Start Guide

QEC-M-02: Modbus TCP Communication with CIMON SCADA

86Duino Coding IDE 501

EtherCAT / Modbus Library

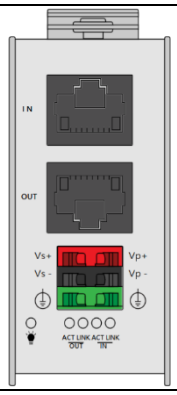
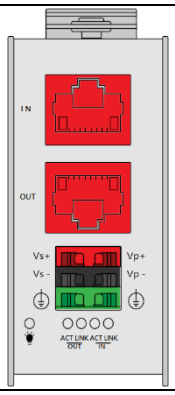
(Version 1.0)

Preface

In this guide, we will show you how to use the EtherCAT MDevice QEC-M-02 with CIMON SCADA through Modbus TCP communication. The QEC-M-02 works as a Modbus TCP Slave, allowing CIMON SCADA to monitor and control local I/O, remote EtherCAT I/O, external Modbus RTU I/O, and a single-axis stepper motor controller.

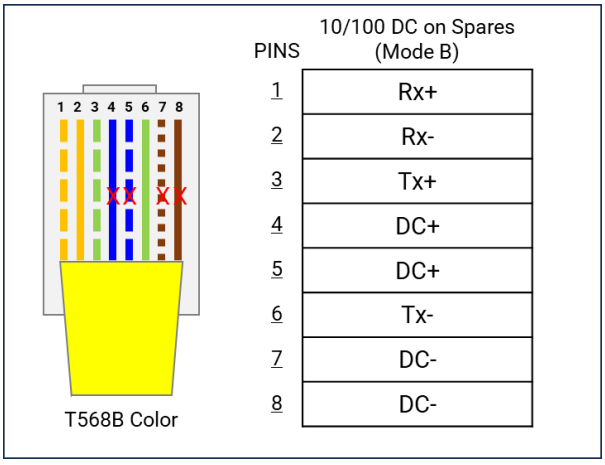
Notes QEC's PoE (Power over Ethernet)

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.

	
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:



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

Introduction

This document provides a quick start guide for using CIMON SCADA to communicate with the QEC-M-02 through Modbus TCP. It covers the basic workflow from QEC setup and 86Duino sketch upload to CIMON SCADA Tag configuration, screen design, and CIMON X testing.

In this setup:

- QEC-M-02 works as the Modbus TCP Slave.
- CIMON SCADA works as the Modbus TCP Master / Client.
- 86Duino IDE is used to upload the control sketch.
- CimonD is used to configure Devices, Tags, and screens.
- CimonX is used to run the project and verify the result.

This guide includes four examples:

Example	Application
Ex 1	QEC-M-02 local I/O control
Ex 2	QEC-R11CFFG EtherCAT I/O control
Ex 3	External Modbus RTU I/O board
Ex 4	QEC-R11MP3S Motor control

Through these examples, users will learn how to:

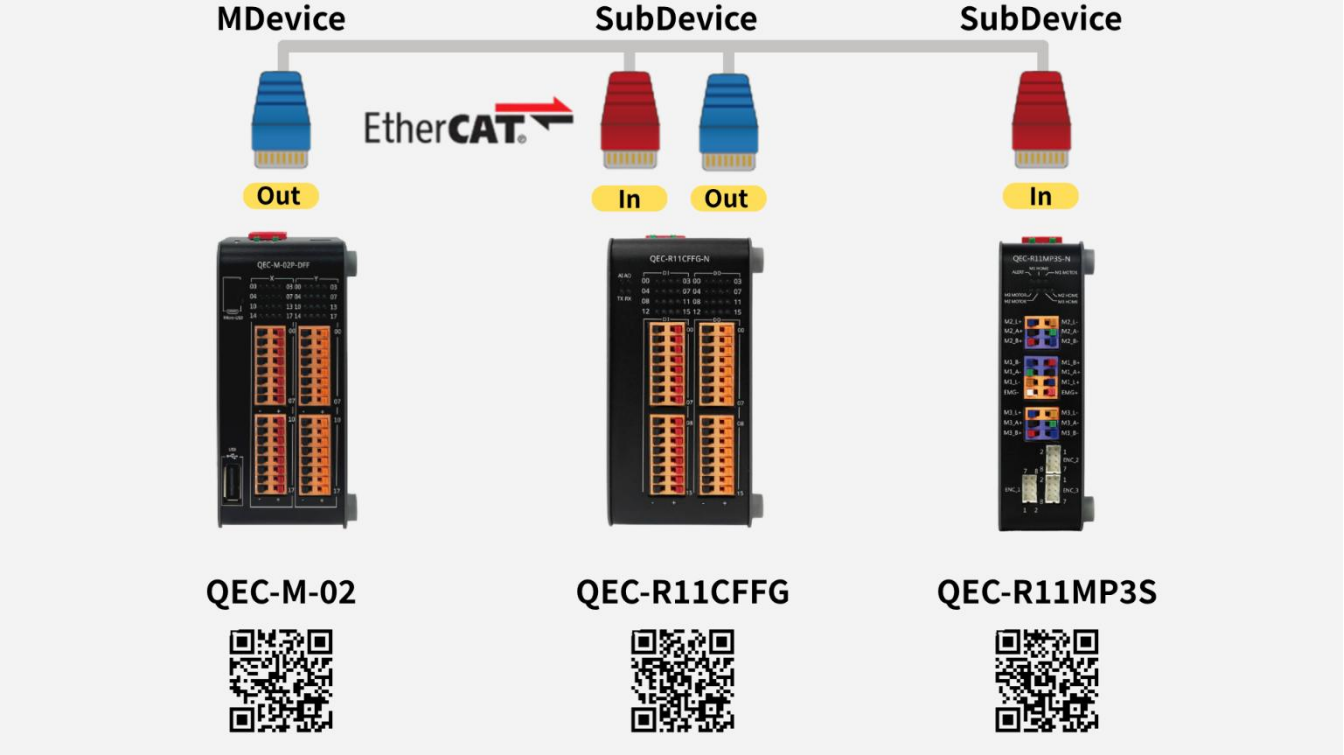
- Set up Modbus TCP communication.
- Map Modbus addresses to CIMON Tags.
- Build basic SCADA screens using Text, Button, Tag Value, Circle, and Image components.
- Verify I/O control, status monitoring, and motor control in CIMON X.

After completing this guide, users can apply the same workflow to other QEC-based monitoring and control applications.

1. Connection and wiring hardware

The following devices are used in this guide:

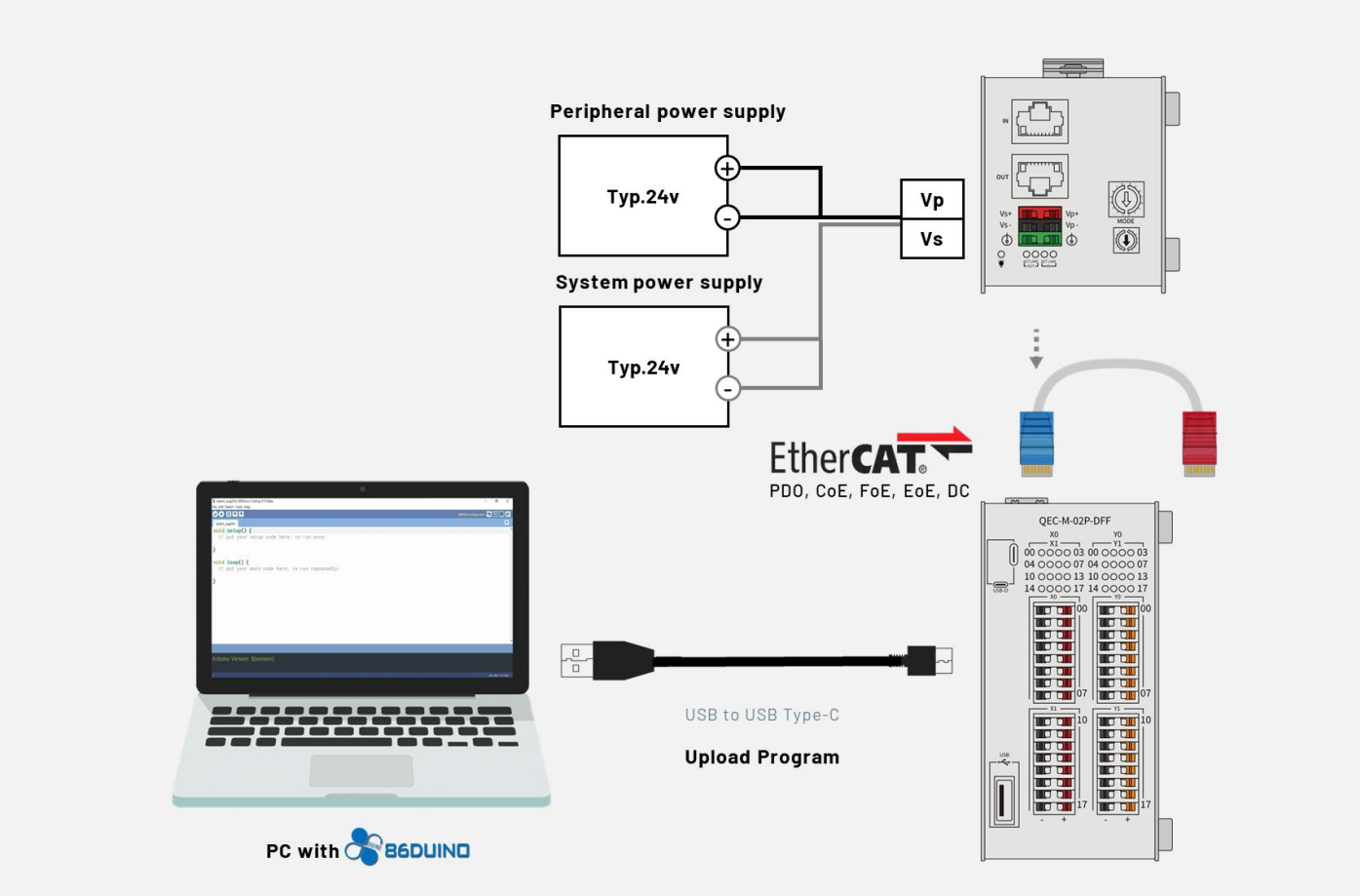
1. QEC-M-02 (EtherCAT MDevice)
2. QEC-R11CFFG series (EtherCAT Compound I/O module, with DIO + AIO + Serial COM port)
3. QEC-R11MP3S series (EtherCAT 3-axis stepper motor controller)
4. A 4-wire two-phase bipolar 42 stepper motor (refer to 86STEP | 86Duino)
5. 24VDC power supply & EU-type terminal cable & LAN cable



1.1 QEC-M-02

QEC EtherCAT MDevice.

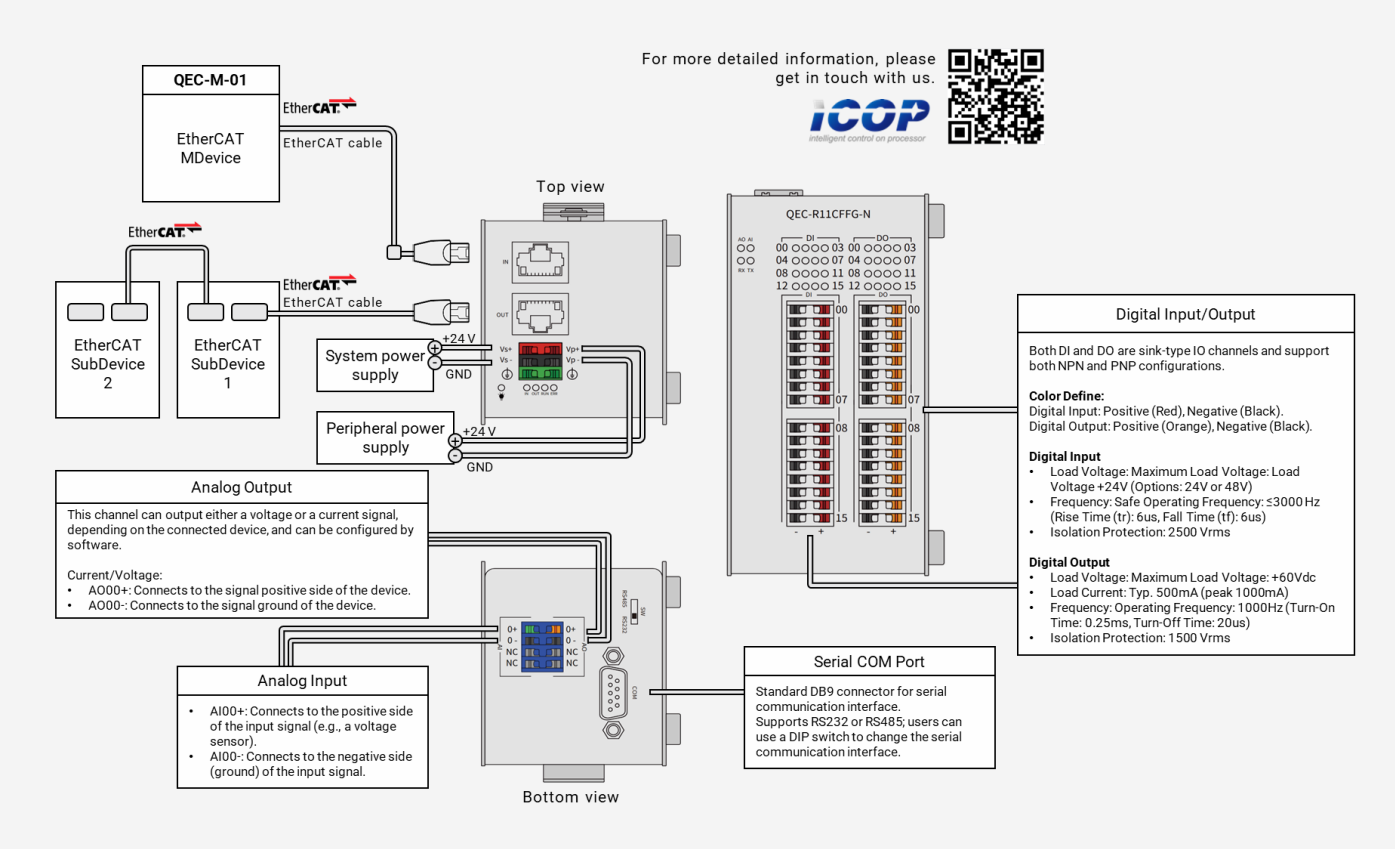
1. Power Supply: Connect to Vs+/Vs- and Vp+/Vp- power supplies via EU terminals for 24V power.
2. EtherCAT Connection: Using the EtherCAT Out port (On the top side) connected to the EtherCAT In port of EtherCAT slave via RJ45 cable.



1.2 QEC-R11CFFG

The **QEC-R11CFFG** is an EtherCAT slave that combines isolated 32-ch Digital I/O (DI16/DO16), analog I/O slots (voltage/current per configuration), and a serial COM port (RS-232/RS-485 selectable).

The diagram shows a typical wiring example with a QEC MDevice and an EtherCAT network.



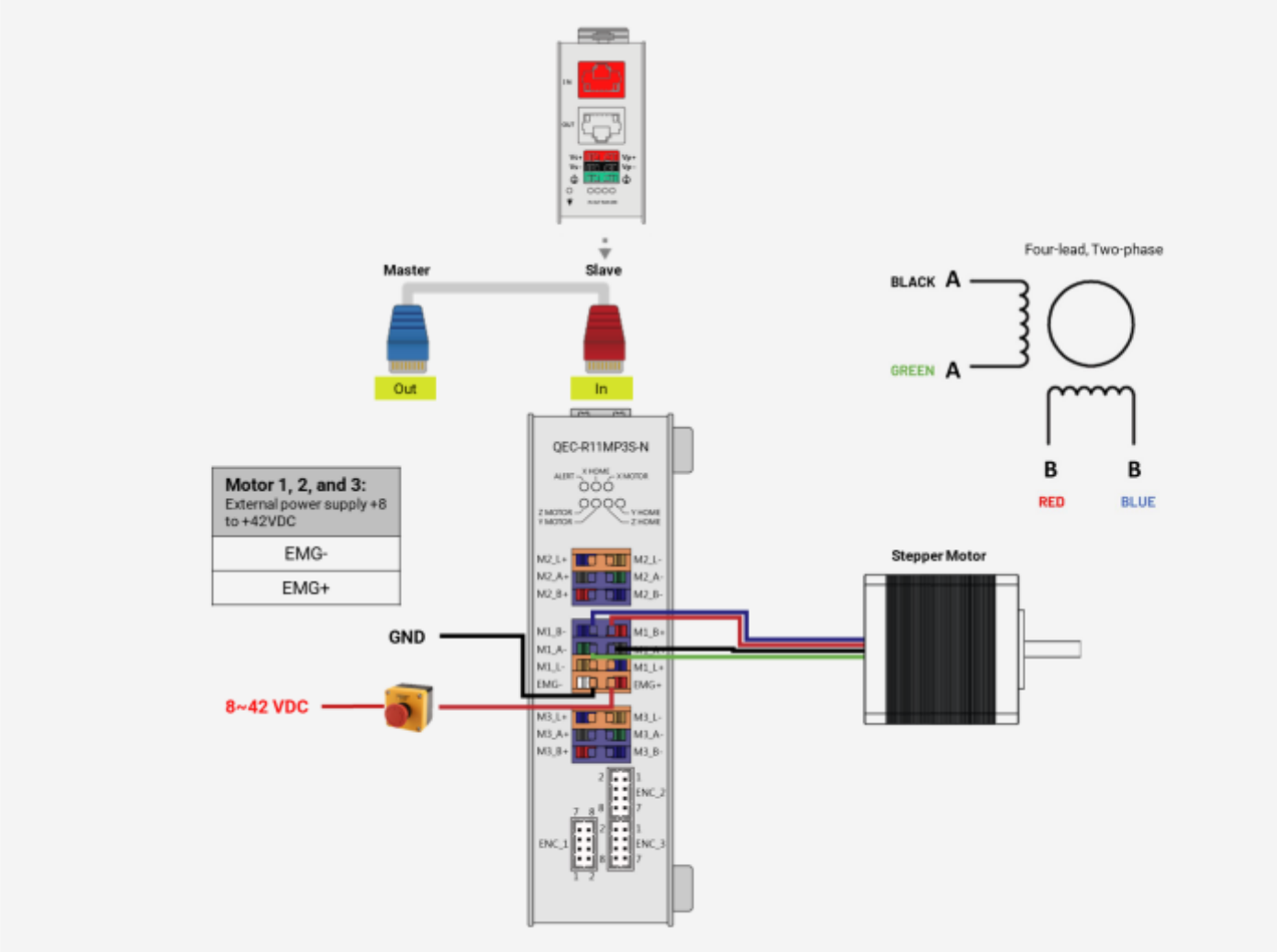
Connections are grouped by function:

- EtherCAT: MDevice → IN; OUT for daisy-chain.
- Power & Grounding:
  - VS+/VS−: system power +24 V/GND
  - VP+/VP−: field I/O power +24 V/GND
- Digital I/O:
  - DI 16 (sink type): supports NPN/PNP; safe op ≤ 3 kHz, tr/tf 6 µs; input +24 V (options 24/48 V).
  - DO 16: up to +60 Vdc, typ. 500 mA (peak 1 A); 1 kHz (Ton 0.25 ms / Toff 20 µs). Add flyback/snubber for inductive loads.
  - Terminal groups: DI00–07 / DI08–15, DO00–07 / DO08–15, each with + / −
  - Isolation: DI 2500 Vrms; DO 1500 Vrms.
- Analog I/O:
  - AO voltage/current (software-selectable): AO00+ to device positive, AO00− to return.
  - AI: AI00+ to sensor positive, AI00− to sensor return

- Serial COM: DB9; DIP switch selects RS-232/RS-485
  For RS-485, use termination/bias and a shared reference.
- Indicators: PWR / RUN / ERR / L/A (see later section)
- Color legend: DI + Red / − Black; DO + Orange / − Black.

1.3 QEC-R11MP3S

The QEC-R11MP3S with PoE function.



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

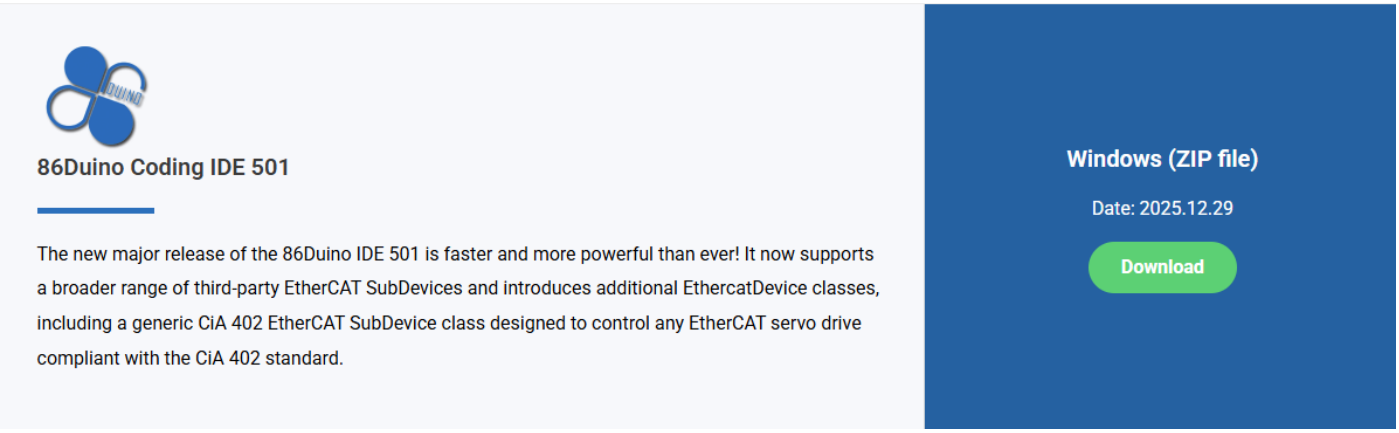
Note: The Emergency Stop is a safety mechanism for quickly shutting down machinery in emergencies via a hardware switch.

2. Connect the 4-wire bidirectional 42-stepper motor to the M1 axis (middle) of the QEC-R11MP3S 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, users can follow the color for wiring reference.

2. Software/Development Environment

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



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

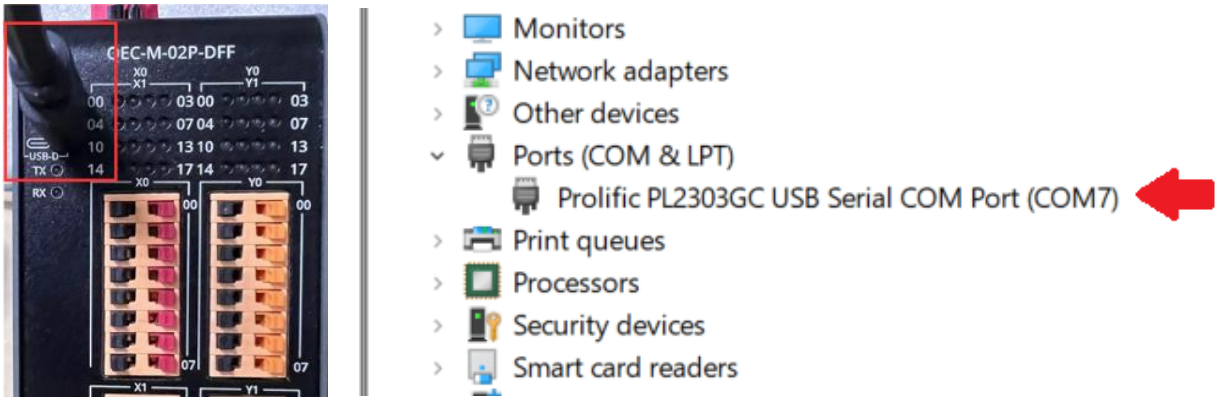
*__Note__: If Windows displays a warning, click Details once and then click the Continue Run button once.

86Duino Coding IDE 501+ looks like below.

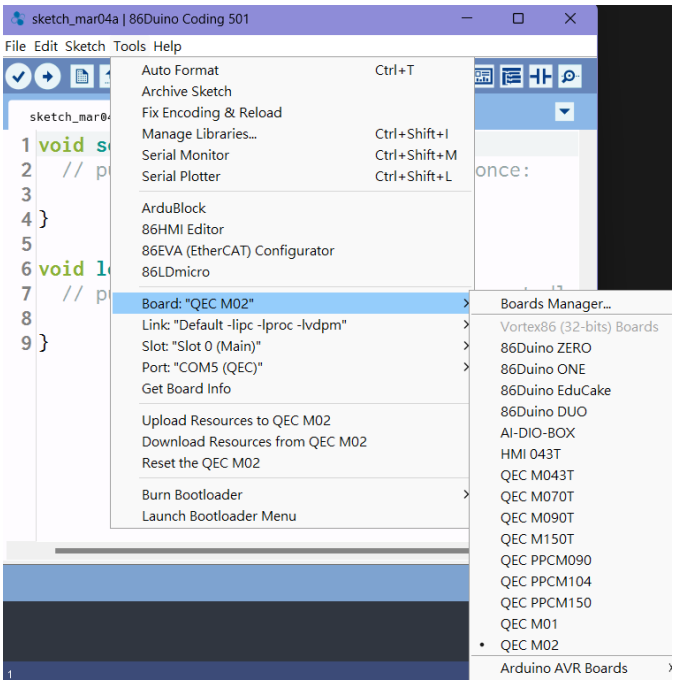
3. Connect to PC and set up the environment

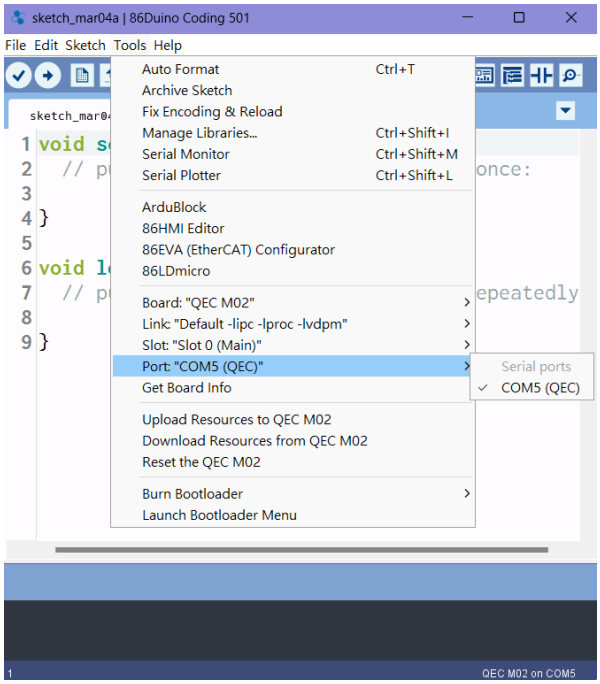
Follow the steps below to set up the environment:

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)).





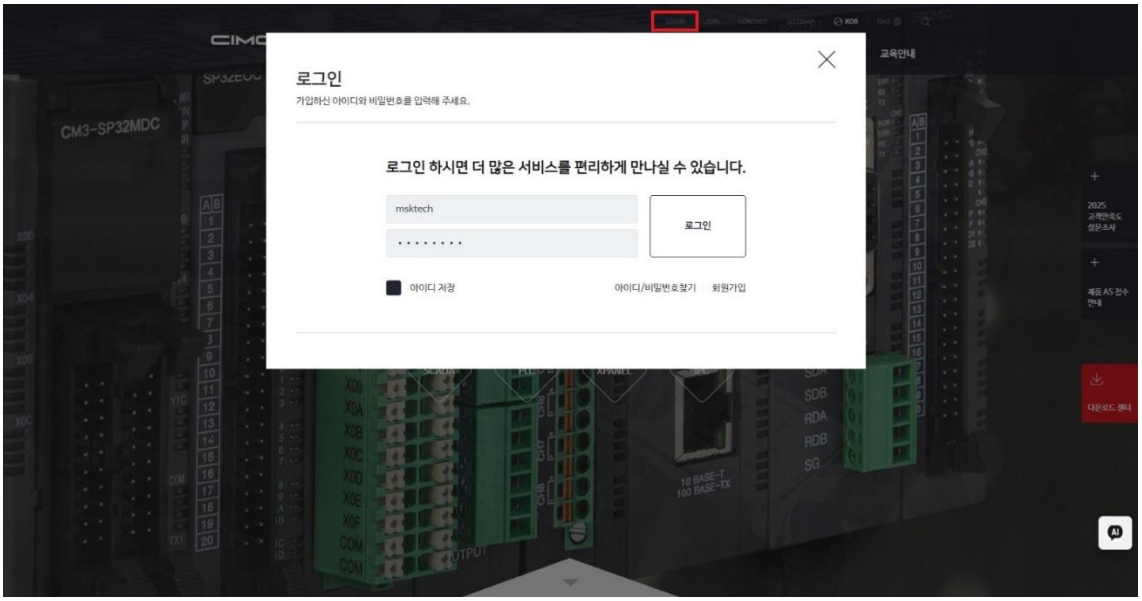
4. Install CIMON SCADA

Step 1: Download CIMON SCADA

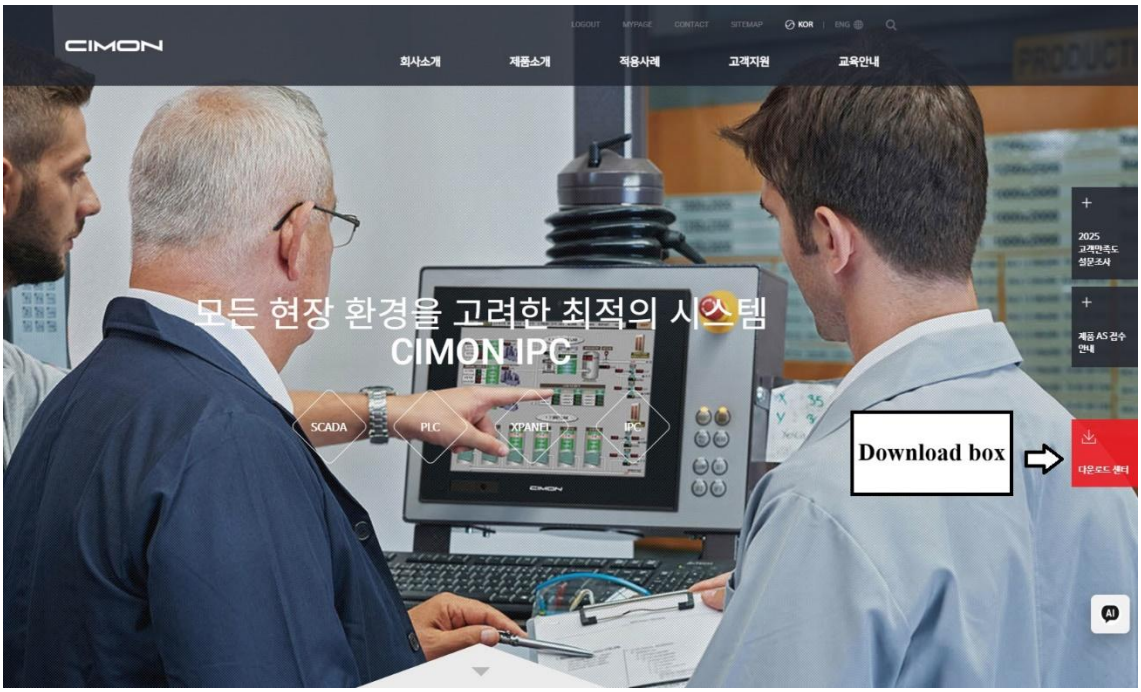
Download CIMON SCADA from https://cimon.co.kr/

Step 2: Log in to the CIMON website

Enter your account information to log in:



Step 3: Go to the Download section



Step 4: Search for the download

In the first dropdown select **SCADA**, in the second select 소프트웨어 **(Software)**, then click the red search button.

Step 5: Select the SCADA version

Select item 57: **CIMON-SCADA Software V3.91.6**

Step 6: Download the installer

Scroll to the bottom of the page to find the download link.

Step 7: Run the installer

Unzip the downloaded archive, then run **setup** to install. After installation, you will see **CimonD** and **CimonX**:

- **CimonD**: The CIMON SCADA development environment — used to create projects, configure communication devices and Tags, and design monitoring screens.
- **CimonX**: The CIMON SCADA runtime environment — used to run completed projects, perform real-time communication, display monitoring screens, and operate controls.

5. Example Code

QEC-M-02 supports remote monitoring and control through the Modbus TCP communication protocol. By connecting CIMON SCADA to QEC-M-02, users can perform functions such as digital I/O control, remote EtherCAT I/O control, external Modbus RTU I/O control, and single-axis motor operation control.

The development is performed using the 86Duino IDE. In each example, QEC-M-02 runs as a Modbus TCP Slave, while CIMON SCADA acts as a Modbus TCP Master / Client to read and write Modbus Tags through the Ethernet network.

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

- Example 1: QEC-M-02 Local I/O Control
- Example 2: QEC-R11CFFG Remote EtherCAT I/O Control
- Example 3: External Modbus RTU I/O Board Control
- Example 4: QEC-R11MP3S Single-Axis Motor Control

Hardware Configuration

- EtherCAT Master Device: QEC-M-02
- EtherCAT Slave Device: QEC-R11MP3S & QEC-R11CFFG
- Communication Protocol: Modbus TCP
- Ethernet MAC/IP: Assigned to QEC-M-02 for network access

Software Configuration

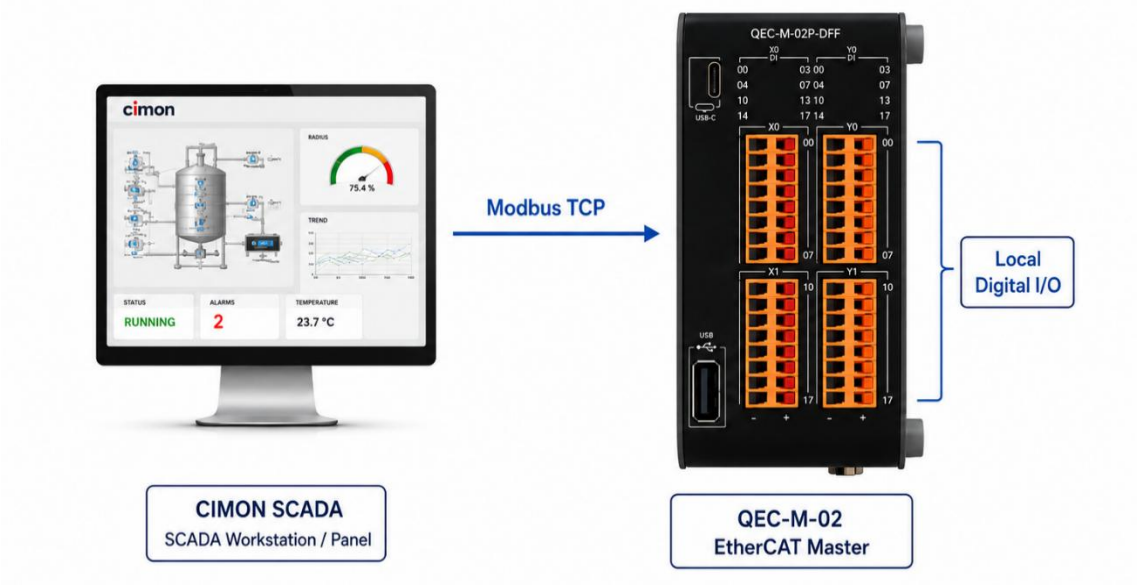
- Development IDE: 86Duino IDE 501+
- Network Library: Ethernet Library
- Visualization Platform: CIMON SCADA

Before running the following examples, please make sure that the QEC-M-02 firmware has been uploaded successfully, the PC and QEC-M-02 are on the same network segment, and CIMON SCADA has been installed as described in the previous sections.

Example 1 – QEC-M-02 local I/O control

This example uses Modbus TCP to allow CIMON SCADA to directly control the digital outputs (DO) and read the digital inputs (DI) of the QEC-M-02 module itself.

A. Hardware Wiring



B. Modbus Address Map

CIMON Address	Internal Addr	Function	Description
00001	0	DO0 control	1 = ON, 0 = OFF
00002	1	DO1 control	1 = ON, 0 = OFF
00003	2	DO2 control	1 = ON, 0 = OFF
00004	3	DO3 control	1 = ON, 0 = OFF
CIMON Address	**Internal Addr**	**Function**	**Description**
10001	0	DI0 status	1 = ON, 0 = OFF
10002	1	DI1 status	1 = ON, 0 = OFF
10003	2	DI2 status	1 = ON, 0 = OFF
10004	3	DI3 status	1 = ON, 0 = OFF

C. Function Description

Function	Description
`setup()`	Initializes the QEC-M-02 local I/O pins, Ethernet connection, and Modbus TCP Slave.
`loop()`	Continuously processes Modbus TCP requests and updates I/O status.
`updateDOFromCimon()`	Reads Coil commands from CIMON SCADA and controls QEC-M-02 DO0~DO3.
`updateDIToCimon()`	Reads QEC-M-02 DI0~DI3 status and sends the values back to CIMON SCADA.

The example code is as follows:

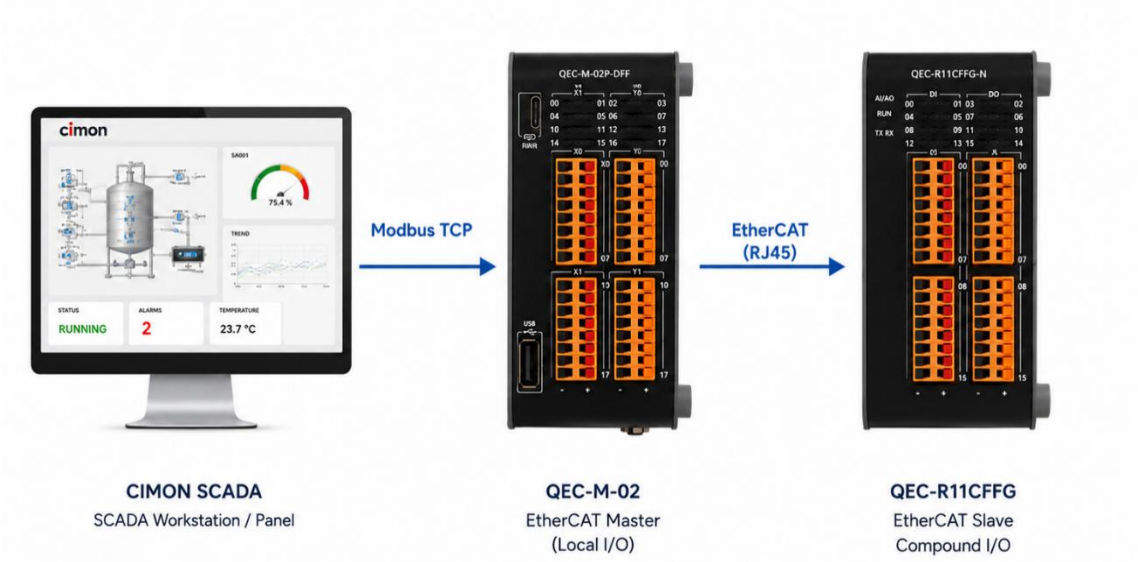
```
#include <Ethernet.h>
#include "Modbus.h"

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x02, 0x50 };
IPAddress localIp(192, 168, 2, 250);  // Modify according to your subnet

#define TCP_SLAVE_ID 1

ModbusSlave tcpBus;
ModbusSlaveNode tcpNode;

#define IO_COUNT 4

int doPins[IO_COUNT] = {0, 1, 2, 3};
int diPins[IO_COUNT] = {0, 1, 2, 3};

bool coilState[IO_COUNT] = {false, false, false, false};
bool inputState[IO_COUNT] = {false, false, false, false};

void setup()
{
  Serial.begin(115200);

  for (int i = 0; i < IO_COUNT; i++) {
    pinMode(doPins[i], OUTPUT);
    digitalWrite(doPins[i], LOW);

    pinMode(diPins[i], INPUT);
  }

  Ethernet.begin(mac, localIp);
  delay(1000);

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

  tcpBus.begin(MODBUS_TCP);
  tcpNode.begin(TCP_SLAVE_ID, tcpBus);

  Serial.println("Modbus TCP Slave Ready");
  Serial.println("CIMON can connect to QEC by Modbus TCP");
```

```
}

void loop()
{
  tcpNode.poll();
  updateDOFromCimon();
  updateDIToCimon();
}

// Update DO0~DO3 from CIMON coil commands
void updateDOFromCimon()
{
  for (int i = 0; i < IO_COUNT; i++) {
    bool cmd = tcpNode.readCoil(i);

    coilState[i] = cmd;

    if (cmd) {
      digitalWrite(doPins[i], HIGH);
    } else {
      digitalWrite(doPins[i], LOW);
    }
  }
}

// Send DI0~DI3 status back to CIMON discrete inputs
void updateDIToCimon()
{
  for (int i = 0; i < IO_COUNT; i++) {
    bool value = digitalRead(diPins[i]);

    inputState[i] = value;

    tcpNode.writeDiscreteInput(i, value);
  }
}
```

Example 2 – QEC-R11CFFG remote EtherCAT I/O control

This example connects to QEC-M-02 via Modbus TCP; QEC-M-02 then drives QEC-R11CFFG DO5~DO7 over EtherCAT.

A. Hardware Wiring



B. Modbus Address Map

CIMON Address	Internal Addr	Function	Description
00006	5	DO5 control	1 = ON, 0 = OFF
00007	6	DO6 control	1 = ON, 0 = OFF
00008	7	DO7 control	1 = ON, 0 = OFF
CIMON Address	**Internal Addr**	**Function**	**Description**
10006	5	DO5 output feedback	1 = ON, 0 = OFF
10007	6	DO6 output feedback	1 = ON, 0 = OFF
10008	7	DO7 output feedback	1 = ON, 0 = OFF

C. Function Description

Function	Description
`setup()`	Initializes EtherCAT, connects QEC-R11CFFG, and starts the Modbus TCP Slave.
`loop()`	Continuously processes Modbus TCP requests and updates remote I/O status.
`updateDOFromCimon()`	Reads Coil commands from CIMON SCADA and controls QEC-R11CFFG DO5~DO7.
`updateDOStatusToCimon()`	Sends the current DO5~DO7 output status back to CIMON SCADA as feedback.

The example code is as follows:

```
#include <Ethernet.h>
#include "Ethercat.h"
#include "Modbus.h"

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x02, 0x50 };
IPAddress localIp(192, 168, 2, 250);  // Modify according to your subnet

#define TCP_SLAVE_ID 1

EthercatMaster master;
EthercatDevice_QECR11CFFG cffg;

ModbusSlave tcpBus;
ModbusSlaveNode tcpNode;

#define IO_COUNT 3  // DO5~DO7

int doPins[IO_COUNT] = {5, 6, 7};

bool coilState[IO_COUNT] = {false, false, false};
bool inputState[IO_COUNT] = {false, false, false};

void setup()
{
  Serial.begin(115200);

  Serial.print("EtherCAT Begin: ");
  Serial.println(master.begin());

  Serial.print("Slave Attach: ");
  Serial.println(cffg.attach(0, master));

  Serial.print("EtherCAT Start: ");
  Serial.println(master.start(1000000, ECAT_SYNC));

  // Set DO5~DO7 to OFF by default
  for (int i = 0; i < IO_COUNT; i++) {
    cffg.digitalWrite(doPins[i], LOW);
  }

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

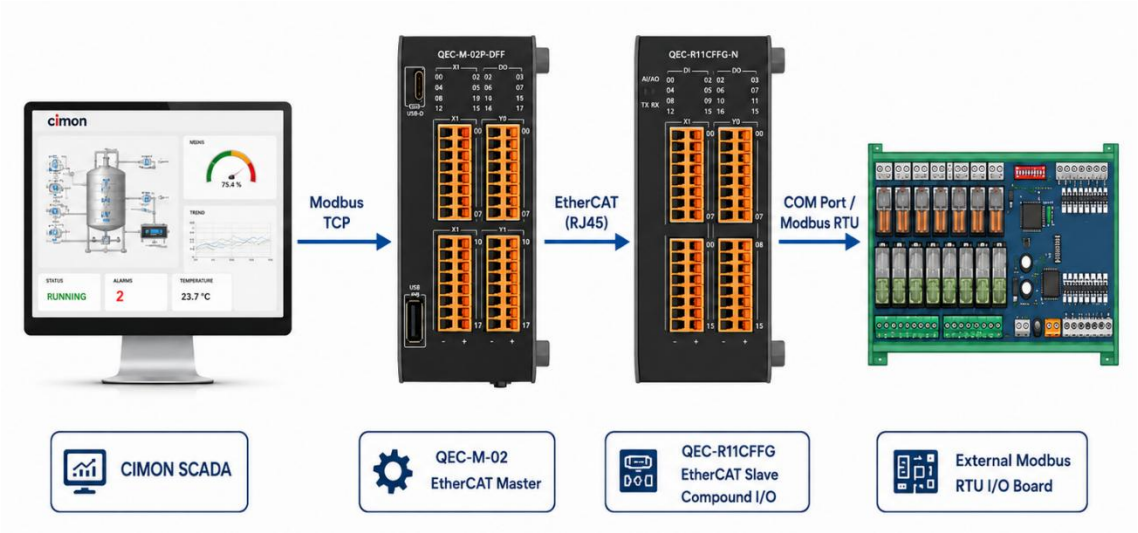
```
  delay(1000);

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

  tcpBus.begin(MODBUS_TCP);
  tcpNode.begin(TCP_SLAVE_ID, tcpBus);

  Serial.println("Modbus TCP Slave Ready");
  Serial.println("CIMON can connect to QEC by Modbus TCP");
}

void loop()
{
  tcpNode.poll();
  updateDOFromCimon();
  updateDOStatusToCimon();
}

// Update DO5~DO7 from CIMON coil commands
void updateDOFromCimon()
{
  for (int i = 0; i < IO_COUNT; i++) {
    bool cmd = tcpNode.readCoil(doPins[i]);

    coilState[i] = cmd;

    if (cmd) {
      cffg.digitalWrite(doPins[i], HIGH);
    } else {
      cffg.digitalWrite(doPins[i], LOW);
    }
  }
}

// Send DO5~DO7 status back to CIMON discrete inputs
void updateDOStatusToCimon()
{
  static uint32_t lastPrintTime = 0;

  for (int i = 0; i < IO_COUNT; i++) {
    bool value = coilState[i];
```

```
    inputState[i] = value;

    tcpNode.writeDiscreteInput(doPins[i], value ? MODBUS_COIL_ON :
MODBUS_COIL_OFF);
  }

  if (millis() - lastPrintTime >= 1000) {
    lastPrintTime = millis();

    Serial.print("DO Feedback to SCADA: ");

    for (int i = 0; i < IO_COUNT; i++) {
      Serial.print("1000");
      Serial.print(doPins[i] + 1);
      Serial.print("(DO");
      Serial.print(doPins[i]);
      Serial.print(")=");
      Serial.print(inputState[i]);
      Serial.print(" ");
    }

    Serial.println();
  }
}
```

Example 3 – External Modbus RTU I/O Board

This example demonstrates how CIMON SCADA uses QEC-M-02 and QEC-R11CFFG to control an external Modbus RTU I/O Board through Modbus TCP, EtherCAT, and RS-485 communication.

A. Hardware Wiring



B. Modbus Address Map

CIMON Address	Internal Addr	Function	Description
00011	10	RTU DO0 control	FC05 sent only on state change
00012	11	RTU DO1 control	FC05 sent only on state change
00013	12	RTU DO2 control	FC05 sent only on state change
00014	13	RTU DO3 control	FC05 sent only on state change
CIMON Address	**Internal Addr**	**Function**	**Description**
10011	10	RTU DI0 status	Polled via FC02 every 200ms
10012	11	RTU DI1 status	Polled via FC02 every 200ms
10013	12	RTU DI2 status	Polled via FC02 every 200ms
10014	13	RTU DI3 status	Polled via FC02 every 200ms

C. Function Description

Function	Description
`setup()`	Initializes EtherCAT, QEC-R11CFFG COM1, Ethernet, and the Modbus TCP Slave.
`loop()`	Processes CIMON SCADA commands, forwards DO control to the RTU I/O Board, and updates DI status.
`writeCoil()`	Sends a Modbus RTU command to control one DO point on the external I/O Board.
`readDI()`	Reads DI status from the external I/O Board through Modbus RTU.
`crc16()`	Calculates the CRC16 checksum required for Modbus RTU communication.
`clearRx()`	Clears the COM1 receive buffer before sending or receiving RTU data.

The example code is as follows:

```
#include <Ethernet.h>
#include "Ethercat.h"
#include "Modbus.h"

// Modbus TCP slave for CIMON connection
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x02, 0x50 };
IPAddress localIp(192, 168, 2, 250);  // Modify according to your subnet

ModbusSlave tcpBus;
ModbusSlaveNode tcpNode;

// EtherCAT master and R11CFFG COM1 for Modbus RTU
EthercatMaster master;
EthercatDevice_QECR11CFFG cffg;

#define TCP_SLAVE_ID 1
#define IO_BOARD_ID 16   // RTU IO board slave ID
#define IO_COUNT    4    // Number of DI/DO points
#define CFFG_COM1   0    // R11CFFG COM1

bool lastDO[IO_COUNT] = {0};   // Store the previous DO status
uint32_t lastDI_t = 0;

// Calculate Modbus RTU CRC16
uint16_t crc16(uint8_t *b, int n) {
  uint16_t c = 0xFFFF;
  for (int i = 0; i < n; i++) {
    c ^= b[i];
    for (int j = 0; j < 8; j++)
      c = (c & 1) ? (c >> 1) ^ 0xA001 : c >> 1;
  }
  return c;
}

// Clear the COM1 receive buffer
void clearRx() {
  uint8_t d[32];
  while (cffg.uartReceive(CFFG_COM1, d, 32) > 0) { cffg.update(); delay(2); }
}

// Write one RTU coil
```

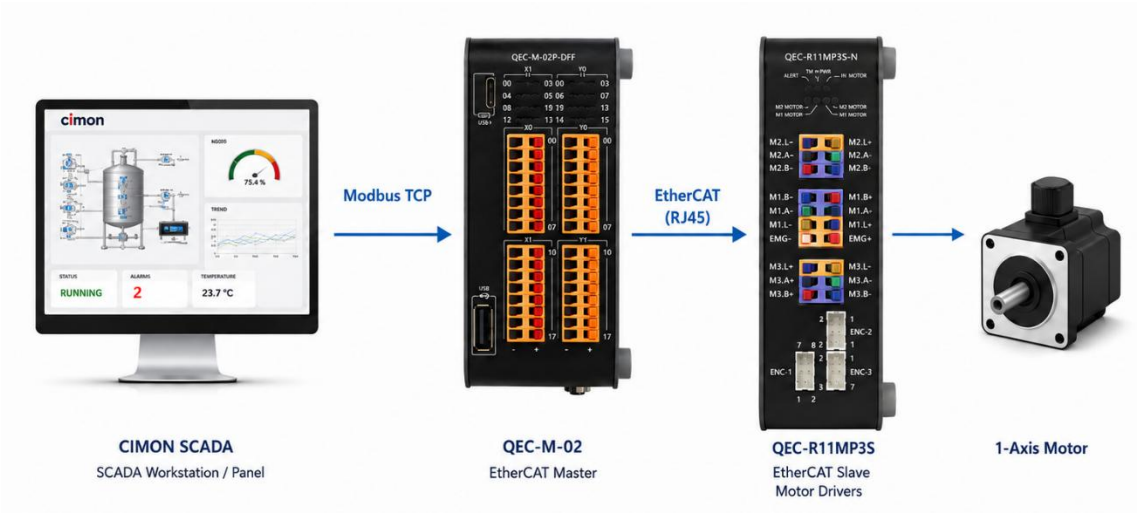
```
void writeCoil(uint8_t id, uint16_t addr, bool s) {
  uint8_t f[8] = { id, 0x05, highByte(addr), lowByte(addr), s ? 0xFF : 0x00,
0x00 };
  uint16_t c = crc16(f, 6);
  f[6] = lowByte(c); f[7] = highByte(c);
  clearRx();
  cffg.uartSend(CFFG_COM1, f, 8);
  delay(30);
}

// Read RTU discrete inputs with a 100 ms timeout
bool readDI(uint8_t id, uint16_t addr, uint16_t cnt, uint8_t *data) {
  uint8_t tx[8] = { id, 0x02, highByte(addr), lowByte(addr), highByte(cnt),
lowByte(cnt) };
  uint16_t c = crc16(tx, 6);
  tx[6] = lowByte(c); tx[7] = highByte(c);

  uint8_t rx[16]; int len = 0;
  clearRx();
  cffg.uartSend(CFFG_COM1, tx, 8);

  uint32_t t = millis();
  while (millis() - t < 100) {
    cffg.update();
    int n = cffg.uartReceive(CFFG_COM1, rx + len, 16 - len);
    if (n > 0) len += n;
    if (len >= 6) break;
    delay(2);
  }

  if (len < 6 || rx[0] != id || rx[1] != 0x02) return false;
  if (crc16(rx, len - 2) != (rx[len-2] | (rx[len-1] << 8))) return false;
  *data = rx[3];
  return true;
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  master.begin();
  cffg.attach(0, master);
```

```
  master.start(1000000, ECAT_SYNC);
  cffg.uartSetBaud(CFFG_COM1, 115200);
  cffg.uartSetFormat(CFFG_COM1, SERIAL_8N1);

  Ethernet.begin(mac, localIp);
  delay(1000);
  tcpBus.begin(MODBUS_TCP);
  tcpNode.begin(TCP_SLAVE_ID, tcpBus);

  Serial.println("Ready");
}

void loop() {
  tcpNode.poll();
  cffg.update();

  // CIMON coils 00011~00014 control RTU DO0~DO3
  for (int i = 0; i < IO_COUNT; i++) {
    bool cmd = tcpNode.readCoil(10 + i) == MODBUS_COIL_ON;
    if (cmd != lastDO[i]) {
      lastDO[i] = cmd;
      writeCoil(IO_BOARD_ID, i, cmd);
      Serial.print("DO"); Serial.print(i);
      Serial.println(cmd ? " ON" : " OFF");
    }
  }

  // Read RTU DI0~DI3 every 200 ms and update CIMON inputs
  if (millis() - lastDI_t >= 200) {
    lastDI_t = millis();
    uint8_t di = 0;
    if (readDI(IO_BOARD_ID, 0, IO_COUNT, &di)) {
      for (int i = 0; i < IO_COUNT; i++)
        tcpNode.writeDiscreteInput(10 + i,
          bitRead(di, i) ? MODBUS_COIL_ON : MODBUS_COIL_OFF);
      Serial.print("DI="); Serial.println(di, BIN);
    }
  }

  delay(20);
}
```

Example 4 – QEC-R11MP3S Single-Axis Motor Control

This example connects to QEC-M-02 via Modbus TCP; QEC-M-02 then drives a QEC-R11MP3S axis via EtherCAT in Profile Position (PP) mode for jog motion control.

A. Hardware Wiring



B. Modbus Address Map

CIMON Address	Internal Addr	Function	Description
00016	15	Jog+ (positive direction)	Moves the motor forward by the JogStep value.
00017	16	Jog- (negative direction)	Moves the motor backward by the JogStep value.
00018	17	Stop	Stops the motor at the current position while keeping the servo enabled.
00019	18	Enable (Servo ON)	Enables the motor servo and prepares the motor for operation.
00020	19	Disable (Servo OFF)	Disables the motor servo and stops motor operation.
CIMON Address	**Internal Addr**	**Function**	**Description**
40001	0	JogStep	Sets the movement distance for each jog command
40002	1	Position Low	Displays the lower 16 bits of the current motor position.
40003	2	Position High	Displays the upper 16 bits of the current motor position.
40004	3	MotorState	Displays the current motor state.

40005	4	LastResult	Displays the result code of the last motor command.
40006	5	EtherCATState	Displays the current EtherCAT communication status.

C. Function Description

Function	Description
`setup()`	Initializes EtherCAT, QEC-R11MP3S, Ethernet, and the Modbus TCP Slave.
`loop()`	Processes CIMON SCADA commands and updates motor status feedback.
`initEC()`	Starts EtherCAT and connects to the QEC-R11MP3S motor axis.
`initTCP()`	Starts Modbus TCP communication and prepares the default Coil and Register values.
`enableServo()`	Enables the motor servo.
`disableServo()`	Disables the motor servo.
`jogMotor()`	Moves the motor in the positive or negative direction according to the JogStep value.
`stopMotor()`	Stops the motor at the current position while keeping the servo enabled.
`handleSCADA()`	Reads CIMON SCADA commands and calls the corresponding motor control function.
`checkStatus()`	Checks whether the motor has reached the target position.
`updateFB()`	Sends motor position, state, result code, and EtherCAT status back to CIMON SCADA.

The example code is as follows:

```
#include <Ethernet.h>
#include "Modbus.h"
#include "Ethercat.h"

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0x02, 0x50 };
IPAddress localIp(192, 168, 2, 250);  // Modify according to your subnet
#define TCP_SLAVE_ID 1
#define R11MP3S_ALIAS  1
#define AXIS_ID       1
#define PP_VEL    48000      // counts/s
#define PP_ACC    1280000    // counts/s^2
#define PP_DEC    1280000    // counts/s^2

// SCADA coils 00016~00020 map to internal indexes 15~19
enum { CJP=15, CJM, CSTP, CEN, CDIS };
enum { RSTEP=0, RPL, RPH, RST, RRES, RECAT };

ModbusSlave tcpBus;
ModbusSlaveNode tcpNode;
EthercatMaster master;
EthercatDevice_QECR11MP3S mp3s;
EthercatDevice_CiA402 *motor = NULL;

long jogStep = 10000, pos = 0;
uint16_t mState = 9, ecatState = 0;
int16_t lastResult = 0;
bool ecatOK = false, motorOK = false, servoOn = false;
bool latch[20] = {0};

static void writePos(long p) {
  tcpNode.writeHoldingRegister(RPL, p & 0xFFFF);
  tcpNode.writeHoldingRegister(RPH, (p >> 16) & 0xFFFF);
}

static void updateFB() {
  tcpNode.writeHoldingRegister(RSTEP, (uint16_t)jogStep);
  tcpNode.writeHoldingRegister(RST,   mState);
  tcpNode.writeHoldingRegister(RRES,  (uint16_t)lastResult);
  tcpNode.writeHoldingRegister(RECAT, ecatState);
  writePos(pos);
}
```

```
static bool ready() {
  if (!ecatOK || !motorOK || !motor) {
    lastResult = 998; mState = 9; updateFB(); return false;
  }
  return true;
}

static bool needServo() {
  if (!servoOn) { lastResult = 997; mState = 8; updateFB(); return false; }
  return true;
}

// Configure PP motion parameters before each command
static void cfgPP() {
  if (!motorOK || !motor) return;
  motor->setCiA402Mode(1);          // PP Mode
  motor->pp_SetMotionProfileType(0); // trapezoidal
  motor->pp_SetVelocity(PP_VEL);
  motor->pp_SetAcceleration(PP_ACC);
  motor->pp_SetDeceleration(PP_DEC);
}

void initEC() {
  ecatOK = motorOK = servoOn = false;
  motor = NULL; mState = 9; lastResult = 0; ecatState = 0;

  int ret = master.begin();
  if (ret) { lastResult=ret; ecatState=9; return; }
  ecatState = 1;

  ret = mp3s.attach(R11MP3S_ALIAS, master);
  if (ret) { lastResult=ret; ecatState=9; return; }
  ecatState = 2;

  ret = master.start(1000000, ECAT_SYNC);  // 1ms cycle
  if (ret) { lastResult=ret; ecatState=9; return; }
  ecatState = 3; ecatOK = true;

  motor = mp3s.cia402GetServo(AXIS_ID);
  if (!motor) { lastResult=999; mState=9; return; }
  motorOK = true;
```

```
  pos = motor->getPositionActualValue();
  mState = 8;  // Ready, but servo is not enabled
}

void initTCP() {
  Ethernet.begin(mac, localIp);
  tcpBus.begin(MODBUS_TCP);
  tcpNode.begin(TCP_SLAVE_ID, tcpBus);

  for (int i = CJP; i <= CDIS; i++) tcpNode.writeCoil(i, MODBUS_COIL_OFF);
  tcpNode.writeHoldingRegister(RSTEP, (uint16_t)jogStep);
  tcpNode.writeHoldingRegister(RST,   9);
}

void enableServo() {
  if (!ready()) return;
  cfgPP(); delay(100);
  int ret = motor->enable();
  lastResult = ret;
  if (!ret) { servoOn=true; mState=0; pos=motor->getPositionActualValue();
writePos(pos); }
  else      { servoOn=false; mState=9; }
  updateFB();
}

void disableServo() {
  if (!ready()) return;
  int ret = motor->disable();
  lastResult = ret;
  if (!ret) { servoOn=false; mState=8; pos=motor->getPositionActualValue();
writePos(pos); }
  else      { mState=9; }
  updateFB();
}

// Use the actual position as the base for relative jog movement
void jogMotor(long step) {
  if (!ready() || !needServo()) return;
  cfgPP();
  pos = motor->getPositionActualValue();
  int ret = motor->pp_Run(pos + step);
  lastResult = ret;
```

```
  mState = ret ? 9 : 1;  // 1 = Moving
  updateFB();
}

// Stop at the current position and keep the servo enabled
void stopMotor() {
  if (!ready() || !needServo()) return;
  cfgPP();
  pos = motor->getPositionActualValue();
  writePos(pos);
  int ret = motor->pp_Run(pos);
  lastResult = ret;
  mState = ret ? 9 : 0;  // 0 = Idle
  updateFB();
}

// Execute once on the rising edge and clear the coil after execution
static void handleCmd(uint8_t coil, void (*fn)()) {
  bool on = tcpNode.readCoil(coil) == MODBUS_COIL_ON;
  if (on && !latch[coil]) {
    latch[coil] = true;
    fn();
    tcpNode.writeCoil(coil, MODBUS_COIL_OFF);
  }
  if (!on) latch[coil] = false;
}

static void cmdJogP() { jogMotor( jogStep); }
static void cmdJogM() { jogMotor(-jogStep); }

void handleSCADA() {
  uint16_t v = tcpNode.readHoldingRegister(RSTEP);
  if (v > 0) jogStep = (long)v;

  handleCmd(CEN,  enableServo);
  handleCmd(CDIS, disableServo);
  handleCmd(CJP,  cmdJogP);
  handleCmd(CJM,  cmdJogM);
  handleCmd(CSTP, stopMotor);
}

void checkStatus() {
```

```
  if (!ecatOK || !motorOK || !motor || !servoOn || mState != 1) return;
  if (motor->pp_IsTargetReached()) {
    pos = motor->getPositionActualValue();
    writePos(pos);
    mState = 2; lastResult = 0;
    updateFB();
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);
  initEC();
  initTCP();
  Serial.println("SCADA->QEC-M-02->R11MP3S Ready");
}

void loop() {
  tcpNode.poll();
  handleSCADA();
  checkStatus();

  static unsigned long t = 0;
  if (millis() - t >= 200) { t = millis(); updateFB(); }

  delay(5);  // Allow the EtherCAT stack to run
}
```

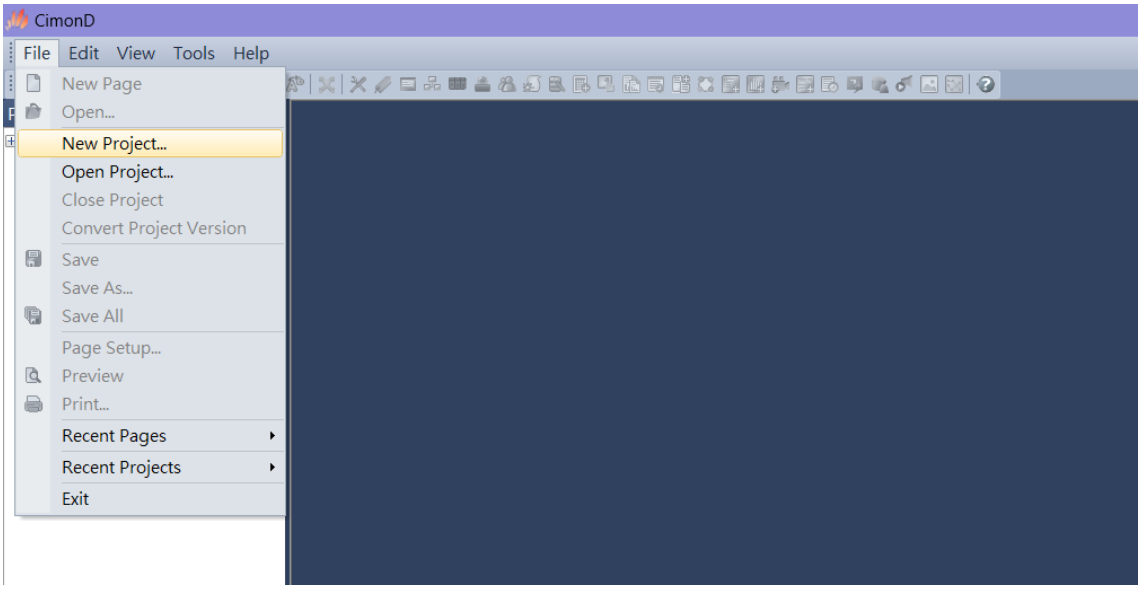
6. CIMON SCADA Setup and Screen Design

This chapter explains how to configure Modbus TCP communication in CIMON SCADA so that CIMON SCADA can connect to QEC-M-02 via Modbus TCP, create Tags, set up screen components, and perform basic read/write tests.

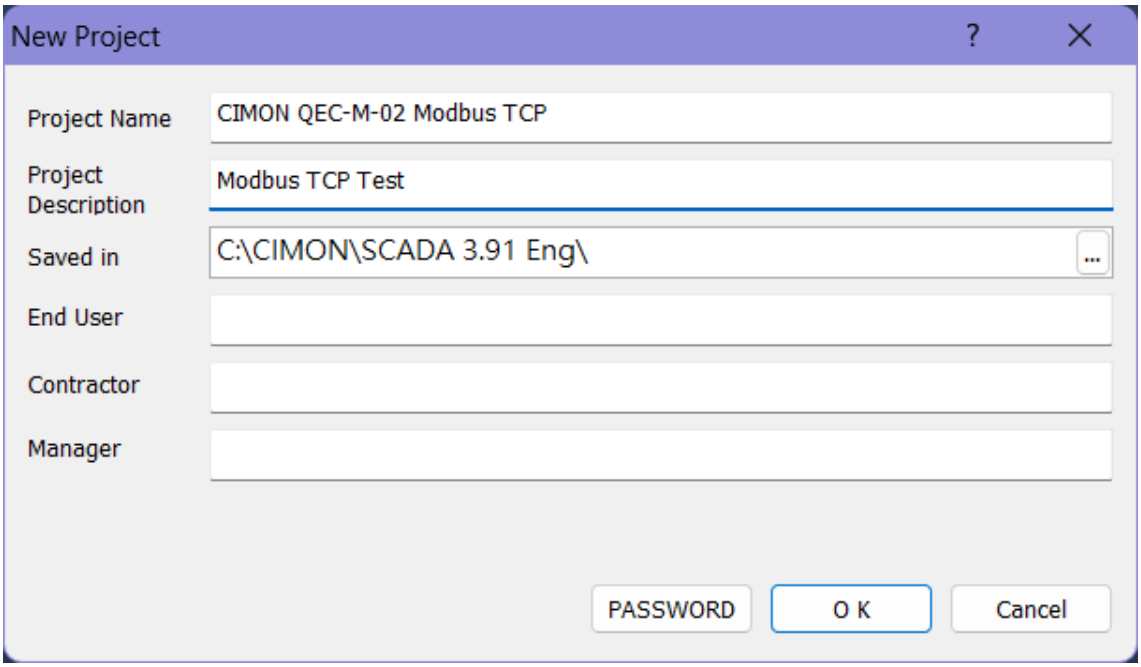
In these examples, QEC-M-02 acts as the Modbus TCP Slave and CIMON SCADA acts as the Modbus TCP Master / Client. After completing the setup, users can control DO outputs and read DI inputs via the CIMON X screen, and extend the project to build motor control and status display screens.

6.1 Create a New Project

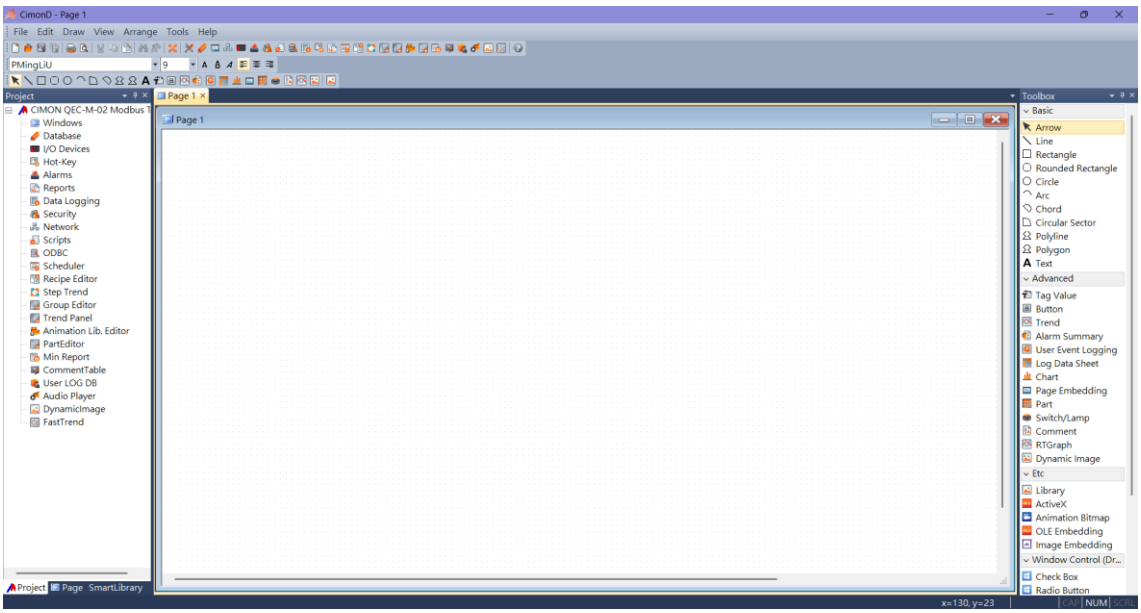
Open CimonD, then click **File** → **New Project** in the upper-left corner to create a new project.



When creating the project, enter a project name and select the storage location. It is recommended to save the project in an easily identifiable location for later use when opening it in CIMON X for testing.

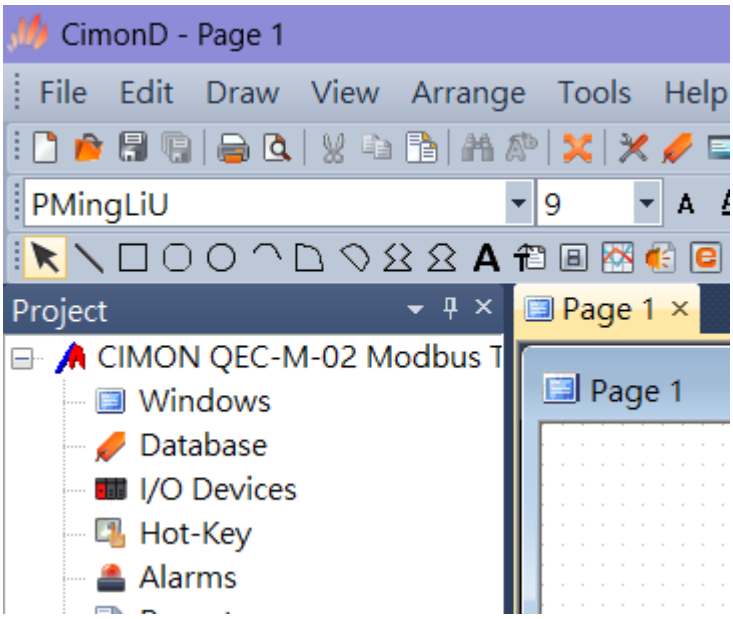


After completing the settings, click **OK** to begin creating the Modbus TCP communication device.



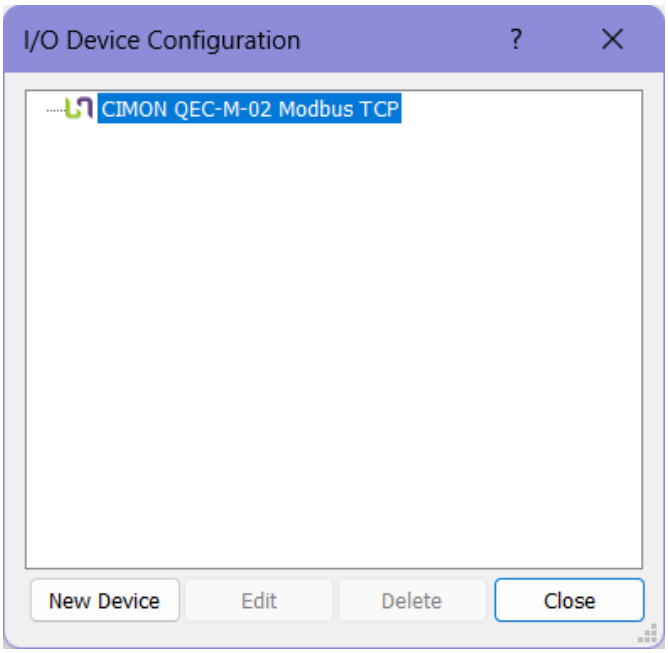
6.2 Create an I/O Device

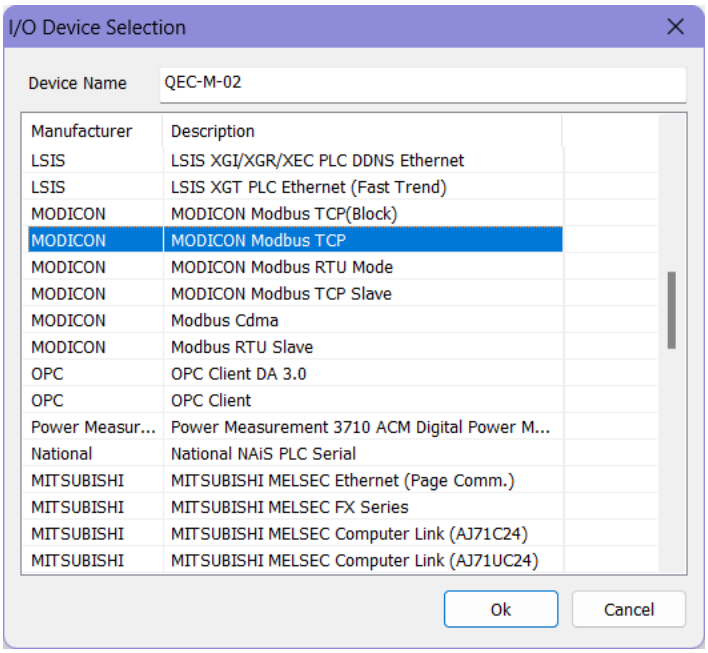
In the left **Project** panel, find **I/O Devices** and double-click it to open the I/O Device Configuration window.



Click **New Device** in the lower-left corner and select the following in the device list:

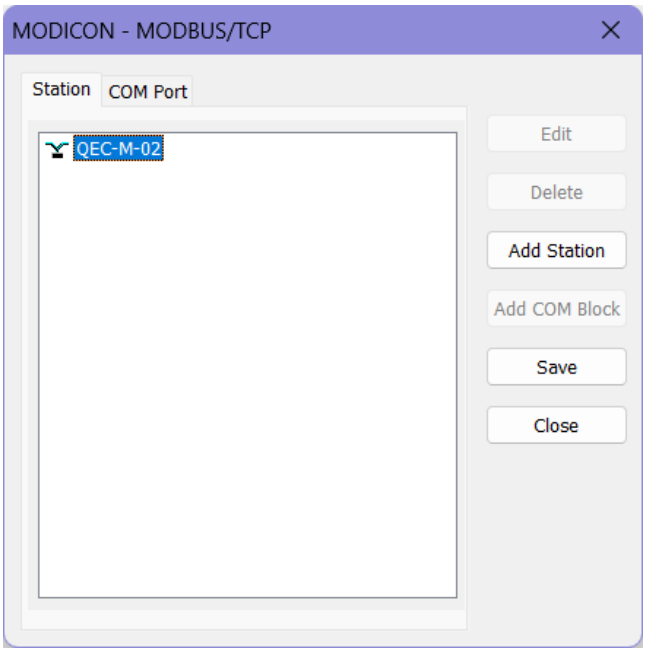
- **Manufacturer**: MODICON
- **Description**: MODICON Modbus TCP
- **Device Name**: Name it according to your project requirements, e.g., QEC-M-02

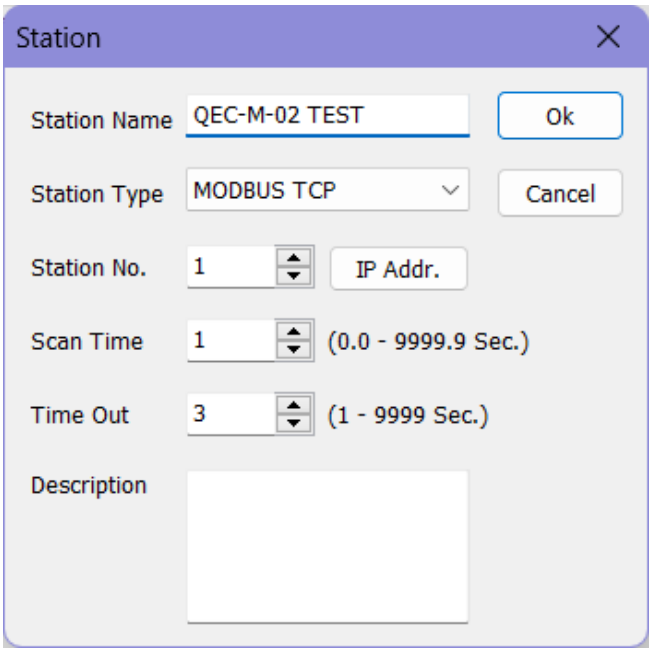




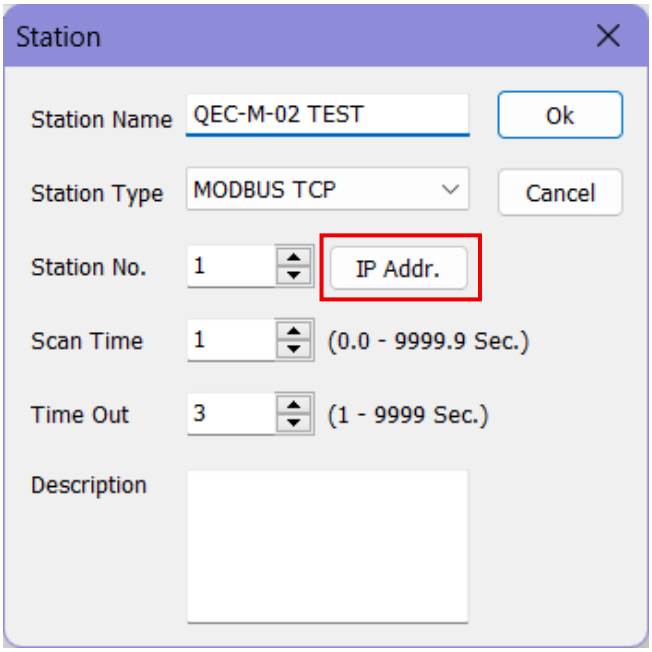
In the MODICON - MODBUS/TCP configuration window, click **Add Station** on the right side.
Configure the settings as follows:

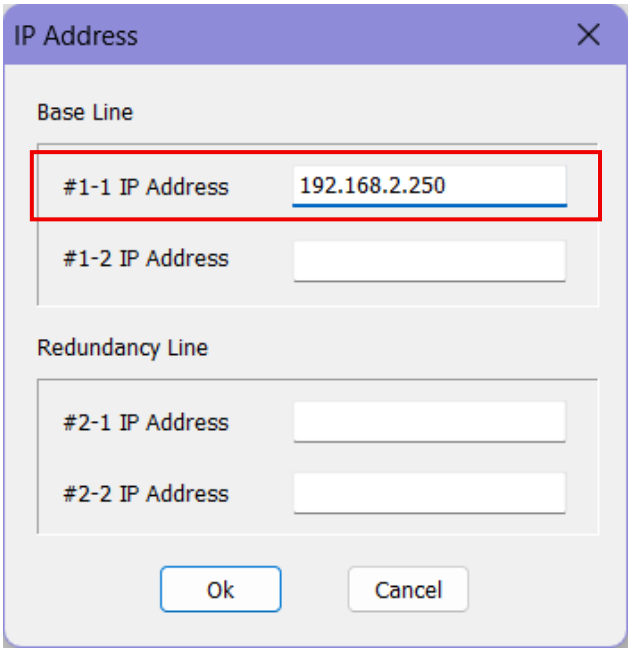
- **Station Name**: Name it as desired, e.g., QEC-M-02 TEST
- **Station Type**: MODBUS TCP
- **Station No.**: 1





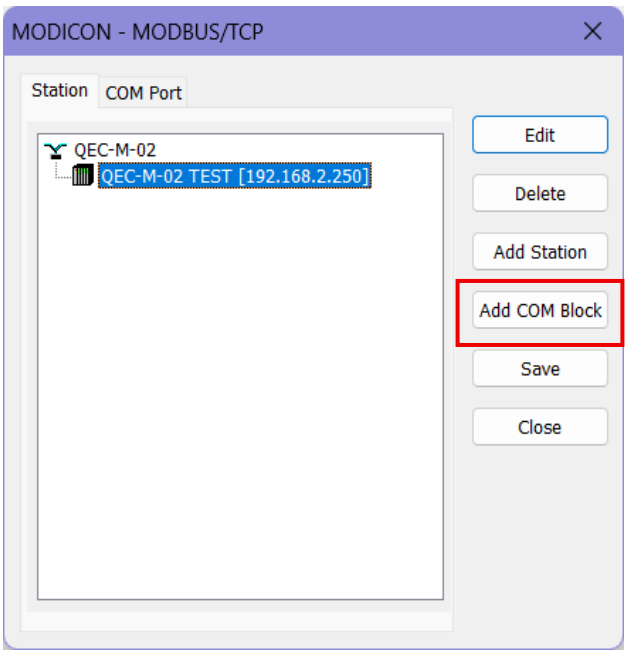
After completing the settings, click **IP Addr.** to configure the IP address.
Enter the IP address of QEC-M-02 in the **#1-1 IP Address** field.
This example uses: **192.168.2.250 ,** After completing the settings, click **OK.**

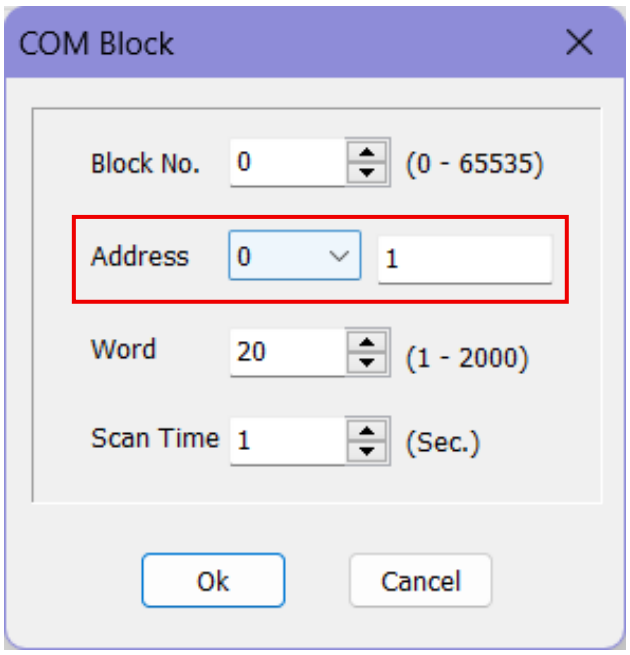


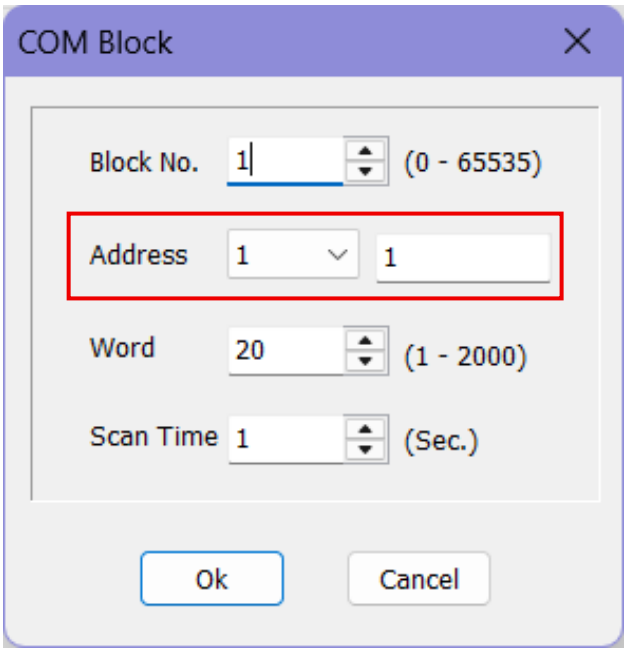


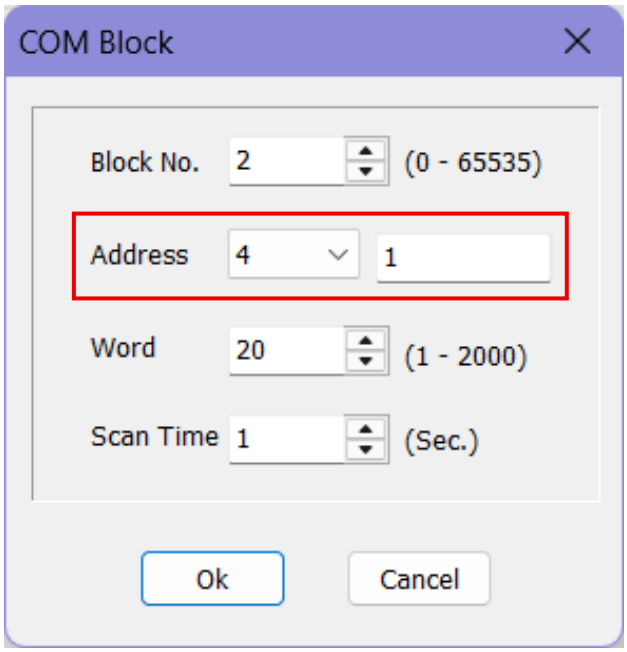
Click **Add COM Block** on the right to create the Modbus data segments that CIMON SCADA needs to read and write. This example creates three COM Blocks corresponding to Coil, Discrete Input, and Holding Register.

Block No.	Modbus Data Type	Address	Purpose
0	Coil	0	DO / command start address
1	Discrete Input	1	DI / status start address
2	Holding Register	4	Register data start address





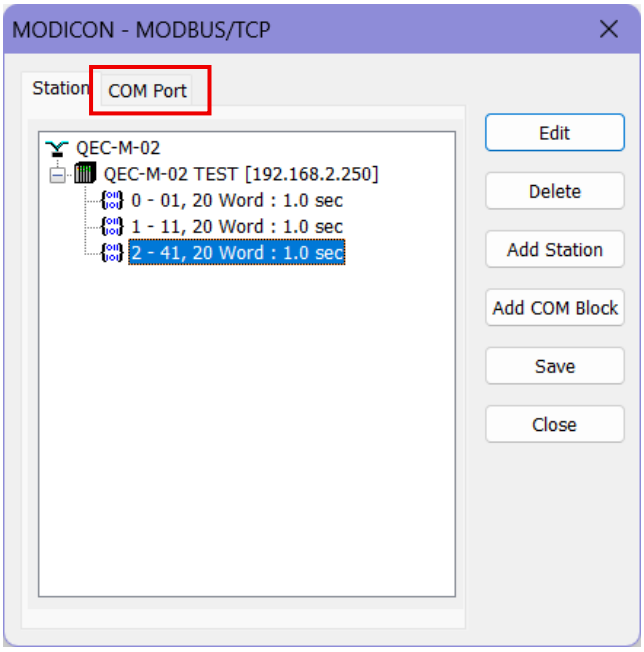


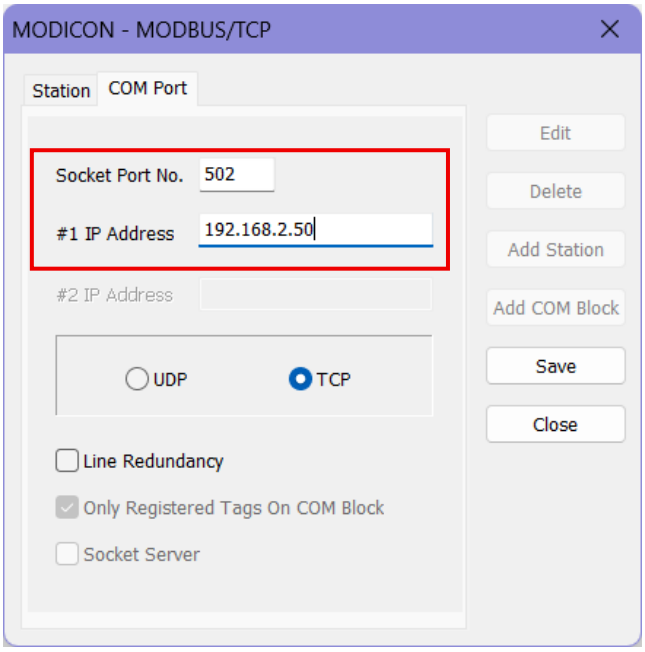


After completing the settings, click **OK**, then proceed to the **COM Port** configuration.
On the **COM Port** page, confirm the following settings:

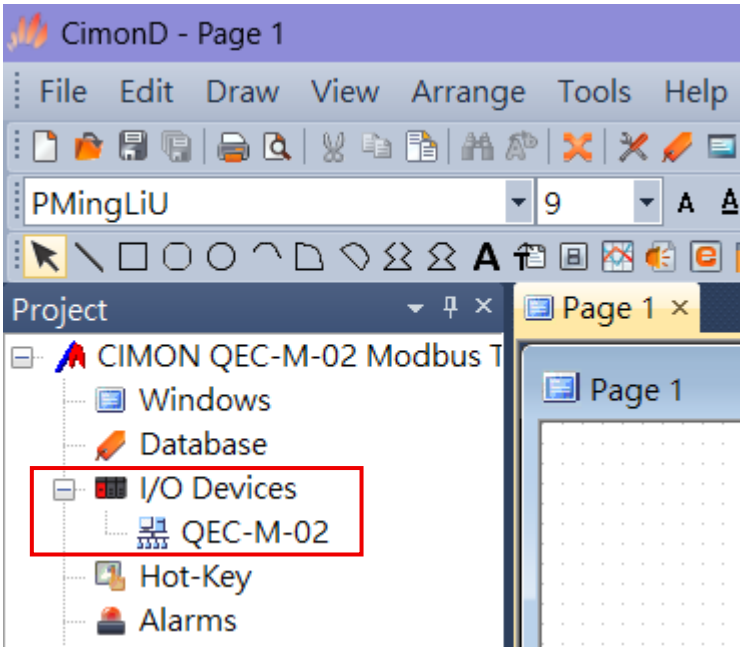
- **Socket Port No.**: 502
- **#1 IP Address**: Local computer IP address
- **Protocol**: TCP

The local computer IP can be found by running ***ipconfig*** in Windows CMD. After completing the settings, click **Save**, then click **Close**.





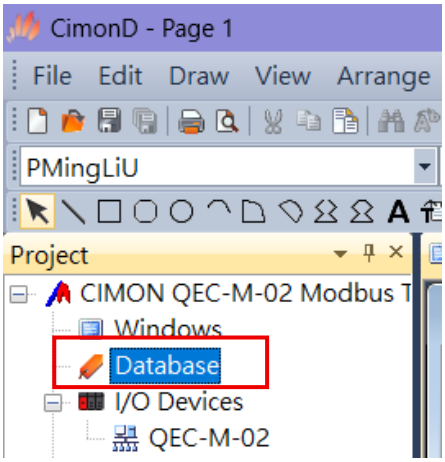
After completing the settings, the Modbus TCP device just created will appear under I/O Devices.

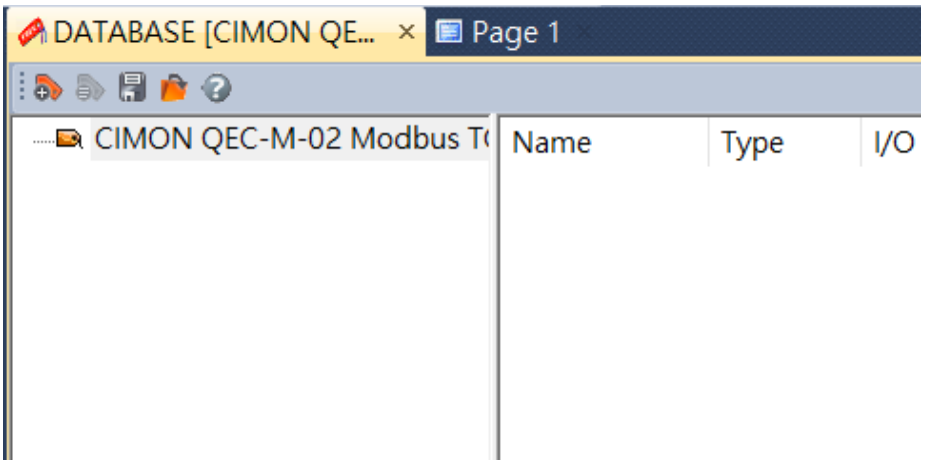


6.3 Create Tags

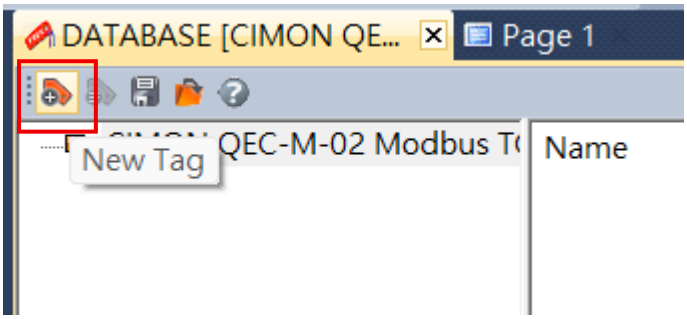
After completing the communication device setup, create CIMON Tags. Tags define the mapping between CIMON SCADA screen components and Modbus addresses. Buttons, Tag Values, LED indicators, and other components will all use Tags to perform read/write operations with QEC-M-02.

In the left **Project** panel, find **Database** and double-click it to enter the Tag configuration screen.



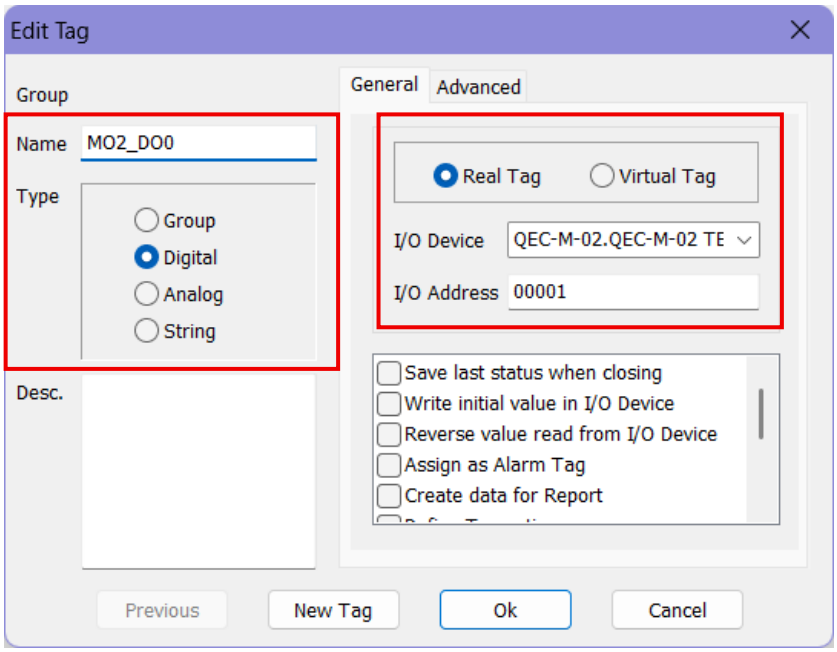


Click **New Tag** in the upper-left corner to create a new Tag.



Using M02_DO0 as an example, configure as follows:

- **Name**: M02_DO0
- **Type**: Digital
- **General**: Real Tag
- **I/O Device**: Select the QEC-M-02 Device / Station created earlier
- **I/O Address**: 00001



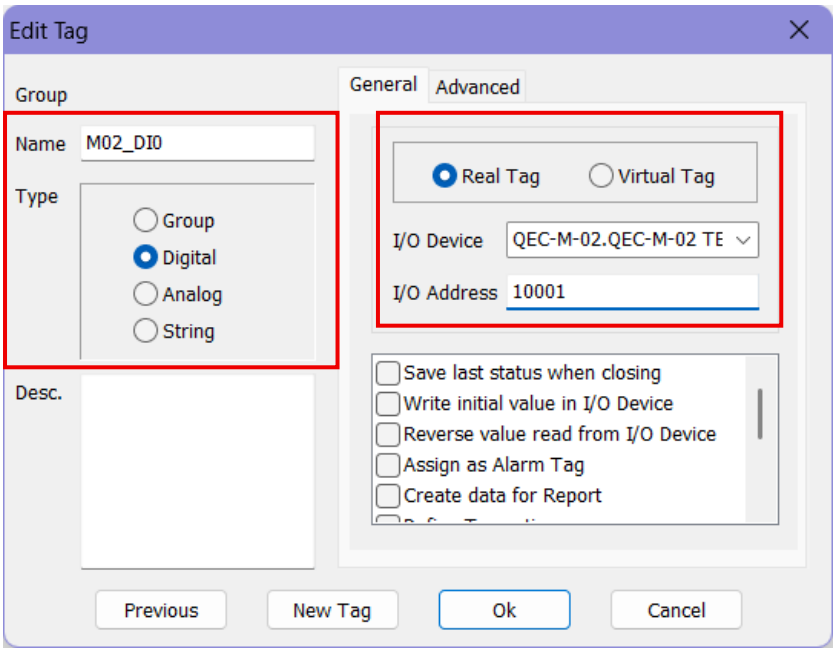
Create the following Tags using the same method:

Tag Name	Type	I/O Address	Description
M02_DO0	Digital	00001	QEC-M-02 DO0
M02_DO1	Digital	00002	QEC-M-02 DO1
M02_DO2	Digital	00003	QEC-M-02 DO2
M02_DO3	Digital	00004	QEC-M-02 DO3

DI Tags are created similarly to DO Tags — also select **Digital** and **Real Tag** — the difference is that the I/O Address uses Discrete Input addresses starting at 10001.

Using M02_DI0 as an example:

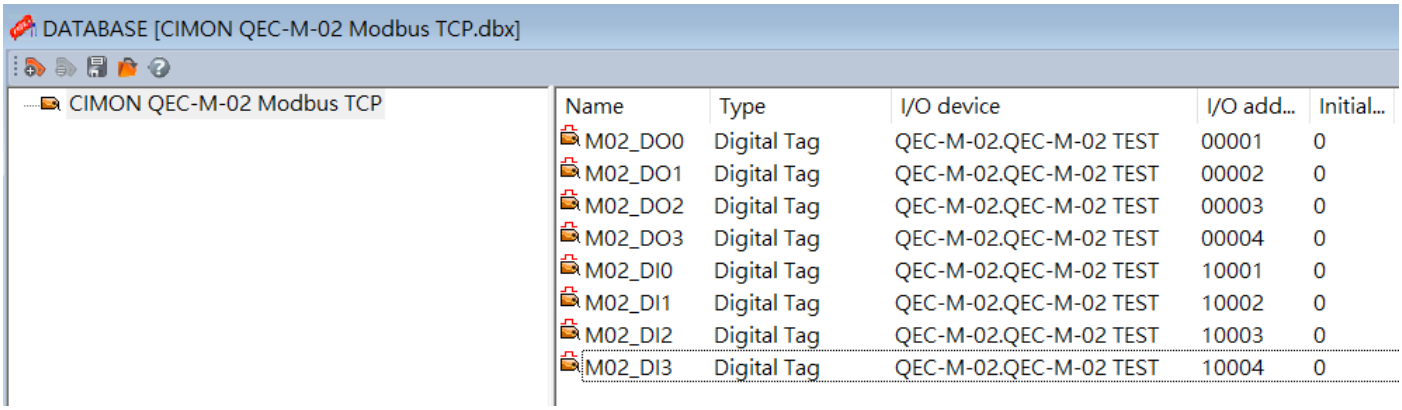
- **Name**: M02_DI0
- **Type**: Digital
- **I/O Address**: 10001



Create the following Tags using the same method:

Tag Name	Type	I/O Address	Description
M02_DI0	Digital	10001	QEC-M-02 DI0
M02_DI1	Digital	10002	QEC-M-02 DI1
M02_DI2	Digital	10003	QEC-M-02 DI2
M02_DI3	Digital	10004	QEC-M-02 DI3

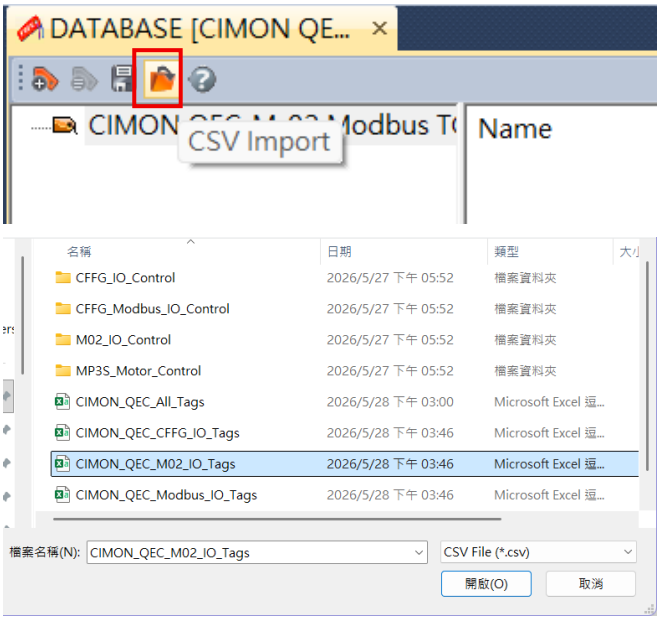
After completing the setup, you can check the Database screen to confirm that each Tag's Name, Type, I/O Device, and I/O Address are correct.



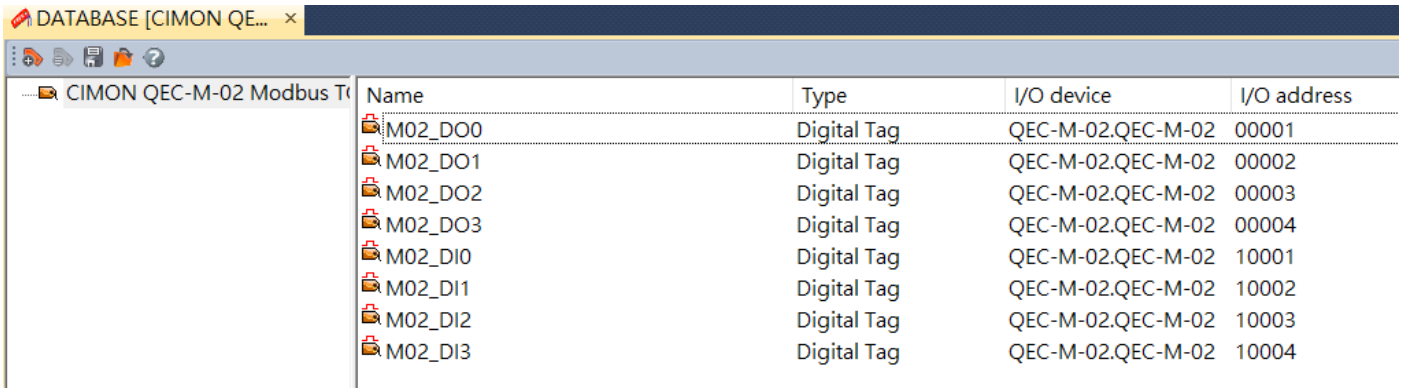
6.4 Import and Export Tags

If you have a prepared Tag CSV file, you can use the import function to quickly create multiple Tags, reducing manual entry time and configuration errors.

In the **Database** page, click **CSV Import**, select the CSV file to import, and execute the import.

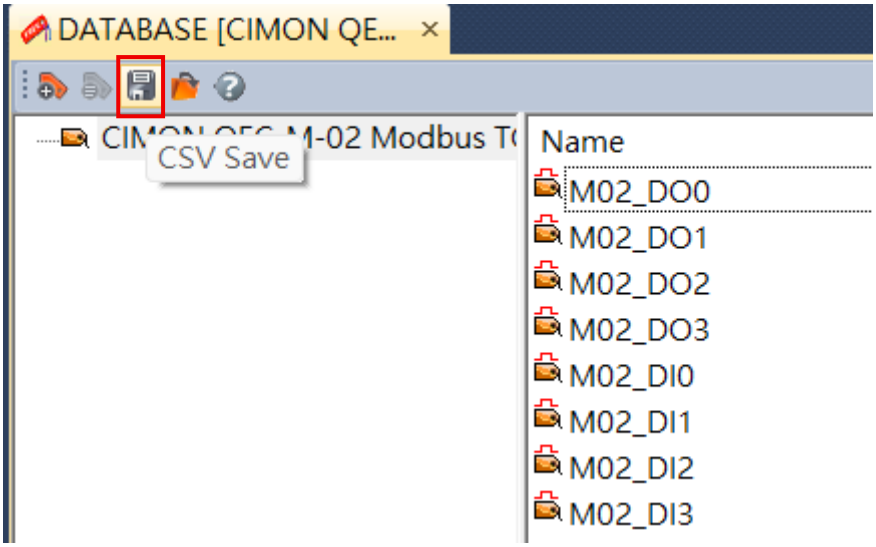


After importing, verify that the Tags appear correctly in the Database list.



***Note:** After importing a CSV, confirm that each Tag's **I/O Device** corresponds to the Device / Station created in the current project. If the I/O Device name in the CSV differs from the current project, screen components may fail to read or write data. In this case, you will need to re-edit the Tag's I/O Device setting.

If you need to export the created Tags to a CSV file, click **CSV Save**.



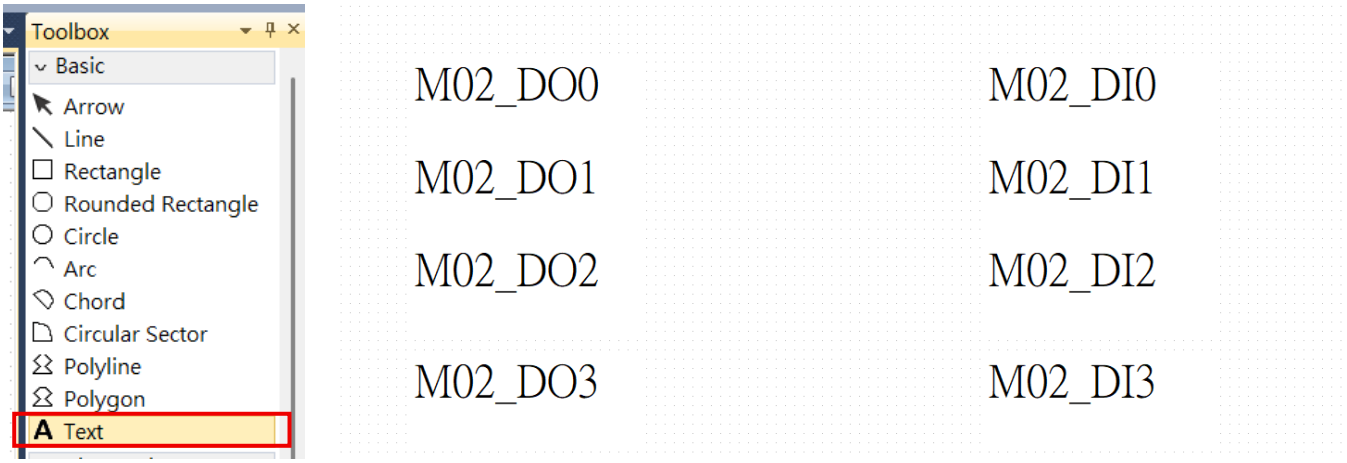
6.5 Add Text Components

After completing Tag configuration, you can begin building the SCADA screen. First add text labels to help users identify each I/O point.

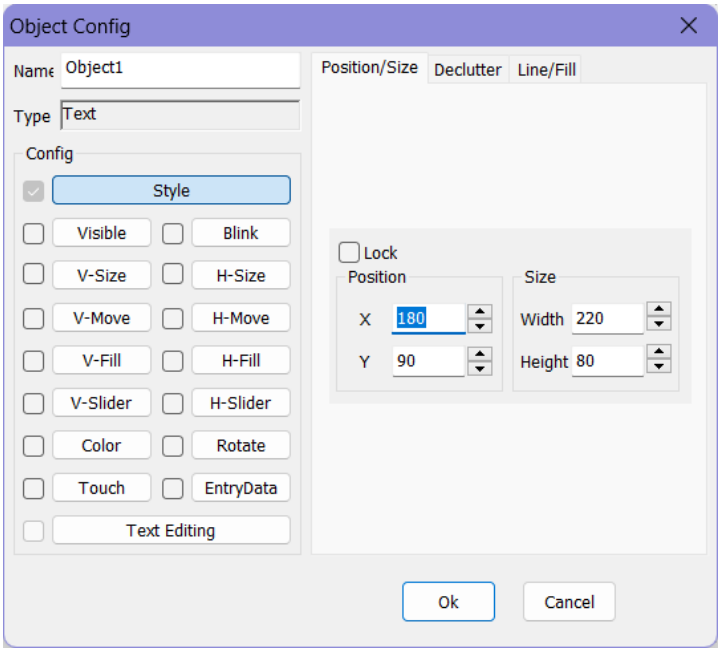
Return to the main screen, select **Basic** → **Text** in the right **Toolbox**, then click once on the canvas to add a text object.

This example creates labels for:

- M02_DO0 ~ M02_DO3
- M02_DI0 ~ M02_DI3



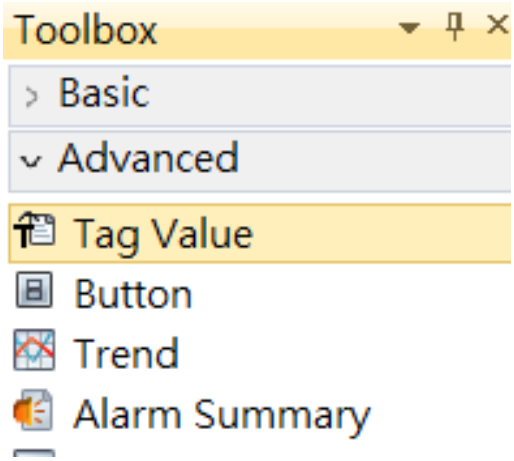
If you need to adjust the text position, size, or style, double-click the text object to open Object Config.



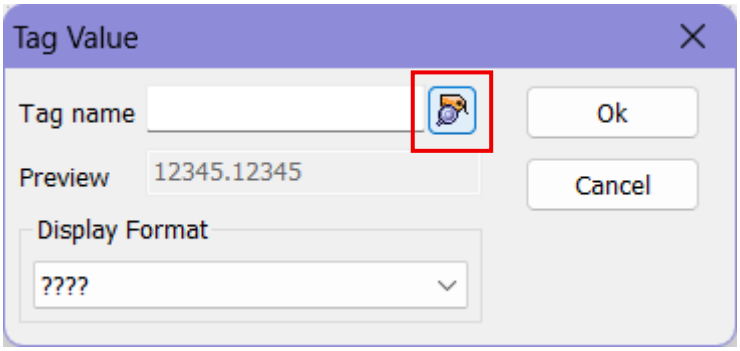
6.6 Add Tag Value Components

Tag Value components display the real-time value of a Tag. This example first demonstrates DI status display by showing M02_DI0 ~ M02_DI3 on the screen.

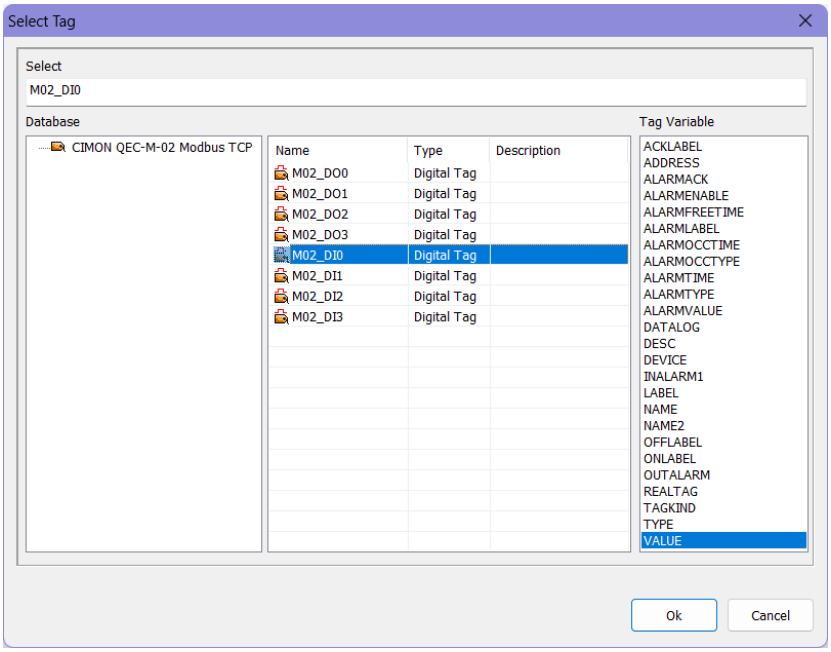
Select **Advanced** → **Tag Value** in the right **Toolbox**, then click once on the canvas.



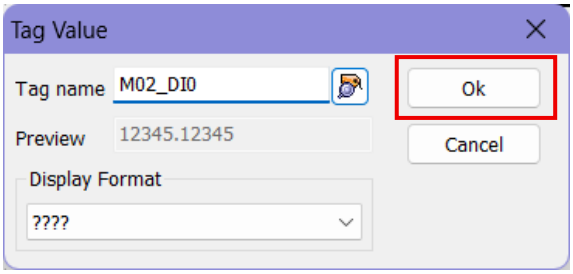
Click the magnifying glass icon next to **Tag name** to open the Select Tag window.



In the Select Tag window, select the Tag to display (e.g., M02_DI0), then click **OK**.



After completing the setup, the corresponding Tag Value will appear on the screen.



Add M02_DI1 ~ M02_DI3 using the same method. You can then resize the Tag Value components and align them with the text labels created earlier. The result is shown in the figure below.

***Extended Note:** Displaying Motor Position

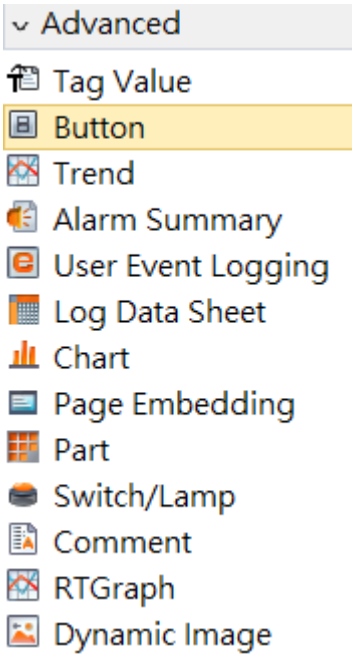
Displaying motor Position works exactly the same way as displaying DI values — it also uses a Tag Value component bound to the corresponding Tag. The only difference is that Position typically uses an Analog Tag or Holding Register-type numeric Tag, whereas DI uses a Digital Tag.

Therefore, to display the current motor position, simply add a Tag Value component and bind the Tag name to the corresponding Motor Position Tag to show the current position value on the screen.

6.7 Add Button Components

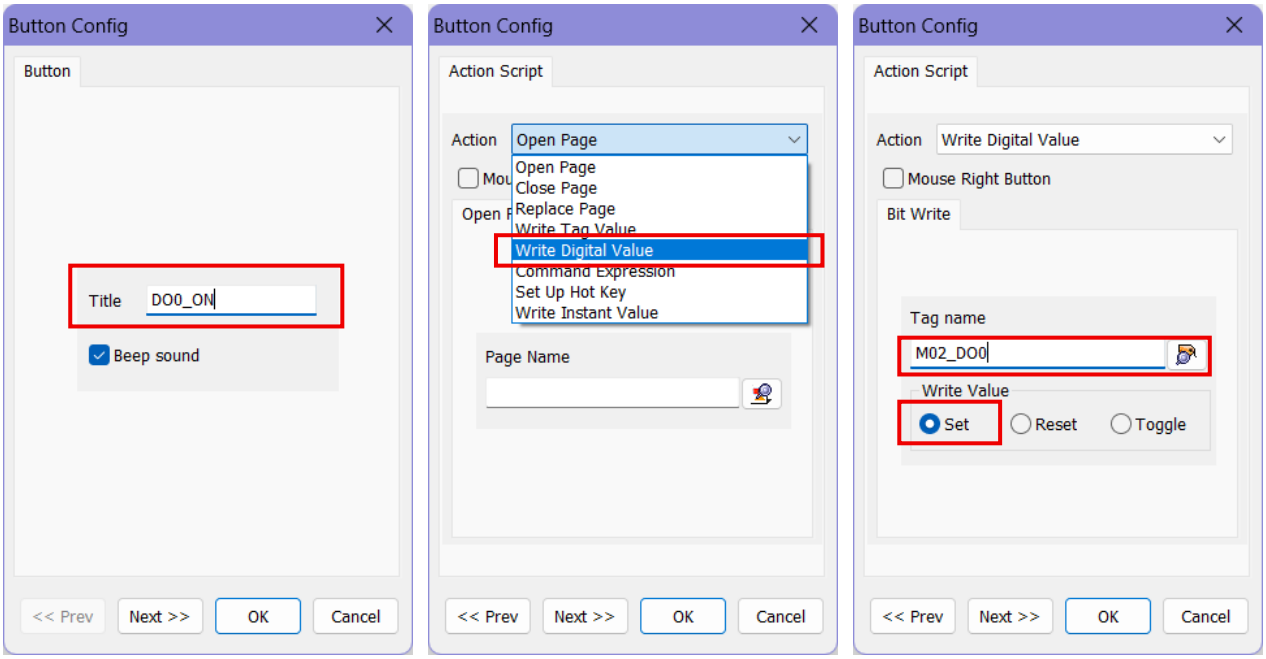
Button components are used to write to Digital Tags. Common uses include DO ON/OFF, JOG, STOP, Enable, and Disable commands. This section uses M02_DO0 ON/OFF as an example.

Select **Advanced** → **Button** in the right **Toolbox**, then click once on the canvas.



In the Button configuration window, set **Title** to: **DO0_ON**
Under **Action**, select **Write Digital Value** and configure:

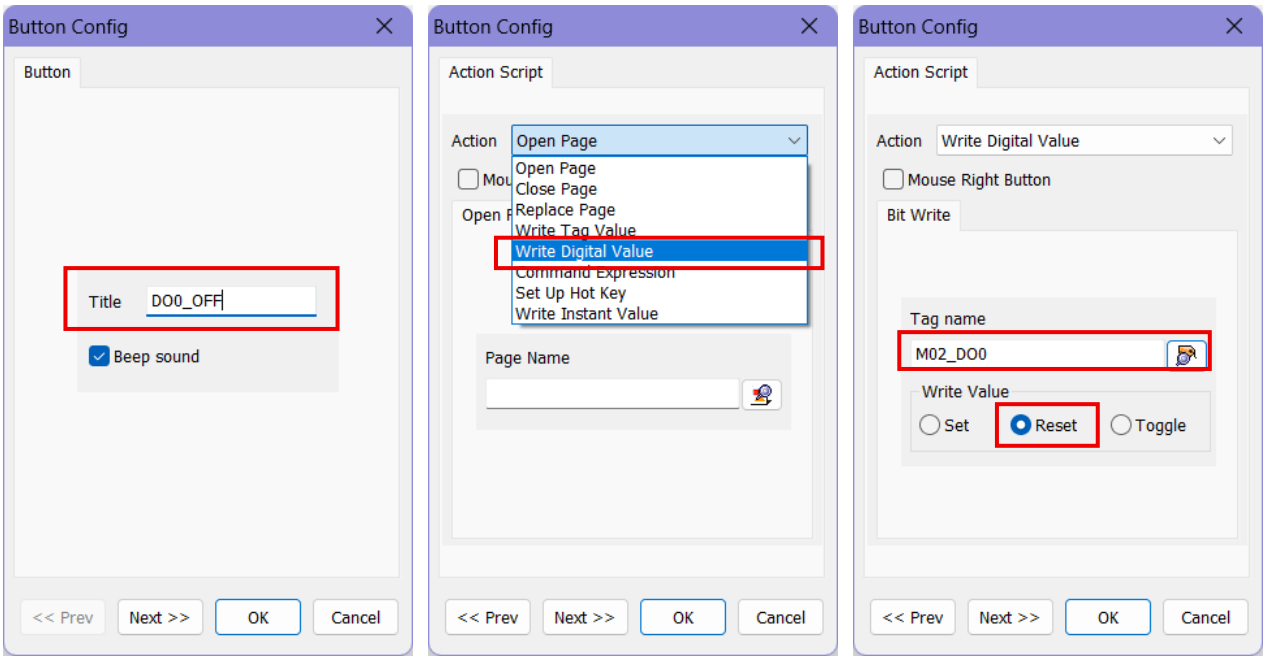
- **Tag name**: M02_DO0
- **Write Value**: Set



After completing the setup, click **OK** to see the button on the canvas.

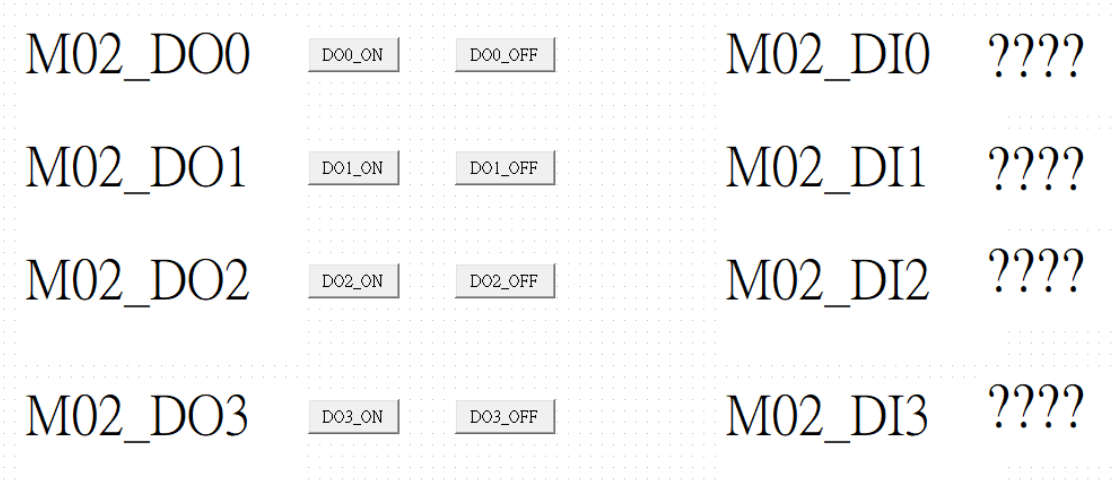
The OFF button is created the same way as the ON button. The differences are:

- **Title**: DO0_OFF
- **Tag name**: M02_DO0
- **Write Value**: Reset



Arrange the DO0_ON and DO0_OFF buttons side by side to keep the screen layout clear and consistent.

Create M02_DO1 ~ M02_DO3 using the same method — simply change the button name and corresponding Tag.



***Extended Note: Motor Control Buttons**

In addition to the M02_DO0 ~ M02_DO3 ON/OFF buttons shown in this section, other control functions can be built using the same approach. For example, motor control buttons such as **JOG+, JOG-, STOP, Enable, and Disable** are also implemented by writing to the corresponding Digital Tags via Button components.

The method is the same as for DO buttons: add a Button component, select **Write Digital Value** in **Action**, choose the corresponding **Tag name**, and set **Write Value** to Set, Reset, or Toggle as needed. Simply change the button name and Tag per function.

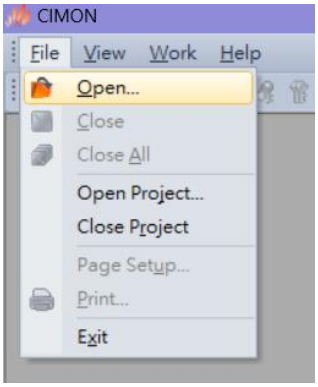
Therefore, this document does not repeat the creation procedure for each motor button. Users can extend the DO ON/OFF button setup described above to create motor control-related buttons.

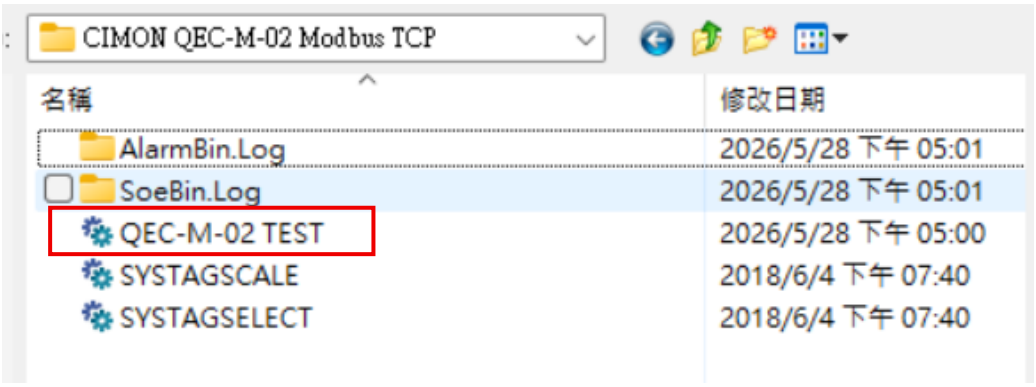
6.8 Testing and Expected Results

After completing the Device, COM Block, Tag, and screen component settings, check the following before running the project:

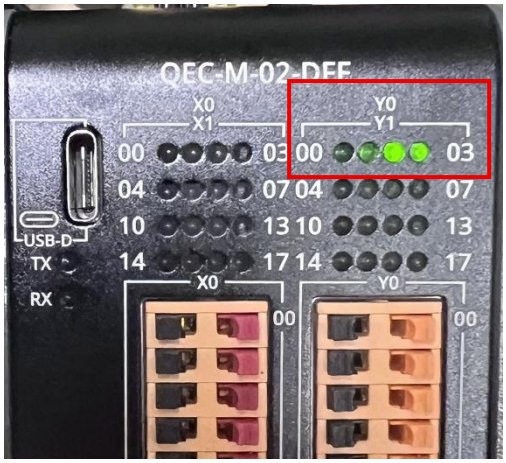
- QEC-M-02 is powered on and running the correct Modbus TCP Slave sketch.
- The PC and QEC-M-02 are on the same subnet.
- The IP address, Port 502, and Slave ID settings are correct.
- Tags are mapped to the correct Device / Station.
- COM Blocks include Coil, Discrete Input, and Holding Register areas.

After confirming, open **CIMON X**, go to **File** → **Open** in the upper-left corner, and select the project created earlier.





Once the project is opened, you will see the operating screen built previously.



Expected Results

- Pressing the DO_ON button writes 1 to the corresponding Coil, and the QEC-M-02 local DO output becomes HIGH.
- Pressing the DO_OFF button writes 0 to the corresponding Coil, and the QEC-M-02 local DO output becomes LOW.
- DI Tag Values display the current 0 / 1 input status.
- In the SCADA example screen above, DO2 and DO3 are turned ON, so the corresponding DO2 and DO3 indicators are lit.

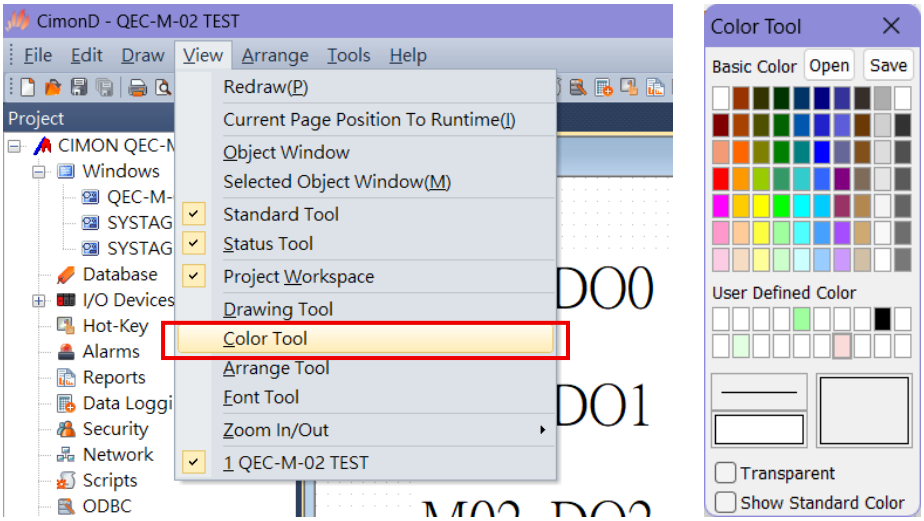
7. Screen Optimization

After completing basic communication, Tag, and screen component setup, the following optimizations can be applied to make the screen easier to read, operate, and demonstrate.

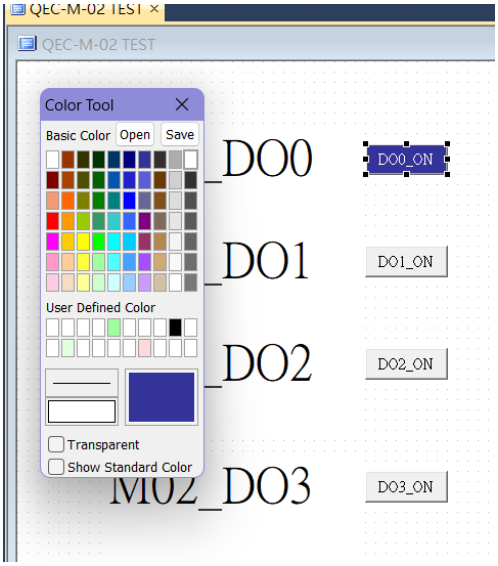
The previous chapters demonstrated the most basic component creation methods. This chapter describes how to organize the same Text, Button, Tag Value, Circle, and Image components into a screen closer to a real-world application. These optimization steps do not change the control logic described earlier — they primarily make the screen easier to read, operate, and present.

7.1 Button Color Settings

To adjust button colors, open the color tool via **View** → **Color Tool**.

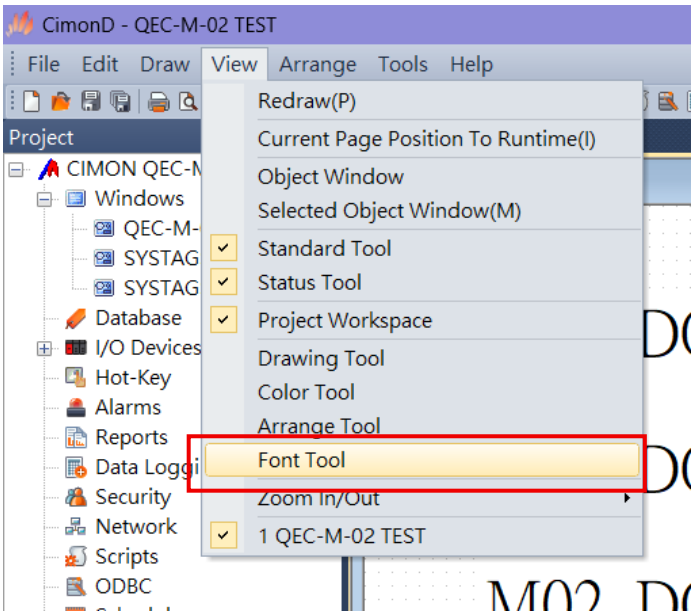


Select a button on the canvas to adjust its text color or background color.
In this example, a dark blue button with white text is used for demonstration. For actual operation screens, light green is recommended for ON buttons, while light red or gray is recommended for OFF or Disable buttons.

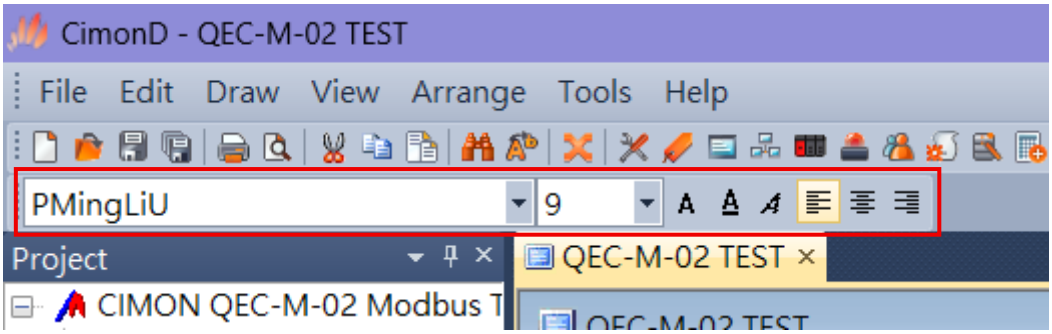


7.2 Font and Size Adjustment

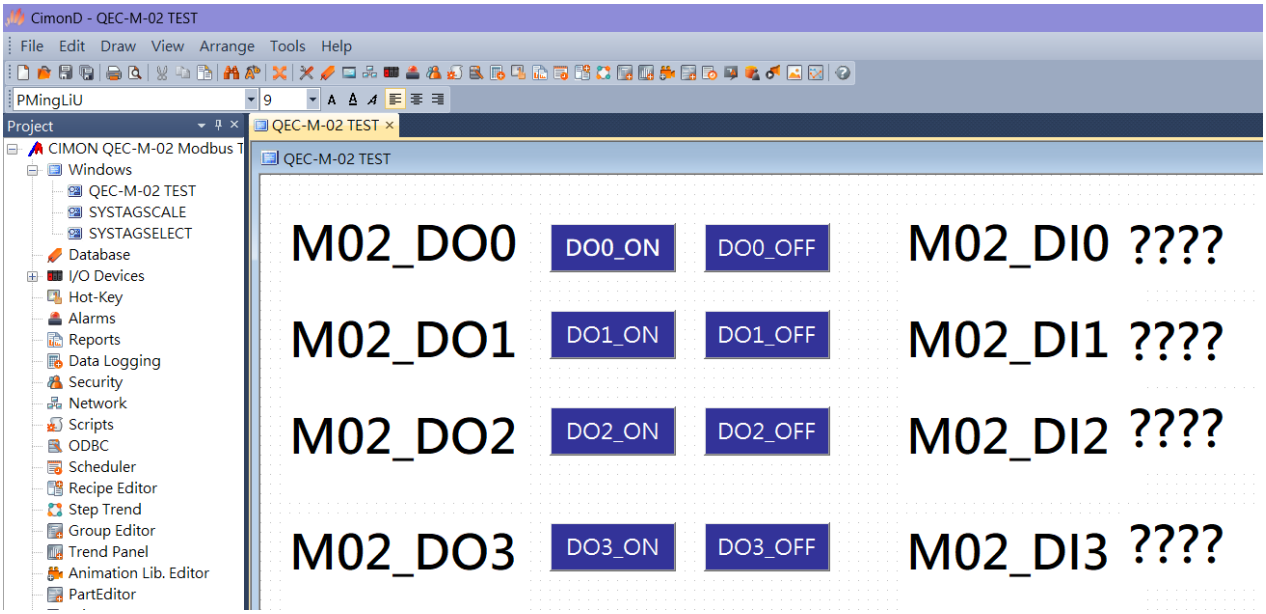
To adjust text size or font, click **View** → **Font Tool**.



Once opened, the top toolbar shows font-related settings where you can adjust font face, size, bold, underline, italic, and text alignment.

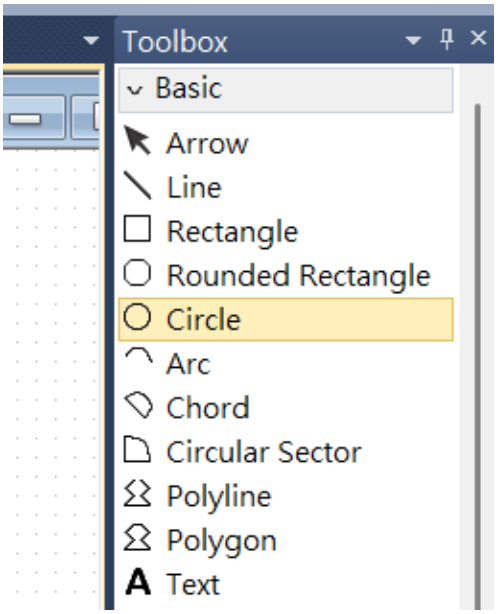


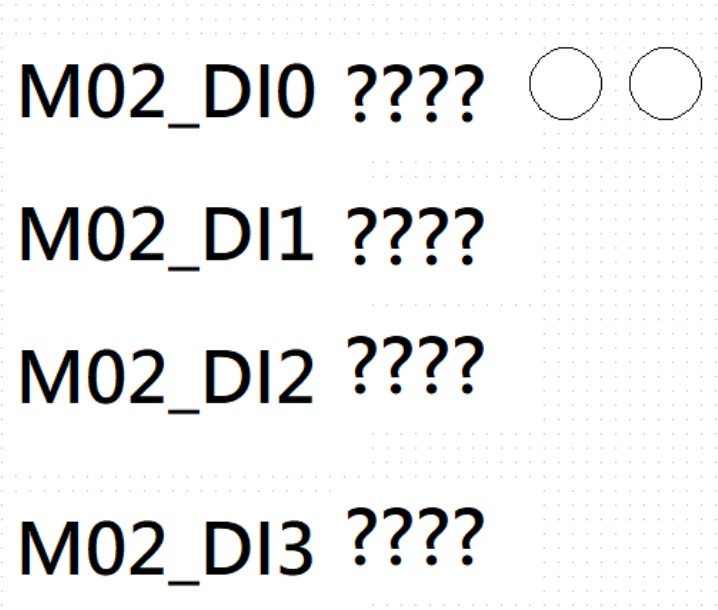
It is recommended to use the same font size for the same type of text and maintain consistent alignment to make the screen easier to read when running in CIMON X.



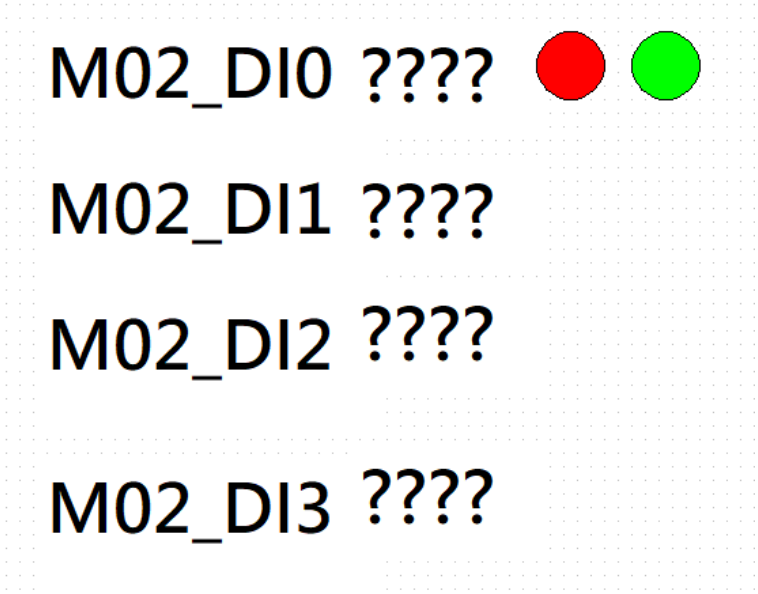
7.3 Show or Hide Elements Based on Tag State

To create an LED-style indicator effect, use Circle components with Visible settings to display circles of different colors based on the Tag state.
First, select **Circle** from the right Toolbox and add two circles to the canvas.

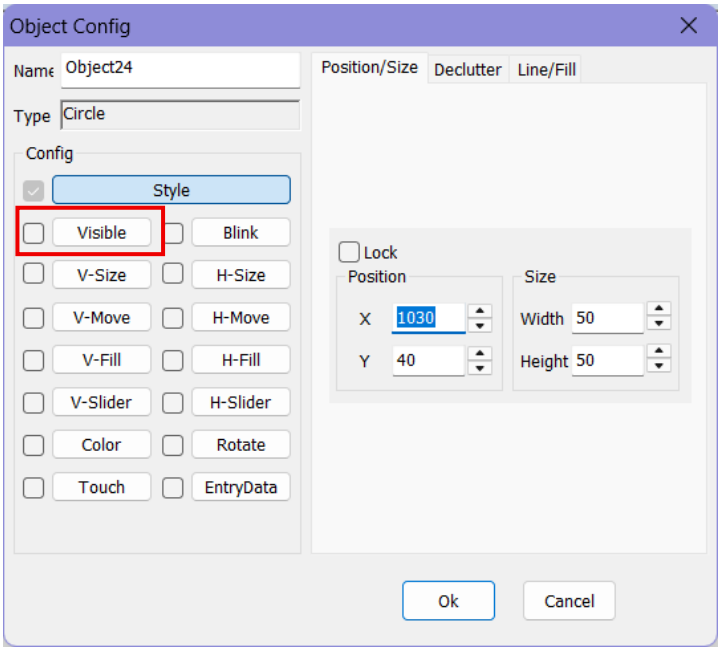




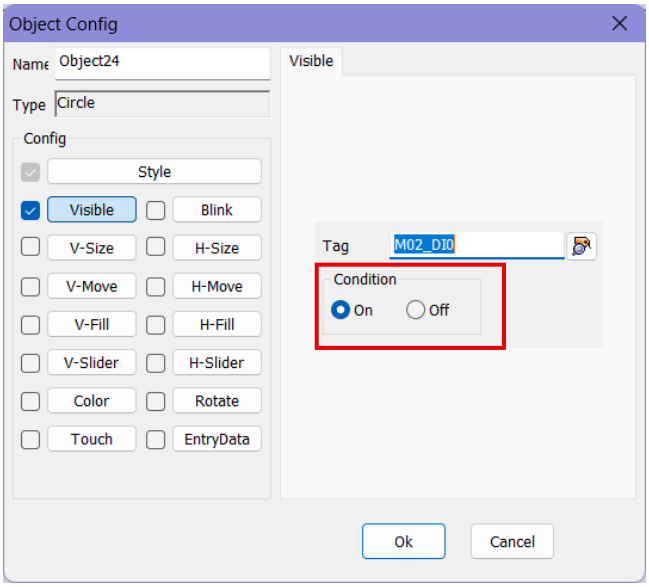
Setting them to different colors (e.g., red and green).

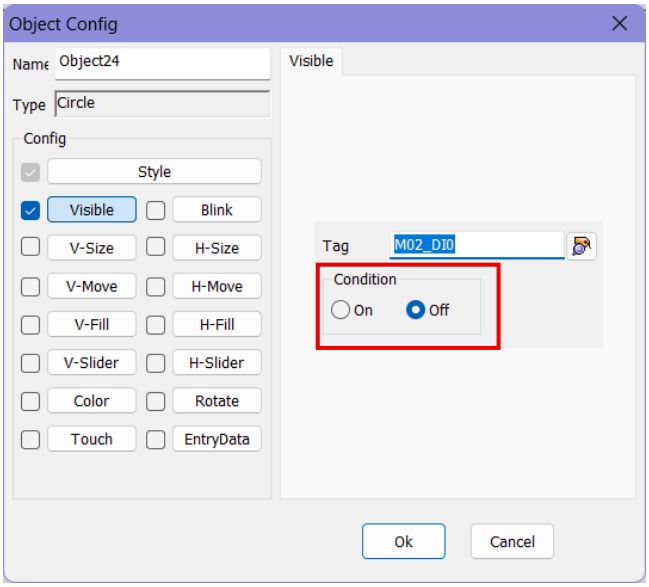


Then double-click a circle to open Object Config and set the **Visible** condition.

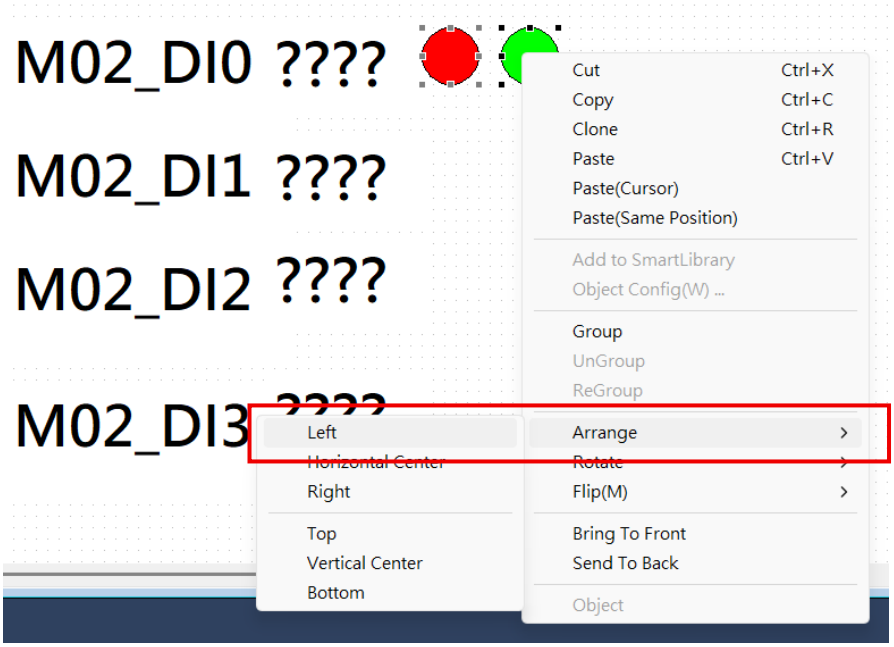


Using M02_DI0 as an example: set the red circle to show when the Tag is OFF, and the green circle to show when the Tag is ON.

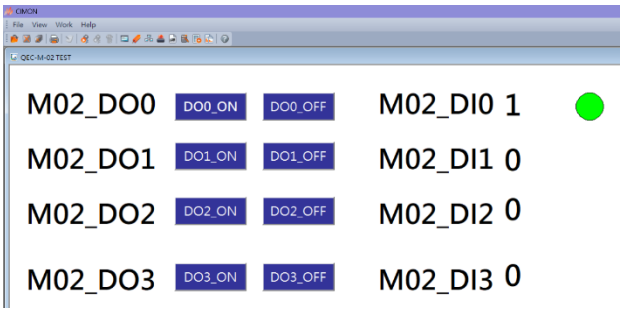


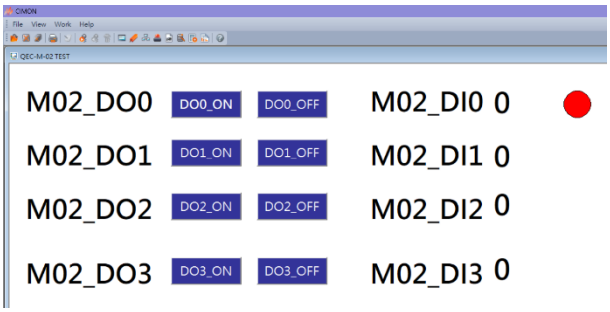


After completing the setup, overlap or align the two circles to create a toggle indicator effect. When running CIMON X, the indicator changes display based on the Tag state.



When running CIMON X, the indicator toggles based on the Tag state: green when DI0=1, red when DI0=0.





Extended Note: Motor Status Display

Motor Status display works the same way as the LED indicator shown above — it also controls element visibility based on Tag state. For example, you can use Enable / Disable-related Tags to show different status indicators or text.

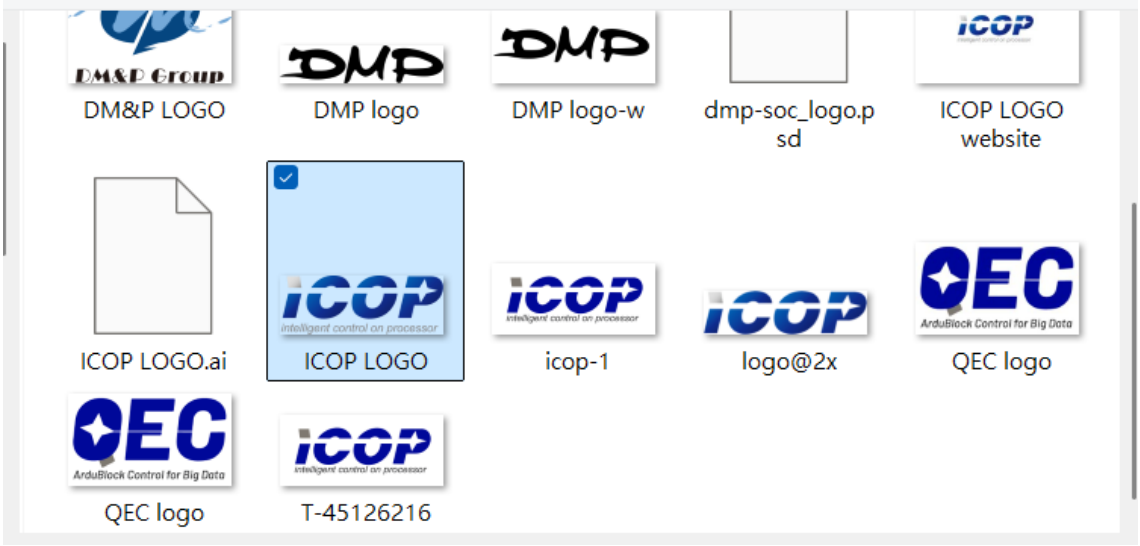
Common approaches:

- When the Enable state is ON, show a green status indicator or “Running / Enabled” text.
- When the Disable state is ON, show a gray or red status indicator, or display “Disabled” text.
- If there is a dedicated Motor Status Tag, set the display condition directly based on that Tag.

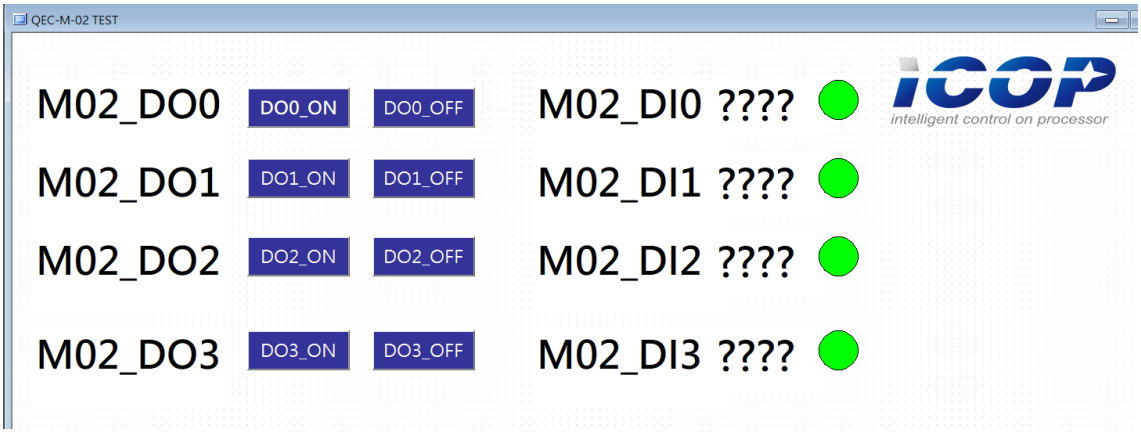
Therefore, Motor Status does not require any new configuration methods — simply apply the Tag show/hide logic from this section, changing the display condition to a motor-related Tag.

7.4 Insert Images or Icons

To add device images, logos, or diagrams to the screen, use **Toolbox → Etc → Image Embedding**.

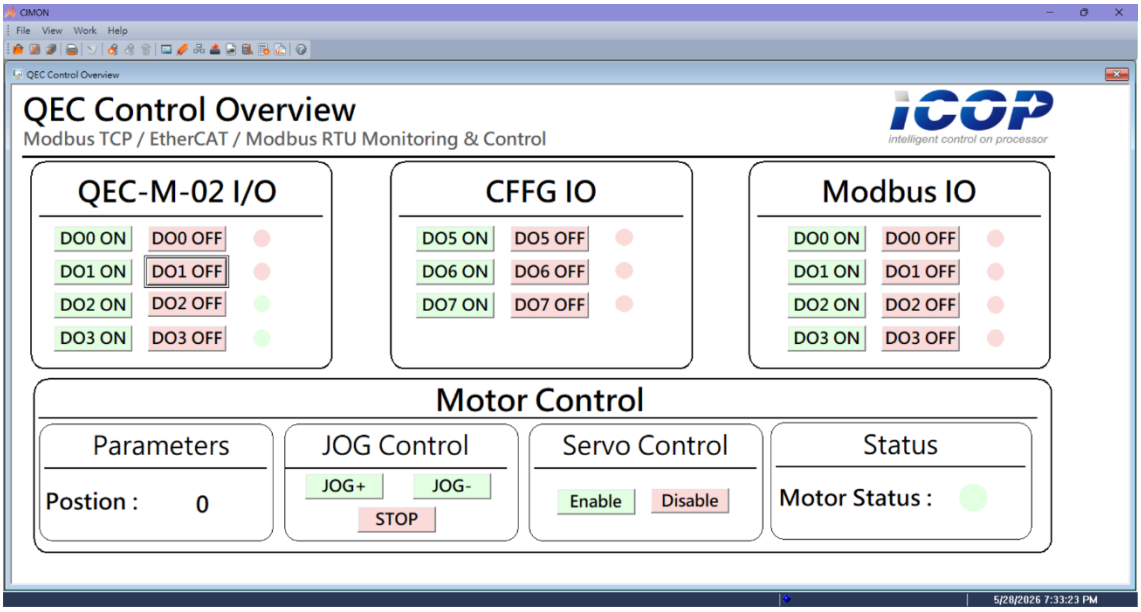


After selecting the image to insert, place it on the screen. Images can display company logos, device appearances, or operation diagrams to make the screen closer to a real-world application.



7.5 Screen Layout Example

After completing basic component setup, it is recommended to divide the screen into sections by device or function so users can quickly understand each area. The figure below shows an optimized example screen that integrates QEC-M-02 I/O, CFFG I/O, Modbus IO, and Motor Control on a single page.



This optimized screen uses the components introduced in the previous sections, including Text, Button, Tag Value, Circle, and Image Embedding. These components can be arranged by function to create a clearer and more systematic UI layout.

The screen can be divided into different areas, such as QEC-M-02 I/O, CFFG I/O, Modbus I/O, and Motor Control. Each area contains the related buttons, values, and status indicators, allowing users to monitor and control each device more easily.

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.

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

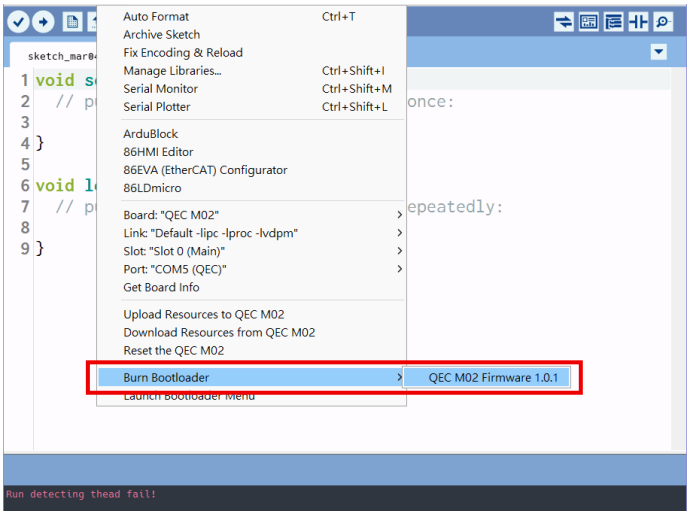
Step 1: Setting up QEC-M

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.

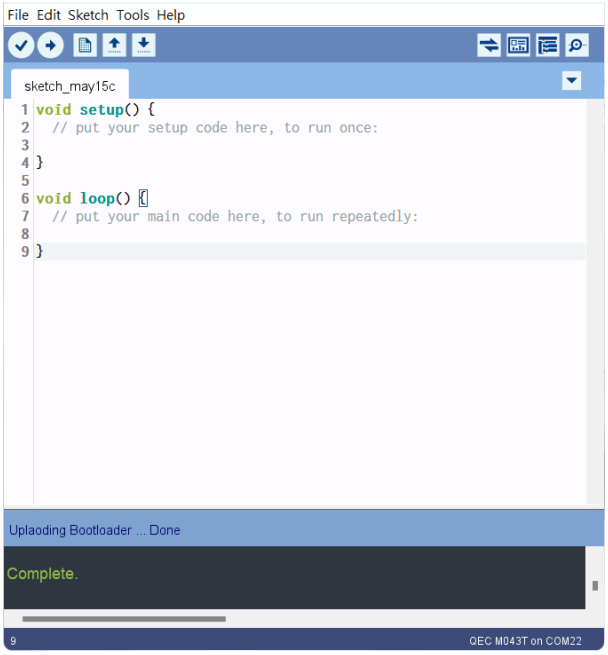
Step 2: Click “Burn Bootloader” button

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:



Step 3: Complete the Update



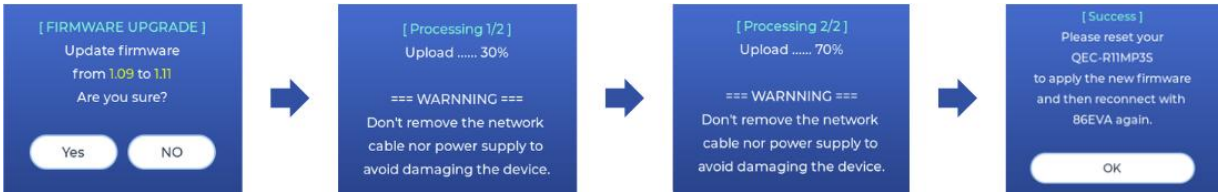
After completing the above steps, your QEC-M has been successfully updated to the latest version of the development environment.

QEC-R11MP3S – Firmware Update

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

Please click it to update/downgrade the device.

After Update finish, please power cycle the device.



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

CIMON SCADA

1. Tag Value displays ???? in CIMON X

Possible causes:

- The Tag value has not been read successfully.
- CIMON X is not running correctly.
- QEC-M-02 is not online.
- The IP address, Port, or Station No. is incorrect.
- The Tag's I/O Device is not mapped to the current Device / Station.

What to check:

- Confirm that CIMON X is running.
- Confirm that QEC-M-02 is powered on and connected.
- Check the IP address, Port 502, and Station No.
- Check whether the Tag's I/O Device is assigned to the correct Device / Station.

2. Button press has no effect

Possible causes:

- The Button is not bound to the correct Tag.
- The Write Value setting is incorrect.
- The QEC-M-02 sketch is not responding.

What to check:

- Check the Button's Tag name.
- Check whether the Write Value is set correctly.
- Confirm that the correct QEC-M-02 sketch is running.

3. Cannot read or write after CSV import

Possible cause:

- The CSV I/O Device name does not match the current project.

What to check:

- Re-assign the Tag's I/O Device in the Database screen.

4. DI status is not updating

Possible causes:

- The Discrete Input address is incorrect.
- The COM Block setting is incorrect.

What to check:

- Check the 10001 starting address.

- Check the Block No. 1 settings.

5. Holding Register value is not displayed

Possible causes:

- The Register address is incorrect.
- The Tag type is incorrect.
- The Holding Register COM Block is not configured correctly.

What to check:

- Check the 40001 starting address.
- Check the Holding Register Block setting.
- Check the Tag Type.

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.