User Manual

QEC-M-01

DM&P Vortex86EX2 Processor

EtherCAT MDevice

(Revision 3.1)

REVISION

DATE	VERSION	DESCRIPTION
2022/06/24	**Version1.0A**	**First Release.**
2023/03/28	**Version1.1A**	**Modified Start Guide.**
2023/08/08	**Version2**	**Updated Product Specifications.**
2023/11/05	**Version2.1**	**Updated Hardware system.**
2024/01/14	**Version2.2**	**Updated Getting Started.**
2024/12/05	**Version3.0**	**Change Master to MainDevice.** **Added the Bootloader Menu Usage.** **Added RS485 and Modbus Usage.** **Added USB Usage.** **Added Ethernet Usage.** **Updated EtherCAT Information.**
2025/03/14	**Version3.1**	**Change MainDevice to MDevice** **Added ENI import method.** **Updated EtherCAT Function List.** **Updated 86Duino IDE 500 to 501.**

COPYRIGHT

Ver.3.1 March, 2025

TRADEMARKS ACKNOWLEDGMENT

For more detailed information or if you are interested in other ICOP products, please visit our official websites at:

- Global: www.icop.com.tw
- USA: www.icoptech.com
- Japan: www.icop.co.jp
- Europe: www.icoptech.eu
- China: www.icop.com.cn

For technical support or drivers download, please visit our websites at:

- https://www.icop.com.tw/resource_entrance

For EtherCAT solution service, support or tutorials, 86Duino Coding IDE 500+ introduction, functions, languages, libraries, etc. Please visit the QEC website:

- QEC: https://www.qec.tw/

This Manual is for the QEC series.

SAFETY INFORMATION

- Read these safety instructions carefully.
- Please carry the unit with both hands and handle it with caution.
- Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)
- Make sure the voltage of the power source is appropriate before connecting the equipment to the power outlet.
- To prevent the QEC device from shock or fire hazards, please keep it dry and away from water and humidity.
- Operating temperature between -20 to +70°C/-40 to +85°C (Option).
- When using external storage as the main operating system storage, ensure the device's power is off before connecting and removing it.
- Never touch un-insulated terminals or wire unless your power adaptor is disconnected.
- Locate your QEC device as close as possible to the socket outline for easy access and avoid force caused by the entangling of your arms with surrounding cables from the QEC device.
- If your QEC device will not be used for a period of time, make sure it is disconnected from the power source to avoid transient overvoltage damage.

WARNING!

DO NOT ATTEMPT TO OPEN OR TO DISASSEMBLE THE CHASSIS (ENCASING) OF THIS PRODUCT. PLEASE CONTACT YOUR DEALER FOR SERVICING FROM QUALIFIED TECHNICIAN.

CONTENT

Ch. 1

General Information

1.1 Introduction

ICOP's QEC-M-01 is an EtherCAT MDevice based on Vortex86EX2 processor. The development environment uses industrial Arduino-like software, **86Duino IDE**, supporting EtherCAT API, which performs real-time field monitoring and big data collection and supports graphical programming tools, making it easy to hire software engineers and shorten the market time.

It offers real-time EtherCAT communication between EtherCAT MDevice and EtherCAT sub-devices. Except for the EtherCAT Library of 86Duino IDE, QEC MDevice also provides Modbus, Ethernet TCP/IP, CAN bus, etc. industrial communication protocols and uses a rich high-level C/C++ programming language for rapid application development.

The QEC MDevice has precise synchronization (min.125μs), and its 86Duino IDE provides less than 1us jitter time in the minimum cycle time; it could apply to highly synchronized and precision automatic applications, like motion control and I/O control. (Read More: EtherCAT MDevice Benchmark)

QEC-M-01 has a built-in high endurance 2GB SLC eMMC, designed to provide a stable and reliable operating system. Users can upload the developed executable files and required images or data, such as HMI images, to the QEC-M-01's SLC via the 86Duino IDE without affecting the performance of the master system. It has two networks for EtherCAT Cable Redundancy, one Giga LAN for external network connection, RS485 signal pins, HD Audio, and USB; All provide an off-the-shelf API to use. Users can quickly collect data over EtherCAT and transfer data to a server via Giga LAN, and then big data can easily to construct by MySQL Library.

QEC-M-01 can also monitor hardware information on temperature, voltage, and current. These features allow users to track the system's carbon footprint and estimate its lifespan. Its dimension is 107.45 x 77.39 x 34 mm, could be mounted via Din-Rail. Operating temperature is from -20°C to +70°C; it can be placed directly in the field outdoors and ensures that the machine can work in harsh environments.

1.1.1 QEC EtherCAT MDevice Architecture

The EtherCAT MDevice software is primarily divided into two parts, each running on the respective systems of the Vortex86EX2 CPU. They are responsible for the following tasks:

- **EtherCAT MDevice Library**
  - Provides C/C++ application interfaces:
    - Initialization interface.
    - Configuration interface.
    - Process Data (PDO) access interface.
    - CAN application protocol over EtherCAT (CoE) access interface.
    - File access over EtherCAT (FoE) access interface.
    - SubDevice Information Interface (SII) access interface.
    - Distributed Clocks (DC) access interface.
- **EtherCAT MDevice Firmware**
  - Executes the EtherCAT MDevice Core.
  - Controls the Primary/Secondary Ethernet Driver, sending EtherCAT frames

The programs are designed to run on the FreeDOS operating system and have been compiled using the GCC compiler provided by the DJGPP environment.

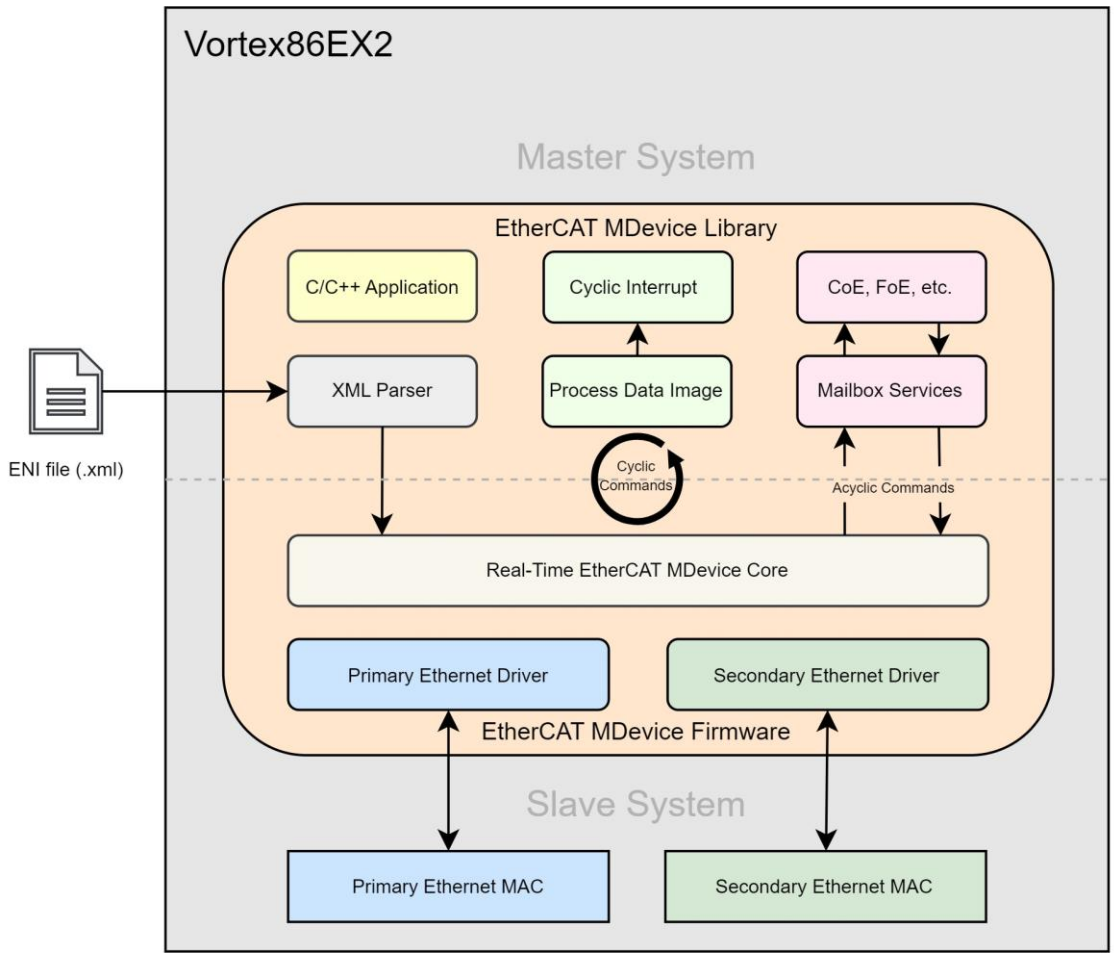
1.1.2 Hardware Platform

The EtherCAT MDevice software only runs on the Vortex86EX2 CPU produced by DM&P, which features a dual-system architecture. It is divided into Master System and Slave System, each running its own operating system, with communication between systems facilitated by Dual-Port RAM and event interrupts.

Their respective tasks are as follows:

- **Master System**
  - User's EtherCAT application.
  - User's HMI application.
  - User's Ethernet application.
  - And so on.
- **Slave System**
  - Only responsible for running the EtherCAT MDevice Firmware.

As most applications run on the Master System, the EtherCAT MDevice Firmware running on the Slave System is free from interference by other applications. This setup allows it to focus on executing the EtherCAT MDevice Core, ensuring the synchronization and real-time capabilities of EtherCAT.



1.1.3 Dual-System Synchronization

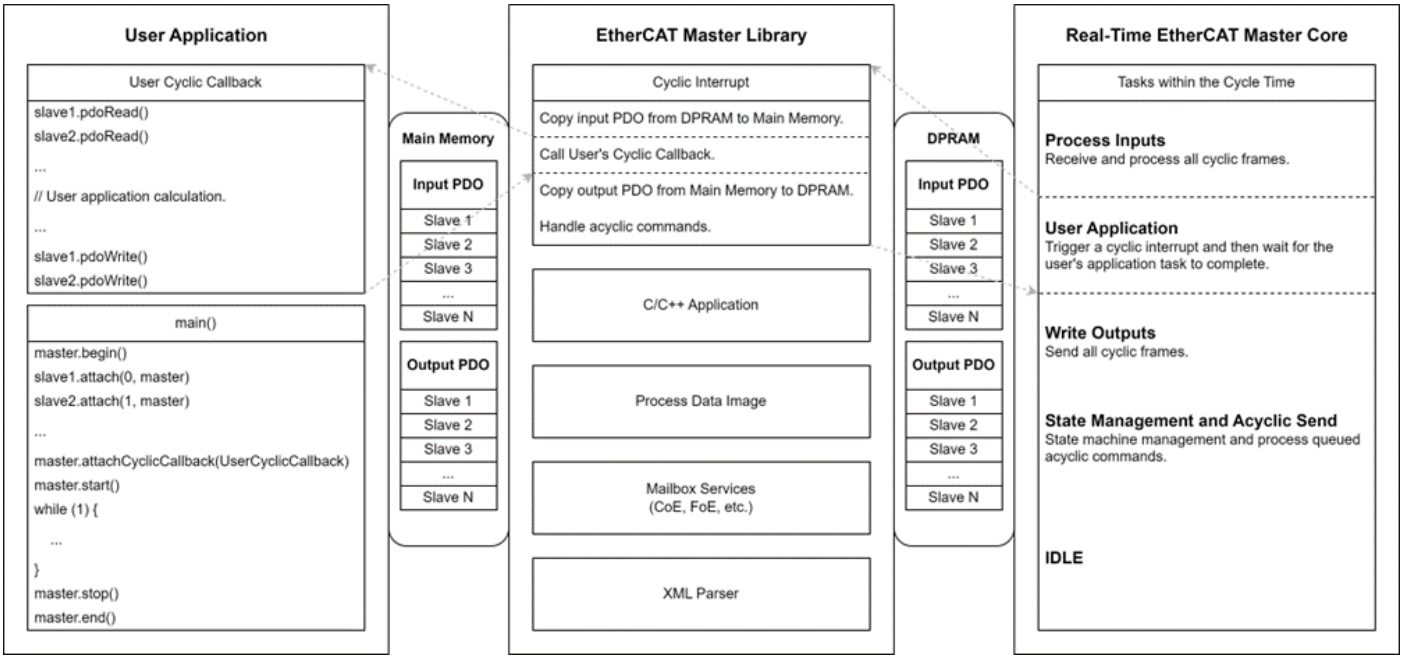
The primary focus of this section is the synchronization of dual-system PDO data. As illustrated in the diagram below, the User Application and EtherCAT MDevice Library blocks run on the Master System, while the Real-Time EtherCAT MDevice Core runs on the Slave System.

When the EtherCAT MDevice Core reaches the ***Process Inputs*** stage, it receives all cyclic frames from the Ethernet Driver and copies Input PDO data to the DPRAM.

Upon reaching the ***User Application*** stage, the EtherCAT MDevice Core triggers a Cyclic Interrupt to the Master System. Upon receiving the Cyclic Interrupt, the Master System executes the interrupt handling procedure of the EtherCAT MDevice Library. It moves Input PDO data from DPRAM to Main Memory, calls the user-registered Cyclic Callback, transfers Output PDO data from Main Memory to DPRAM after the Cyclic Callback completes, processes acyclic commands, and concludes the interrupt handling procedure. At this point, both the EtherCAT MDevice Core's ***User Application*** and the interrupt handling procedure are completed simultaneously.

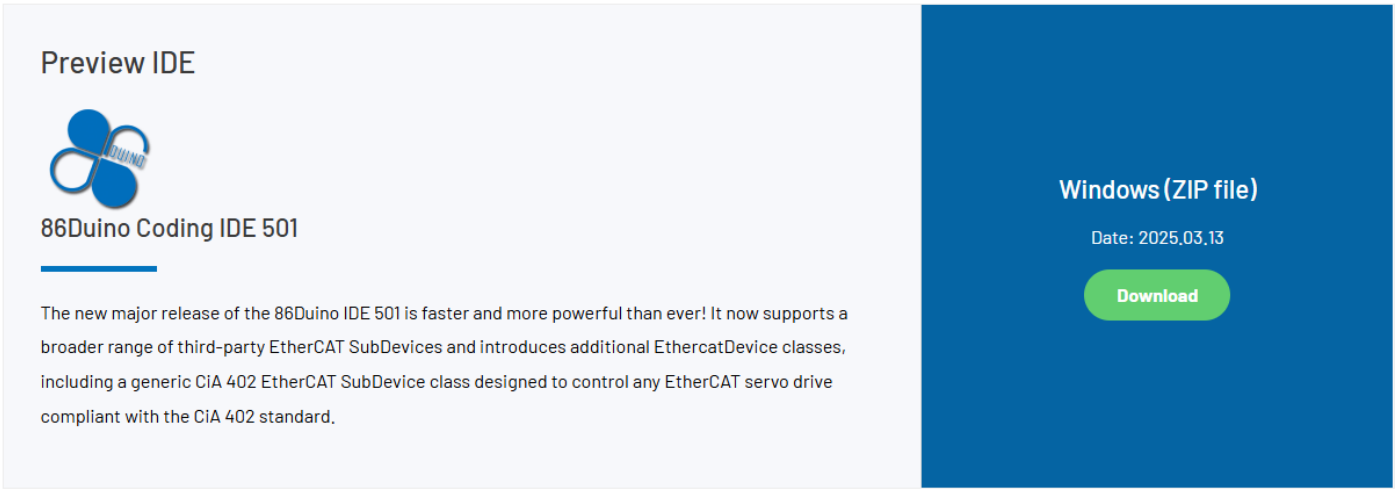
When the EtherCAT MDevice Core reaches the ***Write Outputs*** stage, it copies Output PDO data from DPRAM to the Ethernet Driver's DMA and sends frames.

These tasks are executed periodically in a cyclic manner, following the outlined procedural steps, ensuring the synchronization of dual-system PDO data.



1.1.4 Software Support

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from https://www.qec.tw/software/.



QEC MDevice software, 86Duino IDE, also offers a configuration utility: **86EVA**, a graphic user interface tool for users to edit parameters for the EtherCAT network; its functions are as follows:

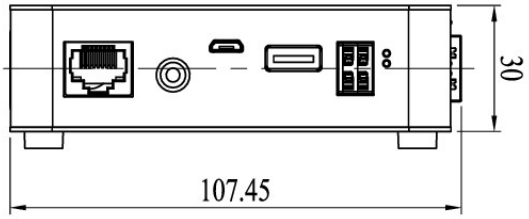
- EtherCAT slave devices scanning.
- Import ENI file.
- Setting EtherCAT MDevice.
- Configure EtherCAT SubDevices.

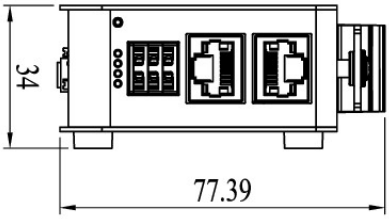
For other detailed functions, please refer to the 86EVA User Manual.

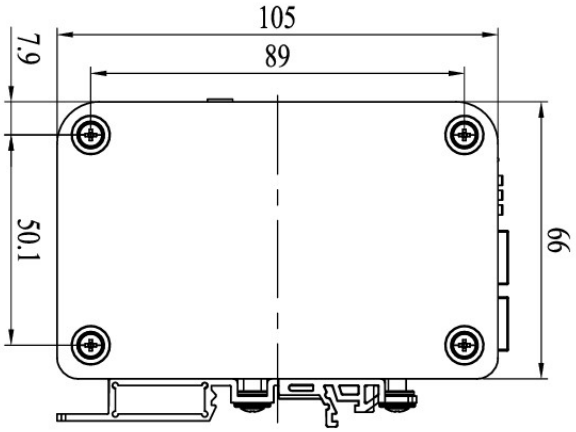
1.2 Specifications

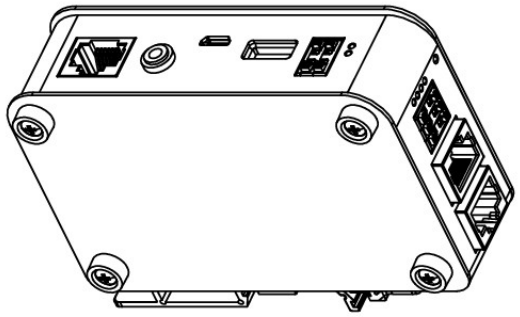
CPU	DM&P Vortex86EX2 Processor, Master 533MHz/Slave 400MHz
Memory	512MB/1GB DDRIII Onboard
Storage	32MB SPI Flash/2GB SLC eMMC
LAN	1Gbps Ethernet RJ45 x1 10/100Mbps Ethernet RJ45 x2 for EtherCAT
I/O Connector	Power DC Input/Output Connector x1 USB 2.0 Host x1 Micro USB (Type-B) x1 (Upload/Debug only) Audio Connector x1 RS485 x1 RJ45 x3
Protocol	EtherCAT, Modbus, Ethernet, etc
Ethernet Standard	IEEE 802.3
EtherCAT Cycle Time	125 μs (min.)
Power Connector	6-pin Power Input /Output
Power Requirement	4-pin Power Input/Output & 2-pin FGND
Power Consumption	5W
Operating Temperature	-20 to +70°C -40 to +85°C (Option)
Dimension	107.45 x 66 x 34mm (Without DIN-Rail)
Weight	270 g
Mounting	DIN-Rail
Certifications	CE, FCC, VCCI
Internal Monitoring	Temperature, Voltage, Current, Startup time
Certifications	CE, FCC, VCCI
Software Support	86Duino Coding IDE 500+

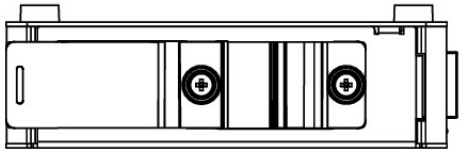
1.3 Dimension

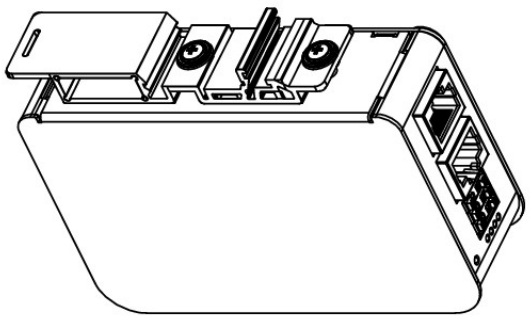










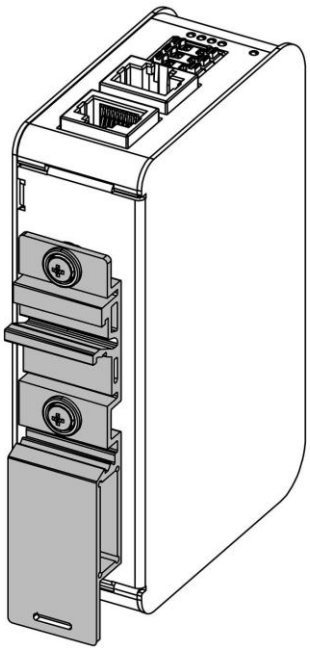


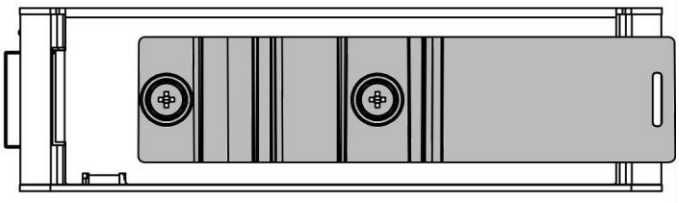
(Unit: mm)

1.4 Mounting Instruction

QEC-M-01 is an easy-install design to help you maintain your modules easily. Please refer to Ch.3.1 DIN-Rail installation.

- **DIN-Rail**





1.5 Ordering Information

Type	LCD size	(Below is the customization function, the unfilled fields do not need to be filled in; if the customer does not require, it will be directly shipped standard material number, such as QEC-M-01)							
		PoE	-	Feature		-	Wide Temp.	-	Coating
				Memory	Storage				
QEC-M	01	X		X	X		X		X

1. Type: Code 1~4

M: EtherCAT MDevice

2. LCD size: Code 5~6

01: without LCD

3. PoE: Code 7

P: RJ45 PoE Device, Red Plastic Housing

None or E: RJ45 w/o power, Black Plastic Housing

(Standard: None)

4. Feature: Code 8~9

X (Memory): G: 1G DDRIII / M: 512M DDRIII (Standard: 512MB)

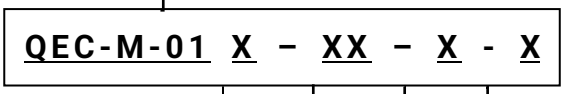
X (Storage): 1, 2, 4: eMMC size (Standard: 2G)

5. Wide Temp.: Code 10

X: -40 to +85°C / R: -20 to +70°C (Standard: -20 to +70°C)

6. Coating: Code 11

C: Yes / N: Normal



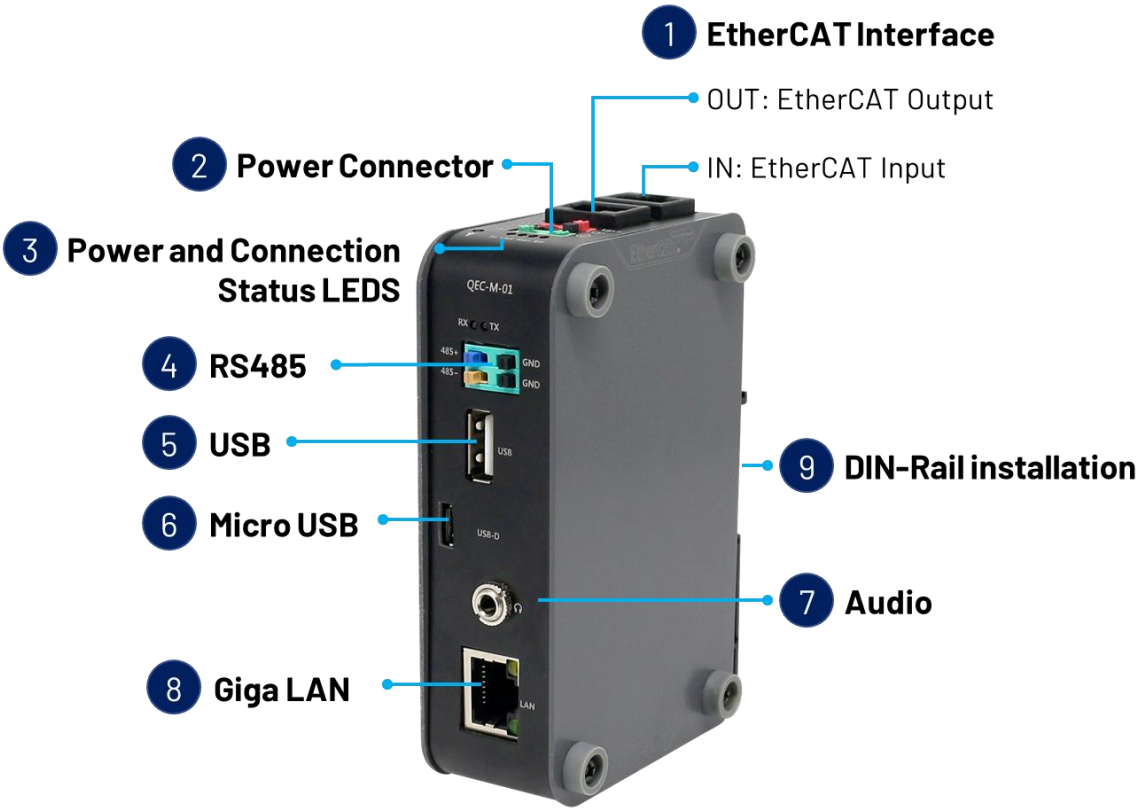
1.5.1 Ordering Part Number:

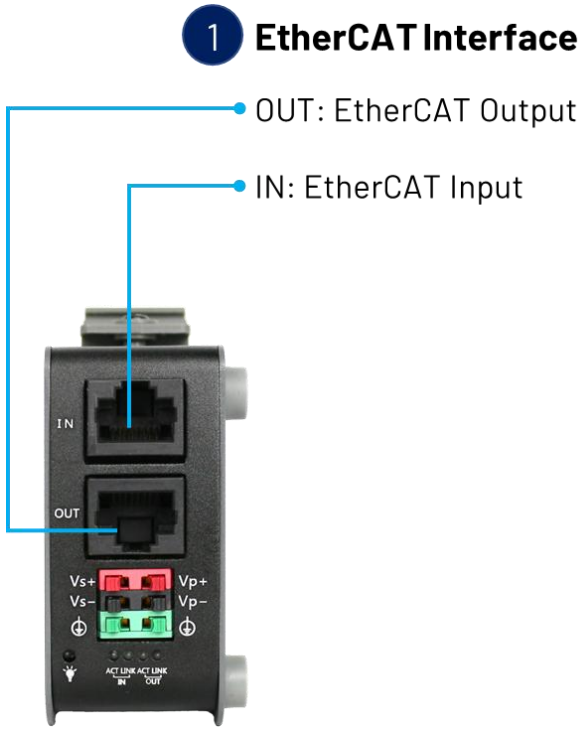
- **QEC-M-01**: Vortex86EX2 Processor 533MHz-based EtherCAT MDevice.
- **QEC-M-01P**: Vortex86EX2 Processor 533MHz-based EtherCAT MDevice.
- **QEC-M-01P-G4-X-C**: Vortex86EX2 Processor 533MHz-based EtherCAT MDevice /PoE/1G DDRIII Memory/4G eMMC Storage/Wide Temp./Coating

Ch. 2

Hardware System

2.1 General Technical Data



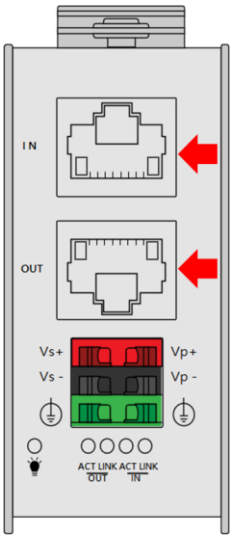


2.2 General Summary

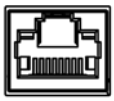
No.	Description		Type Narrative	Pin #
1	EtherCAT Interface	IN	External RJ45 Connector	8-pin
		OUT	(Gold finger)	8-pin
2	Power Connector		Terminal Block Interface	6-pin
3	Power and Connection Status LEDs		External Status LEDs	-
4	RS-485		Terminal Block Interface	4-pin
5	USB		Standard USB 2.0	-
6	Micro USB (Debug and Upload port)		Micro USB (Type-B)	-
7	Audio		HD Audio	-
8	Giga LAN		External RJ45 Connector (Gold finger)	8-pin
9	DIN-Rail		-	-

2.2.1 EtherCAT Interface

RJ45 Connectors.



EC IN

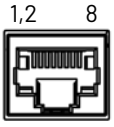


Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN with the Red Housing; Regular LAN with Black Housing.

* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

EC OUT



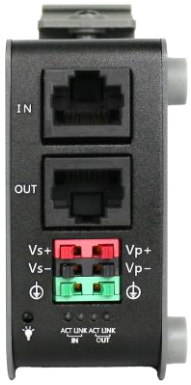
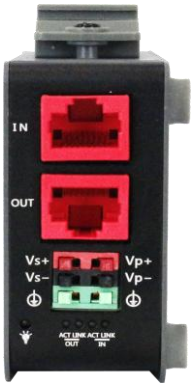
Pin #	Signal Name	Pin #	Signal Name
1	LAN2_TX+	2	LAN2_TX-
3	LAN2_RX+	4	VS+
5	VP+	6	LAN2_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN with the Red Housing; Regular LAN with Black Housing.

* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

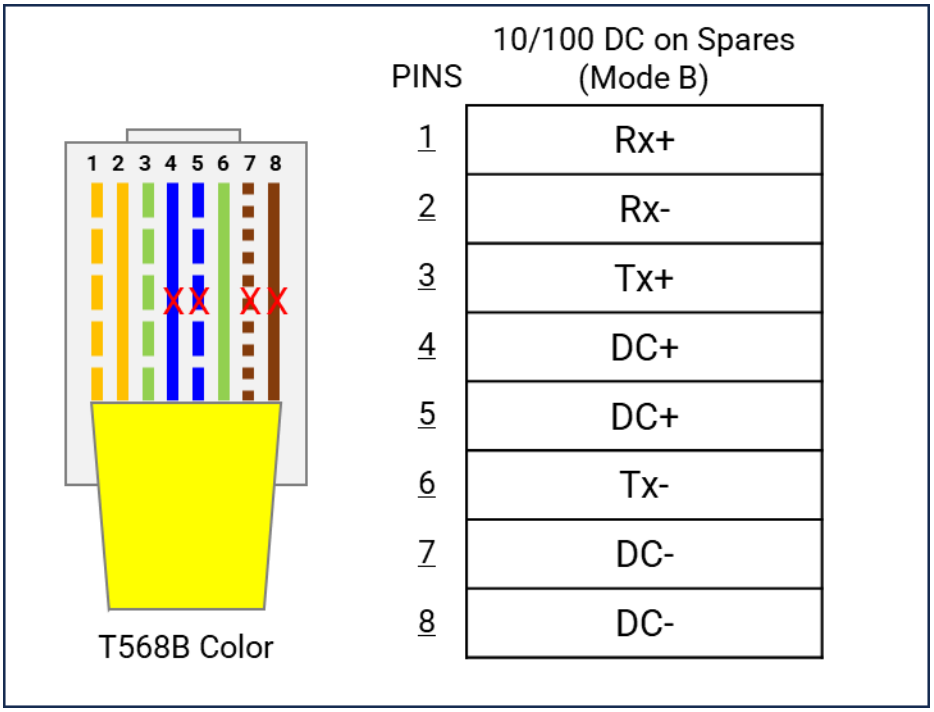
Note. QEC's PoE (Power over Ethernet)

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.

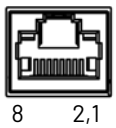
	
Non-PoE type	PoE type

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. 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).



2. 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:



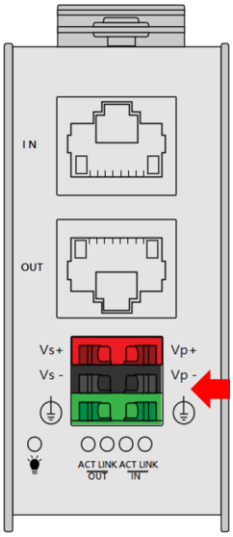
Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

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

2.2.2 Power Connector

Euroblock Connectors.
4-pins Power Input/Output & 2-pins FGND.



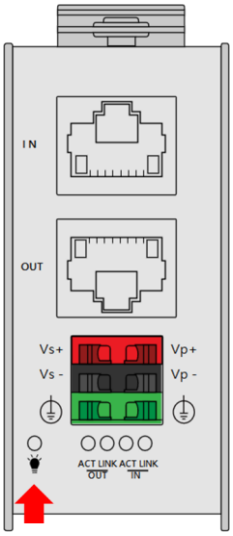
Vs for system power; Vp for peripheral power and backup power.

Pin #	Signal Name	Pin #	Signal Name
1	Vs+	2	Vp+
3	Vs- (GND)	4	Vp- (GND)
5	F.G	6	F.G

* Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)

2.2.3 Power and Connection Status LEDs

Power Status LED



Power input is 24V (typical). The LED status provide high/low voltage warning.

Notation	States	Condition	Description
PWR	Green LED On	Voltage <= 50V and >= 45V Voltage <= 26V and >= 19V	When Vs and Vp voltages are confirmed to be normal, the Green LED will remain steady on.
	Green LED On Red LED On	Voltage < 45V and > 26V Voltage < 19V and > 12V	LEDs will alternately flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.
	Orange LED On	Voltage > 50V or < 12V	Orange LED (Green + Red) will continuously flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.

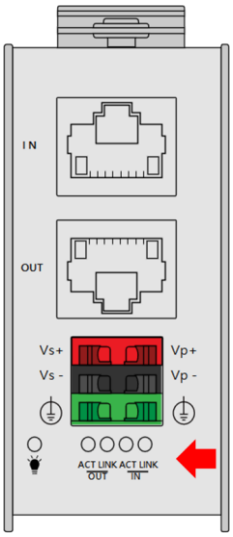
* Vs power status will be displayed first.

Power ERROR Code table (Red LED Flashing Display (2 seconds/cycle))：

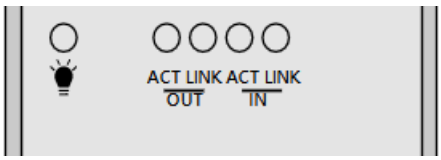
Long Light	Short Flash	Description
0 Long Light	After microchip completes the Bootloader test, it proceeds to the APP program stage.	
	1 short flash	Microchip communication with the EtherCAT chip failed.
	2 short flashes	EtherCAT chip internal RAM test failed.
	5 short flashes	Quartz oscillator on the board abnormality.
1 Long Light	Indicates the microchip Bootloader stage during startup, APP program not yet executed.	
	1 short flash	microchip internal SRAM failed.
	2 short flashes	APP software CHECKSUM failed.
2 Long Lights	Not yet defined.	

* Note: If you encounter any of the above abnormal states, please contact us.

Connection Status LEDs



There are two LED for each LAN ports.

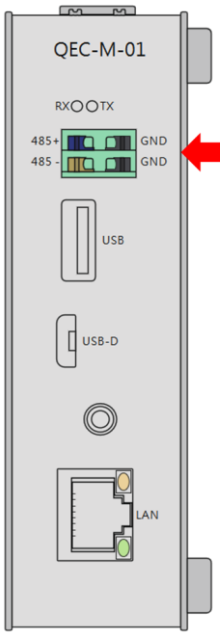


Please refer to the table below for the LAN port LED indications.

LAN	Notation	Color	States	Description
OUT	ACT	Green	Green LED Blinking	Data Activity
	LINK	Orange	Orange LED On	100Mbps Connection
IN	ACT	Green	Green LED Blinking	Data Activity
	LINK	Orange	Orange LED On	100Mbps Connection

2.2.4 RS-485

The QEC-M-01 provides a set of RS485+ and RS485- signal pins and two ground (GND) pins for the RS-485 communication interface. The RS-485 is located on the device's front panel (as shown in the diagram) and is labeled as 485+/-.



The pin functions are as follows:

Pin #	Signal Name	Pin #	Signal Name
2	RS485+	1	GND
4	RS485-	3	GND

Usage Guide:

- RS485+ (Pin 2): Positive signal for RS-485 communication.
- RS485- (Pin 4): Negative signal for RS-485 communication.
- GND (Pins 1, 3): Ground pins for reference.

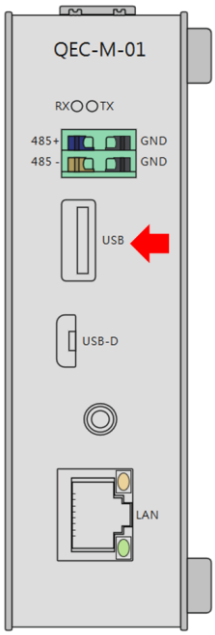
For configuration, please refer to the Ch 4.7 RS485 Communication Section.

2.2.5 USB

The QEC-M-01 has a Standard USB 2.0 port with hot-plug support. You can use this port to connect:

- USB Disk: For file storage, transfer, or accessing configuration data.

The USB port is located on the device's front panel (as shown in the diagram) and is labeled as USB.



Notes on USB Disk Usage:

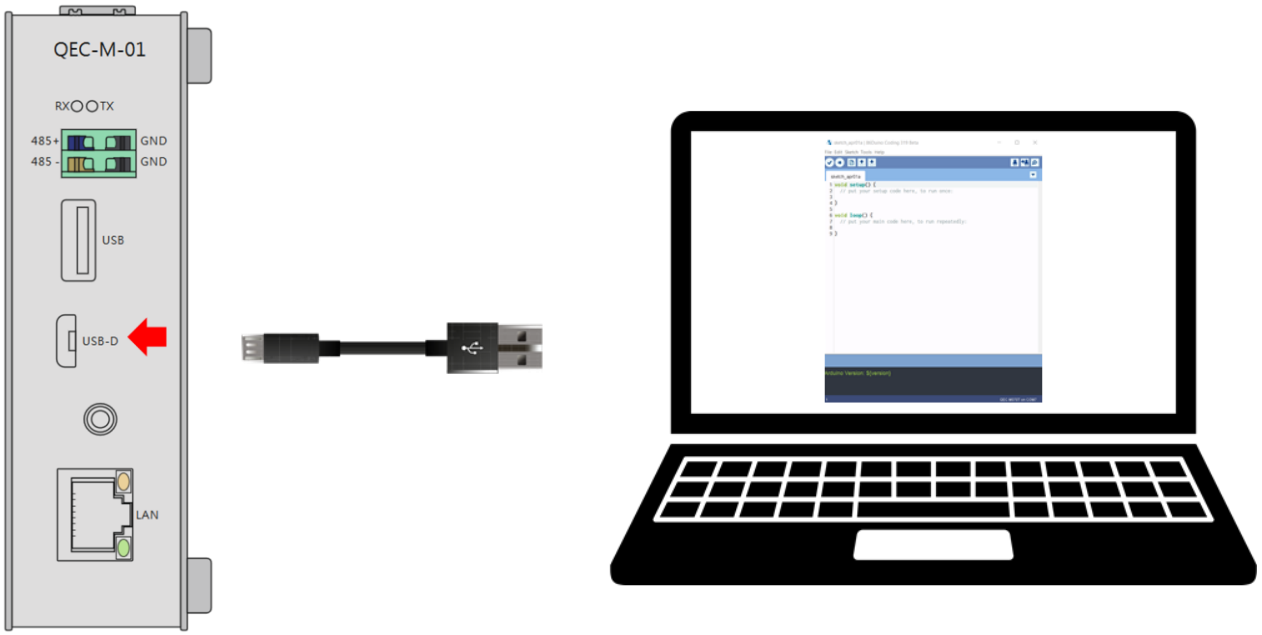
- Ensure the USB disk is formatted in a supported file system (e.g., FAT32 or NTFS).
- Large file transfers may be subject to USB 2.0 speed limitations.

For more details on using USB storage devices, please refer to the Ch 4.8 USB Device Usage Section.

2.2.6 Micro USB

The QEC-M-01 features a Micro USB port primarily used for programming uploads and debugging. You can connect the device to a computer using a USB to micro-USB cable to upload code and configure the system.

The Micro USB port is located on the device's front panel (as shown in the diagram) and is labeled as USB-D.

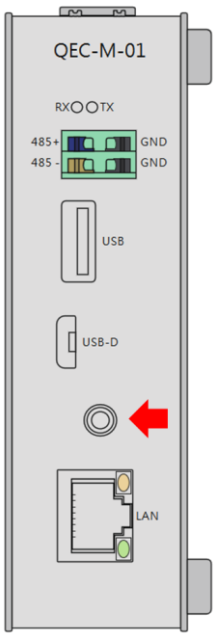


For instructions on programming and setting up the QEC-M-01, please refer to Chapter 4: Getting Started.

2.2.7 Audio

The QEC-M-01 features an HD Audio (Line-Out) port for connecting external audio devices such as speakers or headphones.

The HD Audio port is located on the device's front panel (as shown in the diagram).



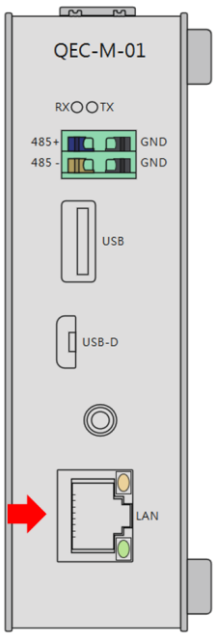
Compatible with standard 3.5mm audio jacks.

For more details on using USB storage devices, please refer to the Ch 4.10 Audio Usage Section.

2.2.8 Giga LAN

The QEC-M-01 features one Giga LAN port and is dedicated to external Ethernet communication for general network use.

The Giga LAN port is located on the device's front panel (as shown in the diagram).



Pin Definitions:

Pin #	Signal Name	Pin #	Signal Name
L1	GTX+	L2	GTX-
L3	GRX+	L4	GTXC+
L5	GTXC-	L6	GRX-
L7	GRXD+	L8	GRXD-

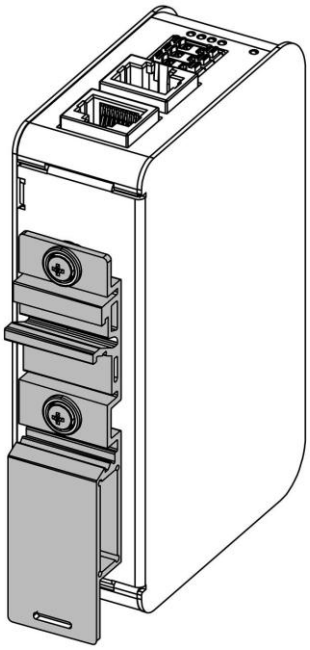
The Giga LAN port supports high-speed Ethernet for external communication. To drive GigaLAN, you can refer to Ethernet library or Modbus Library.

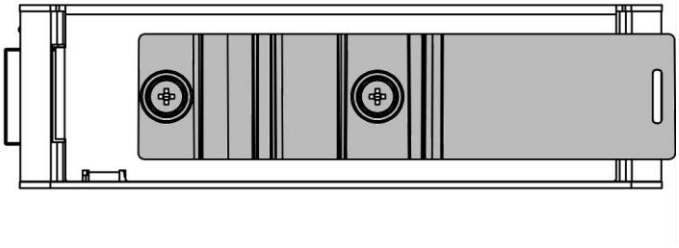
For more details on using USB storage devices, please refer to the Ch 4.9 Giga LAN Configuration Section.

2.2.9 DIN-Rail installation

QEC-M-01 is an easy-install design to help you maintain your modules easily. Please refer to Ch.3.1 DIN-Rail installation.

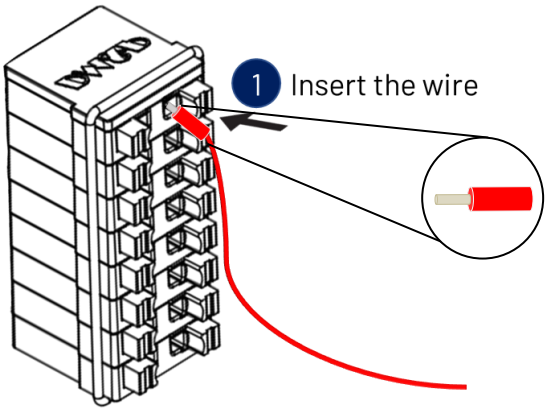
- **DIN-Rail**

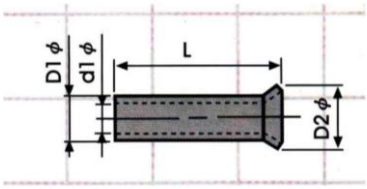




2.3 Wiring to the Connector

2.3.1 Connecting the wire to the connector

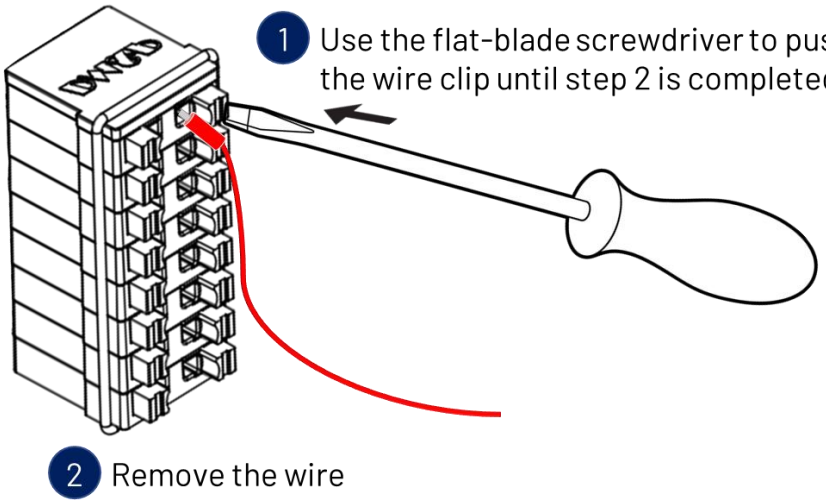




Insulated Terminals Dimensions (mm)

Position	L	ØD1	Ød1	ØD2
CN 0.5-6	6.0	1.3	1.0	1.9
CN 0.5-8	8.0	1.3	1.0	1.9
CN 0.5-10	10.0	1.3	1.0	1.9

2.3.2 Removing the wire from the connector



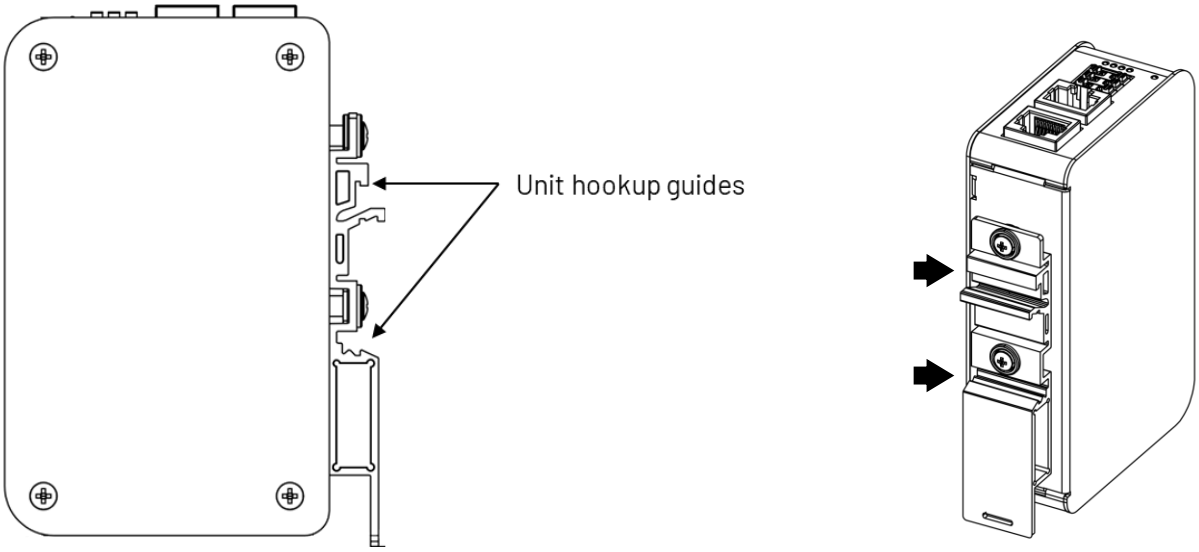
Ch. 3

Hardware Installation

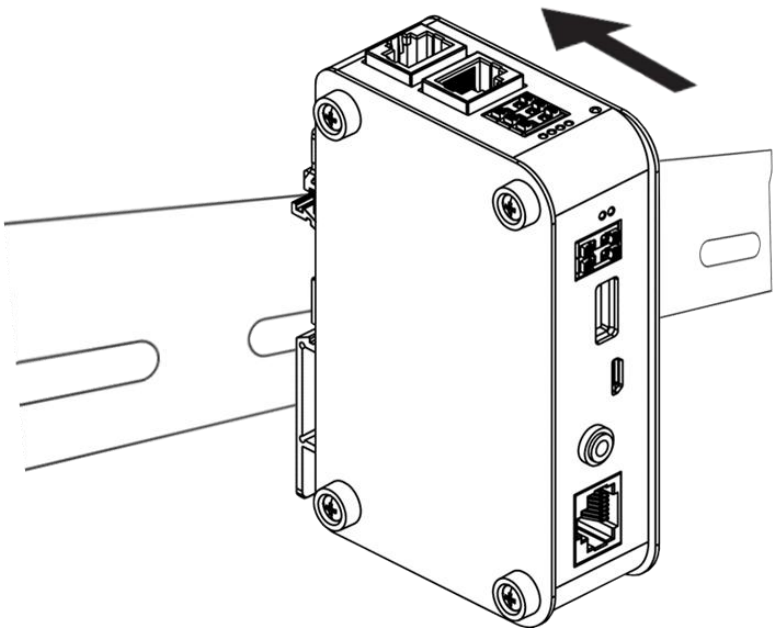
This section describes how to install QEC-M-01. Please turn OFF the power supply before you mount QEC-M-01. Always mount QEC-M-01 one at a time.

3.1 DIN-Rail installation

Slide in the QEC-M-01 on the hookup guides and press the QEC-M-01 with a certain amount of force against the DIN track until the DIN track mounting hook lock into place.



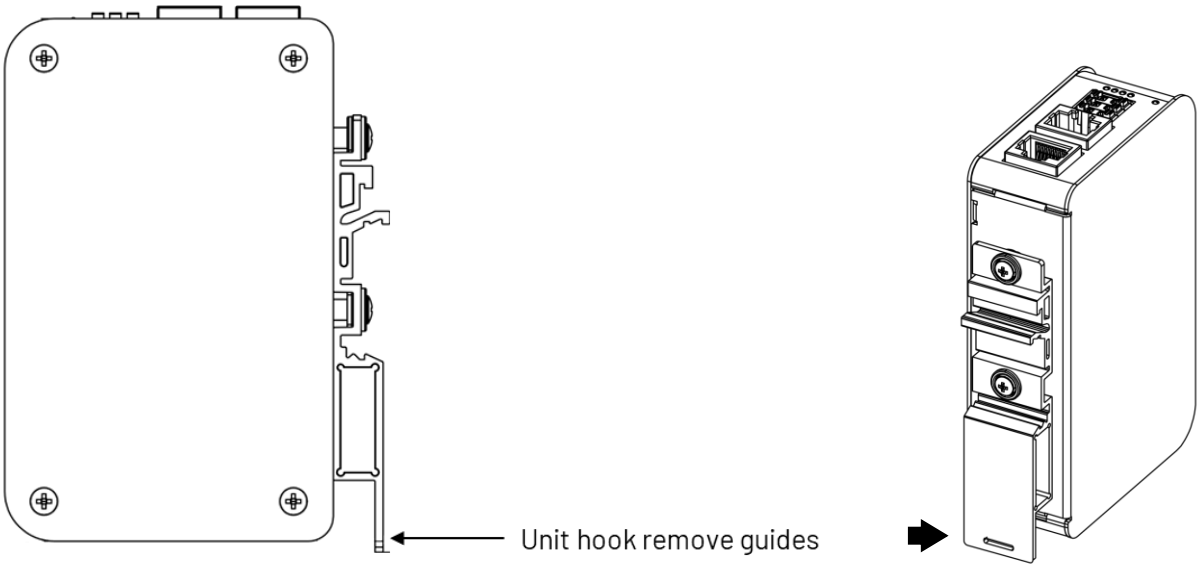
When you mount the QEC-M-01, releasing the DIN track mounting hook on the QEC-M-01 is unnecessary. After you mount the QEC-M-01, make sure it is locked to the DIN track.



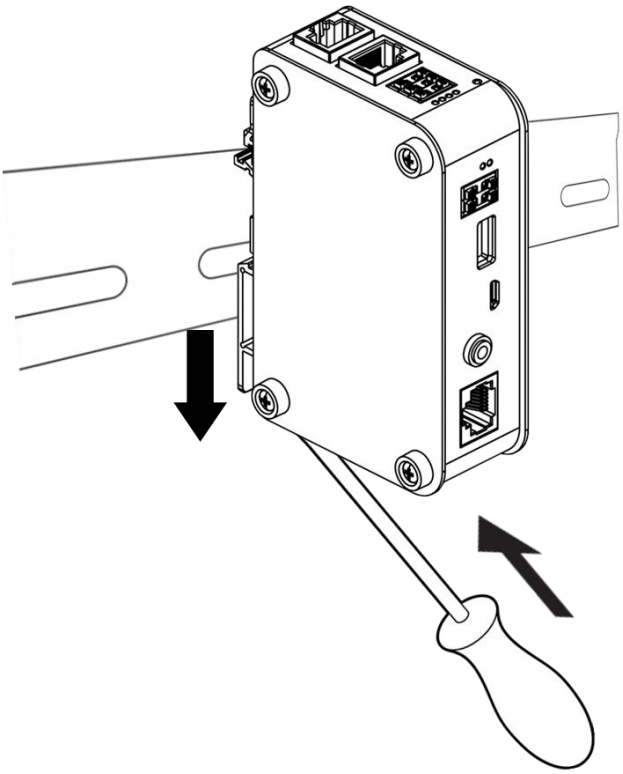
*Note: Always turn OFF the Unit power supply before connecting and removing the QEC-M-01.

3.2 Removing QEC-M-01 Unit

Use a flat-blade screwdriver to remove the DIN Track mounting hook on the unit.



Pull down the flat-blade screwdriver against the DIN track until the mounting hook being removed from the track.



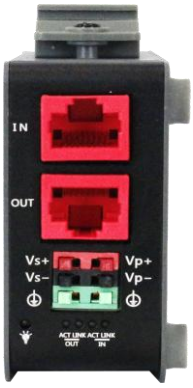
Ch. 4

Getting Started

This chapter explains how to start with QEC-M-01 and its software, 86Duino Coding IDE.

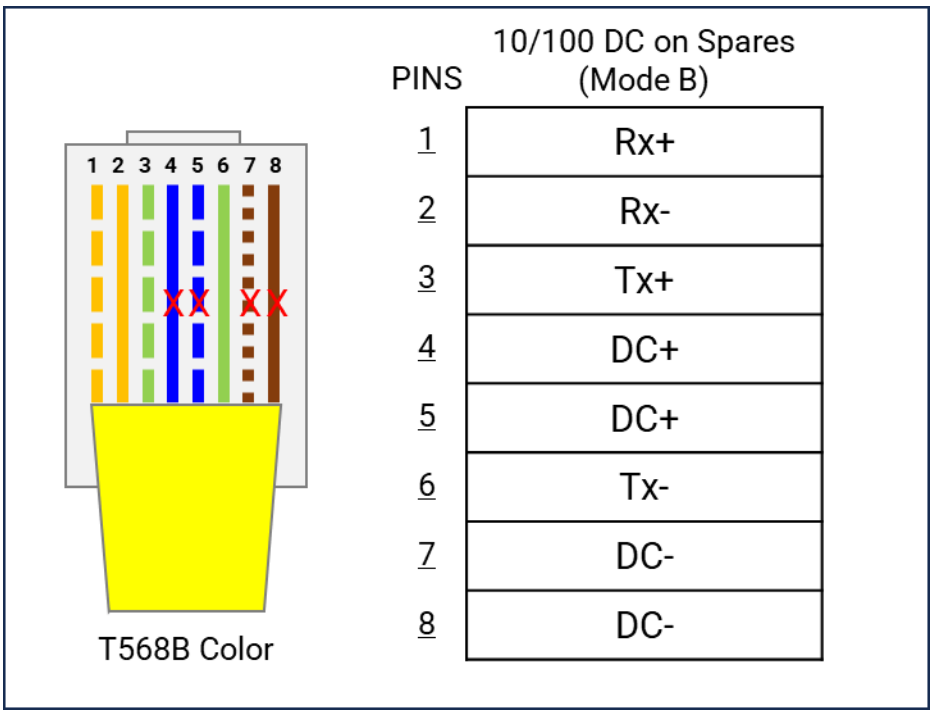
Note. QEC's PoE (Power over Ethernet)

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.

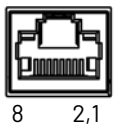
	
Non-PoE type	PoE type

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:

4. 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).



5. 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:



Pin #	Signal Name	Pin #	Signal Name
1	LAN1_TX+	2	LAN1_TX-
3	LAN1_RX+	4	VS+
5	VP+	6	LAN1_RX-
7	VS- (GND)	8	VP- (GND)

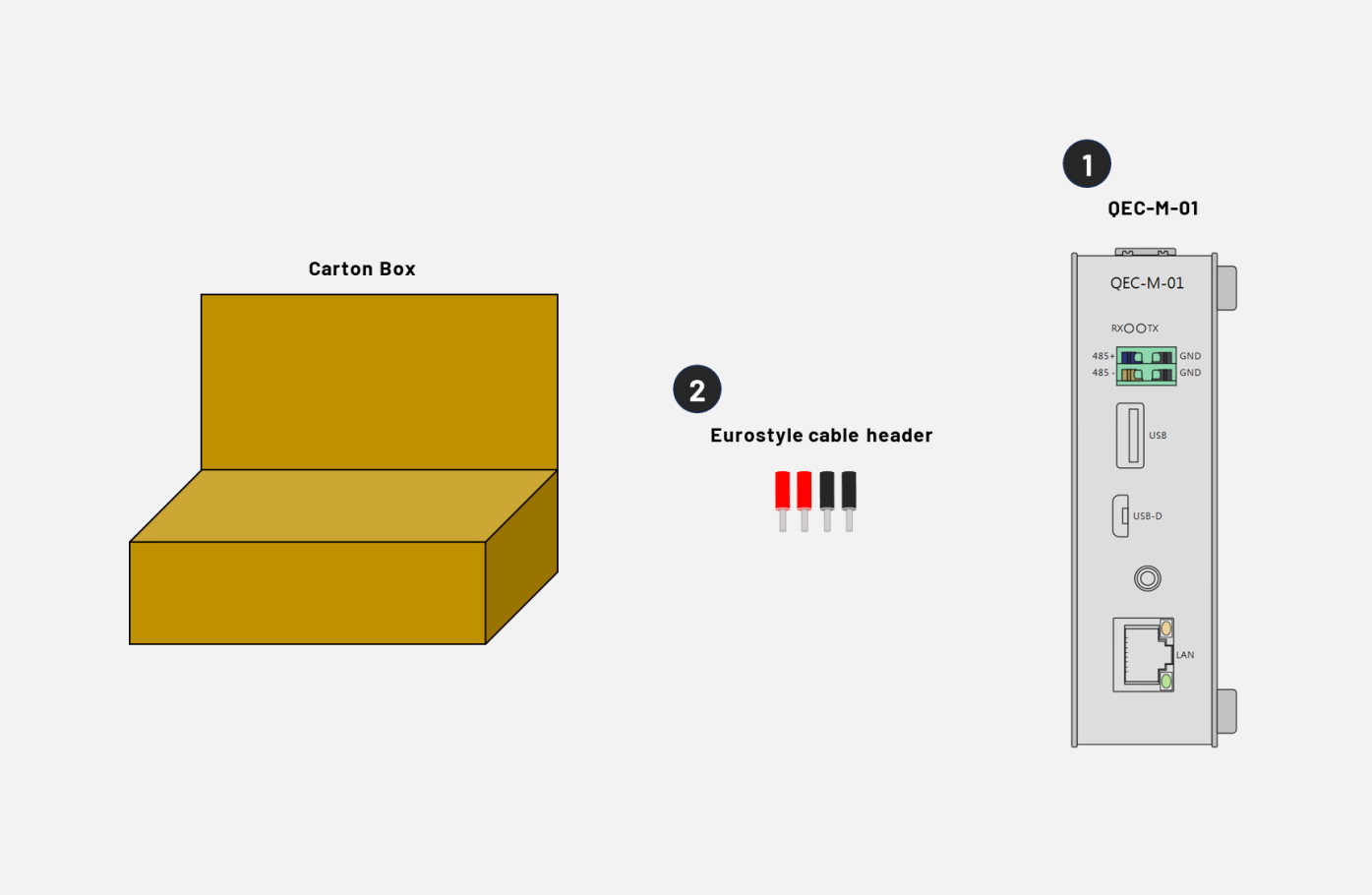
* PoE LAN with the Red Housing; Regular LAN with Black Housing.
* L4, L5, L7, L8 pins are option, for RJ45 Power IN/OUT.

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

4.1 Package Contents

The package includes the following items:

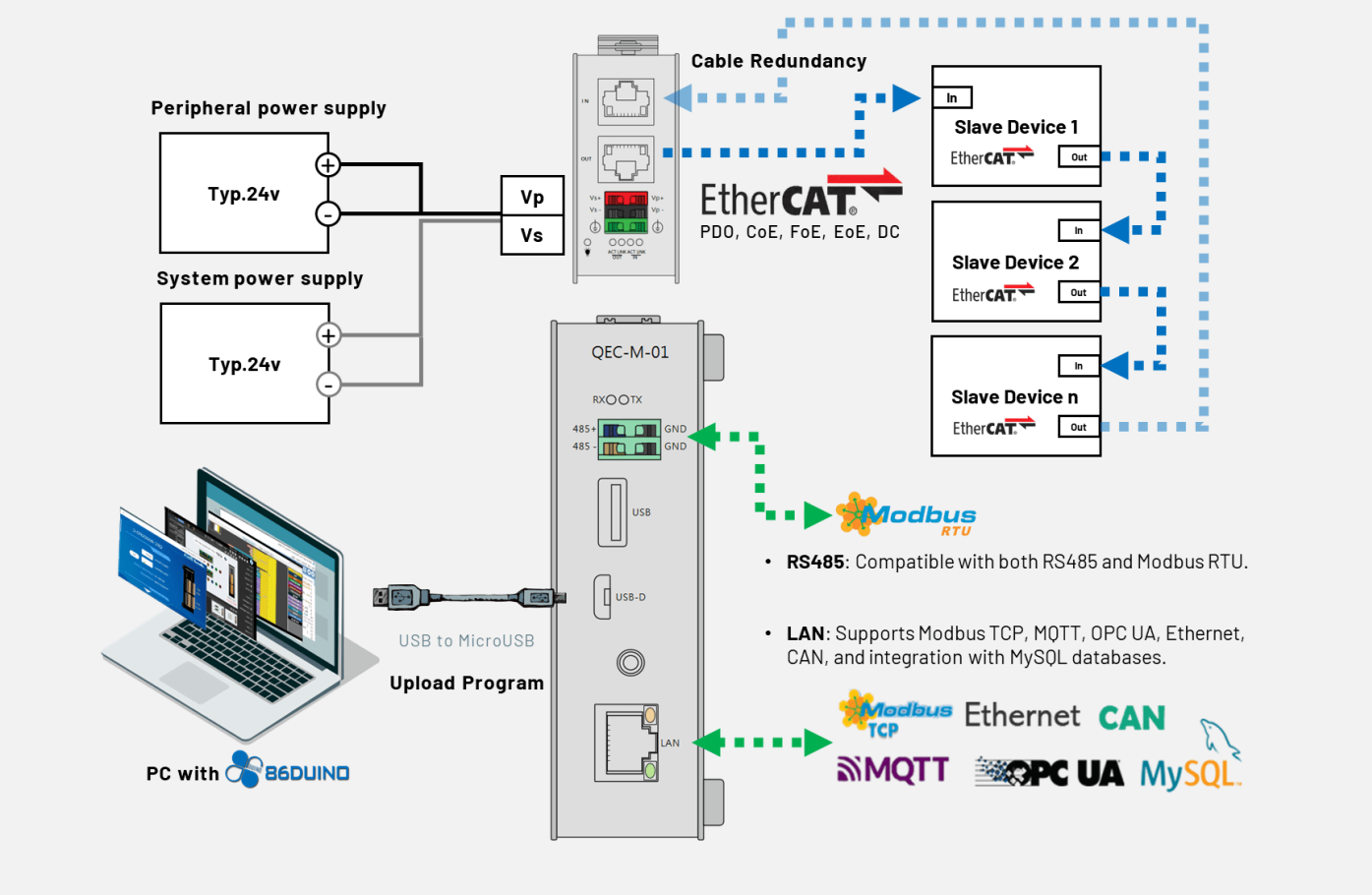
1. QEC-M-01
2. Cable-set (Euro-type connector)



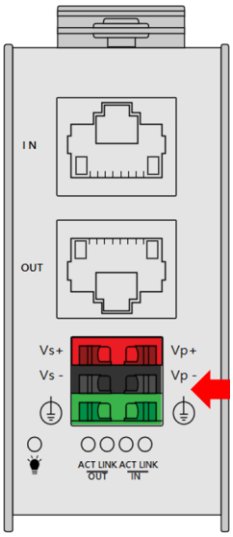
Please get in touch with our sales channels if any of the package items are missing or damaged. Also, feel free to reuse the shipping materials and carton for further storing and shipping needs in the future,

4.2 Hardware Configuration

The development environment will be pre-installed before the QEC-M is shipped to customers. Users must download the software (see 4.3 Software/Development Environment) and follow this user manual to set up the device.

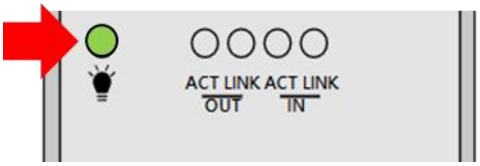


4.2.1 Plug in the power supply



There are two groups of power supplies in QEC-M-01, Vs and Vp. The voltage requirement for both supplies' ranges from 19V to 50V wide voltage.

After powering it on, the power LED lights up (green).



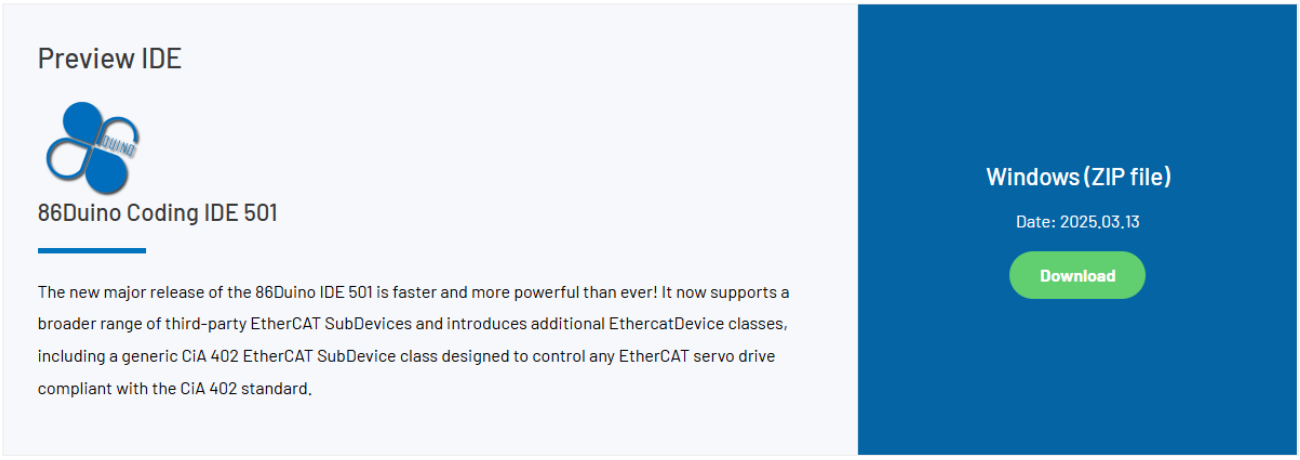
Power input is 24V (typical). The LED status provide high/low voltage warning.

Notation	States	Condition	Description
PWR	Green LED On	Voltage <= 50V and >= 45V Voltage <= 26V and >= 19V	When Vs and Vp voltages are confirmed to be normal, the Green LED will remain steady on.
	Green LED On Red LED On	Voltage < 45V and > 26V Voltage < 19V and > 12V	LEDs will alternately flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.
	Orange LED On	Voltage > 50V or < 12V	Orange LED (Green + Red) will continuously flash (at 0.3-second intervals) until the Vs and Vp voltages are correct.

* Vs power status will be displayed first.

4.3 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.

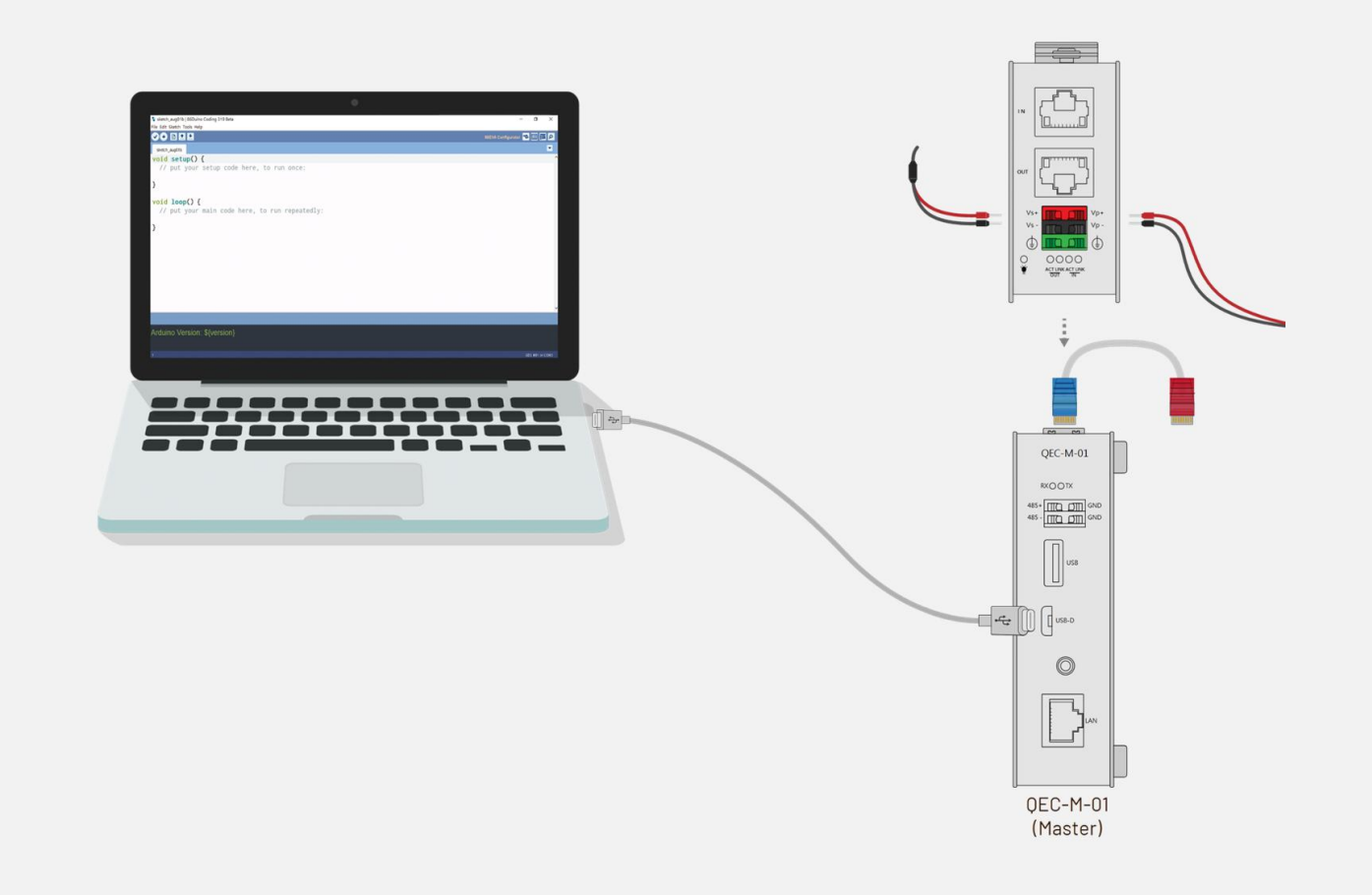
* Note: If Windows displays a warning, click Details and then click the Continue Run button once.

86Duino Coding IDE 501+ looks like below.

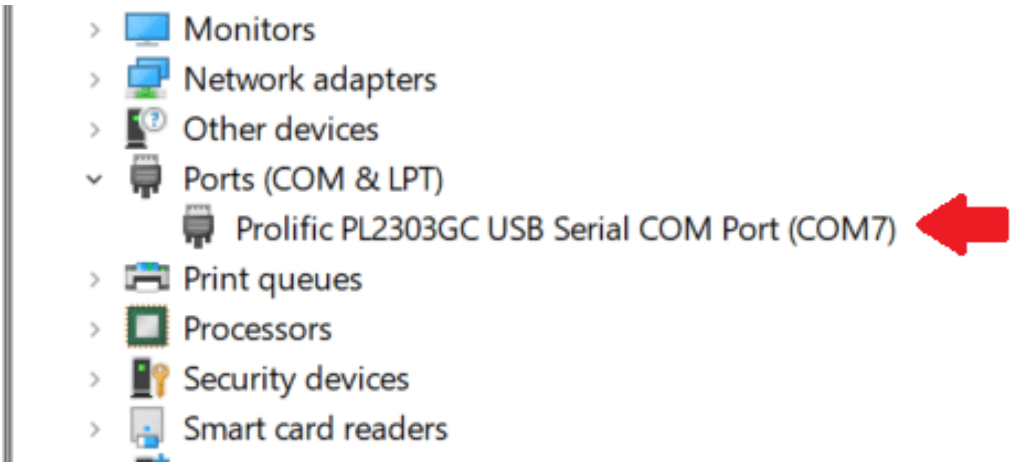
4.4 Connect to your PC and set up the environment

Follow the steps below to set up the environment:

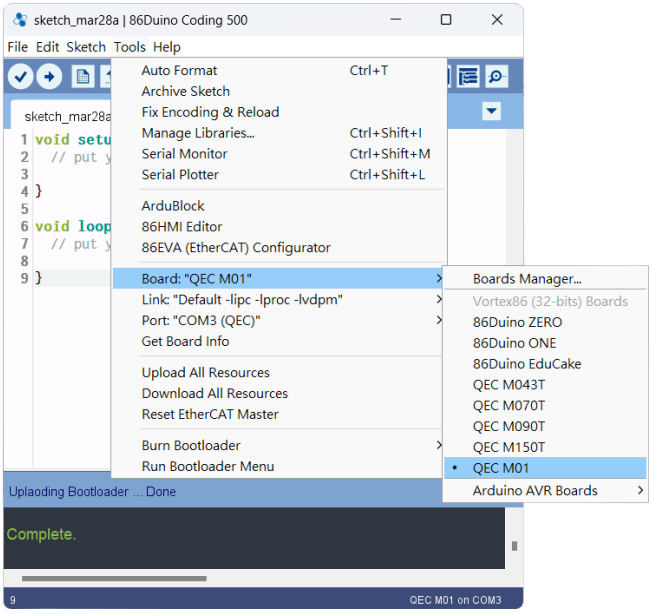
1. Connect the QEC-M-01 to your PC via a Micro USB to USB cable (86Duino IDE installed).



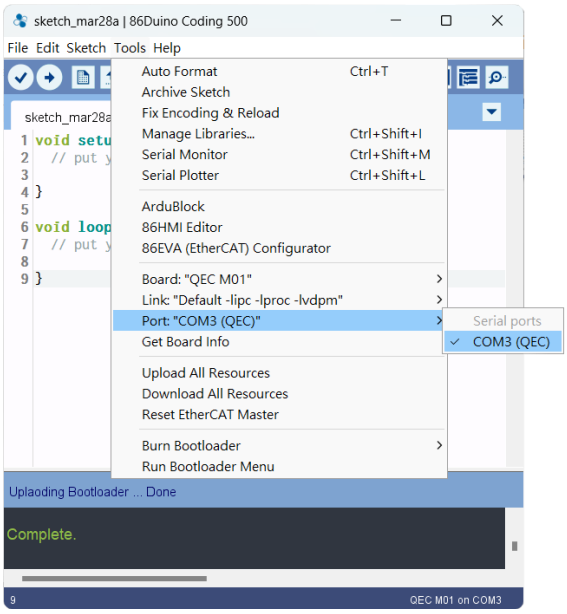
2. Turn on the QEC power.
3. Open “Device Manager” -> “Ports (COM & LPT)” in your PC and expand the ports; you should see that the “Prolific PL2303GC USB Serial COM Port (COMx)” is detected; if not, you will need to install the required drivers.
   (For Windows PL2303 driver, you can download here)



4. Open the 86Duino IDE.
5. Select the correct board: In the IDE's menu, select “Tools” ->” Board”- > QEC-M-01 (or the QEC-M 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-M MDevice (in this case, COM3 (QEC)).



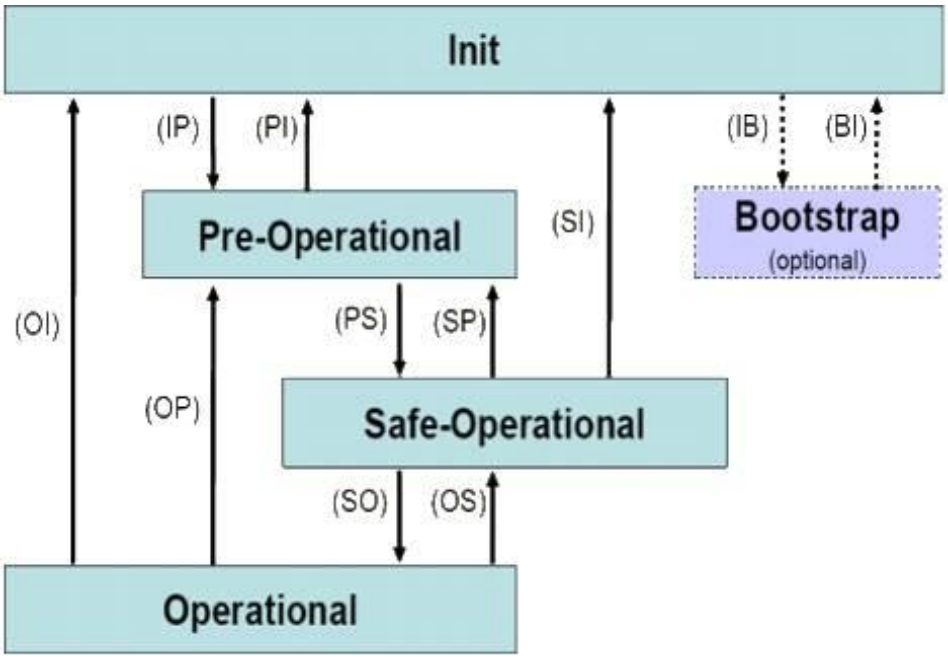
4.5 EtherCAT Communication

This section introduces two primary methods to configure your EtherCAT Slave devices through the QEC EtherCAT MDevice: Write code and Use 86EVA with code. Both methods are designed to offer flexibility and efficiency depending on your familiarity and requirements.

4.5.1 EtherCAT State Machine (ESM)

To set up and transition EtherCAT Slave devices through various operational states using the QEC EtherCAT MDevice. It is crucial for understanding the state transitions and operational flow within an EtherCAT network.

The state of the EtherCAT slave is controlled via the EtherCAT State Machine (ESM). Depending upon the state, different functions are accessible or executable in the EtherCAT slave. Specific commands must be sent by the EtherCAT MDevice to the device in each state, particularly during the bootup of the slave.



A distinction is made between the following states:

- Init
- Pre-Operational
- Safe-Operational and
- Operational
- Boot

The regular state of each EtherCAT slave after bootup is the OP state.

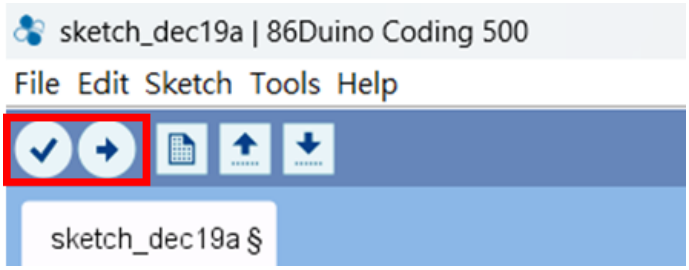
The EtherCAT MDevice (QEC-M-01) can be configured in the EtherCAT network via the EtherCAT library and programmed with the control action in the 86Duino IDE. The 86Duino development environment has two main parts: setup() and loop(), which correspond to initialization and main programs.

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

}

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

}
```

Before operating the EtherCAT network, you must configure it once. The process should be from Init to OP state in EtherCAT devices.

To implement EtherCAT communication, users must use the EtherCAT Library of 86Duino Coding IDE 500+. For detailed information, please refer to Ch.5 Software Function.

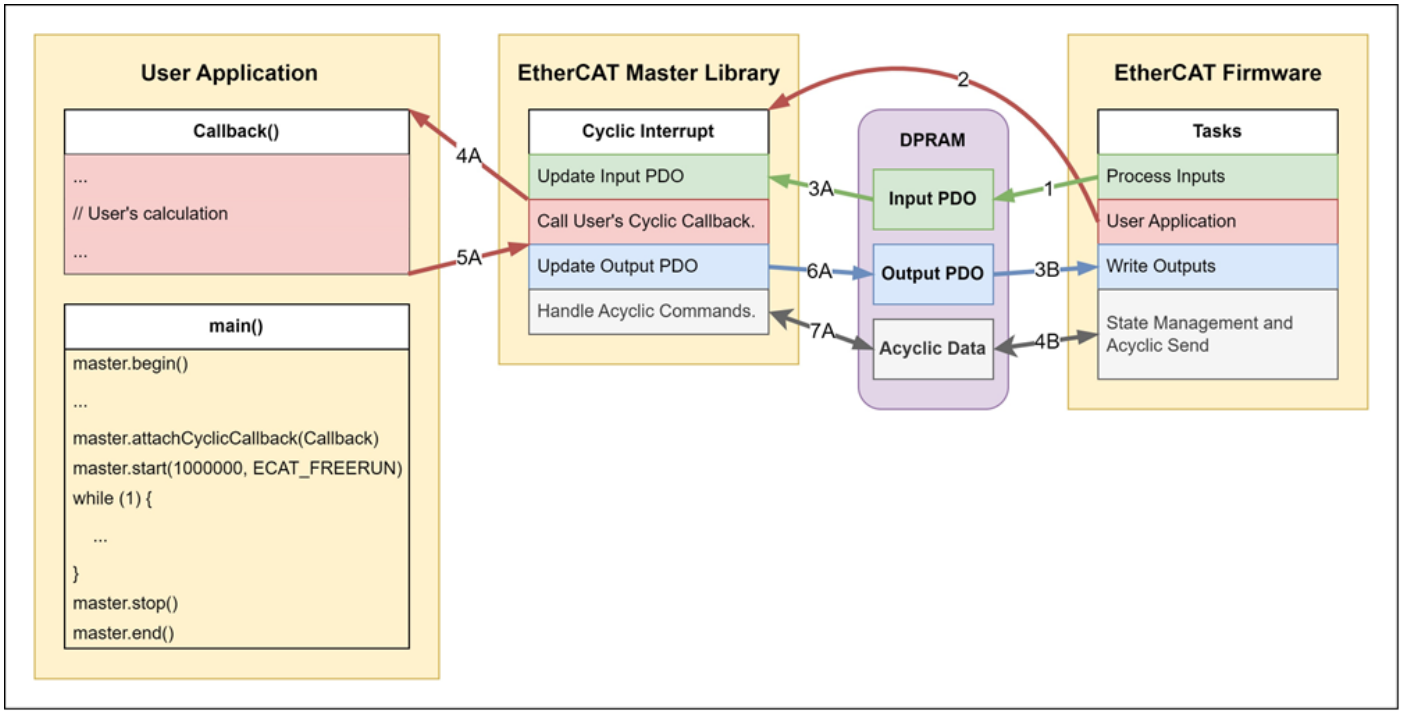
*Note: Once the code is written, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload.



Example 1: Free Run Mode

In "Free Run" mode the local cycle is triggered through a local timer interrupt of the application controller. The cycle time can be modified by the master (optional) in order to change the timer interrupt. In "Free Run" mode the local cycle operates independent of the communication cycle and/or the master cycle.

The Free Run Mode diagram for QEC MDevice:



This is the free-run mode without dual-system synchronization. As shown in the diagram, the numbered arrows indicate the sequence of operations. However, a branching occurs at step 3 because, after the EtherCAT firmware triggers a cyclic interrupt to the EtherCAT MDevice library (step 2), it does not wait for the EtherCAT MDevice library and directly continues with the next action. The two systems operate independently, with no synchronization. If the user has registered a cyclic callback, the cyclic interrupt will call it, as shown in step 4A.

Here is the example code:

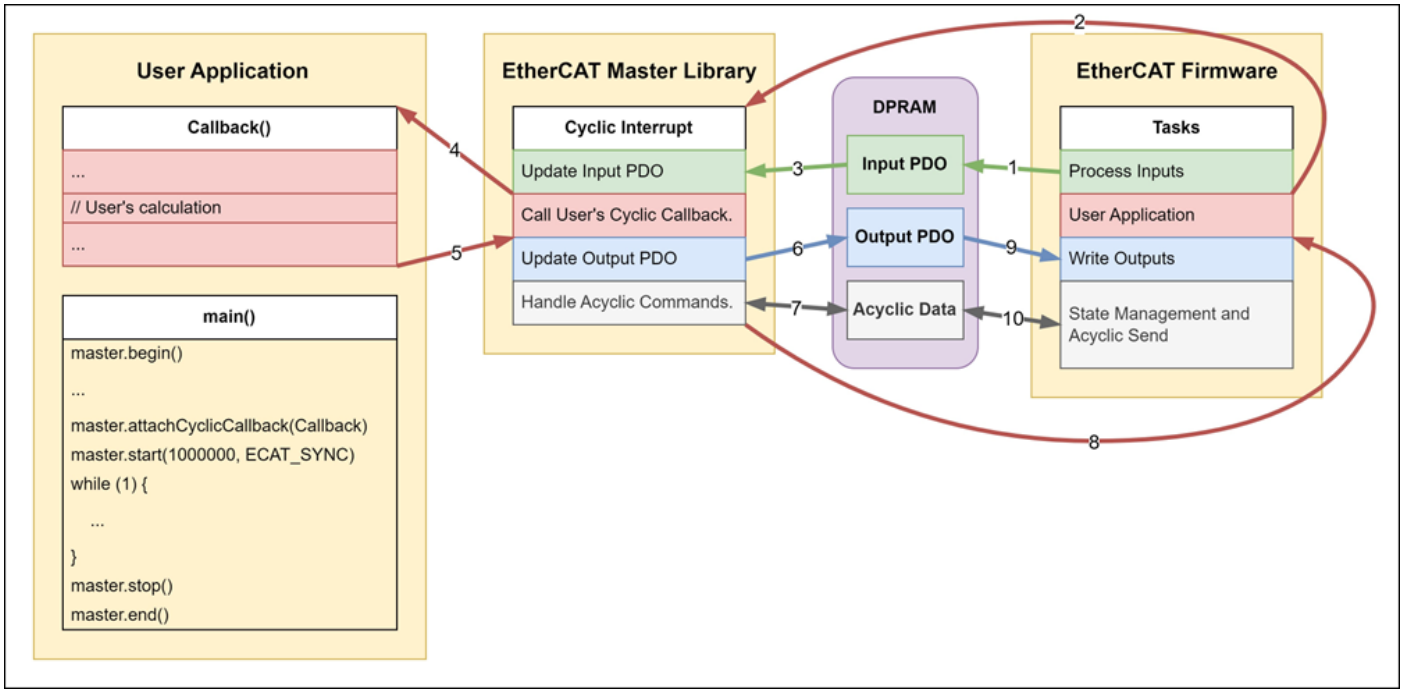
```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

void setup() {
  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start(1000000, ECAT_FREERUN);
}

void loop() {
  // ...
}
```

Example 2: SYNC Mode

The local cycle is started when the SM2 event [with cyclical outputs] or the SM3 event [without cyclical outputs] is received. If the outputs are available, the slave is generally synchronized with the SM2 event. If no outputs are available, the slave is synchronized with the SM3 event, e.g. for cyclical inputs.
In this mode the following options are available:

- Synchronous with SM2/3 event
- Synchronous with SM2/3 event, shifting of the "Input Latch" time

The SYNC Mode diagram for QEC MDevice:



This mode offers the highest level of dual-system synchronization. As shown in the diagram, the numbered arrows indicate the sequence of operations, with no branching present. After the EtherCAT firmware triggers a cyclic interrupt to the EtherCAT MDevice library (step 2), it waits for an ACK response from the EtherCAT MDevice library (step 8) before proceeding with the next action. If the user has registered a cyclic callback, the cyclic interrupt will call it, as shown in step 4. As long as the user reads the current input process data within the cyclic callback, processes it, calculates the output process data, and writes it back, the current cycle will send the output process data to the EtherCAT network, fulfilling the requirements of real-time control systems.

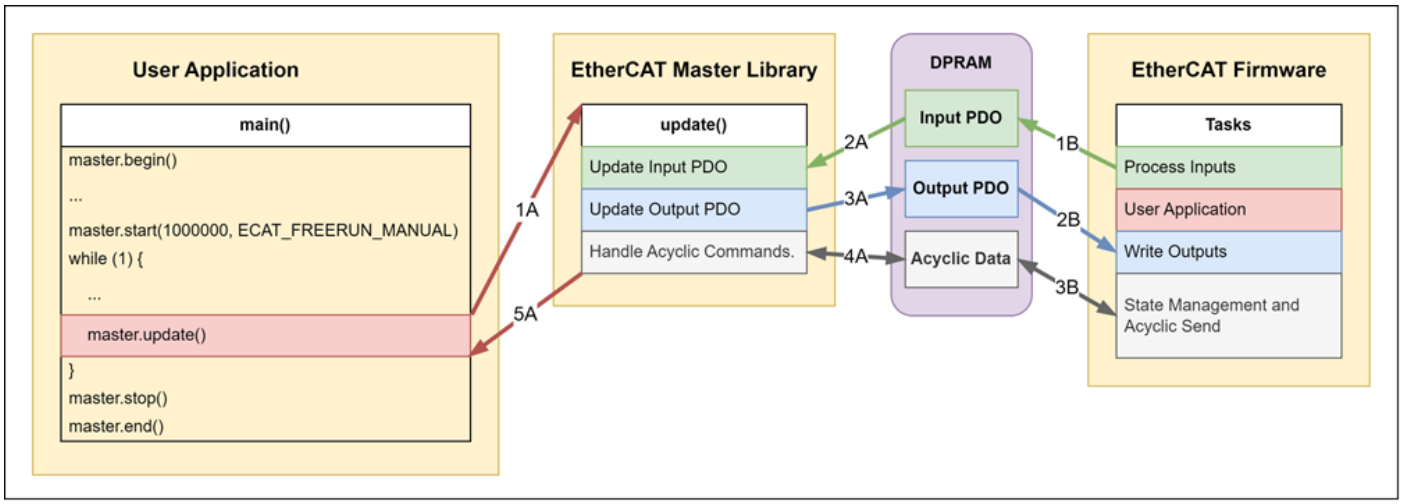
Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

void setup() {
  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start(1000000, ECAT_SYNC);
}

void loop() {
  // ...
}
```

Example 3: Free Run Manual Mode

This is also a free-run mode without dual-system synchronization. The primary difference from the ECAT_FREERUN mode is that there is no cyclic interrupt to update process data and handle acyclic commands. Instead, the user must manually call EthercatMaster::update() to update process data and handle acyclic commands. Additionally, since there is no cyclic interrupt in this mode, the cyclic callback will not be called. As indicated by the numbered arrows in the diagram, the two systems operate independently, with no synchronization.

The Free Run Manual Mode diagram for QEC MDevice:



Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

void setup() {
  master.begin();
  master.start(1000000, ECAT_FREERUN_MANUAL);
}

void loop() {
  // ...
  master.update();
}
```

4.5.2 SubDevice Information

The QEC MDevice EtherCAT library provides functions to obtain information about EtherCAT SubDevices on the network. These include querying the number of slaves on the network, retrieving a SubDevice's Vendor ID, Product Code, and Alias Address by its sequence number, and reverse querying the slave number using the information above. This information is used to identify the type of slave device and choose the appropriate EtherCAT SubDevices class to attach.

Example 1: Using EthercatMaster class

Show SubDevice information using EthercatMaster class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;

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

  master.begin(); // Initialize EtherCAT Master in Pre-OP state

  Serial.println("Starting EtherCAT Master...");

  // Print Out All Slave Information
  for (int i = 0; i < master.getSlaveCount(); i++) {
    Serial.print("Slave ");
    Serial.print(i);
    Serial.print(" VID: ");
    Serial.print(master.getVendorID(i), HEX);
    Serial.print(", PID: ");
    Serial.print(master.getProductCode(i), HEX);
    Serial.print(", Rev: ");
    Serial.print(master.getRevisionNumber(i), HEX);
    Serial.print(", Ser: ");
    Serial.print(master.getSerialNumber(i), HEX);
    Serial.print(", Alias: ");
    Serial.print(master.getAliasAddress(i));
```

```
    Serial.println();
  }
}

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

}
```

Example 2: Using EthercatDevice_Generic class

Show SubDevice information using EthercatDevice_Generic class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

char name[256];

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

  master.begin(); // Initialize EtherCAT Master in Pre-Operational state

  for (int i = 0; i < master.getSlaveCount(); i++) {
    slave.attach(i, master); // Attach the slave to the master
    Serial.print("Slave ");
    Serial.println(i);

    Serial.print("               Name: ");
    Serial.println(slave.getDeviceName(name, 256));

    Serial.print("          Vendor ID: 0x");
    Serial.println(slave.getVendorID(), HEX);

    Serial.print("       Product Code: 0x");
    Serial.println(slave.getProductCode(), HEX);

    Serial.print("    Revision Number: 0x");
    Serial.println(slave.getRevisionNumber(), HEX);

    Serial.print("      Serial Number: 0x");
    Serial.println(slave.getSerialNumber(), HEX);

    Serial.print("      Alias Address: ");
    Serial.println(slave.getAliasAddress());
```

```
    Serial.print("    Mailbox Protocol: 0x");
    Serial.println(slave.getMailboxProtocol(), HEX);

    Serial.print("        CoE Details: 0x");
    Serial.println(slave.getCoEDetails(), HEX);

    Serial.print("        FoE Details: 0x");
    Serial.println(slave.getFoEDetails(), HEX);

    Serial.print("        EoE Details: 0x");
    Serial.println(slave.getEoEDetails(), HEX);

    Serial.print("       SoE Channels: ");
    Serial.println(slave.getSoEChannels());

    Serial.print("       DC Supported: ");
    Serial.println(slave.isSupportDC());
  }
}

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

}
```

Example 3: Using EthercatDevice_CiA402 class

Show SubDevice information using EthercatDevice_CiA402 class.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 slave;

char name[256];

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

  master.begin();  // Initialize EtherCAT Master in Pre-Operational state

  for (int i = 0; i < master.getSlaveCount(); i++) {
    // Attach the slave to the master
    if (slave.attach(i, master) < 0) {
      continue; // Skip this slave if attachment fails
    }

    Serial.print("Slave ");
    Serial.println(i);

    Serial.print("               Name: ");
    Serial.println(slave.getDeviceName(name, 256));

    Serial.print("          Vendor ID: 0x");
    Serial.println(slave.getVendorID(), HEX);

    Serial.print("       Product Code: 0x");
    Serial.println(slave.getProductCode(), HEX);

    Serial.print("    Revision Number: 0x");
    Serial.println(slave.getRevisionNumber(), HEX);

    Serial.print("      Serial Number: 0x");
    Serial.println(slave.getSerialNumber(), HEX);
```

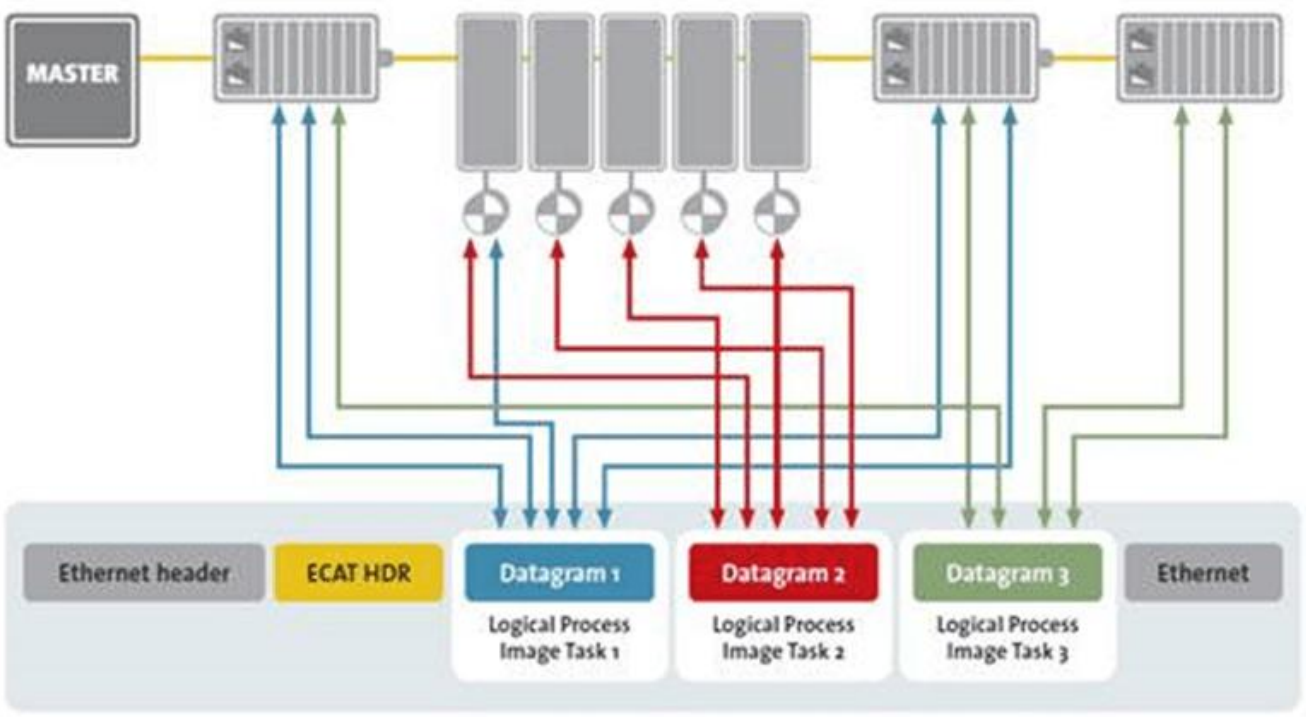
```
    Serial.print("      Alias Address: ");
    Serial.println(slave.getAliasAddress());

    Serial.print("    Mailbox Protocol: 0x");
    Serial.println(slave.getMailboxProtocol(), HEX);

    Serial.print("        CoE Details: 0x");
    Serial.println(slave.getCoEDetails(), HEX);

    Serial.print("        FoE Details: 0x");
    Serial.println(slave.getFoEDetails(), HEX);

    Serial.print("        EoE Details: 0x");
    Serial.println(slave.getEoEDetails(), HEX);

    Serial.print("       SoE Channels: ");
    Serial.println(slave.getSoEChannels());

    Serial.print("       DC Supported: ");
    Serial.println(slave.isSupportDC());
  }
}

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

}
```

4.5.3 Process Data Objects (PDO)

Process Data refers to real-time communication data exchanged between the MDevice and SubDevice in an EtherCAT network. This data includes information used for control, monitoring, and communication purposes. The EtherCAT MDevice cyclically transmits process data to control and monitor all slaves, ensuring high synchronization and low latency.

The Fieldbus Memory Management Units (FMMU) in the EtherCAT SubDevice Controller (ESC) can mapping dual-port memory to logical address. All slave nodes check the EtherCAT frames sent by the EtherCAT MDevice, comparing the logical address of the process data with the configured address in the FMMU. If a match is found, the output process data is transferred to dual-port memory, and the input process data is inserted into the EtherCAT frame.

Overall, process data is an essential part of EtherCAT technology and is suitable for real-time applications in robot control, CNC control, automation control, and other fields.



EtherCAT: Exchange of internet packets and data. (Source of information: http://www.ethercat.org/)

Example 1: Read a bit data from Input PDO

Read a bit from Input PDO using pdoBitRead().
A 16-channel digital input EtherCAT SubDevice has 2-byte Input PDOs, with each bit corresponding to a digital input channel. The states of channels 0 and 9 will be printed with a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    Serial.begin(115200); // Initialize Serial Monitor
    while (!Serial);      // Wait for Serial Monitor to be ready

    master.begin();       // Initialize EtherCAT Master
    slave.attach(0, master); // Attach the first EtherCAT slave to the master
    master.start();       // Start EtherCAT communication
}

void loop() {
    // Read and display the state of PDO bits
    Serial.print("Bit0 => ");
    Serial.print(slave.pdoBitRead(0)); // Read PDO bit 0
    Serial.print(", Bit9 => ");
    Serial.println(slave.pdoBitRead(9)); // Read PDO bit 9

    delay(1000); // Wait for 1 second
}
```

Example 2: Read a byte data from Input PDO

Read data from Input PDO using pdoRead8().
A 16-channel digital input EtherCAT SubDevice has 2-byte Input PDOs, with each bit corresponding to a digital input channel. The states of channels 0 and 9 will be printed with a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  Serial.begin(115200); // Initialize Serial Monitor
  while (!Serial);      // Wait for Serial Monitor to be ready

  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master
  master.start();       // Start EtherCAT communication
}

void loop() {
  // Read specific bits using pdoRead8 and print their values
  Serial.print("Bit0 => ");
  Serial.print((slave.pdoRead8(0) >> 0) & 1); // Read PDO byte 0, extract bit 0
  Serial.print(", Bit9 => ");
  Serial.println((slave.pdoRead8(1) >> 1) & 1); // Read PDO byte 1, extract bit 9

  delay(1000); // Wait for 1 second before reading again
}
```

Example 3: Write a bit data to Output PDO

Write a bit to Output PDO using pdoBitWrite().
A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();       // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master
  master.start();       // Start EtherCAT communication
}

void loop() {
  // Set Bit 0 and Bit 9 to 1
  slave.pdoBitWrite(0, 1); // Write 1 to Bit 0
  slave.pdoBitWrite(9, 1); // Write 1 to Bit 9
  delay(1000);             // Wait for 1 second

  // Set Bit 0 and Bit 9 to 0
  slave.pdoBitWrite(0, 0); // Write 0 to Bit 0
  slave.pdoBitWrite(9, 0); // Write 0 to Bit 9
  delay(1000);             // Wait for 1 second
}
```

Example 4: Write a byte data to Output PDO

Write data to Output PDO using pdoWrite8().
A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    master.begin();       // Initialize EtherCAT Master
    slave.attach(0, master);
    master.start();       // Start EtherCAT communication
}

void loop() {
    // Write 0x01 to PDO byte 0 and 0x02 to PDO byte 1
    slave.pdoWrite8(0, 0x01); // Set byte 0 to 0x01
    slave.pdoWrite8(1, 0x02); // Set byte 1 to 0x02
    delay(1000);              // Wait for 1 second

    // Write 0x00 to PDO byte 0 and byte 1
    slave.pdoWrite8(0, 0x00); // Set byte 0 to 0x00
    slave.pdoWrite8(1, 0x00); // Set byte 1 to 0x00
    delay(1000);              // Wait for 1 second
}
```

4.5.4 CANopen over EtherCAT (CoE) Functions

CoE (CAN application over EtherCAT) is a CANopen protocol based on the EtherCAT network. It enables communication using the CANopen protocol over EtherCAT networks. The Object Dictionary contains parameters, application data and the mapping information between process data interface and application date (PDO mapping). Its entries can be accessed via Service Data Objects (SDO).

The SDO services primarily consist of two types of commands. The SDO command is utilized for accessing objects stored in the Object Dictionary, while the SDO information command is employed to retrieve details about these objects.

Example 1: SDO Upload

SDO Upload using sdoUpload8().
The usage of sdoUpload16(), sdoUpload32(), and sdoUpload64() is similar to sdoUpload8(), except for the difference in the return value type.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();      // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  // Read and print the value of SDO 0x1C12.0
  Serial.print("1C12h.0 => ");
  Serial.println(slave.sdoUpload8(0x1C12, 0x00)); // Upload 8-bit SDO data from
0x1C12.0

  // Read and print the value of SDO 0x1C13.0
  Serial.print("1C13h.0 => ");
  Serial.println(slave.sdoUpload8(0x1C13, 0x00)); // Upload 8-bit SDO data from
0x1C13.0
```

```
}

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

}
```

Example 2: SDO Upload with abort code

SDO Upload using sdoUpload() with abort code.
Initiate an SDO Upload command to read a value from a non-existent object, expecting an abort code of 0x06020000. For more information about abort codes, please refer to SDO Abort Code.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t abortcode;   // Variable to store the abort code
uint8_t value;       // Variable to store the uploaded value

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

  master.begin();      // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  // Attempt to upload SDO data
  if (slave.sdoUpload(0xFFFF, 0xFF, &value, sizeof(value), &abortcode) ==
ECAT_ERR_DEVICE_COE_ERROR) {
    Serial.print("Abort Code: 0x");
    Serial.println(abortcode, HEX); // Print the abort code in hexadecimal format
  }
}

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

}
```

Example 3: SDO Download

SDO Download using sdoDownload8().
The usage of sdoDownload16(), sdoDownload32(), and sdoDownload64() is similar to sdoDownload8(), except for the difference in the input parameter types.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);

  slave.sdoDownload8(0x1C12, 0x00, 0);
  slave.sdoDownload8(0x1C13, 0x00, 0);
}

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

}
```

Example 4: SDO Download with abort code

SDO Download using sdoDownload() with abort code.
Initiate an SDO Download command to write a value to a non-existent object, expecting an abort code of 0x06020000. For more information about abort codes, please refer to SDO Abort Code.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t abortcode;
uint8_t value;

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

  master.begin();
  slave.attach(0, master);

  if (slave.sdoDownload(0xFFFF, 0xFF, &value, sizeof(value), &abortcode) ==
ECAT_ERR_DEVICE_COE_ERROR) {
    Serial.print("Abort Code: 0x");
    Serial.println(abortcode, HEX); // Print the abort code in hexadecimal format
  }
}

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

}
```

Example 5: Print the PDO mapping configuration

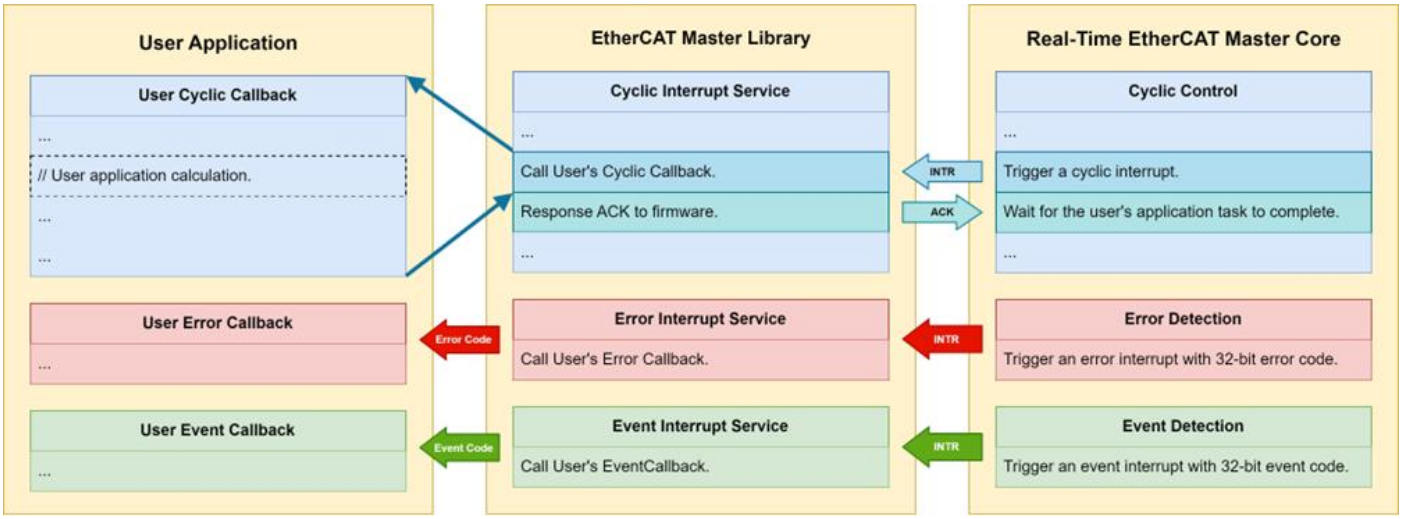
Print the PDO mapping configuration.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t assign_nr, mapping_nr;
uint16_t mapping;
uint32_t entry;

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

  master.begin();
  slave.attach(0, master);

  // Process RxPDO Assignments
  assign_nr = slave.sdoUpload8(0x1C12, 0x00);
  for (int m = 0; m < assign_nr; m++) {
    mapping = slave.sdoUpload16(0x1C12, m + 1);
    Serial.print("  RxPDO");
    Serial.print(m + 1);
    Serial.print(" (");
    Serial.print(mapping, HEX);
    Serial.println("h)");

    mapping_nr = slave.sdoUpload8(mapping, 0x00);
    for (int n = 0; n < mapping_nr; n++) {
      entry = slave.sdoUpload32(mapping, n + 1);
      Serial.print("    ");
      Serial.print(entry, HEX);
      Serial.println("h");
    }
  }

  // Process TxPDO Assignments
  assign_nr = slave.sdoUpload8(0x1C13, 0x00);
```

```
  for (int m = 0; m < assign_nr; m++) {
    mapping = slave.sdoUpload16(0x1C13, m + 1);
    Serial.print("  TxPDO");
    Serial.print(m + 1);
    Serial.print(" (");
    Serial.print(mapping, HEX);
    Serial.println("h)");

    mapping_nr = slave.sdoUpload8(mapping, 0x00);
    for (int n = 0; n < mapping_nr; n++) {
      entry = slave.sdoUpload32(mapping, n + 1);
      Serial.print("    ");
      Serial.print(entry, HEX);
      Serial.println("h");
    }
  }
}

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

}
```

4.5.5 Cyclic Callback



This library provides three types of callbacks as follows:

- **Cyclic Callback**
  The purpose of the Cyclic Callback is to allow users to implement periodic control systems such as motion control, CNC control, and robot control. The Real-Time EtherCAT MDevice Core triggers cyclic interrupts to the EtherCAT MDevice Library at specified cycle time, then waiting for an ACK to ensure process data synchronization. If a user has registered a Cyclic Callback, it will be invoked to achieve periodic control.

- **Error Callback**
  When the Real-Time EtherCAT MDevice Core detects an error, it will trigger an error interrupt and pass a 32-bit error code to the EtherCAT MDevice Library. If the user has registered an error callback, the system will invoke that callback to inform the user of the specific error.

  The error codes supported by the Error Callback are as follows:

Definition	Code	Description
ECAT_ERR_WKC_SINGLE_FAULT	2000001	Working counter fault occurred.
ECAT_ERR_WKC_MULTIPLE_FAULTS	2000002	Multiple working counter faults occurred.
ECAT_ERR_SINGLE_LOST_FRAME	2000003	Frame was lost.
ECAT_ERR_MULTIPLE_LOST_FRAMES	2000004	Frames were lost multiple times.
ECAT_ERR_CABLE_BROKEN	2000007	The cable is broken.
ECAT_ERR_WAIT_ACK_TIMEOUT	2001000	Firmware timeout waiting for cyclic interrupt ACK.

- **Event Callback**
  When the Real-Time EtherCAT MDevice Core detects an event, it triggers an event interrupt and passes a 32-bit event code to the EtherCAT MDevice Library. If the user has registered an event callback, the system will invoke that callback to inform the user of the specific event.
  The event codes supported by the Event Callback are as follows:

Definition	Code	Description
ECAT_EVT_STATE_CHANGED	1000001	The EtherCAT state of the MDevice has changed.
ECAT_EVT_CABLE_RECONNECTED	1000002	The cable has been reconnected.

Example 1: Cyclic callback

A 16-channel digital output EtherCAT SubDevice has 2-byte Output PDOs, with each bit corresponding to a digital output channel. Channels 0 and 9 will be toggled at a frequency of 1 Hz.

B15	B14	B13	B12	B11	B10	**B9**	B8	B7	B6	B5	B4	B3	B2	B1	**B0**

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;
int toggle = 0;       // Toggle variable
int cycle_count = 0;  // Cycle count variable

void myCallback() {
  if (++cycle_count < 1000) // Increment and check the cycle count
    return;
  cycle_count = 0;         // Reset cycle count

  toggle = !toggle;        // Toggle the state
  slave.pdoBitWrite(0, toggle); // Write the toggle value to Bit 0
  slave.pdoBitWrite(9, toggle); // Write the toggle value to Bit 9
}

void setup() {
  master.begin();      // Initialize EtherCAT Master
  slave.attach(0, master); // Attach the first EtherCAT slave to the master

  master.attachCyclicCallback(myCallback); // Attach cyclic callback
  master.start(1000000); // Start EtherCAT Master with 1 ms cycle time
}

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

}
```

Example 2: Error callback

Print the count of each type of error once per second.

Here is the example code:

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

// Error counters
int wkc_single_fault_cnt = 0;
int wkc_multiple_faults_cnt = 0;
int single_lost_frame_cnt = 0;
int multiple_lost_frames_cnt = 0;
int cable_broken_cnt = 0;
int wait_ack_timeout_cnt = 0;

// Error callback function
void myErroCallback(uint32_t errorcode) {
  switch (errorcode) {
    case ECAT_ERR_WKC_SINGLE_FAULT:
      wkc_single_fault_cnt++;
      break;
    case ECAT_ERR_WKC_MULTIPLE_FAULTS:
      wkc_multiple_faults_cnt++;
      break;
    case ECAT_ERR_SINGLE_LOST_FRAME:
      single_lost_frame_cnt++;
      break;
    case ECAT_ERR_MULTIPLE_LOST_FRAMES:
      multiple_lost_frames_cnt++;
      break;
    case ECAT_ERR_CABLE_BROKEN:
      cable_broken_cnt++;
      break;
    case ECAT_ERR_WAIT_ACK_TIMEOUT:
      wait_ack_timeout_cnt++;
      break;
  }
}
```

```
void setup() {
  Serial.begin(115200);
  while (!Serial);

  master.attachErrorCallback(myErroCallback); // Attach error callback
  master.begin();      // Initialize EtherCAT Master
  master.start();      // Start EtherCAT communication
}

void loop() {
  // Print error counts to Serial Monitor
  Serial.print("ECAT_ERR_WKC_SINGLE_FAULT     = ");
  Serial.println(wkc_single_fault_cnt);
  Serial.print("ECAT_ERR_WKC_MULTIPLE_FAULTS  = ");
  Serial.println(wkc_multiple_faults_cnt);
  Serial.print("ECAT_ERR_SINGLE_LOST_FRAME    = ");
  Serial.println(single_lost_frame_cnt);
  Serial.print("ECAT_ERR_MULTIPLE_LOST_FRAMES = ");
  Serial.println(multiple_lost_frames_cnt);
  Serial.print("ECAT_ERR_CABLE_BROKEN        = ");
  Serial.println(cable_broken_cnt);
  Serial.print("ECAT_ERR_WAIT_ACK_TIMEOUT     = ");
  Serial.println(wait_ack_timeout_cnt);

  delay(1000); // Wait 1 second before the next print
}
```

Example 3: Event callback

Print the count of each type of event once per second.

Here is the example code:

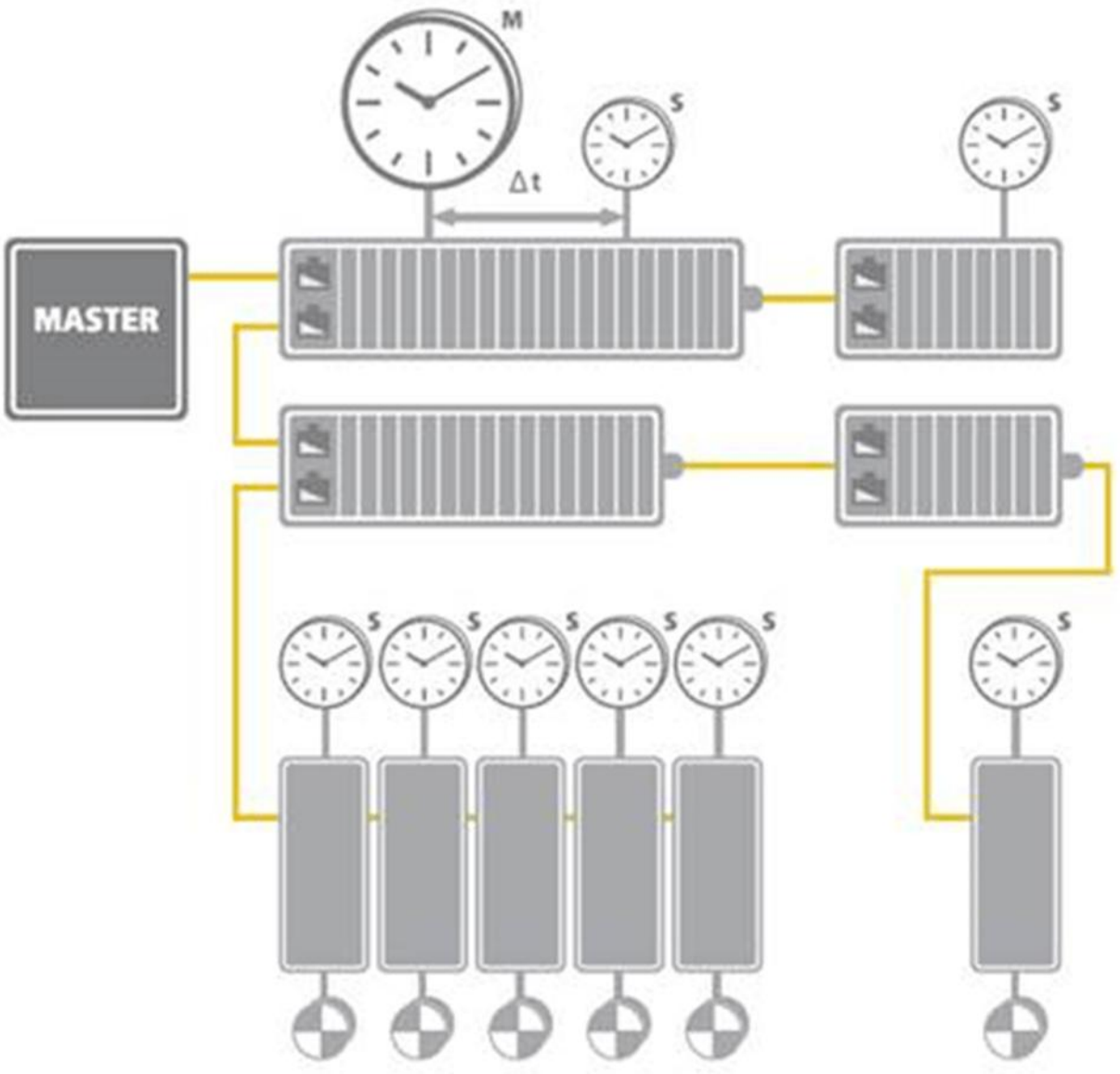
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

// Event counters
int state_changed_cnt = 0;
int cable_reconnected_cnt = 0;

// Event callback function
void myEventCallback(uint32_t eventcode) {
  switch (eventcode) {
    case ECAT_EVT_STATE_CHANGED:
      state_changed_cnt++;
      break;
    case ECAT_EVT_CABLE_RECONNECTED:
      cable_reconnected_cnt++;
      break;
  }
}

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

  master.attachEventCallback(myEventCallback); // Attach event callback
  master.begin();      // Initialize EtherCAT Master
  master.start();      // Start EtherCAT communication
}

void loop() {
  // Print event counts to Serial Monitor
  Serial.print("ECAT_EVT_STATE_CHANGED    = ");
  Serial.println(state_changed_cnt);
  Serial.print("ECAT_EVT_CABLE_RECONNECTED = ");
  Serial.println(cable_reconnected_cnt);
```

```
  delay(1000); // Wait 1 second before the next update
}
```

4.5.6 Distributed Clock (DC)

In applications with spatially distributed processes requiring simultaneous actions, exact synchronization is particularly important. For example, this is the case for applications in which multiple servo axes execute coordinated movements.

In contrast to completely synchronous communication, whose quality suffers immediately from communication errors, distributed synchronized clocks have a high degree of tolerance for jitter in the communication system. Therefore, the EtherCAT solution for synchronizing nodes is based on such distributed clocks (DC).

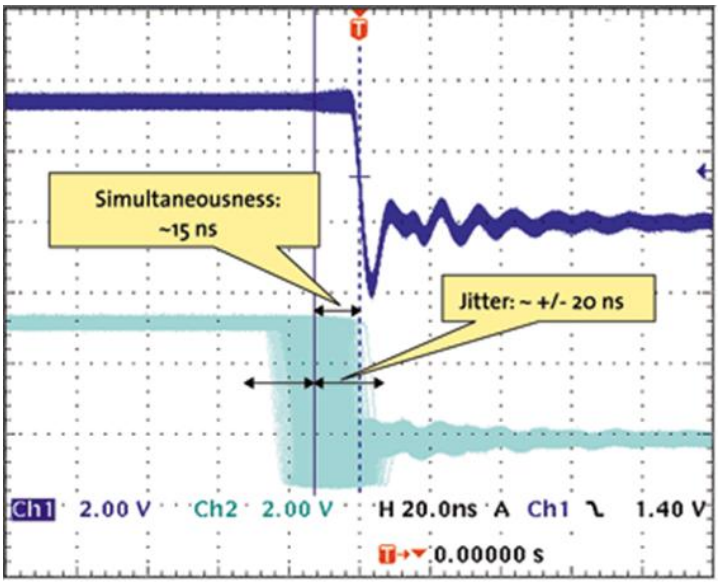


EtherCAT: Illustration of Distributed Clock (DC). (Source of information: http://www.ethercat.org/)

The synchronization mechanism of EtherCAT is based on IEEE-1588 Precision Clock Synchronization Protocol and extends the definition to so-called Distributed-clock (DC). To put it simple, every EtherCAT ESC maintains a hardware-based clock and the minimum time interval is 1 nano-second (64 bits in total). The time maintained by EC-Slave is called Local system time.

With accurate internet time synchronization mechanism and dynamic time compensation mechanism (*1), EtherCAT DC technology can guarantee that the time difference among every EC-Slave local system time is within +/- 20 nano-seconds. The following diagram is a scope view of two slave devices' output digital signals. We can see that the time difference between the I/O signal from two EC-SubDevices is around 20 nano-seconds.

(*1) Please refer to EtherCAT standard document ETG1000.4



Synchronicity and Simultaneousness: Scope view of two distributed devices with 300 nodes and 120 m of cable between them. (Source of information: http://www.ethercat.org/)

Example 1: Enable DC synchronization

Implementing position control on a CiA 402 EtherCAT SubDevice using the EthercatDevice_Generic class. The CiA 402 control mode is set to cyclic synchronous position mode, and DC synchronization is enabled for precise timing.

The default PDO mapping is as follows:

- Output PDO (RxPDO)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Controlword		Target Position			

- Input PDO (TxPDO)

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Statusword		Position Actual Value			

Here is the example code:

```
#include <Ethercat.h>

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t position = 0;

// Cyclic callback function
void myCyclicCallback() {
  // Check if the drive is in the correct state
  if ((slave.pdoRead8(0) & 0x6F) != 0x27)
    return;

  // Increment and write the new position
  slave.pdoWrite32(2, position += 1000);
}

void setup() {
  master.begin();       // Initialize EtherCAT Master

  slave.attach(0, master);   // Attach the first EtherCAT slave
  slave.setDc(1000000);      // Set Distributed Clock synchronization to 1 ms
  slave.sdoDownload8(0x6060, 0x00, 8); // Set operation mode to CSP (Cyclic
Synchronous Position)
```

```
  master.attachCyclicCallback(myCyclicCallback); // Attach the cyclic callback
  master.start(1000000, ECAT_SYNC); // Start EtherCAT Master with 1 ms cycle time
and synchronization

  // Initialize position and control word
  slave.pdoWrite32(2, position = slave.pdoRead32(2)); // Set initial position
  slave.pdoWrite8(0, 0x80); // Reset fault
  delay(1000);
  slave.pdoWrite8(0, 0x06); // Switch to "Shutdown" state
  delay(1000);
  slave.pdoWrite8(0, 0x07); // Switch to "Switch On" state
  delay(1000);
  slave.pdoWrite8(0, 0x0F); // Switch to "Operation Enable" state
  delay(1000);
}

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

}
```

4.5.7 86EVA, an EtherCAT Configuration Tool

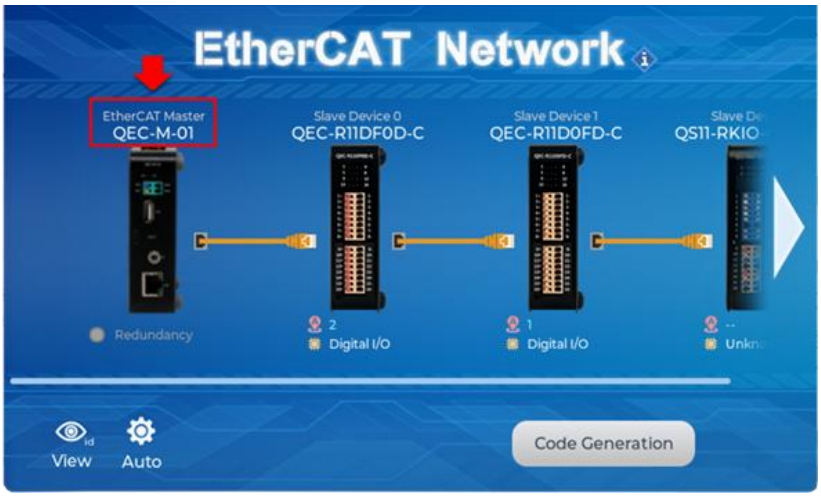
86EVA is a graphical EtherCAT configurator based on the EtherCAT Library in the 86Duino IDE and is one of the development kits for 86Duino. The user can use it to configure the EtherCAT network quickly and start programming.

The following information about 86EVA will focus on QEC EtherCAT SubDevices, with features including:

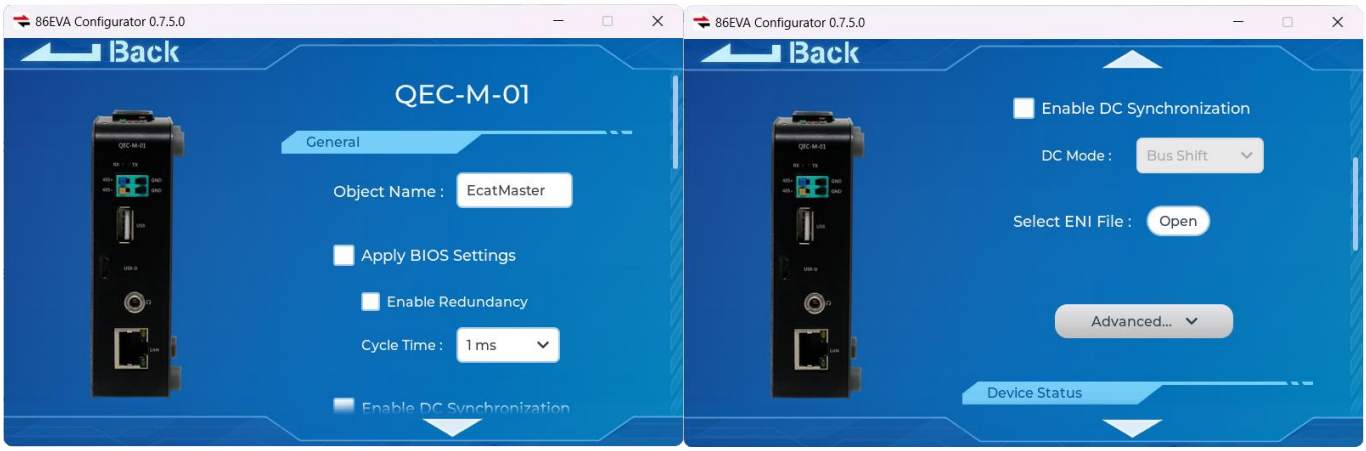
1. Automatically generated Arduino language (via EtherCAT-Based Virtual Arduino)
2. Automatically scan for network devices.
3. EtherCAT MDevice Settings:
    - Set MDevice Object Name
    - Set Cycle Time
    - Set Redundancy Options
    - Optional ENI file
4. EtherCAT SubDevice Settings:
    - Set SubDevice Object Name
    - Set SubDevice Alias
    - SubDevice I/O Mapping can be set
    - Display secondary device information
    - View internal information

The 86Duino IDE development environment is specifically designed for the QEC MDevices, which includes QEC-M-01, QEC-M-02, QEC-M-043T, QEC-M-070T, QEC-M-090T, and QEC-M-150T. After connecting and completing the scanning process in 86EVA, users can view the information of the EtherCAT MDevice with the product image.

- QEC-M-01:



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



You can set the EtherCAT MDevice Object Name, Apply BIOS Settings, Enable the EtherCAT Cable Redundancy option, set EtherCAT Cycle Time, and Select the ENI file if you need.

In **“Advanced”**, you can set **“Error Reactions”** for EtherCAT Network, including “Set All Slaves to Safe-OP When an Error Occurs” and “Auto Restart When Error-Free”.

You can also see the voltage, current, and system temperature in **“Device Status”** area.

For more detailed information, please refer to [86EVA, EtherCAT-Based Virtual Arduino].

Example 1: QEC Digital I/O Control

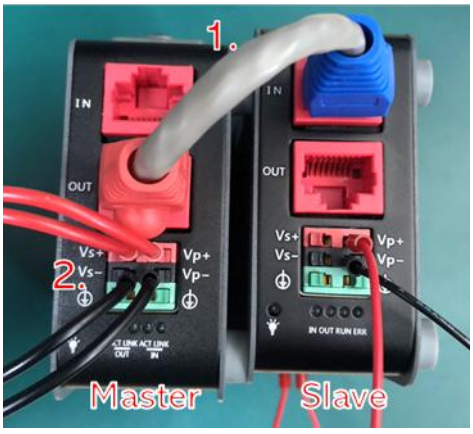
This example is using a QEC-M-01P to control the QEC-R11D88H-N (EtherCAT SubDevice with 8-ch Digital Input and 8-ch Digital Output). The following devices are used here:

1. QEC-M-01P (EtherCAT MDevice/PoE)
2. QEC-R11D88H-N (EtherCAT SubDevice 8-ch digital input and 8-ch digital output/PoE)
3. 24V power supply
4. 24V LED

QEC-M-01P

QEC EtherCAT MDevice with PoE function.

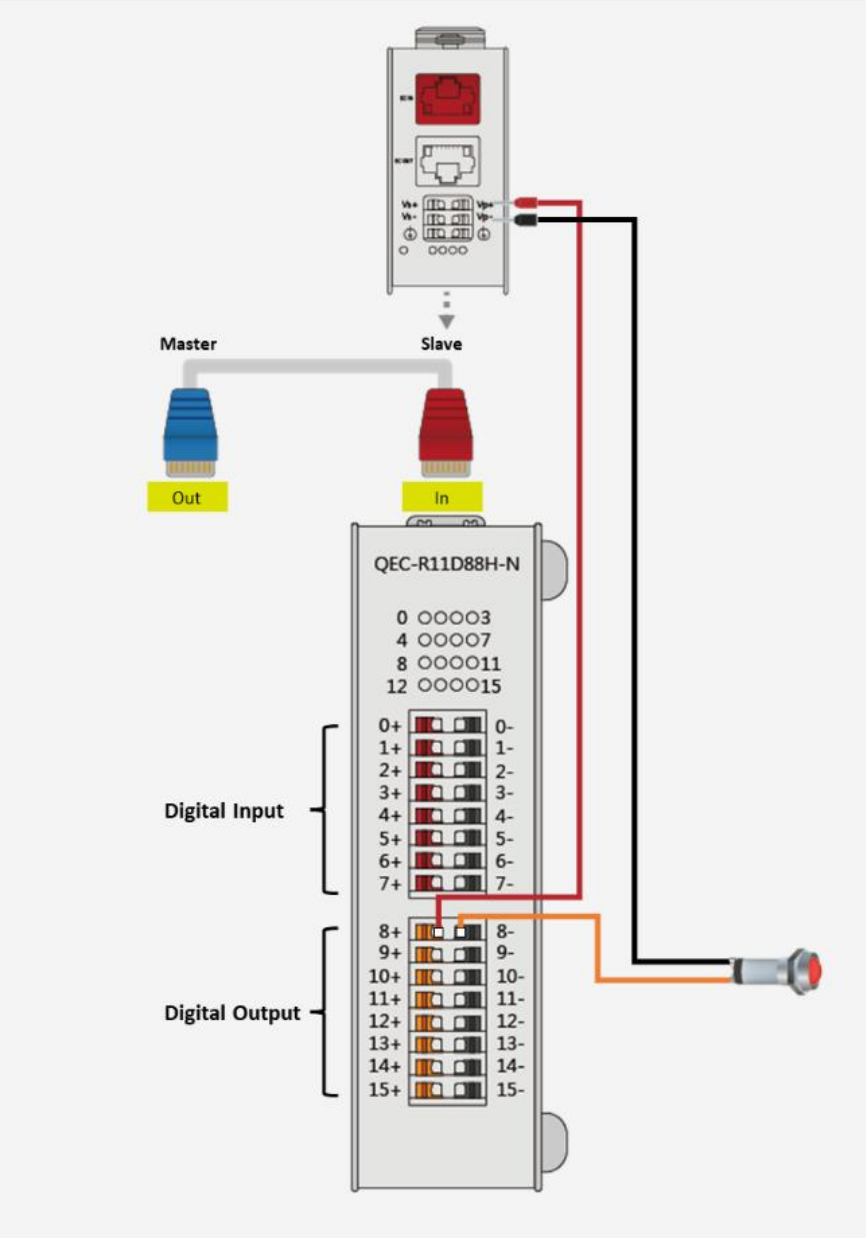
1. Using the EtherCAT Out port (top side) connected to the EtherCAT In port of QEC-R11D88H via RJ45 cable (powered by PoE).
2. Connect to Vs+/Vs- and Vp+/Vp- power supplies via EU terminals for 24V power.



QEC-R11D88H-N

- Connect from VP+ to DO 0+.
- Connect the 24V LED+ terminal to DO 0-.
- Connect the 24V LED- end to the VP-.

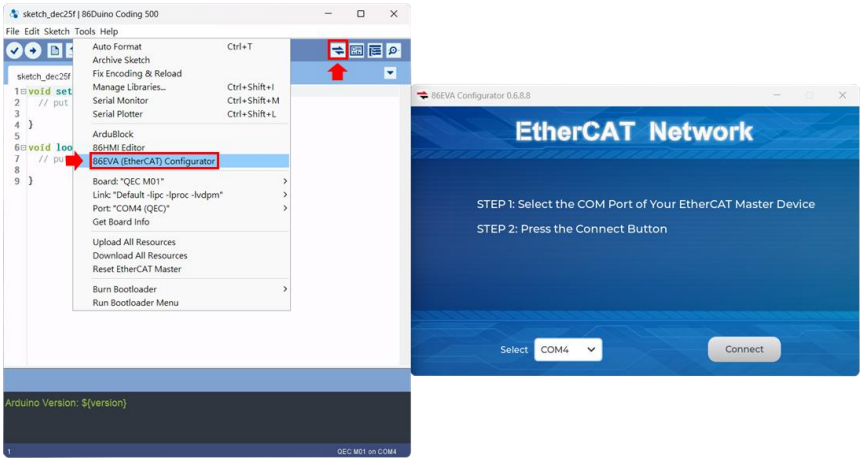
Wiring Diagram:



The following example is setting Pin0 of Digital Output to HIGH for 4 seconds and then changing it to LOW for 1 second.

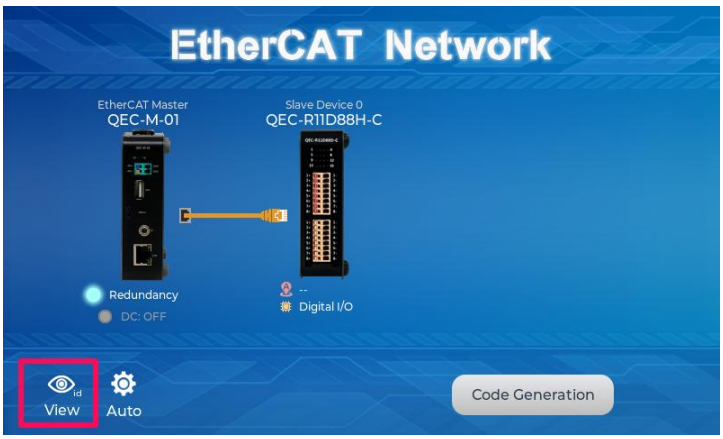
Step 1: Turn on 86EVA and scan

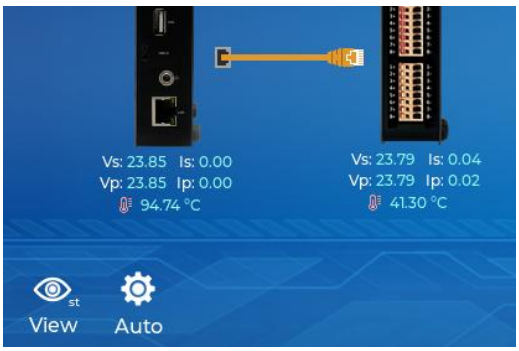
The 86EVA tool can be opened via the following buttons.



Once you have confirmed that the correct COM port has been selected of QEC-M-01P, press the Connect button to start scanning the EtherCAT network.

The connected devices will be displayed after the EtherCAT network has been scanned. Press the "View" button in the lower left corner to check the device's status.





Step 2: Set the parameters

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

QEC-M-01:

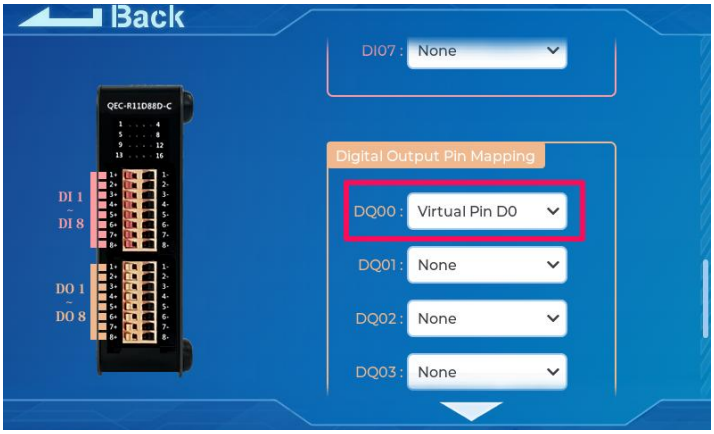
Click on the image of the QEC-M-01 to see the parameter settings.

If you are developing for the first time, please use the preset settings first and click "**Back**" in the upper left corner to return.

QEC-R11D88H-N:

Click on the image of the QEC-R11D88H to see the parameter settings.

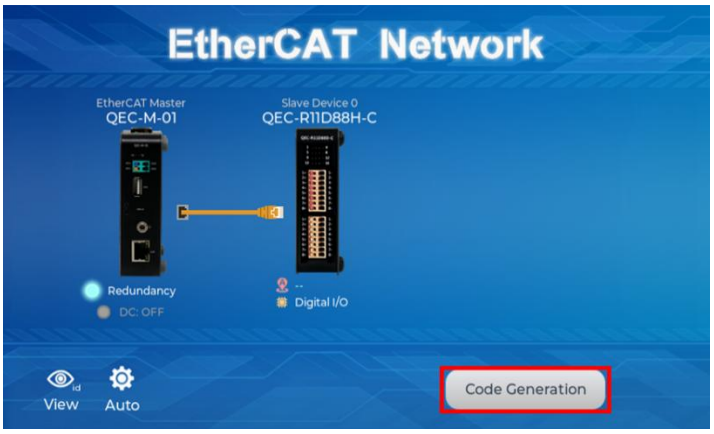
Continue down to the “**Digital Output Pin Mapping”** area. Among them, we select "**Virtual Pin D0**" in the drop-down box of DQ00 of Digital Output Pin Mapping and click "**Back**" in the upper left corner to return.



This action is to set the Digital Output Pin0 of QEC-R11D88H to the virtual D0 pin of EVA.

Step 3: Generate the code

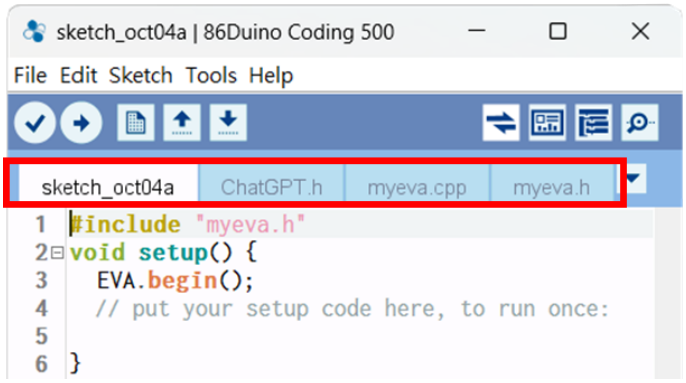
Once you've set your device's parameters, go back to the home screen and press the "Code Generation" button in the bottom right corner.



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_oct04a: Main Project (.ino, depending on your project name)
- ChatGPT.h: Parameters to provide to ChatGPT referred
- myeva.cpp: C++ program code of 86EVA
- myeva.h: Header file of 86EVA



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.

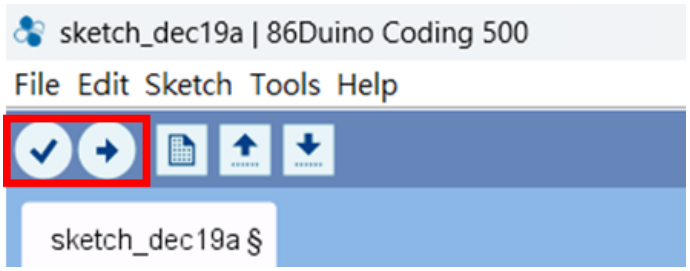
1. #include “myeva.h” : Include EVA Header file
2. EVA.begin(); in setup() : Initialize the EVA function

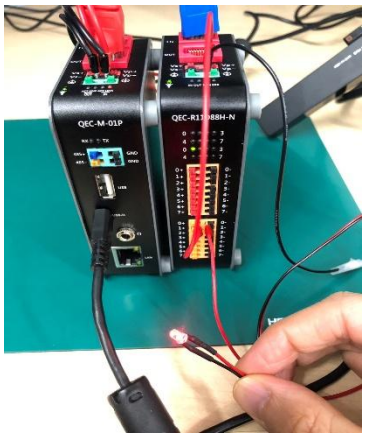
Step 4: Write the code

The following example is setting the D0 pin of EVA (Pin0 for Digital Output) to HIGH for 4 seconds and then changing it to LOW for 1 second.

```
#include "myeva.h" // Include EVA function

void setup() {
  EVA.begin(); // Initialize EVA function.
  // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:
    // Set EVA D0 pin to HIGH
    EVA.digitalWrite(0, HIGH);
    delay(4000); // Wait for 4 seconds

    // Set EVA D0 pin to HIGH
    EVA.digitalWrite(0, LOW);
    delay(1000); // Wait for 1 second
}
```

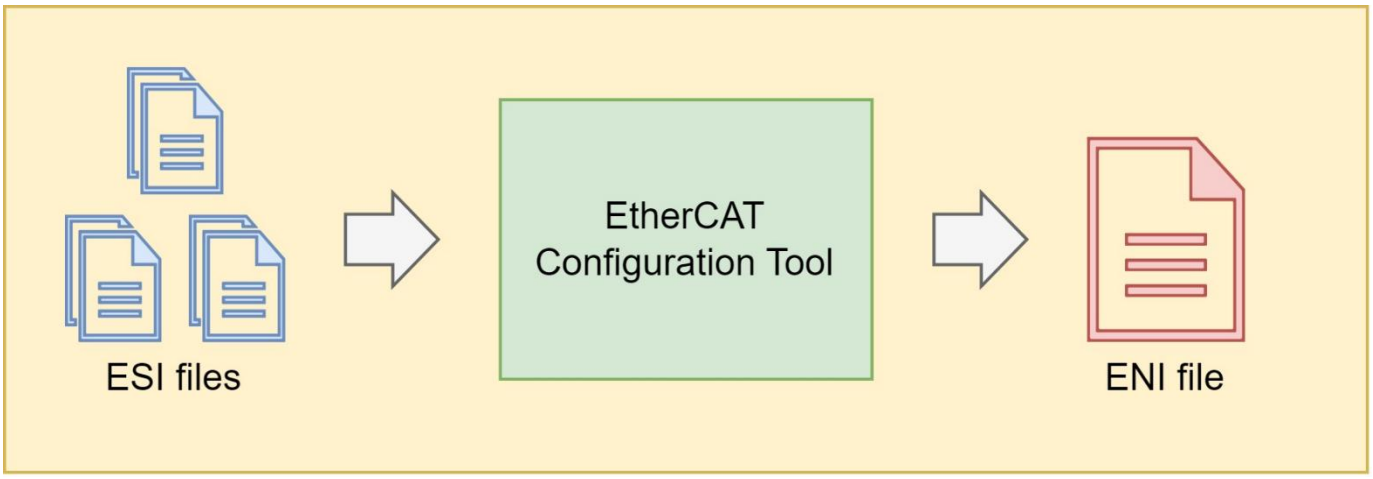
Note: Once the code is written, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload.





4.5.8 Import ENI to QEC-M-01

The EtherCAT Network Information (ENI) file contains the essential settings needed to configure an EtherCAT network. This XML-based file includes general information about the MDevice and the configurations of every slave device connected to it. Using the EtherCAT Configuration Tool, you can read ESI files or perform an online scan of the network to detect all connected slaves. You can then configure relevant EtherCAT settings, such as PDO mapping and enabling DC, and export the ENI file.



The EtherCAT Technology Group requires that the EtherCAT MDevice Software support at least one of the following methods in the Network Configuration section: Online Scanning or Reading ENI.
The EtherCAT Library in the 86Duino IDE 500+ of QEC EtherCAT supports both methods. However, when reading ENI, the library currently extracts only partial information from the ENI file for network configuration. For more details, please refer to A.1 About ENI Configuration in 86Duino IDE.

Method 1: Using Code

After setting up your 86Duino IDE environment, please put the ENI file on a USB disk and insert it into your QEC EtherCAT MDevice.

*Note: the USB disk readings via QEC MDevice will be under the P:\\ path.)

Step 1: Write the Import Code:

- In your project code, use the `EthercatMaster::begin()` function to import the ENI file.
- Example code:

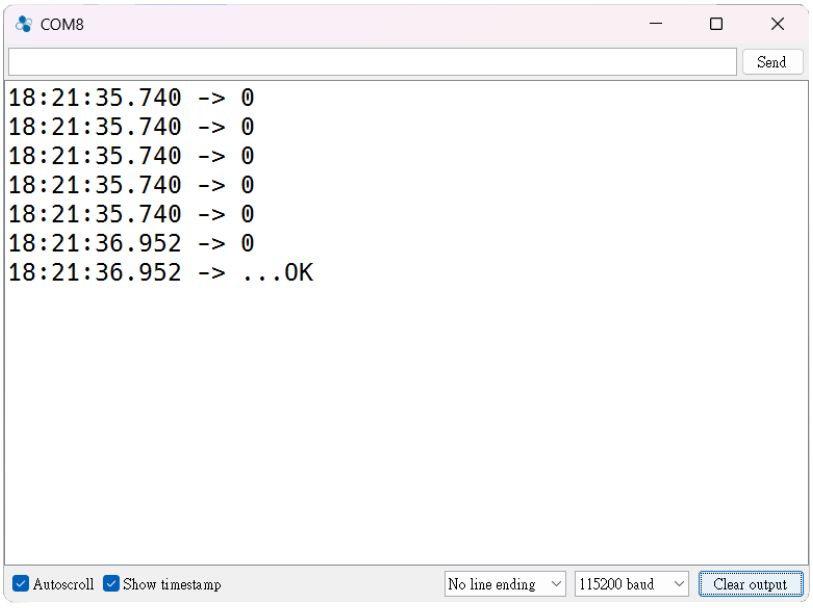
```
#include "Ethercat.h"

#define Device 4

EthercatMaster master;
EthercatDevice_Generic slave[Device];

void setup() {
    Serial.begin(115200);
    Serial.println(master.begin(ECAT_ETH_0, "P:\\test.xml"));
    // Note: the USB disk readings via QEC Master will be under the P:\\ path.
    for (int i = 0; i < Device; i++) {
        Serial.println(slave[i].attach(i, master));
    }
    Serial.println(master.start());
    Serial.println("...OK");
}

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

Step 2: Upload the Code:

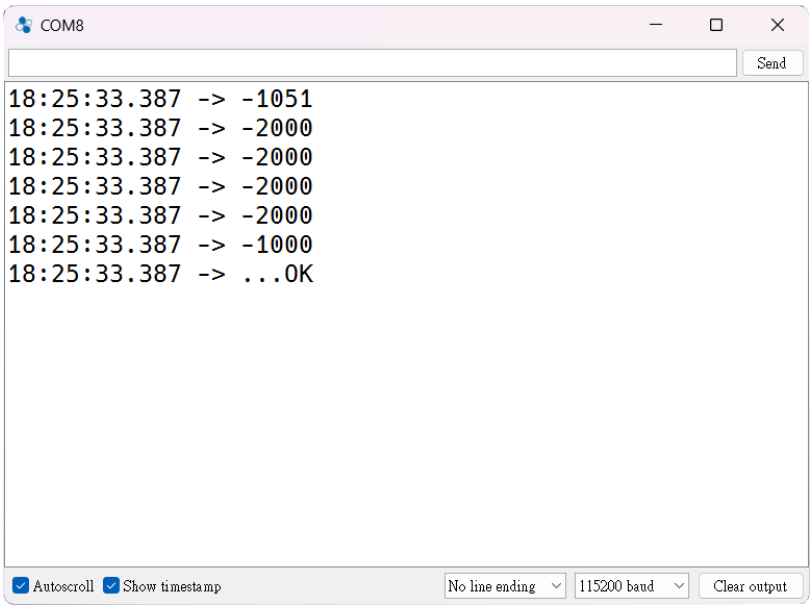
- Upload the code to your QEC EtherCAT MDevice.

Step 3: Verify the ENI File Import:

- Verify that the ENI file is correctly imported and the network is operational via Serial Monitor.
- If the ENI file is imported successfully, you will see the following return value in Serial Monitor:



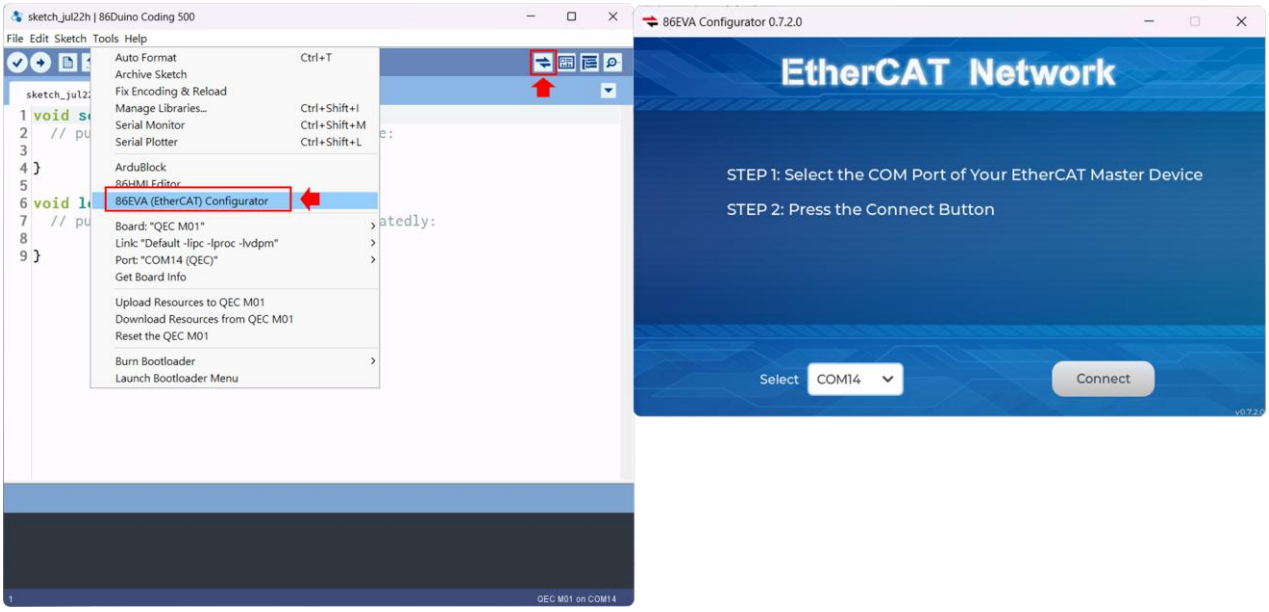
- If there is an error, such as -1051 (which means `ECAT_ERR_MASTER_ENI_MISMATCH`), refer to the EtherCAT API user manual for other error codes and their meanings.



Method 2: Using 86EVA

Step 1: Turn on 86EVA and scan:

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

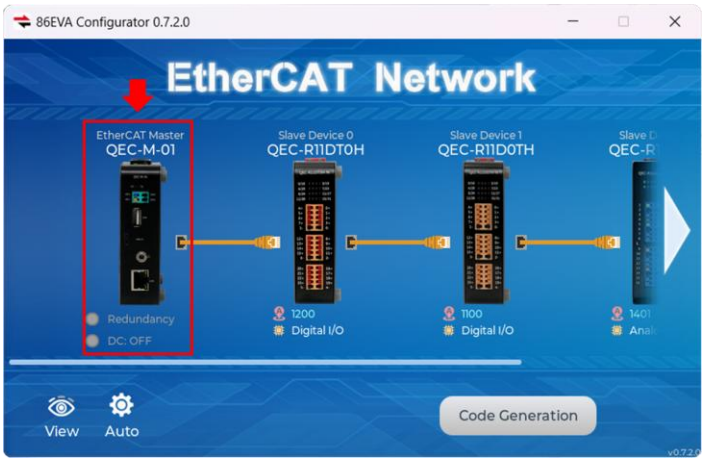


- Once you have confirmed that the correct COM port has been selected of QEC-M-01P, press the Connect button to start scanning the EtherCAT network.

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

Step 2: Import ENI file:

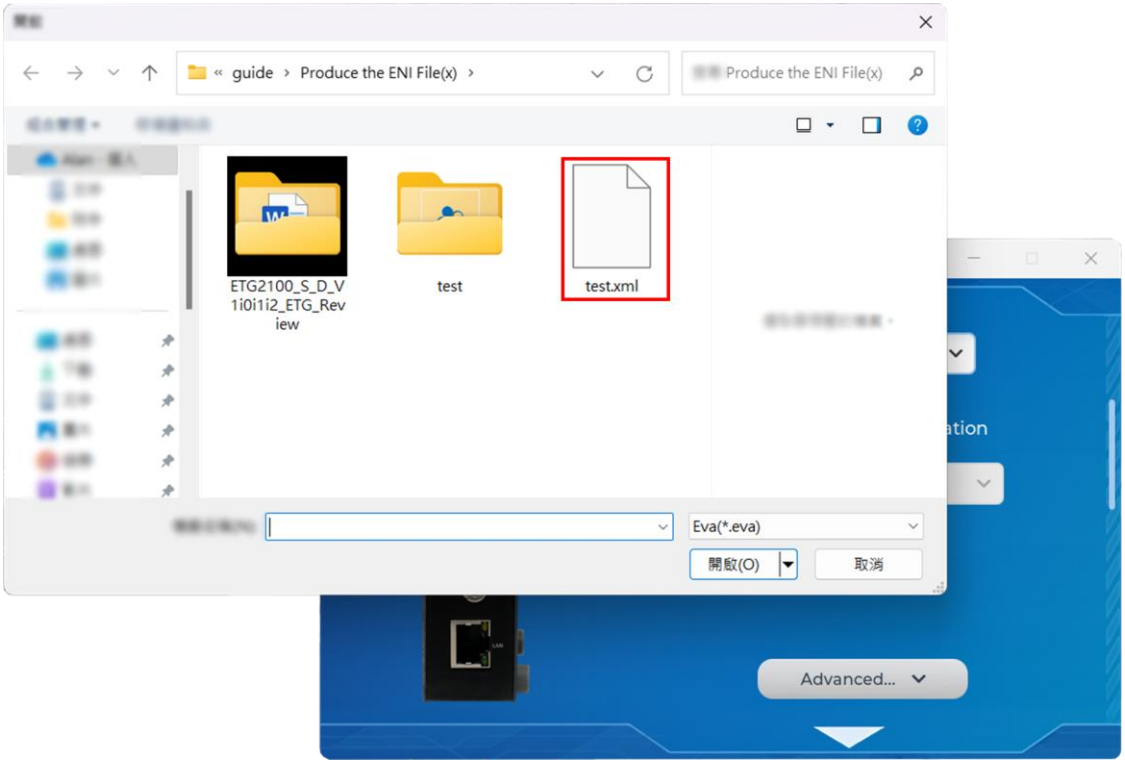
- Press twice on the QEC-M-01 to enter the corresponding parameter setting screen.



- You can see the parameter settings for the QEC-M-01.

- Click on the “Open” button next to the “Select ENI file” in the General area.

- Browse where you saved the ENI file in Chapter 2, and open it.

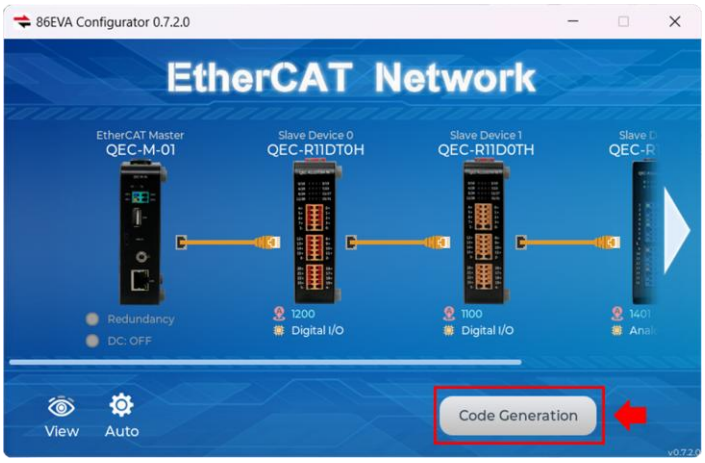


- After you import the ENI file into the 86EVA, you can see the ENI file name.

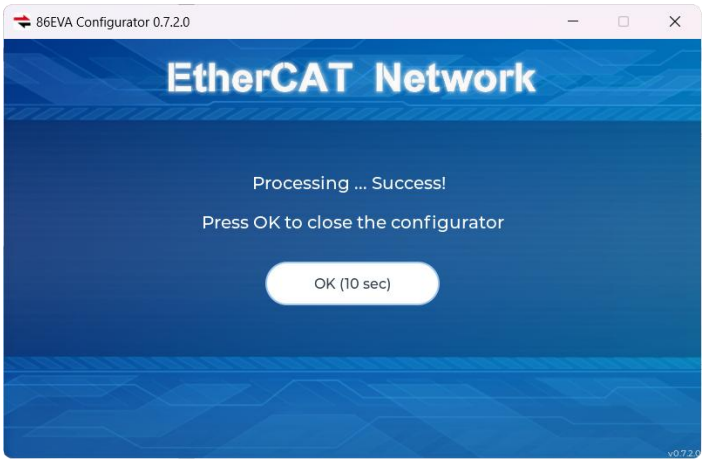
- Please click “Back” button in the upper left corner to return.

Step 3: Generate the code:

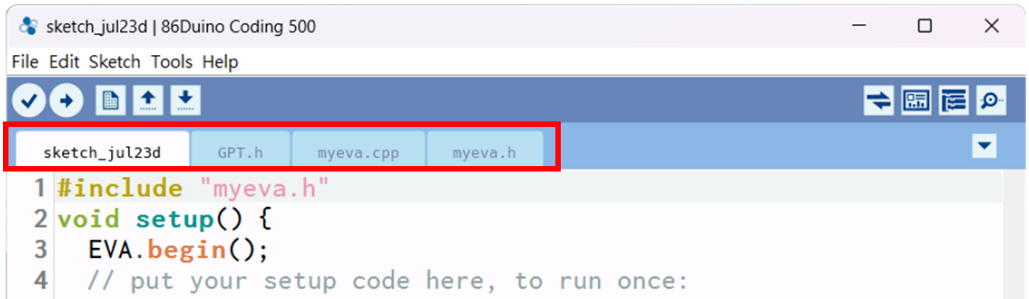
- Go back to the home screen and press the "Code Generation" button in the bottom right corner.



- 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:



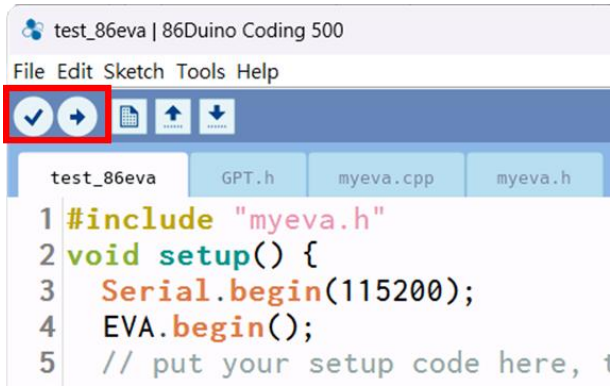
a. sketch_jul23d: Main Project (.ino, depending on your project name).
b. GPT.h: Parameters to provide to ChatGPT referred.
c. myeva.cpp: C++ program code of 86EVA.
d. myeva.h: Header file of 86EVA.

Step 4: Write the code:

- Please insert Serial.begin(115200); before EVA.begin(); in void setup() {}, so the EVA program return value can display in Serial Monitor (in baud rate 115200).

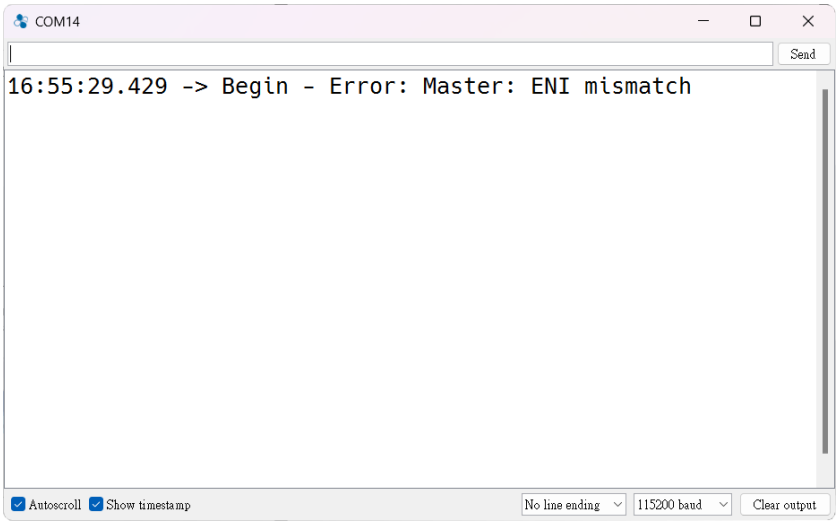
```
#include "myeva.h"

void setup() {
  Serial.begin(115200);
   EVA.begin();
  // put your setup code here, to run once:
}

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

- Once the code is completed, click on the toolbar to compile, and to confirm that the compilation is complete and error-free, you can click to upload. The program will run when the upload is complete.



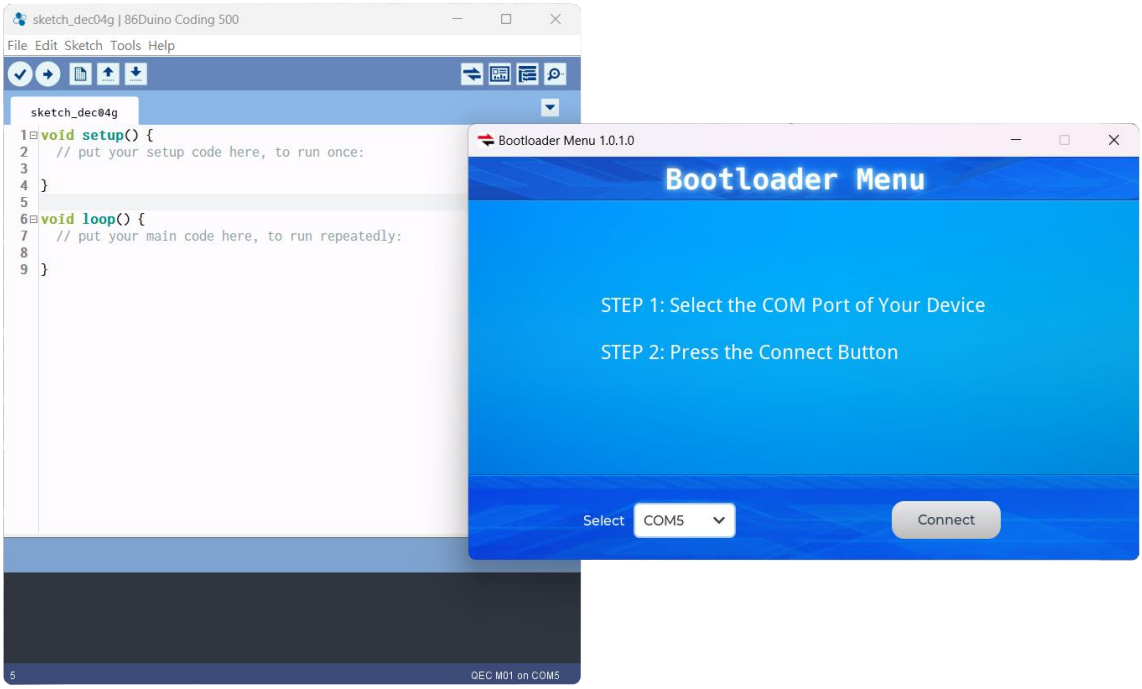
Step 5: Verify the ENI File Import:

- Verify that the ENI file is correctly imported and the network is operational via Serial Monitor.
- If the ENI file is imported successfully, the Serial Monitor will return nothing; if there is an error, it will return the error directly to the Serial Monitor.



4.6 Bootloader Menu Usage

This section introduces the Bootloader Menu, which provides a user-friendly interface for configuring and managing your QEC MDevice.

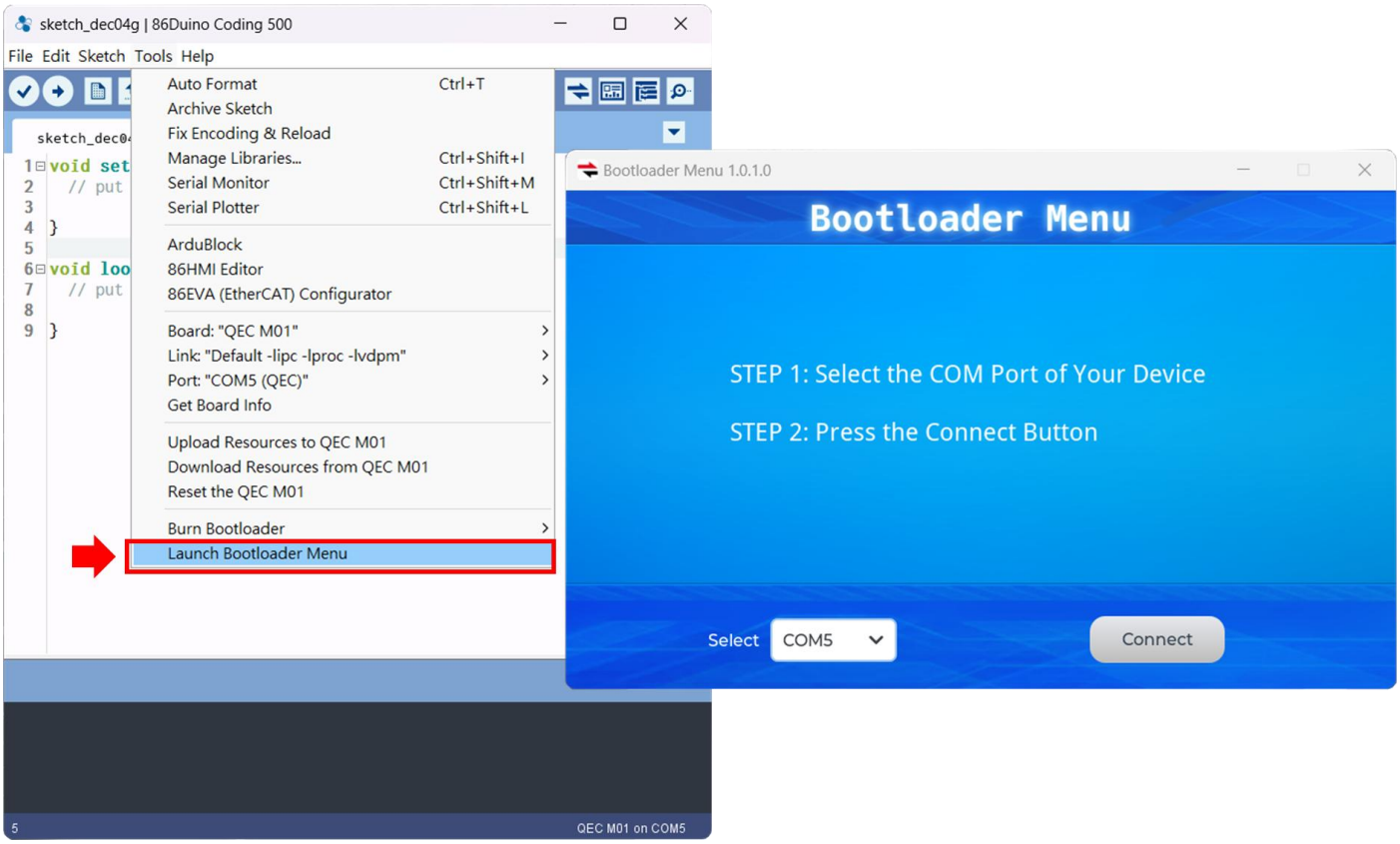


The Bootloader is the first program executed when the 86Duino powers on. It is critical in bridging the 86Duino hardware and the development environment. The Bootloader communicates with the 86Duino Coding IDE via the USB Device/Programming Port. It allows users to upload their sketch programs seamlessly from the development environment to the 86Duino hardware.
The Bootloader accepts user-uploaded sketch programs and writes them to the 86Duino device's onboard memory. After successfully uploading a sketch, the Bootloader executes the program automatically.

This section is a detailed guide to navigating and using the bootloader menu's features effectively.

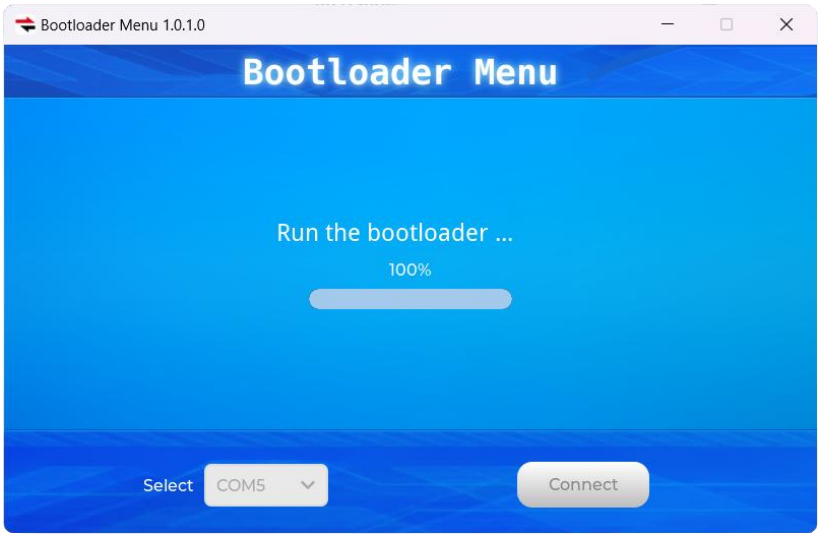
4.6.1 Turn on Bootloader Menu

The Bootloader Menu can be opened via the following buttons.

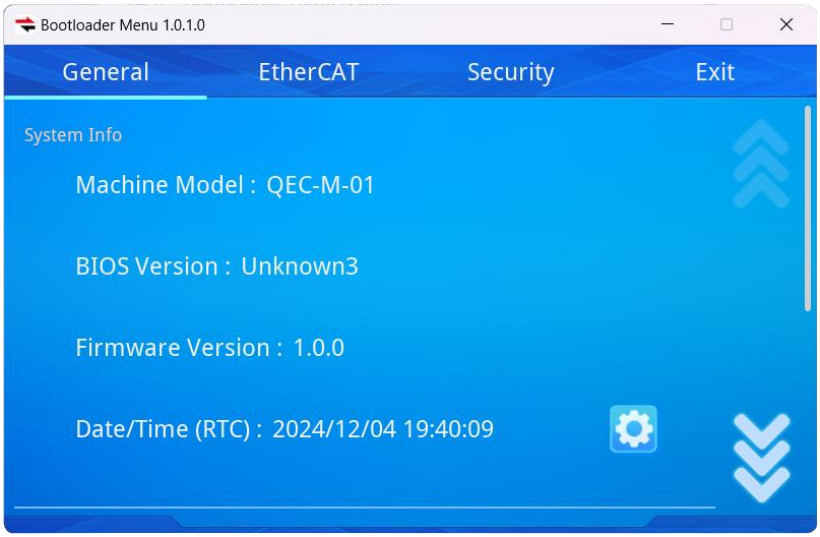


Please select the correct COM port and then click the “Connect” button.

Once you have confirmed that the correct COM port has been selected of QEC-M-01, press the Connect button to start the bootloader menu.



After the connection, you can see the bootloader menu like below the picture.



The Top Menu in the Bootloader Menu provides easy access to key settings and features.

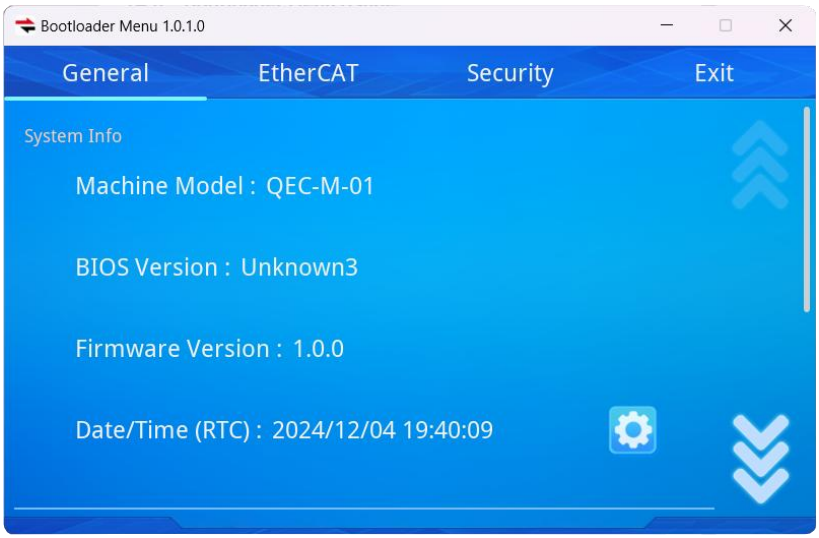
It has the following tabs:

1. General:
    - View system info (e.g., model, BIOS version, firmware version, and RTC).
    - Access basic settings like enabling the BIOS menu or allowing IDE connections.
2. EtherCAT:
    - Configure the BIOS default EtherCAT cycle time and enable/disable redundancy.
3. Security:
    - Set or update the bootloader password to secure access.
4. Exit:
    - Save or discard changes and reboot the device.

4.6.2 General Page

The General Page displays essential system information and provides basic configuration options.

System Info:

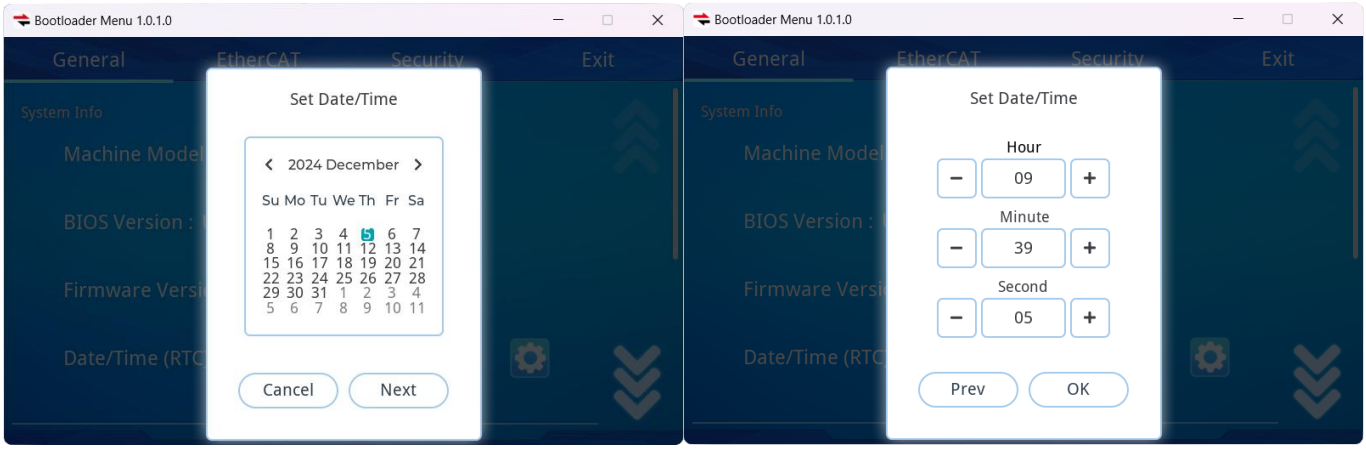


- Machine Model: Displays the current hardware model (e.g., QEC-M-01).
- BIOS Version: Displays the installed BIOS version.
- Firmware Version: Shows the current firmware version installed.
- Date/Time (RTC): Displays the system's real-time clock. Users can adjust this setting.

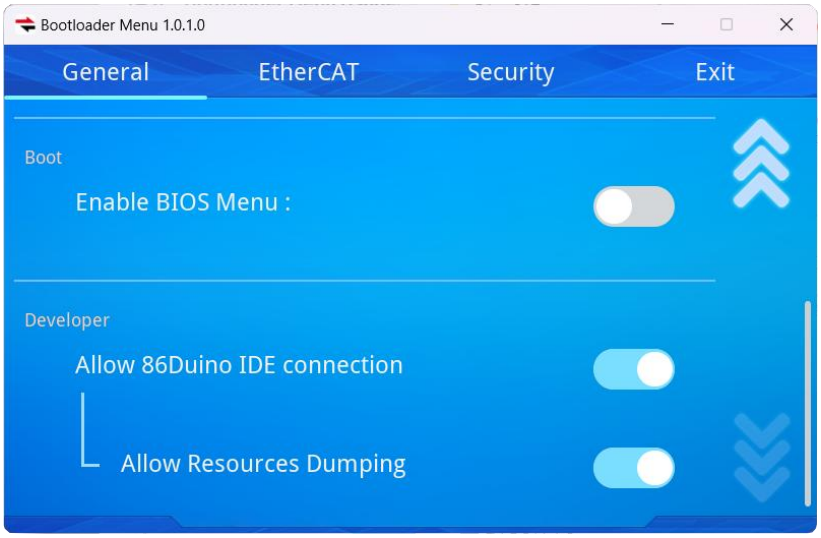
Additional Information:

To adjust the Date/Time (RTC):

1. Click the gear icon to open the "Set Date/Time" window.
2. Use the displayed options to modify the date and time settings.
3. Click OK to save the changes, or cancel to discard them.



Boot and Developer Options:



Boot:

- Enable BIOS Menu: Enable or disable BIOS Menu.

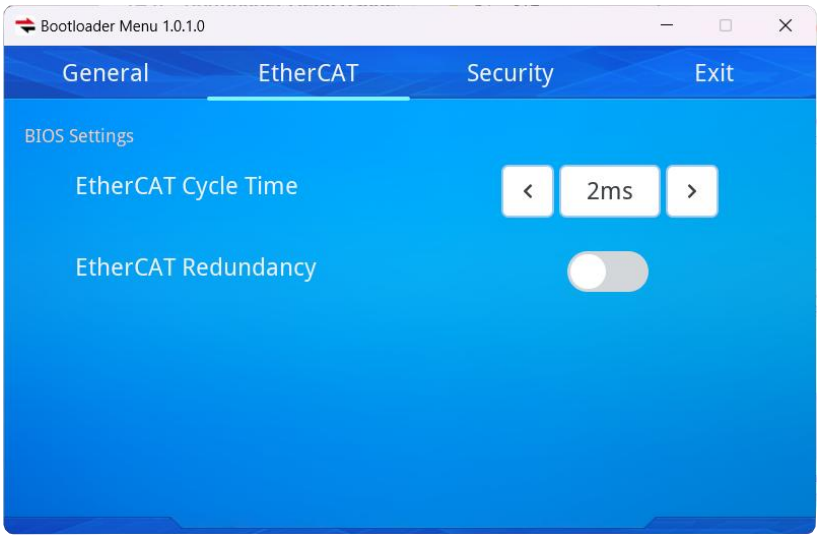
Developer:

- Allow 86Duino IDE connection: for development purposes.
- Allow Resources Dumping: Enable or disable Resources Dumping for debugging.

4.6.3 EtherCAT Page

The EtherCAT Page allows users to configure EtherCAT settings, including Cycle Time and Redundancy, which are part of the BIOS defaults.

BIOS Settings:



- EtherCAT Cycle Time: Configures the default EtherCAT communication cycle time. Options available: 62.5µs, 125µs, 250µs, 500µs, 1ms, and 2ms.
- EtherCAT Redundancy: Enables or disables the default EtherCAT redundancy configuration.

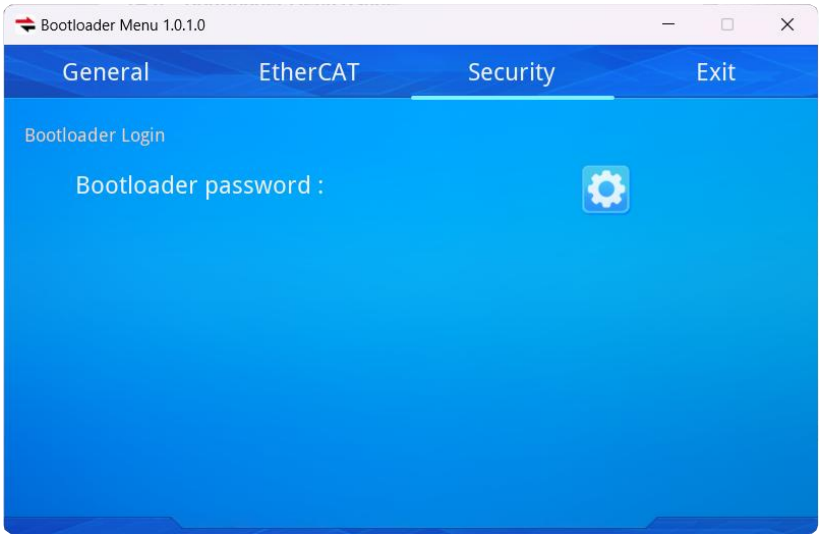
Additional Information:
These BIOS settings directly influence the EtherCAT configuration inside the 86EVA Configurator, as shown in the QEC MDevice page.

Enabling redundancy in the EtherCAT Page will automatically enable the Redundancy checkbox in the 86EVA Configurator, and setting the Cycle Time in the BIOS will apply it as the default value in the configurator.

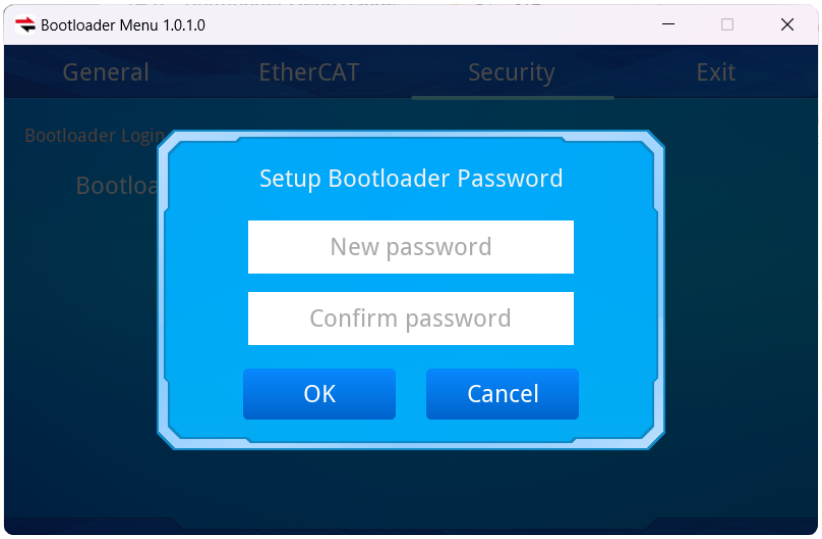
4.6.4 Security Page

The Security Page allows users to set and manage a bootloader password to protect the program and configuration. This feature ensures only authorized users can access the Bootloader Menu.

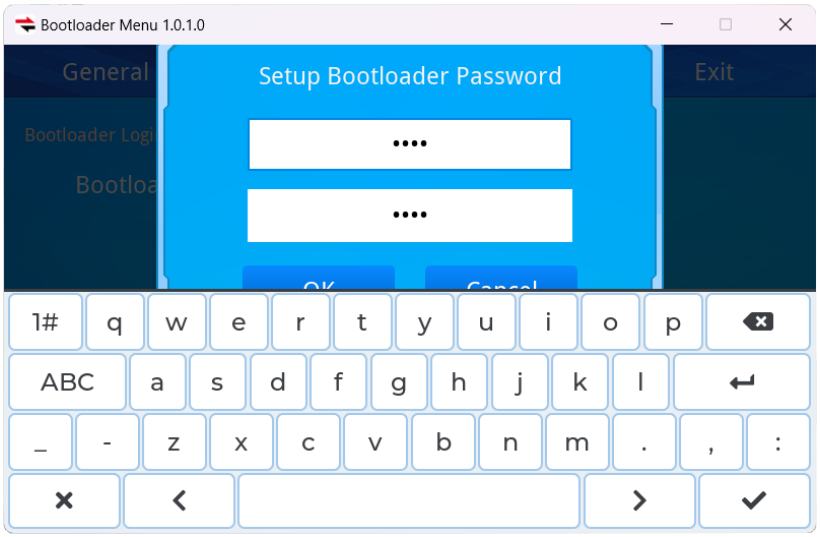
Bootloader Login:



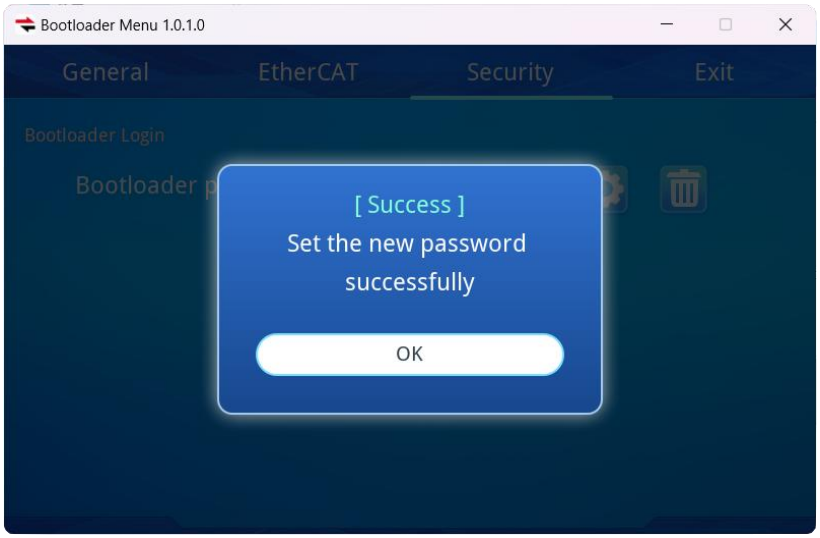
- Bootloader password: Users can click gear icon to open the “Setup Bootloader Password” window, as shown below.



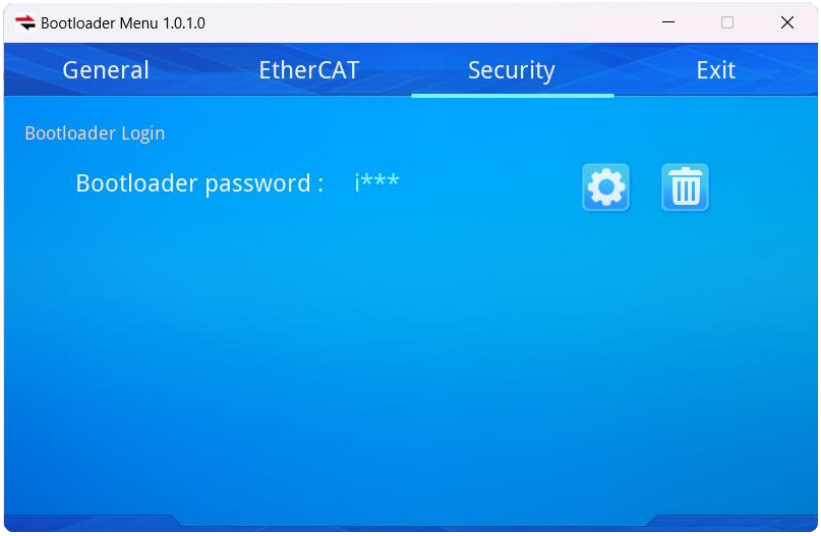
After clicking the text input area, the keyboard will appear on the windows.



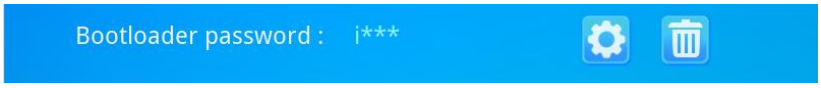
When users finish setting the bootloader password, click “OK”. A [Success] window will appear to let the users know that the new password has been set successfully.



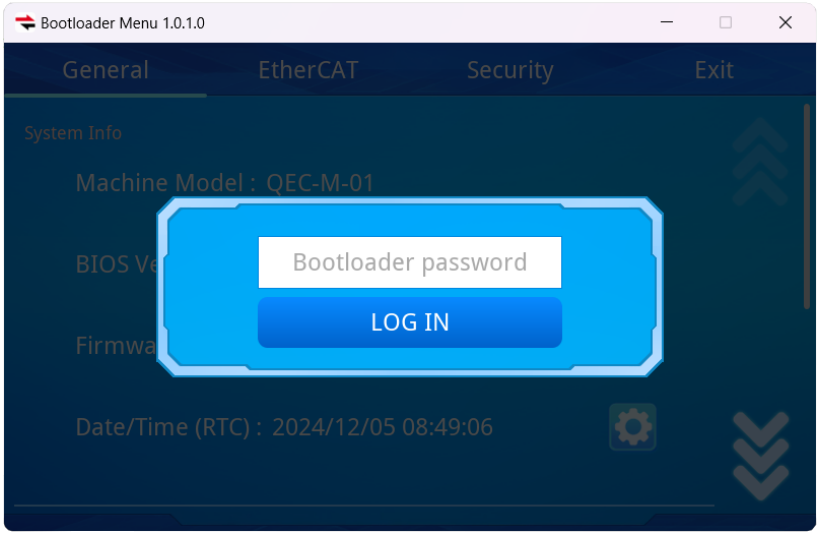
Then, the Security page will look like this.



Users can click the gear icon to change the password and click the garbage can icon to delete the password.



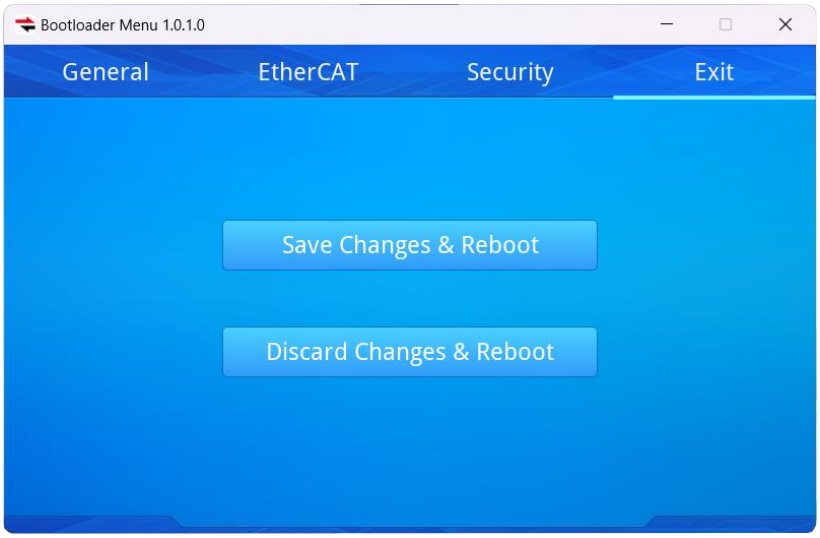
Once users save the bootloader menu configuration, the password will be saved, and the bootloader menu will show when users open it next time.



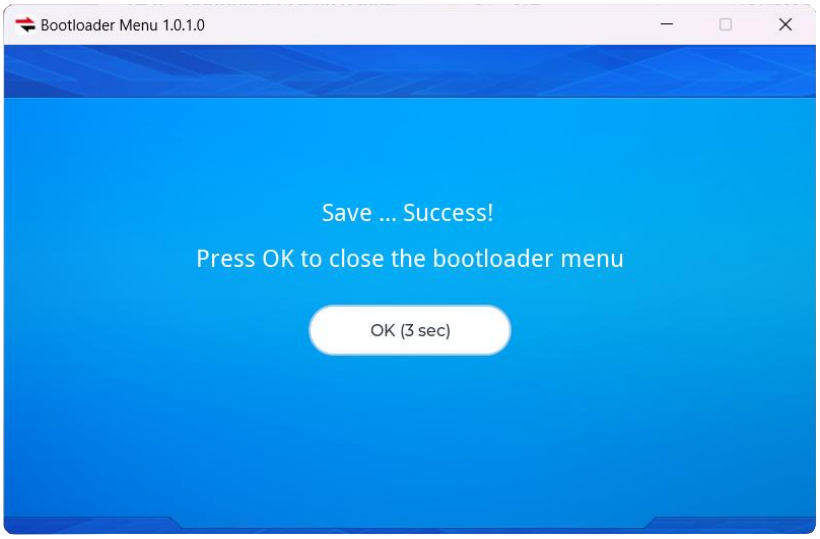
4.6.5 Exit Page

The Exit Page allows users to save or discard changes made in the Bootloader Menu before rebooting the QEC-M-01. This ensures all settings are properly applied or reverted based on user preference.

Exit Page:



- Save Changes & Reboot: If users select this option, all changes made in the Bootloader Menu will be saved. The Bootloader Menu will display a confirmation message: "Save... Success" to let users know that the configuration has been successfully saved.

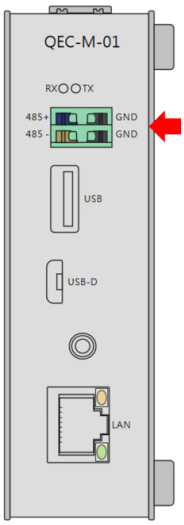
  Users can click OK to close the Bootloader Menu immediately, or wait for the automatic close in 5 seconds.

- Discard Changes & Reboot: Selecting this option will discard any unsaved changes and reboot the QEC-M-01 with the previously saved configuration.
  The Bootloader Menu will close directly without saving.

4.7 RS485 Communication

This section introduces the RS485 configuration and control on your QEC MDevice.
The RS485 Communication section introduces the features and usage of the RS485 interface on the QEC-M-01. RS485 is a versatile serial communication standard, suitable for long-distance, noise-resistant data transmission in industrial environments.
For more detailed information about RS485, please refer to http://en.wikipedia.org/wiki/RS-485.

RS485 Pin Configuration:



RS485 is a versatile communication standard that supports multi-device communication on a shared bus, utilizing differential signaling to enhance noise immunity. It is particularly suited for applications such as industrial automation and Modbus communication. In the 86Duino environment, RS485 communication can be implemented using the Serial Library for basic data transmission or the Modbus Library for protocol-based communication.

4.7.1 Serial Communication

To use the RS485 interface for basic serial communication, refer to the Serial Library documentation.
Users can use Serial485 to configure the RS485 pins on the QEC MDevice. Serial485 is used the same way as Serial1, Serial2, and Serial3 in 86Duino IDE, but the half-duplex mode isn't defined for this port.

Below is a simple example:

```
void setup() {
  Serial485.begin(115200);
  Serial.begin(115200);
  Serial.println("Start");
}

void loop() {
  Serial485.write("hello RS485");
  delay(10);

  for (int i = 0; i < 11; i++) {
    while (Serial485.available() < 1);
    Serial.print(Serial485.read());
    Serial.print(",");
  }
  Serial.println();
}
```

Key Functions:

- Serial.begin(baud_rate): Initializes communication with the specified baud rate.
- Serial.read(): Reads incoming data from the RS485 interface.
- Serial.write(data): Sends data through the RS485 interface.

For more information, refer to the 86Duino Serial Library Reference.

4.7.2 Modbus RTU Communication

For more advanced communication protocols, the RS485 interface can be used with the Modbus Library.
86Diuno IDE supports the Modbus communication protocol, an industrial communication standard published in 1979 for communication between automated electronic devices such as Programmable logic controllers (PLCs).
Modbus is a master/slave based protocol where a master node communicates with multiple slave nodes on the network. Each node has a unique address. When the master node sends a packet to the specified address, only the slave node at the corresponding address receives and parses the packet and executes and responds to commands based on the packet contents.

86Duino's Modbus library has the following features:

- It supports Modbus RTU, TCP, and ASCII sub-protocols.
- Runs as both Modbus master and Modbus slave nodes.
- Support Modbus gateway function

This section provides the foundation for implementing robust RS485 communication using the QEC-M-01. For additional examples and advanced configurations, consult the linked documentation.
For detailed usage, visit the 86Duino Modbus Library Documentation.

Example 1: Send Modbus Master RTU 8 bit data output

- Set the host RS485 baud rate to 115200.
- Configure the Modbus master to use RTU mode and connect it to the host RS485 on QEC MDevice.
- Main Loop Program: The Modbus slave has an ID of 30 and 8 Coils. Write 1 to the odd-numbered Coils and 0 to the even-numbered Coils. After a 1-second interval, write 0 to the odd-numbered Coils and 1 to the even-numbered Coils. Repeat this process in a continuous loop.

Here is the example code:

```
#include "Modbus.h"

ModbusMaster bus;

void setup() {
  Serial.begin(3000000);
  Serial485.begin(115200);    // Set the RS485 baud rate
  bus.begin(MODBUS_RTU, Serial485);    // Initialize Modbus RTU
}

void loop() {
  static bool state = false;
  uint16_t coils = state ? 0xAAAA : 0x5555;    // Odd coils 1, even coils 0 and vice
versa
  bus.writeMultipleCoils(30, 0, 8, &coils);
  state = !state;
  delay(1000);        // Wait for 1 second
}
```

Example 2: Read Modbus Master RTU 8 bit data input

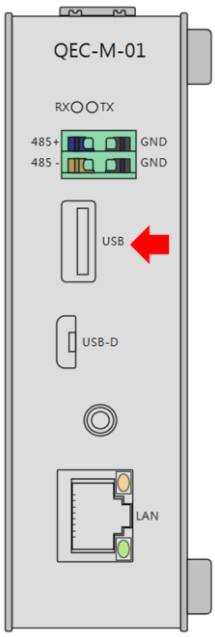
- Set the host RS485 baud rate to 115200.
- Configure the Modbus master to use RTU mode and connect it to the host RS485.
- Main Loop Program: The Modbus slave has an ID of 30. Continuously read the value of the 4th Holding Register and print it using Serial.print every 1 second in a loop.

Here is the example code:

```
#include "Modbus.h"

ModbusMaster bus;

void setup() {
  Serial.begin(3000000);
  Serial485.begin(115200);    // Set the RS485 baud rate
  bus.begin(MODBUS_RTU, Serial485);    // Initialize Modbus RTU
}

void loop() {
  static uint32_t lastTime = 0;
  uint32_t currentTime = millis();
    // Check if one second has passed
    if (currentTime - lastTime >= 1000) {
      uint16_t data[1];
      uint8_t result = bus.readHoldingRegisters(30, 4, 1, data);
      if (result == 0) {
        Serial.print("Holding Register 4 Value: ");
        Serial.println(data[0]);
      } else {
        // Handle error
      }
      lastTime = currentTime;   // Update the last time a write was attempted
    }
}
```

4.8 USB Device Usage

This section ensures proper usage of the USB device and file storage on the QEC-M-01.
The QEC-M-01 features a Standard USB 2.0 port with hot-plug support, allowing users to connect USB storage devices for file storage, transfer, or accessing configuration data.



Users can utilize the SD Library to read from and write to the USB folder. For example, it is possible to create .txt files using this library. The SD library allows for reading from and writing to SD cards in the MicroSD/SD slot of 86Duino. The library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. The file names passed to the SD library functions can include paths separated by forwarding slashes, /, e.g. “directory/filename.txt“. Because the working directory is always the root of the SD card, a name refers to the same file whether or not it includes a leading slash (e.g., “/file.txt” is equivalent to “file.txt“). The library supports opening multiple files.
However, for placing files in a specific folder, the SD Library cannot be used. Instead, users must rely on the standard C language function fopen() to specify the file's location manually.

Warnings: Slot Recommendations:
The QEC-M-01 has four storage slots: A, B, C, and P.

- A and B Slots:These are reserved for system files. Users are strongly discouraged from storing files in these slots, as doing so could corrupt the system and require a factory reset or hardware reprogramming of the 32MB flash.
- C Slot: This slot refers to the EMMC and is recommended for storing user files safely.
- P Slot: This slot refers to the USB device and is also suitable for user file storage.

Example 1: Save .txt in USB disk

Below is an example of creating a file using fopen() in C language. We save a .txt file on an external USB disk. This demonstrates specifying the file location and ensuring safe storage in the appropriate slot.

Here is the example code:

```
char* strs[] = {"Hello, world", "Hello, world2", "Hello, world3"};

void setup() {
   FILE *fp;
   char str[256];

   Serial.begin(115200);

   // Write strings to the test.txt file in the P slot
   fp = fopen("P:\\test.txt", "w");
   for (int i=0; i<sizeof(strs)/sizeof(char*); i++)
      fprintf(fp, "%s\n", strs[i]);
   fclose(fp);

   // Read the strings and print them
   fp = fopen("P:\\test.txt", "r");
   while (fgets(str, sizeof(str), fp)!=NULL ) {
      Serial.print(str);
   }
   fclose(fp);
}

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

}
```

Additional Example: Save .txt in EMMC storage

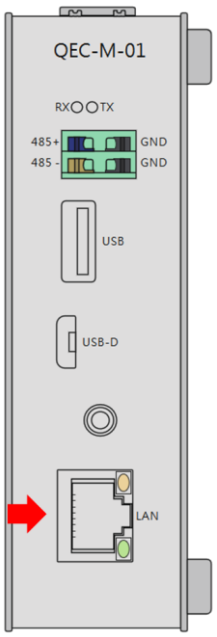
Below is an example of creating a file using fopen() in C language. We save a .txt file on an internal EMMC storage. This demonstrates specifying the file location and ensuring safe storage in the appropriate slot.

Here is the example code:

```
char* strs[] = {"Hello, world", "Hello, world2", "Hello, world3"};

void setup() {
   FILE *fp;
   char str[256];

   Serial.begin(115200);

   // Write strings to the test.txt file in the P slot
   fp = fopen("C:\\test.txt", "w");
   for (int i=0; i<sizeof(strs)/sizeof(char*); i++)
      fprintf(fp, "%s\n", strs[i]);
   fclose(fp);

   // Read the strings and print them
   fp = fopen("C:\\test.txt", "r");
   while (fgets(str, sizeof(str), fp)!=NULL ) {
      Serial.print(str);
   }
   fclose(fp);
}

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

}
```

4.9 Giga LAN Configuration

This section introduces the Giga LAN configuration and control for your QEC MDevice.
The QEC-M-01 features one Giga LAN port dedicated to external Ethernet communication for general network use.



To drive the Giga LAN, you can utilize either the Ethernet Library or the Modbus Library in the 86Duino environment.

Ethernet Library:

- Provides tools for general networking, including:
- Static and dynamic IP configuration.
- TCP/UDP communication.
- Hosting web servers.

For more details, refer to the Ethernet Library Documentation.

Modbus Library:

- Allows communication with Modbus TCP devices using Ethernet.
- Ideal for industrial automation and monitoring applications.

For more information, refer to the Modbus Library Documentation.

4.9.1 Ethernet Communication

This section introduces the Ethernet configuration and control for your QEC MDevice.
The CPU of the QEC-M series contains a built-in 10/100/1000Mbps LAN interface, which can access via the Ethernet library. The Arduino Ethernet Shield isn't needed. This library can serve as either a server accepting incoming connections or a client making outgoing ones. In 86Duino Coding, the library supports up to four concurrent connections (incoming or outgoing or a combination); and in later versions, up to 128 concurrent connections.

Example 1: DHCP-based IP printer

This sketch uses the DHCP extensions to the Ethernet library to get an IP address via DHCP and print the address obtained.

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
EthernetClient client;

void setup() {
  Serial.begin(115200);
  while (!Serial);
  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0)
    Serial.println("Failed to configure Ethernet using DHCP");

  // print your local IP address:
  Serial.print("My IP address: ");
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
    // print the value of each byte of the IP address:
    Serial.print(Ethernet.localIP()[thisByte], DEC);
    Serial.print(".");
  }
  Serial.println();
}

void loop() {
}
```

Example 2: Web Server

A simple web server that shows the value of the analog input pins.

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
EthernetServer server(80);

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

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connnection: close");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
```

```
          // add a meta refresh tag, so the browser pulls again every 5 seconds:
          client.println("<meta http-equiv=\"refresh\" content=\"5\">");
          // output the value of each analog input pin
          for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
            int sensorReading = analogRead(analogChannel);
            client.print("analog input ");
            client.print(analogChannel);
            client.print(" is ");
            client.print(sensorReading);
            client.println("<br />");
          }
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          // you're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // you've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
    Serial.println("client disonnected");
  }
}
```

Example 3: Web client

This sketch connects to a website (http://www.google.com).

Here is the example code:

```
#include <Ethernet.h>

byte mac[] = {  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress server(173, 194, 33, 104); // Google
EthernetClient client;

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

  if (Ethernet.begin(mac) == 0)
    Serial.println("Failed to configure Ethernet using DHCP");
  delay(1000);
  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop() {
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }
}
```

Example 4: MySQL Connector

This example demonstrates connecting to a MySQL server from a QEC MDevice Ethernet port. For more information and documentation, visit the wiki: https://github.com/ChuckBell/MySQL_Connector_Arduino/wiki.

Here is the example code:

```
#include <Ethernet.h>
#include <MySQL_Connection.h>

byte mac_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress server_addr(10,0,1,35);  // IP of the MySQL *server* here
char user[] = "root";              // MySQL user login username
char password[] = "secret";        // MySQL user login password

EthernetClient client;
MySQL_Connection conn((Client *)&client);

void setup() {
  Serial.begin(115200);
  while (!Serial); // wait for serial port to connect
  Ethernet.begin(mac_addr);
  Serial.println("Connecting...");
  if (conn.connect(server_addr, 3306, user, password)) {
    delay(1000);
    // You would add your code here to run a query once on startup.
  }
  else
    Serial.println("Connection failed.");
  conn.close();
}

void loop() {

}
```

4.9.2 Modbus TCP Communication

This section introduces the Modbus TCP configuration and control for your QEC MDevice. Please refer to 4.7.2 Modbus RTU Communication about 86Duino IDE Modbus Library.

Example 1: Simple Modbus Master TCP

This example uses the Ethernet and ModbusMaster libraries to establish Modbus TCP communication over Ethernet. The program initializes an Ethernet connection with a static IP address and configures a Modbus TCP master to interact with a slave device with ID 11.

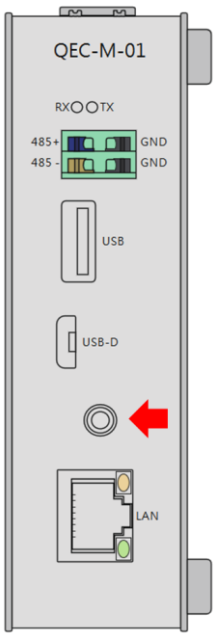
Here is the example code:

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

ModbusMaster bus;
ModbusMasterNode node;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 101);
IPAddress serverIp(192, 168, 1, 102);

int led = 0;
uint32_t value = 0;

void setup()
{
  Ethernet.begin(mac, localIp);

  /* Modbus TCP Mode via Ethernet. */
  bus.begin(MODBUS_TCP, serverIp);

  /* Slave node initialize. */
  node.attach(11, bus);
}

void loop()
{
  uint16_t reg[2];
```

```
  /* Write 1 coil to address 0 of the slave with ID 11. */
  node.writeSingleCoil(0, led);

  /* Write 2 word to holding registers address 16 of the slave with ID 11. */
  reg[0] = value & 0xFFFF;
  reg[1] = (value >> 16) & 0xFFFF;
  node.setTransmitBuffer(0, reg[0]);
  node.setTransmitBuffer(1, reg[1]);
  node.writeMultipleRegisters(16, 2);

  /* Read 2 word from holding registers address 16 of the slave with ID 11. */
  node.readHoldingRegisters(16, 2);
  Serial.print("From Node 1 Holding Register: ");
  value = node.getResponseBuffer(0) | (node.getResponseBuffer(1) << 16);
  Serial.println(value);

  /* Read 1 word from input registers address 2 of the slave with ID 11. */
  node.readInputRegisters(2, 1);
  Serial.print("             Input Register: ");
  Serial.print(node.getResponseBuffer(0));
  Serial.println();

  led = !led;
  value++;

  delay(1000);
}
```

4.10 Audio Usage

This section introduces the HD Audio control for your QEC MDevice.
The QEC-M-01 features one HD Audio port, allows users play music via Audio Library.



This library is available from 86Duino Coding 102 and implements the API of Arduino Due's Audio library to access the onboard HD Audio interface of 86Duino One, EduCake, and PLC.

Example 1: Simple Audio Player

Demonstrates the use of the Audio library for the QEC-M-01.
This example demonstrates the use of the Audio library with the QEC-M-01 to play a .wav audio file stored on a USB storage device. The program reads audio data from the file, processes it using the Audio library, and outputs it in real-time.

Here is the example code:

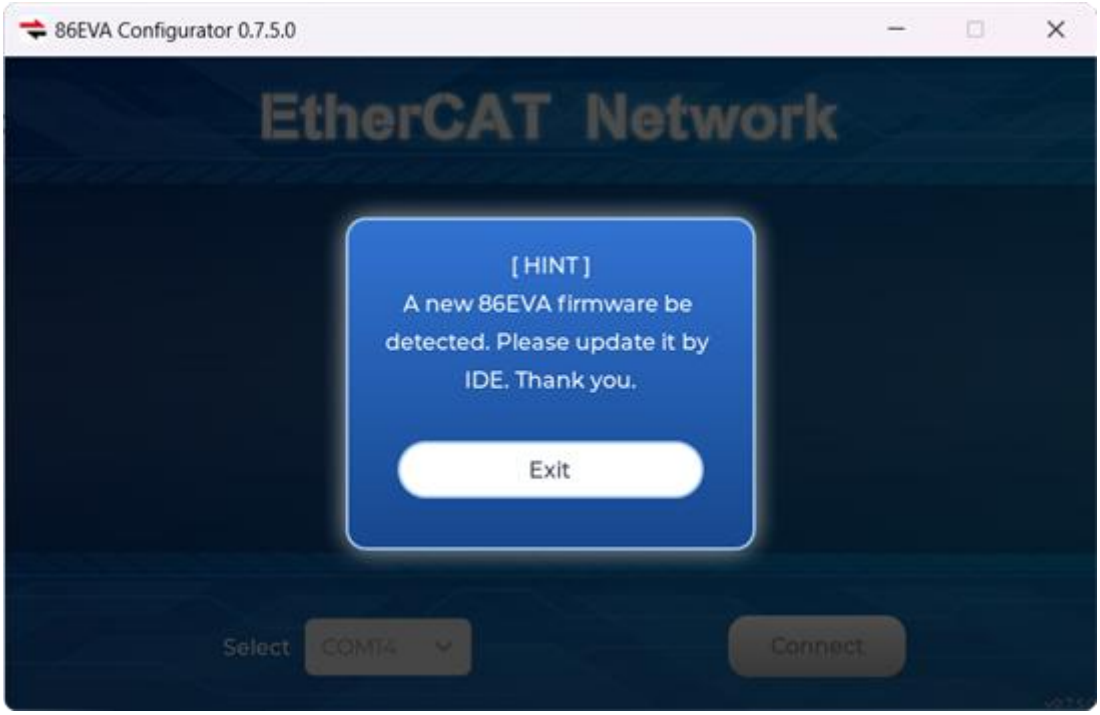
```
#include <SD.h>
#include <Audio.h>

void setup() {
  Serial.begin(9600);
  SD.setBank(USBDISK);
  Serial.print("Initializing SD card...");
  if (!SD.begin(4)) {
```

```
    Serial.println(" failed!");
    return;
  }
  Serial.println(" done.");
  Audio.begin(88200, 100);
}

void loop() {
  int count=0;
  File myFile = SD.open("test.wav");
  if (!myFile) {
    Serial.println("error opening fire.wav");
    while (true);
  }

  const int S=1024;
  short buffer[S];

  Serial.print("Playing");

  while (myFile.available()) {
    myFile.read(buffer, sizeof(buffer));
    int volume = 1024;
    Audio.prepare(buffer, S, volume);
    Audio.write(buffer, S);
    count++;
    if (count == 100) {
      Serial.print(".");
      count = 0;
    }
  }
  myFile.close();

  Serial.println("End of file. Thank you for listening!");
  while (true) ;
}
```

4.11 Troubleshooting

4.11.1 Cannot Successfully Upload the 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:

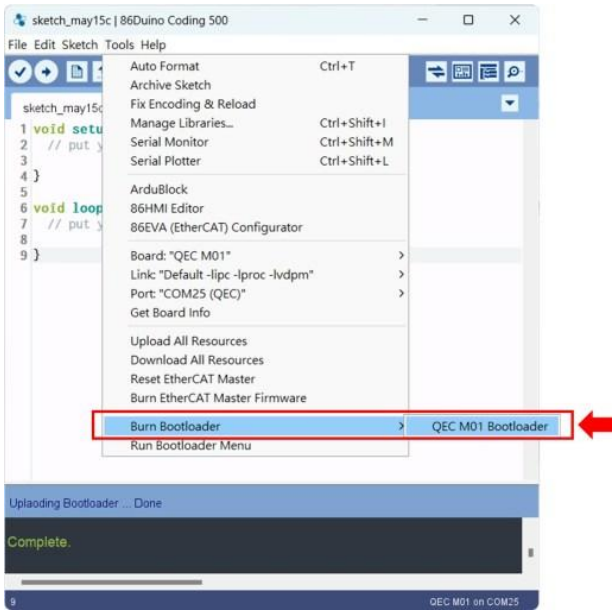
Step 1: Setting up QEC-M

1. Download and install 86Duino IDE 500 (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-01" (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.

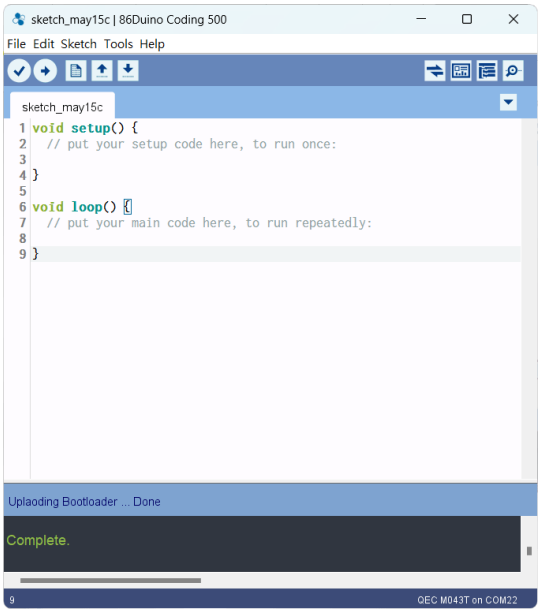
Step 2: Click “Burn Bootloader” button

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-01:



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.

Ch. 5

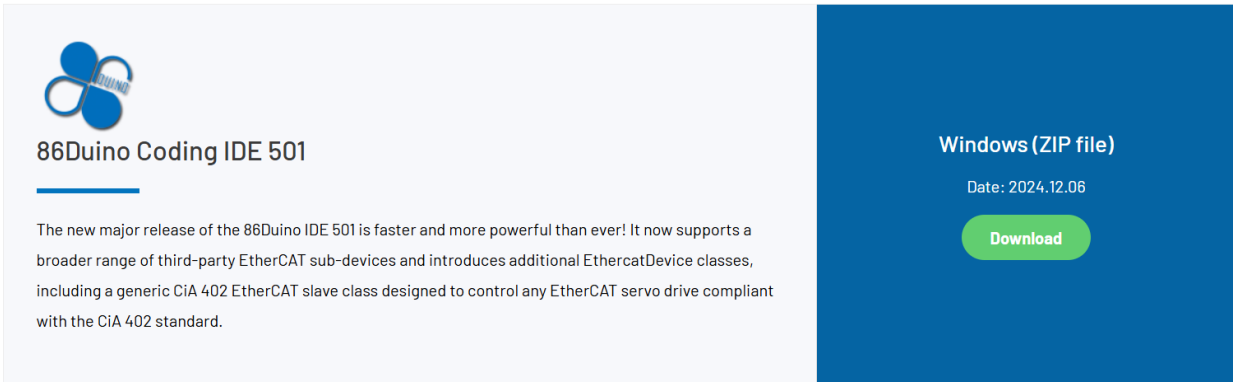
Software Function

5.1 Software Description

The 86Duino Coding IDE 500+ developed by the QEC team designed specifically for industrial-field control systems, bringing simple and powerful functions into related industrial fields through the open-source Arduino.

The 86Duino integrated development environment (IDE) software makes it easy to write code and upload it to 86Duino boards. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software.

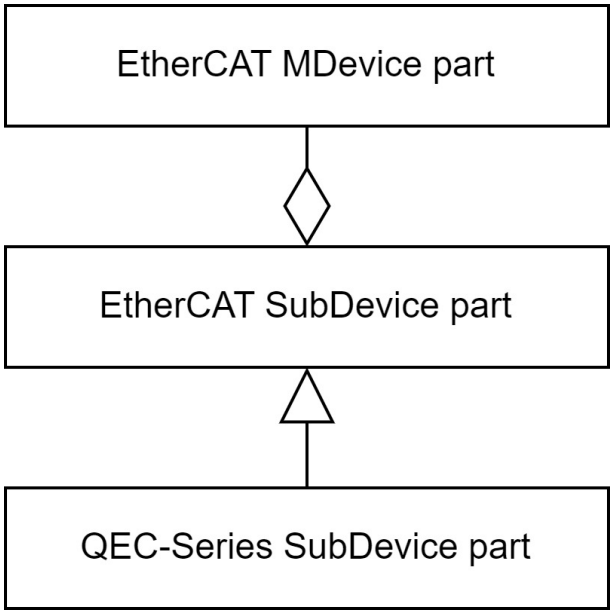
Please visit qec.tw for 86Duino Coding IDE 501+ details.



You can Download here: https://www.qec.tw/software/.

5.2 EtherCAT Function List

QEC-MDevice is an EtherCAT MDevice library implemented in C/C++, which includes classes for the MDevice, generic SubDevice, CiA 402 SubDevice, and dedicated classes for QEC series SubDevices. These classes not only have clearly defined responsibilities but also consider future extensibility.



These classes can be divided into three parts as follows:

- EtherCAT MDevice
  The *EtherCAT MDevice part* not only provides various and flexible MDevice configuration and operation functions but also offers diverse EtherCAT SubDevice operation functions for invocation by the EtherCAT SubDevice part.
- EtherCAT SubDevice
  The *EtherCAT SubDevice part* provides generic EtherCAT SubDevice classes, which can operate functions such as PDOs, CoE, FoE, and also includes CiA 402 SubDevice generic class.
- QEC-Series SubDevice
  The *QEC-Series SubDevice part* provides dedicated functions for ICOP's QEC series SubDevices, enabling users to code in a more user-friendly and concise manner.

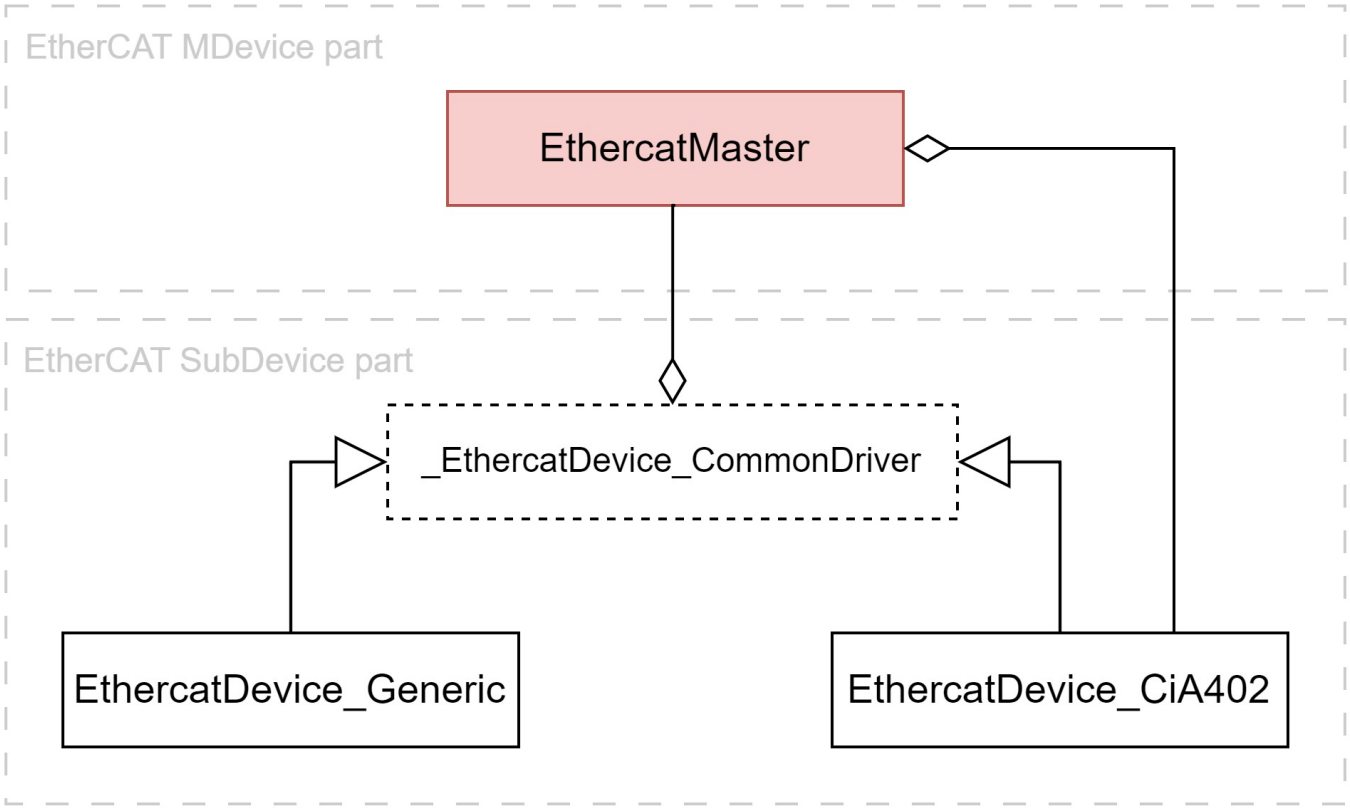
The list below introduces the EtherCAT Library API functions of our QEC MDevice.
Please visit the EtherCAT Library API User Manual for details on the API Function.

5.2.1 EtherCAT MDevice

The EtherCAT MDevice part not only provides various and flexible MDevice configuration and operation functions but also offers diverse EtherCAT SubDevice operation functions for invocation by the EtherCAT SubDevice part.

EthercatMaster is the only class in the EtherCAT MDevice part, it serves as a crucial communication bridge with the EtherCAT firmware. In the Dual-System communication aspect, its responsibilities include communication interface initialization, process data exchange cyclically, handling acyclic transfer interfaces, and managing interrupt events. In the API aspect, it provides functions related to MDevice initialization, MDevice control, and access to SubDevice information.

The main class relationship between the EtherCAT MDevice part and the EtherCAT SubDevice part is association, with the EtherCAT SubDevice part depending on the EtherCAT MDevice part. The class relationships of EthercatMaster are illustrated in the following diagram:



- There is an association between EthercatMaster and _EthercatDevice_CommonDriver, with _EthercatDevice_CommonDriver depending on EthercatMaster.
- There is an association between EthercatMaster and EthercatDevice_CiA402, with EthercatMaster depending on EthercatDevice_CiA402.

Functions:

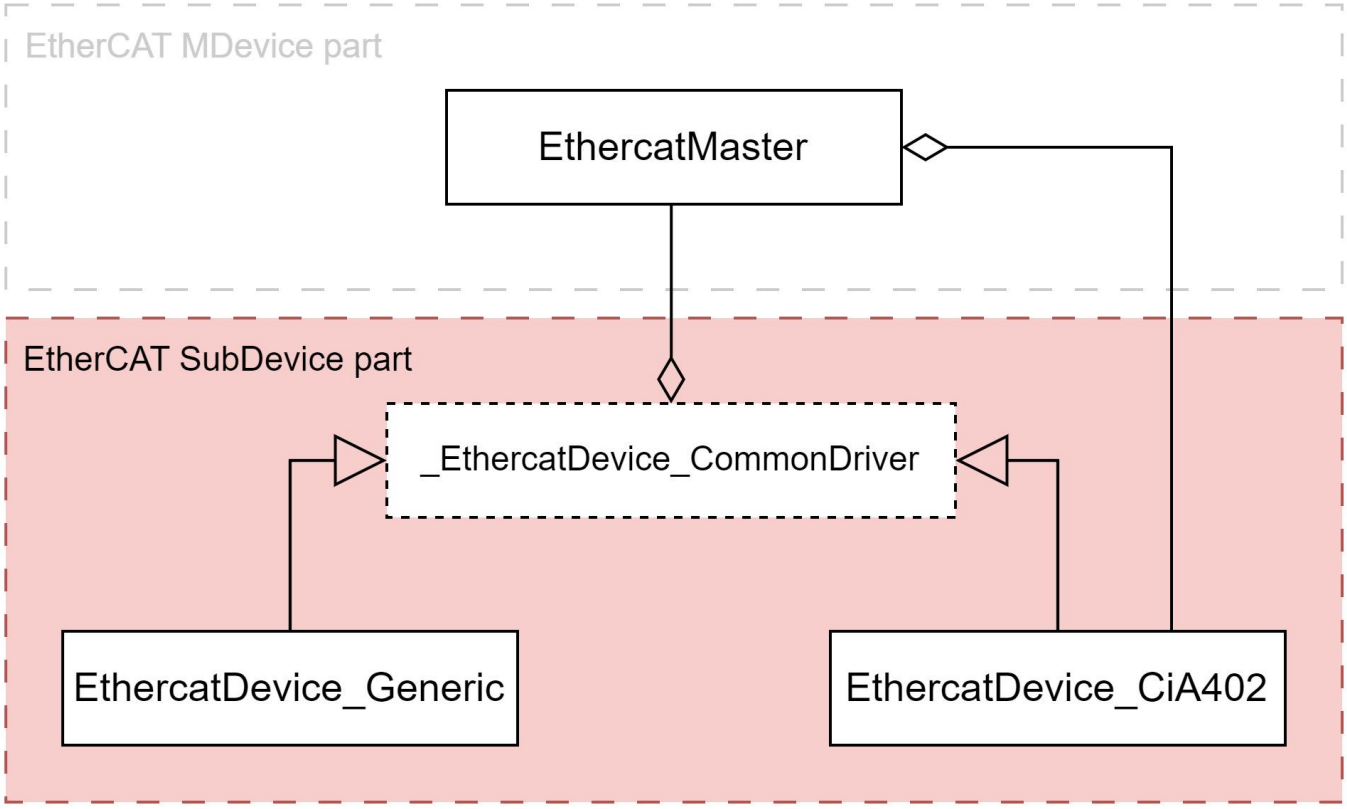
Function Name	Description	Callback Available
Initialization-related functions		
begin()	Initialize the EtherCAT MDevice.	
end()	Deinitialize the EtherCAT MDevice.	
isRedundancy()	Check if the EtherCAT MDevice has cable redundancy enabled.	O
libraryVersion()	Get the EtherCAT MDevice library version.	O
firmwareVersion()	Get the EtherCAT firmware version.	O
readSettings()	Read the current EtherCAT MDevice settings.	
saveSettings()	Save the EtherCAT MDevice settings.	
Control-related functions		
start()	Start the EtherCAT MDevice.	
stop()	Stop the EtherCAT MDevice.	
update()	Update process data and handle acyclic commands.	O
setShiftTime()	Set the Global Shift Time for DC-Synchronous mode.	
getShiftTime()	Get the Global Shift Time for DC-Synchronous mode.	O
getSystemTime()	Get the system time of the current cycle.	O
getWorkingCounter()	Get the working counter for the current cycle.	O
getExpectedWorkingCounter()	Get the expected working counter.	O
Callback-related functions		
attachCyclicCallback()	Register a cyclic callback.	
detachCyclicCallback()	Unregister cyclic callback.	
attachErrorCallback()	Register an error callback.	
detachErrorCallback()	Unregister error callback.	
attachEventCallback()	Register an event callback.	
detachEventCallback()	Unregister event callback.	
errGetCableBrokenLocation1()	Get the cable broken location 1 in error callback.	O[1]
errGetCableBrokenLocation2()	Get the cable broken location 2 in error callback.	O[1]
evtGetMasterState()	Get the EtherCAT MDevice state in event callback.	O[2]
SubDevice information related functions		
getSlaveCount()	Get the number of SubDevices on the network.	O
getVendorID()	Get the vendor ID of the specified SubDevice.	O
getProductCode()	Get the product code of the specified SubDevice.	O
getRevisionNumber()	Get the revision number of the specified SubDevice.	O
getSerialNumber()	Get the serial number of the specified SubDevice.	O
getAliasAddress()	Get the alias address of the specified SubDevice.	O
getSlaveNo()	Find the sequence number of the matching EtherCAT SubDevice on the network.	O

- Note 1: This function can only be called in error callback.
- Note 2: This function can only be called in event callback.

5.2.2 EtherCAT SubDevice

The EtherCAT SubDevice part provides generic EtherCAT SubDevice classes, which can operate functions such as PDOs, CoE, FoE, and also includes CiA 402 SubDevice generic class.

The main class relationship between the EtherCAT SubDevice part and the EtherCAT MDevice part is association, with the EtherCAT SubDevice part depending on the EtherCAT MDevice part. As shown in the diagram below, there is an association relationship between _EthercatDevice_CommonDriver and EthercatMaster.



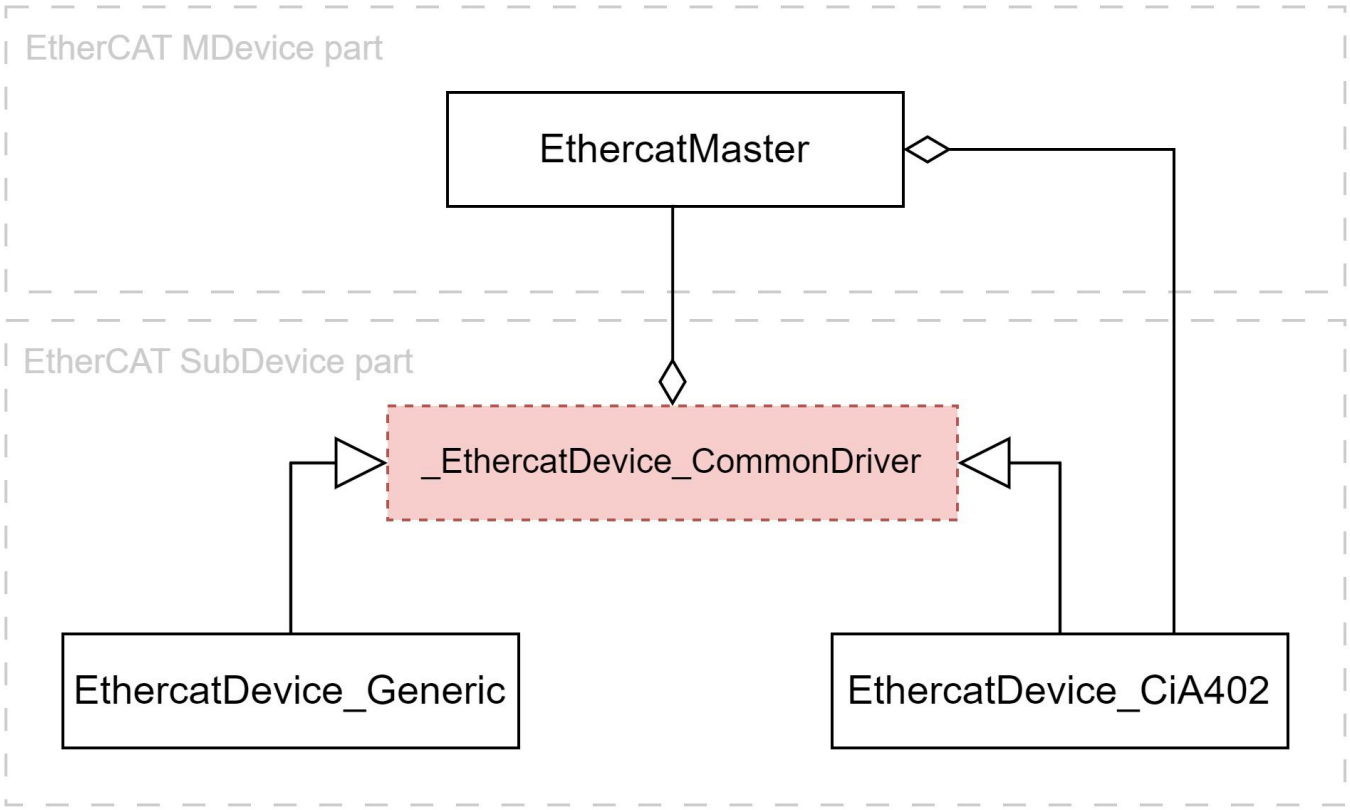
Classes:

- _EthercatDevice_CommonDriver
- EthercatDevice_Generic
- EthercatDevice_CiA402

5.2.2.1 _EthercatDevice_CommonDriver

_EthercatDevice_CommonDriver is an abstract class that not only features functions for accessing SubDevice information but also provides various EtherCAT function access methods, including PDO, SII, CoE, FoE, DC, etc. All EtherCAT SubDevice classes inherit from it.

The class relationships of _EthercatDevice_CommonDriver are illustrated in the following diagram:



- There is an association between EthercatMaster and _EthercatDevice_CommonDriver, with _EthercatDevice_CommonDriver depending on EthercatMaster.
- All other EtherCAT SubDevice classes inherit from _EthercatDevice_CommonDriver.

WARNING: *Prohibited from declaring objects using this class.*

Functions:

Function Name	Description	Callback Available
SubDevice information related functions		
`getVendorID()`	Get the vendor ID.	O
`getProductCode()`	Get the product code.	O
`getRevisionNumber()`	Get the revision number.	O
`getSerialNumber()`	Get the serial number.	O
`getAliasAddress()`	Get the alias address.	O
`getSlaveNo()`	Get the sequence ID on the EtherCAT network.	O

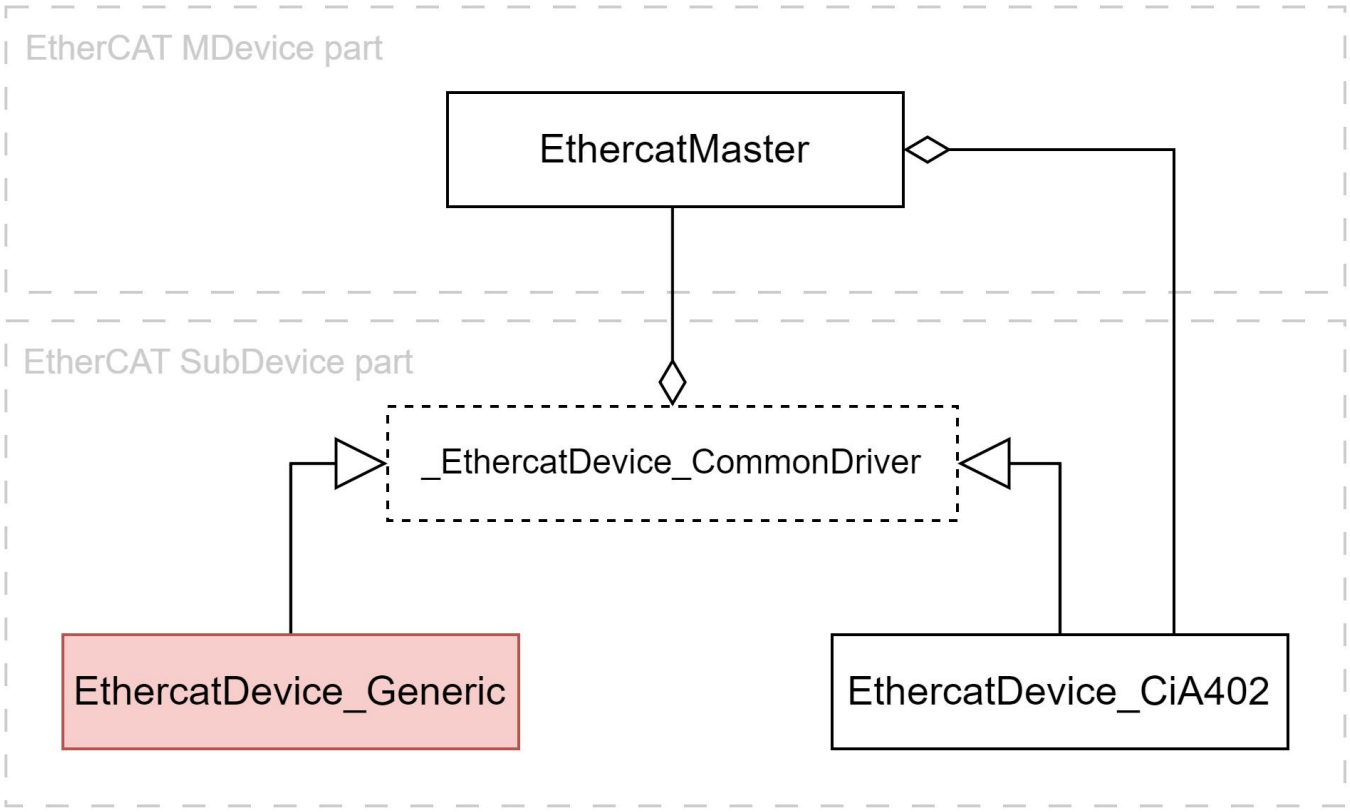
getDeviceName()	Get the device name.	O
getMailboxProtocol()	Get the supported mailbox protocol types.	O
getCoEDetails()	Get the details about CoE supported.	O
getFoEDetails()	Get the details about FoE supported.	O
getEoEDetails()	Get the details about EoE supported.	O
getSoEChannels()	Get the number of SoE channels supported.	O
isSupportDC()	Check if the EtherCAT SubDevice device has DC supported.	O
PDO access functions		
pdoBitWrite()	Write 1-bit output process data.	O
pdoBitRead()	Read 1-bit input process data.	O
pdoGetOutputBuffer()	Get the memory pointer of output process data.	O
pdoGetInputBuffer()	Get the memory pointer of input process data.	O
pdoWrite()	Write multiple bytes of output process data.	O
pdoWrite8()	Write 8-bit output process data.	O
pdoWrite16()	Write 16-bit output process data.	O
pdoWrite32()	Write 32-bit output process data.	O
pdoWrite64()	Write 64-bit output process data.	O
pdoRead()	Read multiple bytes of input process data.	O
pdoRead8()	Read 8-bit input process data.	O
pdoRead16()	Read 16-bit input process data.	O
pdoRead32()	Read 32-bit input process data.	O
pdoRead64()	Read 64-bit input process data.	O
CoE communication functions		
sdoDownload()	Write multiple bytes of data to the object.	
sdoDownload8()	Write 8-bit value to the object.	
sdoDownload16()	Write 16-bit value to the object.	
sdoDownload32()	Write 32-bit value to the object.	
sdoDownload64()	Write 64-bit value to the object.	
sdoUpload()	Read multiple bytes of data from the object.	
sdoUpload8()	Read 8-bit value from the object.	
sdoUpload16()	Read 16-bit value from the object.	
sdoUpload32()	Read 32-bit value from the object.	
sdoUpload64()	Read 64-bit value from the object.	
getODlist()	Get a list of objects existing in the object dictionary.	
getObjectDescription()	Get the object description of the object.	
getEntryDescription()	Get the entry description of the object.	
FoE communication functions		
readFoE()	Read a file from the EtherCAT SubDevice.	
writeFoE()	Write a file to the EtherCAT SubDevice.	
DC configuration functions		
setDc()	Configure DC parameters.	

SII EEPROM access functions		
writeSII()	Write multiple bytes of data to the SII EEPROM.	
writeSII8()	Write 8-bit value to the SII EEPROM.	
writeSII16()	Write 16-bit value to the SII EEPROM.	
writeSII32()	Write 32-bit value to the SII EEPROM.	
readSII()	Read multiple bytes of data from the SII EEPROM.	
readSII8()	Read 8-bit value from the SII EEPROM.	
readSII16()	Read 16-bit value from the SII EEPROM.	
readSII32()	Read 32-bit value from the SII EEPROM.	
Initialization-related functions		
attach()	Initialize the object of this EtherCAT SubDevice class.	
detach()	Deinitialize the object of this EtherCAT SubDevice class.	

5.2.2.2 EthercatDevice_Generic

EthercatDevice_Generic is a generic EtherCAT SubDevice class that can be used to control all EtherCAT SubDevices, including accessing SubDevice information, PDO, CoE, FoE, DC, and more.

The class relationships of *EthercatDevice_Generic* are illustrated in the following diagram:



- EthercatDevice_Generic inherits from _EthercatDevice_CommonDriver.

Base Class

- _EthercatDevice_CommonDriver

Functions

Function Name	Description	Callback Available
Initialization-related functions		
attach()	Initialize the object of this EtherCAT SubDevice class.	
detach()	Deinitialize the object of this EtherCAT SubDevice class.	

5.2.2.3 EthercatDevice_CiA402

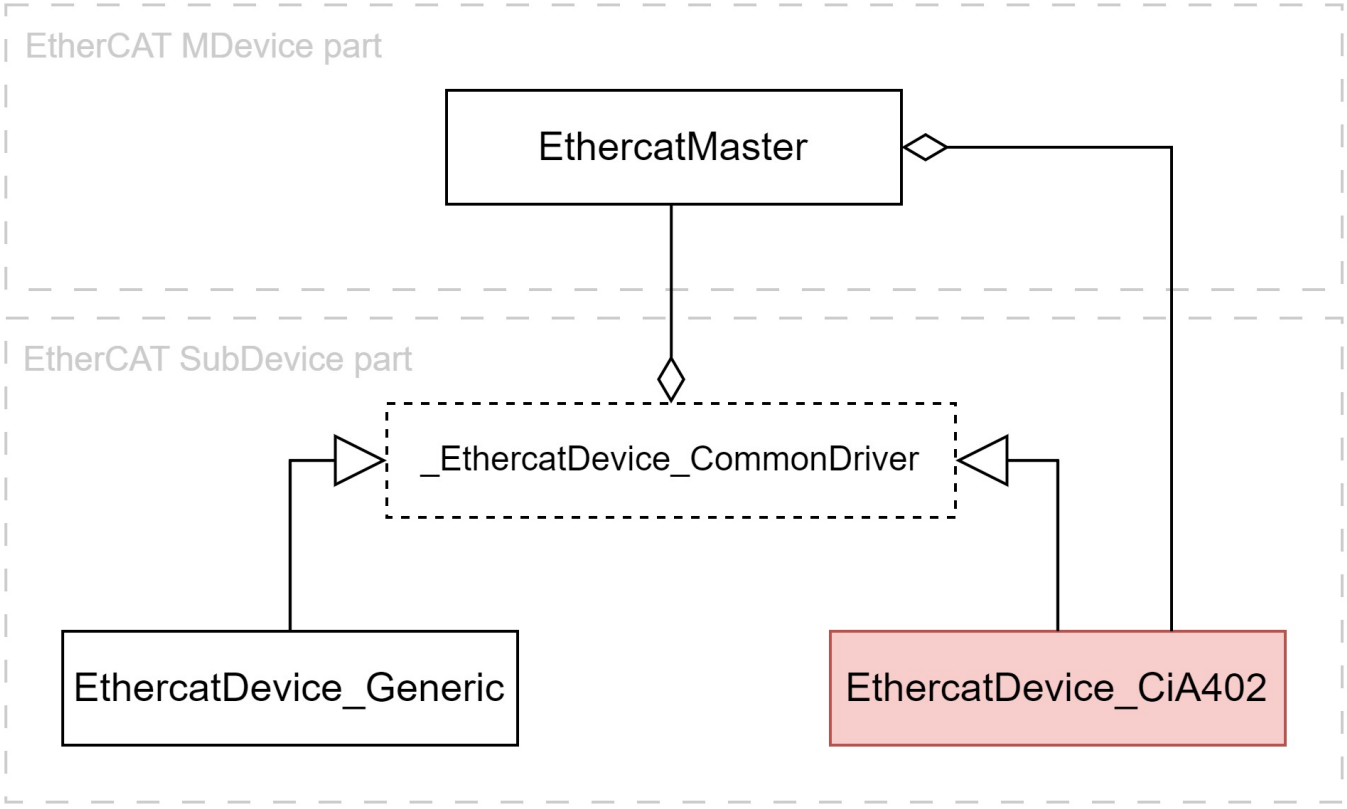
EthercatDevice_CiA402 is a generic CiA 402 EtherCAT SubDevice class designed to control any EtherCAT servo drive that supports the CiA 402 standard. It provides access functions for commonly used CiA 402 objects and operation functions for several CiA 402 operation modes and function groups, including:

- **Operation Modes**
  - Profile Position (pp)
  - Profile Velocity (pv)
  - Profile Torque (tq)
  - Homing (hm)
  - Cyclic Synchronous Position (csp)
  - Cyclic Synchronous Velocity (csv)
  - Cyclic Synchronous Torque (cst)
- **Function Groups**
  - Touch Probe

For more detailed information about CiA 402, please refer to the following documents:

- CiA Draft Standard 402: CANopen device profile drives and motion control
- ETG.6010 Implementation Directive for CiA402 Drive Profile
- User manual for the currently used CiA 402 drive device

The class relationships of EthercatDevice_CiA402 are illustrated in the following diagram:



- *EthercatDevice_CiA402* inherits from *_EthercatDevice_CommonDriver*.

Base Class

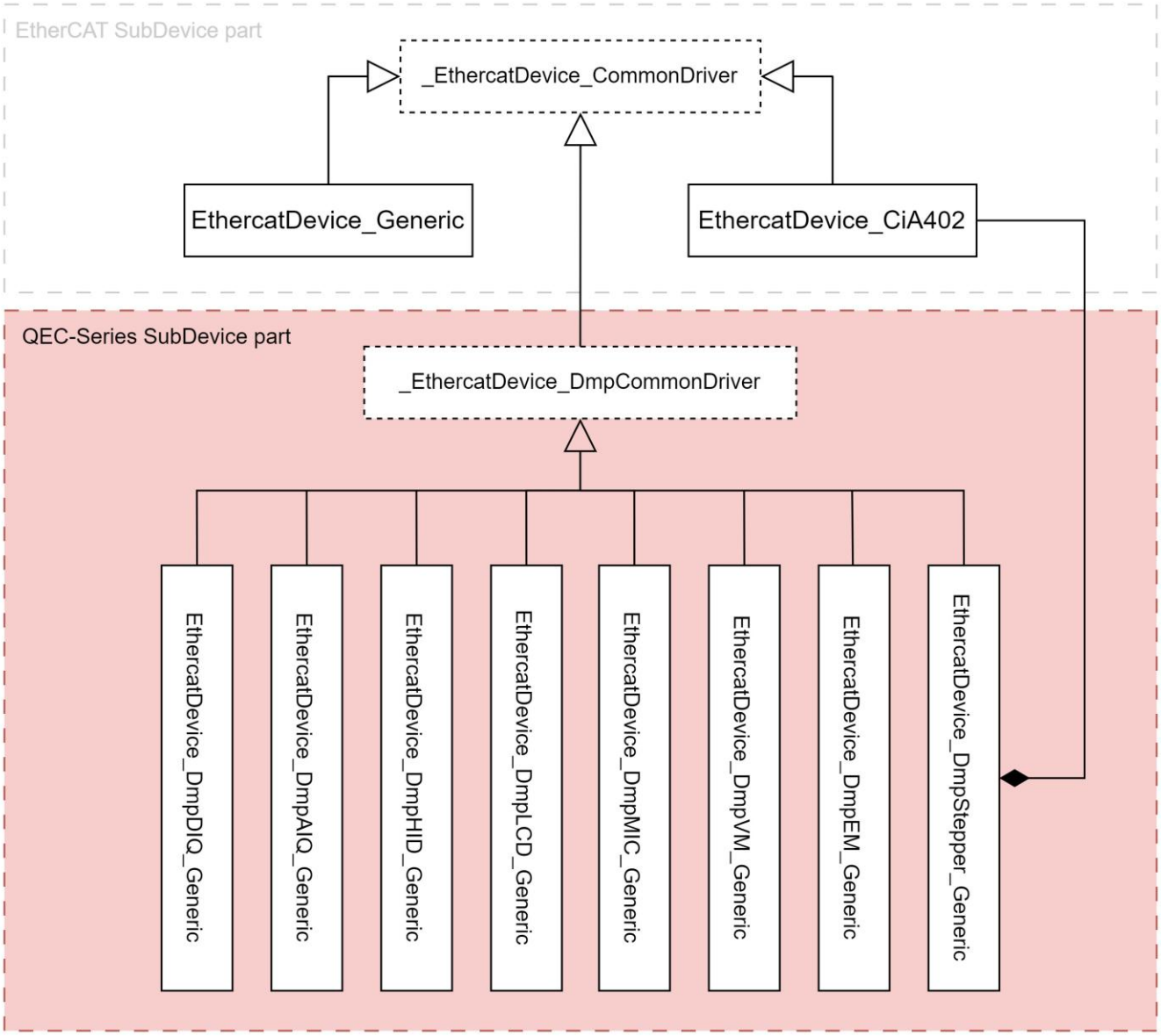
- _EthercatDevice_CommonDriver

For more detailed information about CiA 402 API, please refer to the EtherCAT Library API User Manual - QEC.

5.2.3 QEC-Series SubDevice

The QEC-Series SubDevice part provides dedicated functions for ICOP's QEC series SubDevices, enabling users to code in a more user-friendly and concise manner.

The main class relationship between the QEC-Series SubDevice part and the EtherCAT SubDevice part is association, with the QEC-Series SubDevice part depending on the EtherCAT SubDevice part. As shown in the diagram below, there is an association relationship between _EthercatDevice_DmpCommonDriver and _EthercatDevice_CommonDriver.



Classes

- _EthercatDevice_DmpCommonDriver
- EthercatDevice_DmpDIQ_Generic

- EthercatDevice_DmpAIQ_Generic
- EthercatDevice_DmpHID_Generic
- EthercatDevice_DmpLCD_Generic
- EthercatDevice_DmpStepper_Generic

5.3 Additional Resources

If you want to learn more about the libraries available in the 86Duino IDE or explore the details of the 86Duino programming language, please visit the following links:

- **EtherCAT Library API User Manual:** QEC MDevice is an EtherCAT MDevice compatible with 86Duino Coding IDE 500+. It offers real-time EtherCAT communication between EtherCAT MDevice and EtherCAT Sub-devices. Please refer detailed information at https://www.qec.tw/ethercat/api/ethercat-library-api-user-manual/.

- **86Duino IDE Libraries:** Find an extensive list of libraries supported by 86Duino IDE, along with detailed documentation and examples at https://www.qec.tw/86duino/libraries/.

- **86Duino Language Reference:** Learn about the 86Duino programming language, including its syntax, functions, and usage in the official 86Duino Language Reference at https://www.qec.tw/86duino/86duino-language-reference/.

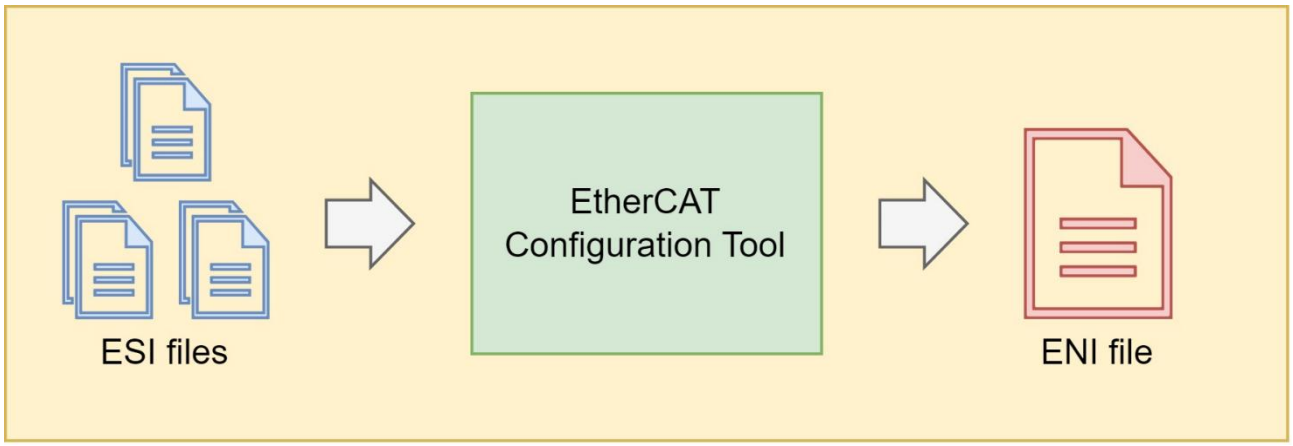
These resources provide valuable information for both beginners and experienced developers using the 86Duino platform. By exploring these links, you can harness the full potential of 86Duino IDE and create innovative projects.

Happy coding with 86Duino IDE!

Appendix

A1. About ENI Configuration in 86Duino IDE

The EtherCAT Network Information (ENI) contains the necessary settings to configure an EtherCAT network. The XML-based file contains general information about the EtherCAT MDevice and the configurations of every SubDevice connected to the EtherCAT MDevice. The EtherCAT Configuration Tool reads the ESI files or online scans the network for all SubDevices, then user can configures relevant EtherCAT settings, such as PDO mapping and enabling DC, and then export the ENI file.



The EtherCAT Technology Group specifies that the EtherCAT MDevice Software must support at least one of the following in the Network Configuration section: Online Scanning or Reading ENI. This library, however, supports both. In the case of Reading ENI, this library currently extracts only partial information from the ENI file for network configuration.

The extracted information includes the following:

EtherCATConfig : Config : Slave : Info

- Elements
  - VendorId
  - ProductCode
- Attribute
  - Identification : Value
- Purpose
  Used to check whether the EtherCAT SubDevices on the network match the SubDevices specified in the ENI file. The checking rules are as follows:
  - Check if the number of SubDevices in the ENI file matches the number of SubDevices on the network.
  - For SubDevices in the ENI file with the Identification: Value attribute, check if there are SubDevices on the network with matching Alias Address and Identification: Value attribute, as well as Vendor ID and Product Code. If such SubDevices exist, it indicates a successful match.
  - For SubDevices with the Identification: Value attribute that fail to match, or those without this attribute, check if the Vendor ID and Product Code of the SubDevice with the same sequence number on the network match.

EtherCATConfig : Config : Slave : Mailbox

- Elements
  - Send : MailboxSendInfoType : Start
  - Recv : MailboxRecvInfoType : Start
- Purpose
  Used to configure the mailbox Physical Start Address of an EtherCAT SubDevice.

EtherCATConfig : Config : Slave : Mailbox : CoE

- Elements
  - InitCmds : InitCmd : Index
  - InitCmds : InitCmd : SubIndex
  - InitCmds : InitCmd : Data
  - InitCmds : InitCmd : Timeout
- Attribute
  - InitCmds : InitCmd : CompleteAccess
- Purpose
  After switching the EtherCAT state machine to the Pre-Operational state, execute the CoE initialization commands for the EtherCAT SubDevices in `EthercatMaster::begin()`.

EtherCATConfig : Config : Slave : ProcessData

- Elements
  - Recv : BitLength
  - Send : BitLength
- Purpose
  The bit length of the output process data and input process data of an EtherCAT SubDevice is provided to the firmware for relevant configuration.

EtherCATConfig : Config : Slave : ProcessData : Sm

- Elements
  - SyncManagerSettings : StartAddress
  - SyncManagerSettings : ControlByte
  - SyncManagerSettings : Enable
- Purpose
  Used to configure the Sync Manager registers for the process data of an EtherCAT SubDevice.

EtherCATConfig : Config : Slave : DC

- Elements
  - CycleTime0
  - CycleTime1
  - ShiftTime
- Purpose

Used to configure the DC parameters of an EtherCAT SubDevice.

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.