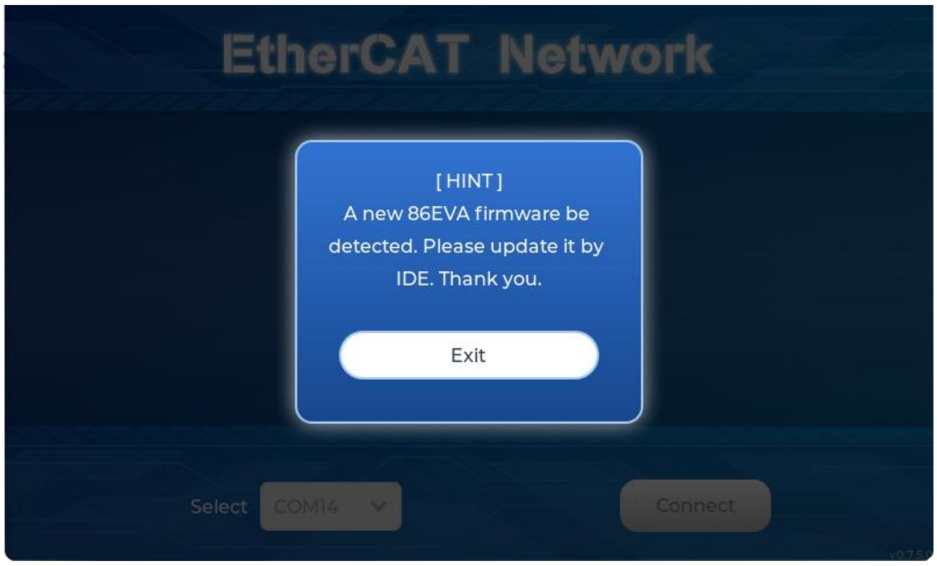
```
18:27:09.776 -> [EcatMaster] getSlaveCount = 1
18:27:09.776 -> [18:27:09] Master | VS=24.097 VP=23.846 IS=0.
18:27:09.776 -> [18:27:09] QEC-R11MP3S(Alias=0) | VS=23.770 V
18:27:10.798 -> [EcatMaster] getSlaveCount = 1
18:27:10.798 -> [18:27:10] Master | VS=24.176 VP=23.768 IS=0.
18:27:10.798 -> [18:27:10] QEC-R11MP3S(Alias=0) | VS=23.750 V
18:27:11.787 -> [EcatMaster] getSlaveCount = 1
18:27:11.787 -> [18:27:11] Master | VS=24.176 VP=23.846 IS=0.
18:27:11.787 -> [18:27:11] QEC-R11MP3S(Alias=0) | VS=23.750 V
18:27:12.788 -> [EcatMaster] getSlaveCount = 1
18:27:12.788 -> [18:27:12] Master | VS=24.176 VP=23.846 IS=0.
18:27:12.788 -> [18:27:12] QEC-R11MP3S(Alias=0) | VS=23.750 V
18:27:13.775 -> [EcatMaster] getSlaveCount = 1
18:27:13.775 -> [18:27:13] Master | VS=24.176 VP=23.846 IS=0.
18:27:13.775 -> [18:27:13] QEC-R11MP3S(Alias=0) | VS=23.750 V
```



Troubleshooting

QEC-M-02 cannot successfully upload code

When you are unable to successfully upload code, please open 86EVA to check if your QEC EtherCAT MDevice's environment is abnormal. As shown in the figure below, please try updating your QEC EtherCAT MDevice's environment, which will include the following three items: Bootloader, EtherCAT firmware, and EtherCAT tool.


[HINT]
A new 86EVA firmware be detected. Please update it by IDE. Thank you.

Now, we will further explain how to proceed with the update:

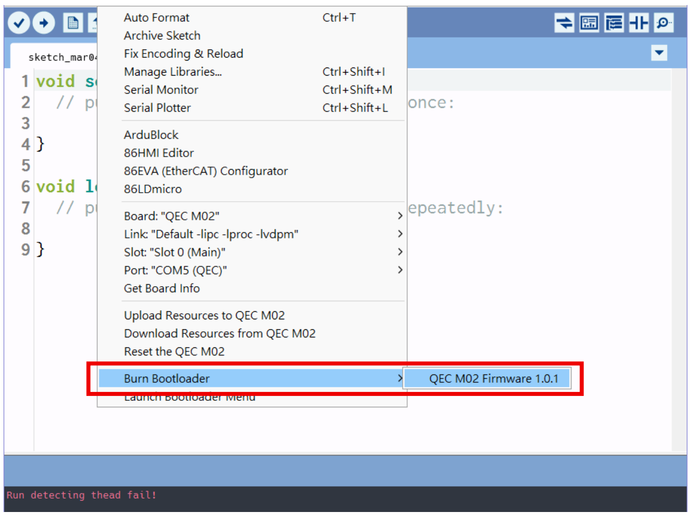
Step 1: Setting up QEC-M

1. Download and install 86Duino IDE 501+ (or a newer version).
   You can download it from Software.
2. Connect the QEC-M: Use a USB cable to connect the QEC-M to your computer.
3. Open 86Duino IDE: After the installation is complete, open the 86Duino IDE software.
4. Select Board: From the IDE menu, choose "**Tools**" > "**Board**" > "**QEC-M-02**" (or the specific model of QEC-M you are using).
5. Select Port: From the IDE menu, choose "**Tools**" > "**Port**" and select the USB port to which the QEC-M is connected.

Step 2: Click “Burn Bootloader” button

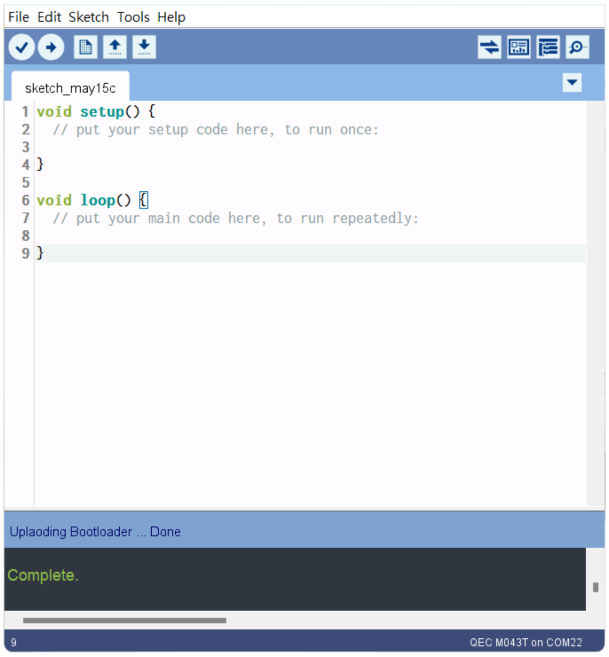
After connecting to your QEC-M product, go to “**Tools**”> “**Burn Bootloader**”.
The currently selected QEC-M name will appear. Clicking on it will start the update process, which will take approximately 5-20 minutes.

- QEC-M-02:



```
Run detecting thead fail!
```

Step 3: Complete the Update



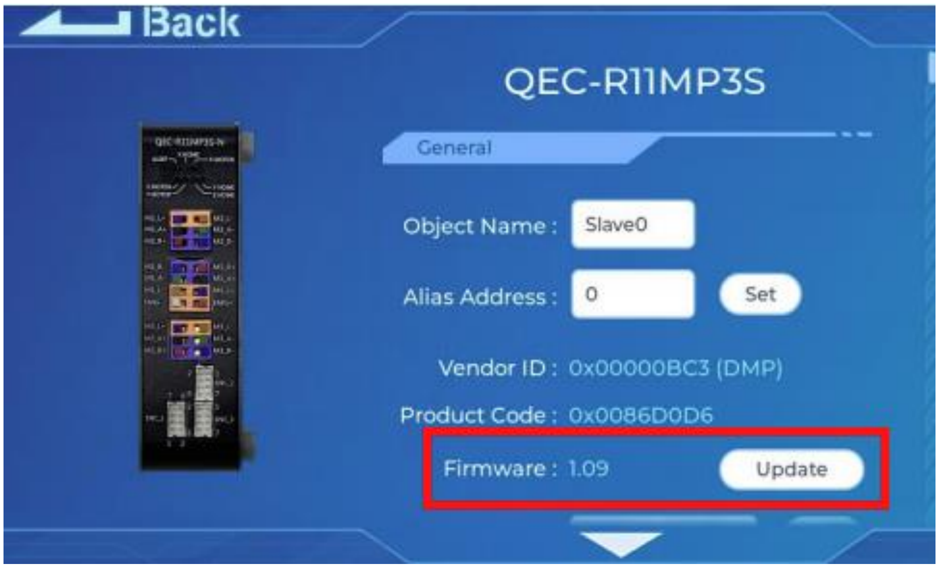
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
Uplaoding Bootloader ... Done
Complete.
```

After completing the above steps, your QEC-M has been successfully updated to the latest version of the development environment.

QEC-R11MP3S – Firmware Update

If there have any difference firmware version for the QEC SubDevice, the “Update” button will appear.



QEC-R11MP3S

Object Name : Slave0

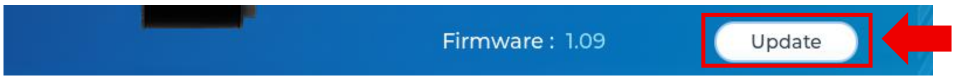
Alias Address : 0

Vendor ID : 0x00000BC3 (DMP)

Product Code : 0x0086D0D6

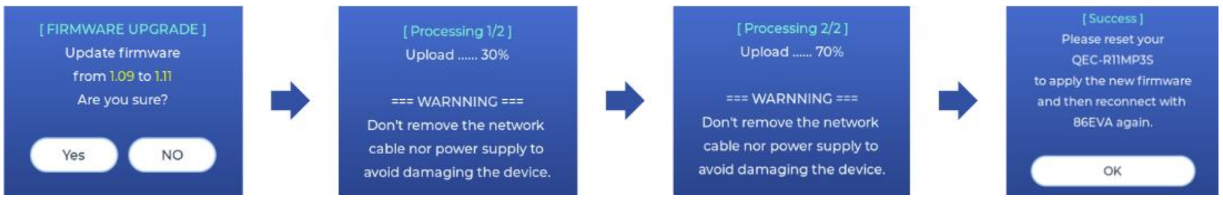
Firmware : 1.09

Please click it to update/downgrade the device.



Firmware : 1.09

After Update finish, please power cycle the device.



[FIRMWARE UPGRADE]
Update firmware
from 1.09 to 1.11
Are you sure?

[Processing 1/2]
Upload 30%

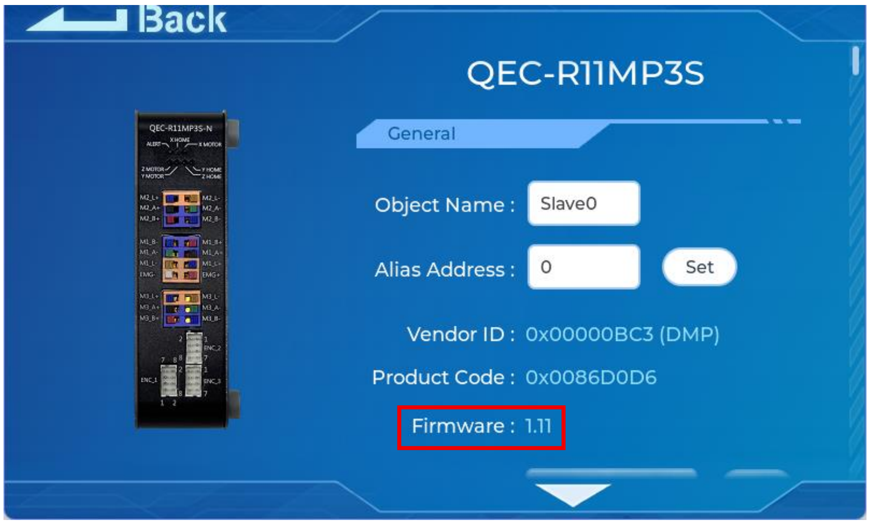
=== WARNNING ===
Don't remove the network cable nor power supply to avoid damaging the device.

[Processing 2/2]
Upload 70%

=== WARNNING ===
Don't remove the network cable nor power supply to avoid damaging the device.

[Success]
Please reset your QEC-R11MP3S to apply the new firmware and then reconnect with 86EVA again.

After power cycling the device, the firmware will be updated to the latest version.



QEC-R11MP3S

Object Name : Slave0

Alias Address : 0

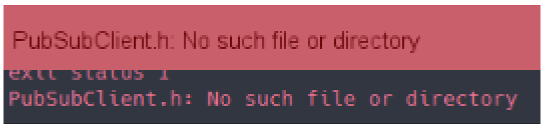
Vendor ID : 0x00000BC3 (DMP)

Product Code : 0x0086D0D6

Firmware : 1.11

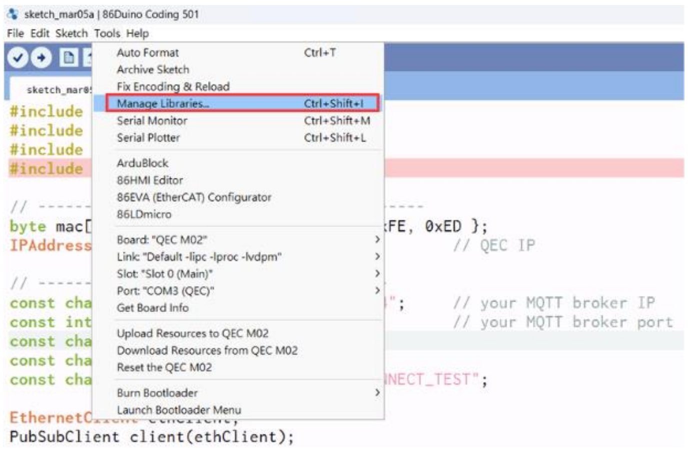
Install PubSubClient library

If you see the error message “PubSubClient.h: No such file or directory” in the window below, please install the PubSubClient library first.



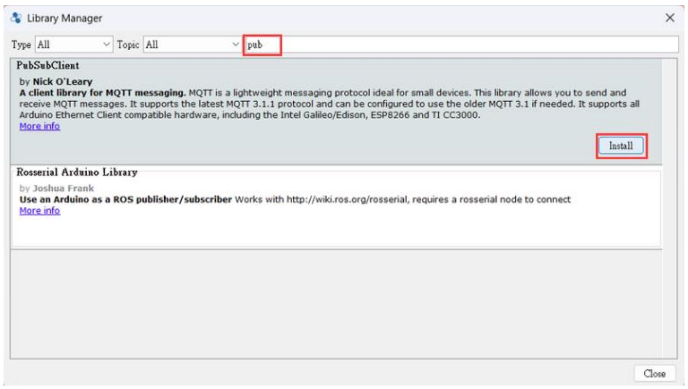
```
PubSubClient.h: No such file or directory
exit status 1
PubSubClient.h: No such file or directory
```

Go to the top menu bar, click Tools, and select **Manage Libraries**.


sketch_mar05a | 86Duino Coding 501
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ArduBlock
86HMI Editor
86EVA (EtherCAT) Configurator
86LDmicro
Board: "QEC M02"
Link: "Default -lipc -lproc -lvdpm"
Slot: "Slot 0 (Main)"
Port: "COM3 (QEC)"
Get Board Info
Upload Resources to QEC M02
Download Resources from QEC M02
Reset the QEC M02
Burn Bootloader
Launch Bootloader Menu

```
PubSubClient client(ethClient);
```

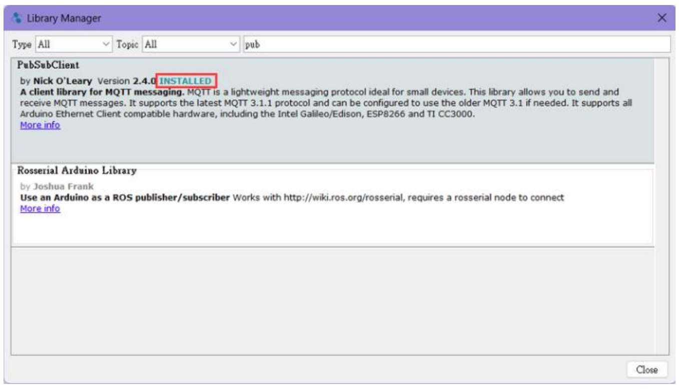
Enter **"pub"** in the search bar, find the **PubSubClient** package, and click the **Install** button.



PubSubClient
by Nick O'Leary
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.
More info

Rosserial Arduino Library
by Joshua Frank
Use an Arduino as a ROS publisher/subscriber Works with http://wiki.ros.org/rosserial, requires a rosserial node to connect
More info

After installation, if you see **INSTALLED**, it means the package has been successfully installed.



PubSubClient
by Nick O'Leary Version 2.4.0 INSTALLED
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000.
More info

Rosserial Arduino Library
by Joshua Frank
Use an Arduino as a ROS publisher/subscriber Works with http://wiki.ros.org/rosserial, requires a rosserial node to connect
More info

Warranty

This product is warranted to be in good working order for a period of one year from the date of purchase. Should this product fail to be in good working order at any time during this period, we will, at our option, replace or repair it at no additional charge except as set forth in the following terms. This warranty does not apply to products damaged by misuse, modifications, accident or disaster. Vendor assumes no liability for any damages, lost profits, lost savings or any other incidental or consequential damage resulting from the use, misuse of, originality to use this product. Vendor will not be liable for any claim made by any other related party. Return authorization must be obtained from the vendor before returned merchandise will be accepted. Authorization can be obtained by calling or faxing the vendor and requesting a Return Merchandise Authorization (RMA) number. Returned goods should always be accompanied by a clear problem description.