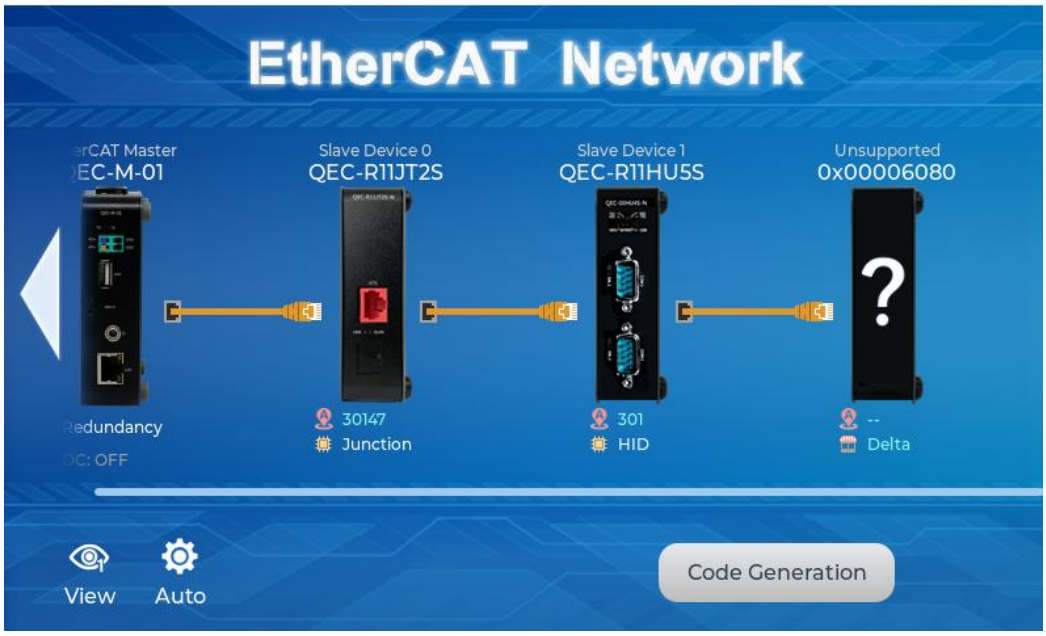


User Manual

86EVA

EtherCAT Configuration Tool

86Duino Coding IDE 501

EVA Library

(Revision 2.3)

REVISE NOTE:

DATE	VERSION	DESCRIPTION
2022/11	V1.0	New Release.
2025/4/9	V2.0	Updated API.
2025/7/11	V2.1	• Clarified Alias Address usage (page 21). • Corrected Auto Mapping DO/DI pin button description (page 22). • Fixed firmware downgrade dialog typo (page 26). • Improved explanation of Virtual Arduino Mapping (page 30). • Adjusted font alignment and spacing (page 40). • Combined the file checking logic descriptions (page 38 & 42). • Revised 3rd-party serial device integration examples (page 51). • Refined graphical serial programming features (page 53). • Updated use of Virtual Servo Configuration settings (page 72). • Added Encoder modes compatibility (page 62). • Replaced GPT.h with MCP.js in 86EVA generation files (page 68). • Clarified the digitalRead() return behavior (page 76). • Delete VirtualSpeaker class (page 97–101).
2025/8/20	V2.2	• Corrected grammar on page 63: “both standard motion applications” → “standard motion applications”. • Added missing value mapping rule for integerDataWrite() API (page 80). • Unified the terminology “MDevice” (pages 91–97 and 108).
2026/11	V2.3	• Updated for 86Duino Coding IDE 501 (86EVA Configurator 0.8.3.0, EtherCAT Library DMP-ECAT-0.9.6). • Added software version information (Ch. 1.2) and terminology mapping between this manual and the 86EVA UI (Ch. 1.4). • Added Ch. 2.3 Quick Start (workflow overview, step-by-step example, and when settings take effect) and Appendix A.4 86EVA Troubleshooting; former Ch. 2.3–2.7 are renumbered to Ch. 2.4–2.8. • Corrected Ch. 3 API descriptions against the IDE 501 library (return values, parameter names, examples using A0), added decimalDataRead() / decimalDataWrite(), and added missing error codes to Appendix A.1 / A.2. • Added Ch. 3.10–3.14 (MyVirtualLCD, MyVirtualMicrophone, MyVirtualSpeaker, MyVirtualEMSensor,

	MyVirtualForceSensor) and other member functions of existing classes; updated the supported platform of the 86Duino IDE (Ch. 1.2.1). • Updated the Virtual Arduino maximum settings to the 16 items available in 86EVA (Ch. 1.1.2, 2.5.4). • Added Other Mapping Types (Ch. 2.6.3.11), Torque Feedback Configuration (Ch. 2.6.4.5), and Supported SubDevices including verified third-party SubDevices (Ch. 2.6.5). • Updated Machine/Servo/Encoder configuration items and button names (Ch. 2.6.4). • Corrected the generated file list (mcp.js, data/Resource/MyEccfg) and launch notes (Ch. 2.4, 2.7). • Corrected heading numbering in Ch. 2.5.1, the MDevice model list, Error Reactions labels, notes in Ch. 2.6.3, the value range in Ch. 2.6.3.2, typos, the PL2303 driver link, and footers.

COPYRIGHT

The information in this manual is subject to change without notice for continuous improvement in the product. All rights are reserved. The manufacturer assumes no responsibility for any inaccuracies that may be contained in this document and makes no commitment to update or to keep current the information contained in this manual.

No part of this manual may be reproduced, copied, translated or transmitted, in whole or in part, in any form or by any means without the prior written permission of the ICOP Technology Inc.

©Copyright 2026 ICOP Technology Inc.
Ver.2.3 November 2026

TRADEMARKS ACKNOWLEDGMENT

ICOP® is the registered trademark of ICOP Corporation. Other brand names or product names appearing in this document are the properties and registered trademarks of their respective owners. All names mentioned herewith are served for identification purpose only.

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 IDE introduction, functions, languages, libraries, etc. Please visit the QEC website:

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

This Manual is for the QEC series.

TABLE OF CONTENTS

Ch. 1 Introduction 1
1.1 About 86EVA 2
1.1.1 Virtual Arduino Concept 3
1.1.2 Virtual Arduino Pin Configuration 4
1.2 About QEC EtherCAT MDevice 5
1.2.1 What is 86Duino IDE? 5
1.2.2 QEC EtherCAT MDevice Architecture 6
1.2.3 Hardware Platform 7
1.2.4 Dual-System Synchronization 8
1.3 Synchronization 9
1.3.1 Free Run 10
1.3.2 SM-Synchronous 11
1.3.3 DC-Synchronous 12
1.4 Terminology 14
Ch. 2 Using 86EVA 15
2.1 Install 86Duino IDE 16
2.2 Connect to PC and set up the environment 17
2.3 Quick Start: Your First 86EVA Project 19
2.3.1 Workflow Overview 19
2.3.2 Example: Control a Digital Output with a Digital Input 20
2.3.3 When Do Settings Take Effect? 26
2.4 Device Scan and User Interface Overview 27
2.5 MDevice Configuration 34
2.5.1 General 35
2.5.2 Error Reactions Function 41
2.5.3 Device Status 42
2.5.4 Virtual Arduino 43
2.6 SubDevice Configuration 44
2.6.1 General 45
2.6.2 Device Status 52
2.6.3 Virtual Arduino Mapping 53
2.6.4 Other Configuration 65
2.6.5 Supported SubDevices 77
2.7 Code Generation 79
2.8 Uploading to MDevice 81
Ch. 3 API Overview 83
3.1 Description 84
3.2 EVA Class 85
3.3 MyVirtualSerial Class 111

3.3.1 BarcodeScanner Class 118
3.3.2 ForceSensor Class 122
3.3.3 3DJoystick Class 130
3.3.4 ECRFID Class 136
3.4 MyVirtualMpg Class 146
3.5 MyVirtualLcm Class 159
3.6 MyVirtualKeypad Class 166
3.7 MyVirtualEncoder Class 170
3.8 MyVirtualCNC Class 180
3.9 MyVirtualServo Class 184
3.10 MyVirtualLCD Class 199
3.11 MyVirtualMicrophone Class 201
3.12 MyVirtualSpeaker Class 202
3.13 MyVirtualEMSensor Class 203
3.14 MyVirtualForceSensor Class 205
Ch. 4 ArduBlock Programming 207
4.1 About ArduBlock 208
4.2 Using ArduBlock 209
4.2.1 Start the ArduBlock 209
4.2.2 Interface introduction 210
4.2.3 Finished, code generation and upload 211
4.3 EtherCAT Class Blocks 212
4.3.1 Pins 213
4.3.2 HID 214
4.3.3 Comm. 215
4.3.4 Motion 216
Appendix 217
A.1 Error List 218
A.2 Error Description and Corrective Actions 222
A.3 About ENI Configuration in 86Duino IDE 245
A.4 86EVA Troubleshooting 248
Warranty 250

Ch. 1

Introduction

1.1 About 86EVA

86EVA (EVA, EtherCAT-Based Virtual Arduino) is a graphical EtherCAT configuration tool that works in conjunction with the EtherCAT Library of the 86Duino IDE.

Starting from IDE501, 86EVA introduces a more intuitive user interface, enhanced device detection logic, and support for a broader range of third-party EtherCAT SubDevices. Users can now view and configure all connected EtherCAT SubDevices in a visualized chain topology, as shown below: For the list of SubDevices recognized by 86EVA, including verified third-party SubDevices, refer to Ch. 2.6.5 Supported SubDevices.



Features of the latest 86EVA with IDE 501:

MDevice Configuration

- Automatically scans and displays all connected SubDevices.
- Set MDevice Object Name, BIOS settings, Cycle Time, Redundancy, Distributed Clock Mode (DC).
- Optional ENI file support.
- Enhanced error handling options.

SubDevice Configuration

- Set SubDevice Object Name, Alias, and I/O Mapping.
- View real-time device status and detailed information.
- Perform firmware updates for supported SubDevices, particularly QEC series modules.

Auto Code Generation

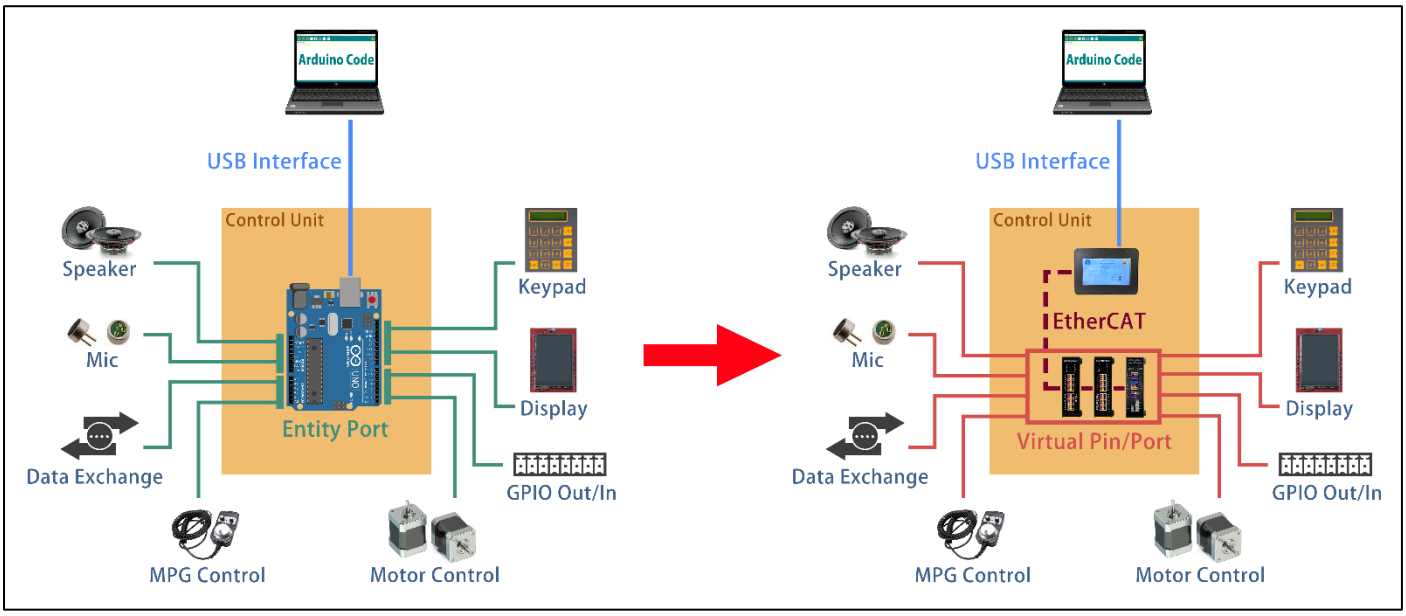
- With a single click on Code Generation, the system automatically converts your EtherCAT configuration into 86Duino C++ code, ready to upload through the 86Duino IDE.

86Duino IDE EtherCAT Library is an industrial-strength Arduino development environment based on Protected-mode OS. It enables real-time EtherCAT communication and offers a rich set of APIs for motion, I/O, and other fieldbus technology communication.

1.1.1 Virtual Arduino Concept

In a traditional Arduino board, the number, order, and function of pins are fixed during manufacturing. In contrast, Virtual Arduino allows developers to dynamically configure pin numbers, types, and functions based on the layout of an EtherCAT network.

Virtual Arduino represents a collection of pins formed by various QEC EtherCAT SubDevices. These "virtual pins" can originate from different hardware modules, but are logically mapped together and accessed as if they belonged to a single Arduino board.



Virtual Arduino in 86EVA allows flexible configuration of logical pins and peripherals based on your EtherCAT network setup. These resources can be mapped from multiple SubDevices and treated as part of a unified Arduino-like development environment.

Virtual Arduino is a key feature for both 86Duino and QEC platforms. It bridges graphical logic tools like **ArduBlock** and **LDmicro** with EtherCAT SubDevices, making it easy to build control logic without needing to understand low-level EtherCAT commands.

1.1.2 Virtual Arduino Pin Configuration

86EVA allows users to configure the maximum number of virtual pins and peripherals available in the Virtual Arduino environment. These virtual resources are mapped from connected SubDevices and used in Arduino-compatible code generation.



The MDevice "**Virtual Arduino**" page of 86EVA provides 16 "**Max Virtua**l …" settings. Each setting can be configured within the following system-defined maximum limits:

Category	Max Configurable
Virtual Digital Pins	1024
Virtual Integer Data Pins	1024
Virtual Decimal Data Pins	1024
Virtual Serials	64
Virtual Servos	384
Virtual Encoders	96
VirtualCNC Controllers	96
Virtual LCDs	64
Virtual LCMs	64
Virtual MPGs	64
Virtual Keypads	64
Virtual Buzzer Pins	64

***Note:** The practical limits may vary depending on the number of SubDevices in use and available memory on the QEC-MDevice.

These limits give developers tremendous flexibility when designing complex automation systems. All configurations are automatically mapped and translated into executable Arduino code by 86EVA.

1.2 About QEC EtherCAT MDevice

QEC (Quicker, Easier Control with EtherCAT) is an EtherCAT controller developed by ICOP. It offers high synchronization and real-time capabilities, along with ease of development. Below, we'll describe the architecture, features, and performance of QEC.

QEC MDevice is an EtherCAT MDevice System compatible with 86Duino Coding IDE 500 and later; the 86EVA functions described in this manual are based on 86Duino Coding IDE 501. It offers real-time EtherCAT communication between EtherCAT MDevice and EtherCAT SubDevices. 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.

Software versions referenced in this manual:

Component	Version
86Duino Coding IDE	501
86EVA Configurator	0.8.3.0
EtherCAT Library	DMP-ECAT-0.9.6 (2025-04-23)
EtherCAT MDevice Firmware	2025-03-05
QEC MCP Server script	QECmcp21.js (v2.0.0)

***Note:** Screenshots in this manual may differ slightly from your screen depending on the 86EVA Configurator version. The version is shown in the title bar of the 86EVA window.

1.2.1 What is 86Duino IDE?

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevices. 86Duino Coding IDE 501 is provided for Windows as a ZIP package. 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/.



1.2.2 QEC EtherCAT MDevice Architecture

The EtherCAT MDevice software only runs on the **Vortex86EX2** CPU produced by DM&P, which features a dual-system 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 a 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.

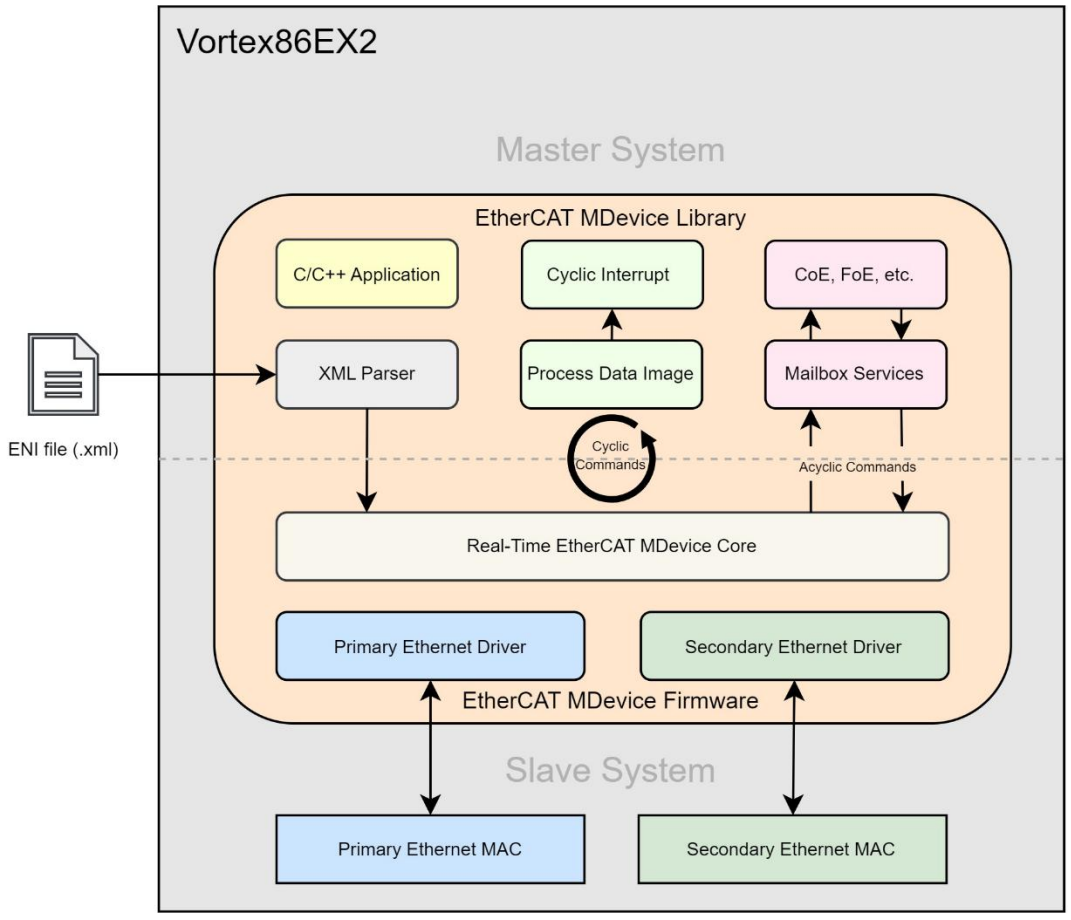
1.2.3 Hardware Platform

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.



1.2.4 Dual-System Synchronization

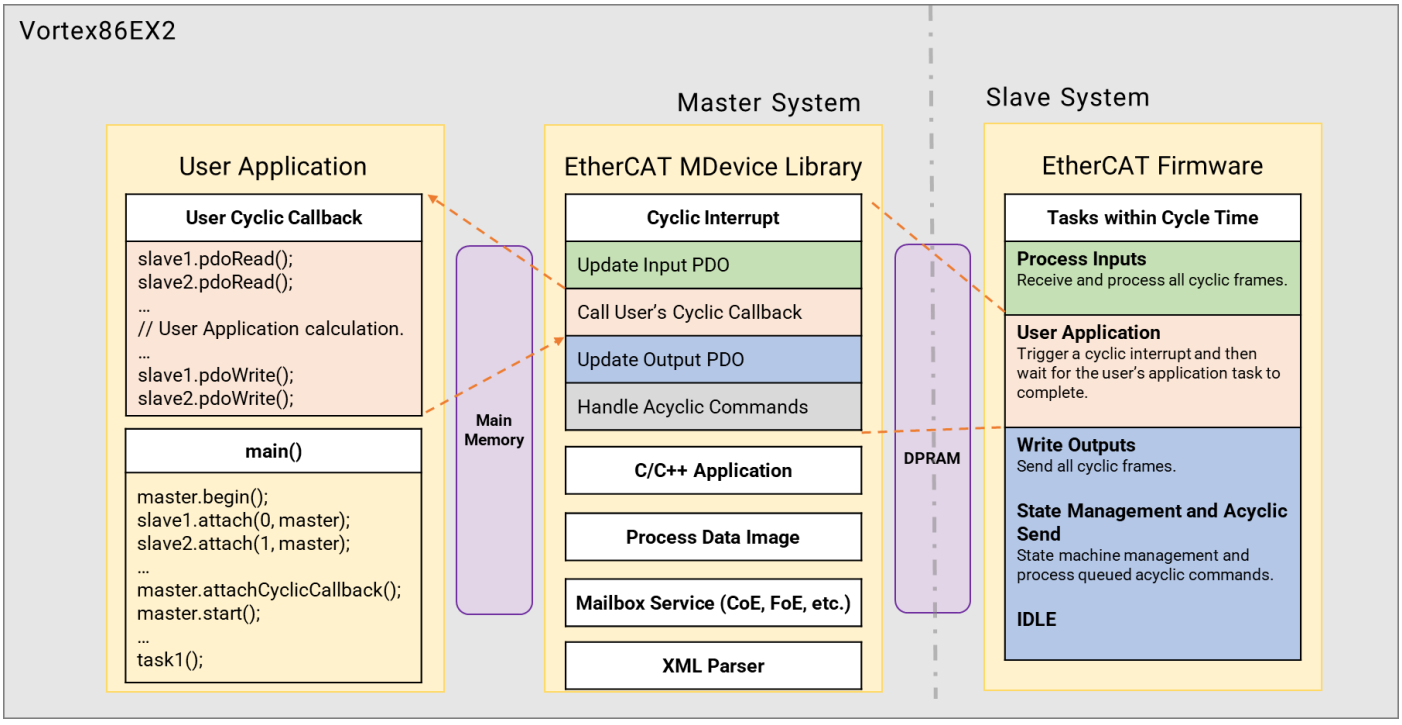
The primary focus of this section is the synchronization of dual-system PDO data. As illustrated in the diagram below, the User Application and EtherCAT MDevice Library blocks run on the Master System, while the Real-Time EtherCAT MDevice Core runs on the Slave System.

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.

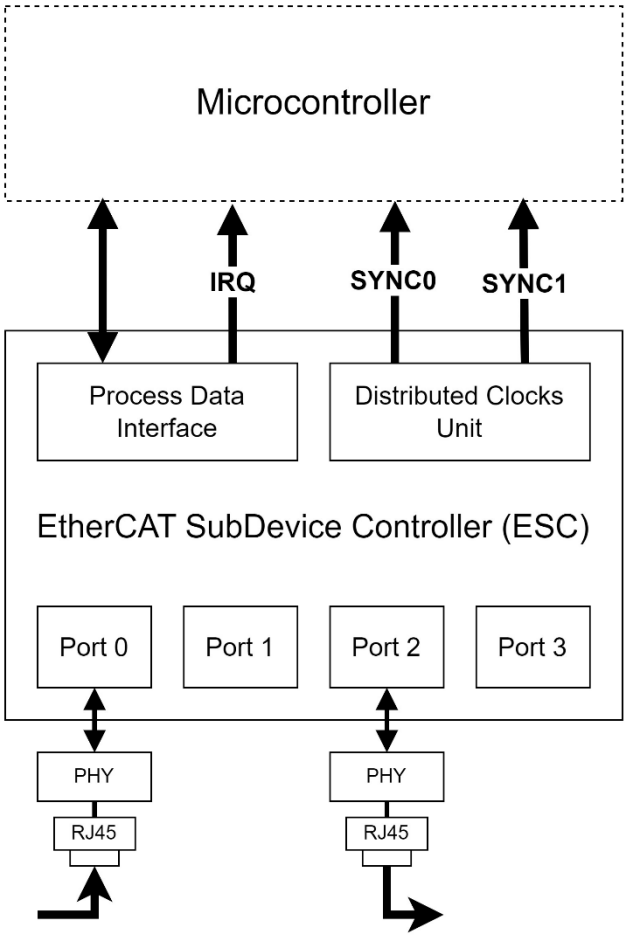
When the EtherCAT MDevice Core reaches the **Write Outputs** stage, it copies Output PDO data from DPRAM to the Ethernet Driver's DMA and sends frames.

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



1.3 Synchronization

The time synchronization among all SubDevices in an EtherCAT network relies on the Distributed Clocks (DC) unit within the EtherCAT SubDevice Controller (ESC), ensuring consistency across the entire system. Typically, the first SubDevice with DC serves as the system reference clock to synchronize other SubDevices with DC. For a more detailed explanation of DC, please refer to Distributed Clocks.



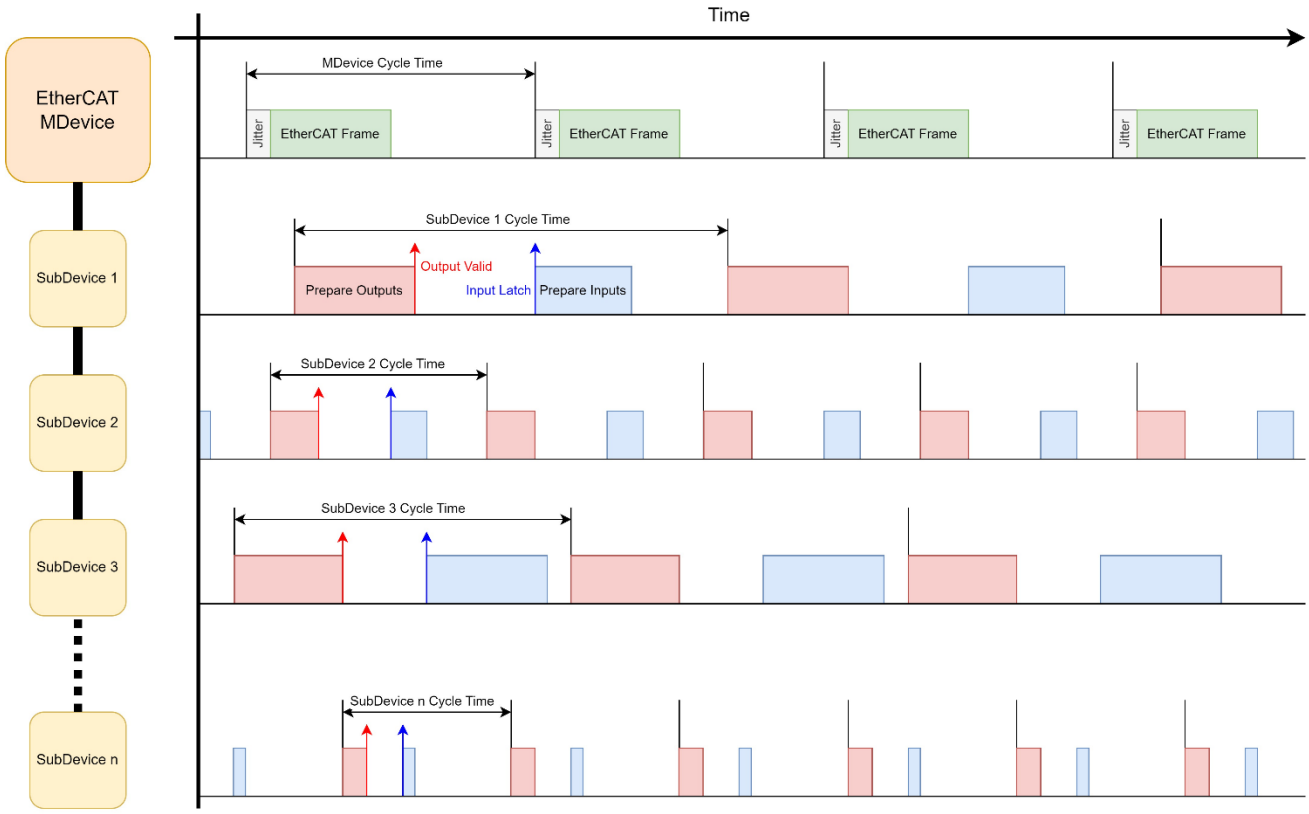
The ESC has three synchronization output pins: ***IRQ***, ***SYNC0***, and ***SYNC1***. The IRQ pin generates a signal to the upper-layer microcontroller (µC) after the ESC receives EtherCAT Cyclic Frames. SYNC0 and SYNC1 pins cyclically generate signals to the µC based on the configuration in the DC related registers of ESC. Hence, if an EtherCAT SubDevice does not have a µC, it does not support synchronization functionality.

There are three synchronization modes in EtherCAT:

- Free Run
- SM-Synchronous
- DC-Synchronous

1.3.1 Free Run

The EtherCAT MDevice and all EtherCAT SubDevices each have their own local timer, and their cycle times are independent, so they are not synchronized. As shown in the diagram below, both the EtherCAT MDevice and SubDevice 1, SubDevice 2, SubDevice 3 to SubDevice n have their own Cycle Time, resulting in inconsistent **Output Valid** and **Input Latch**. This scenario is not suitable for applications with high synchronization requirements.

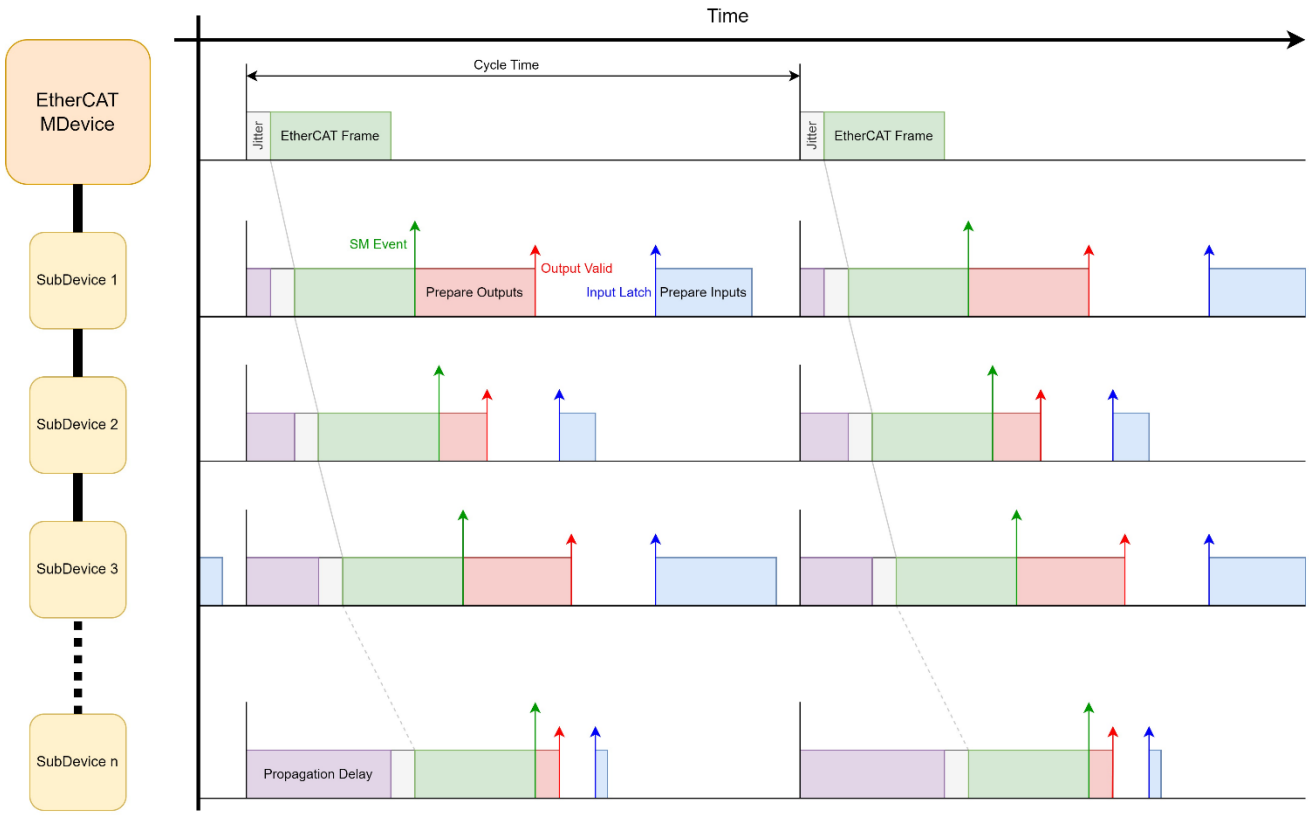


1.3.2 SM-Synchronous

The IRQ pin generates a SM event when the cyclic frame is received by ESC, this event will trigger the execution of the local application in µC. As shown in the diagram below, cyclic frames are received by SubDevices with the same jitter of MDevice in sending them. Even assuming zero jitter, due to finite hardware **Propagation Delay** the last SubDevices will receive the cyclic frames later with respect to the first ones.

Due to the Propagation Delay, there is an offset in the timing of SM events between SubDevices, resulting in an accuracy of SM-Synchronous at the **microsecond** level.

If each SubDevice supports the **Shift Time** in the **SyncManager Parameter objects** (0x1C32.3/0x1C33.3), it is possible to attempt to adjust the Output Valid and Input Latch of all SubDevices to be close to each other. However, due to the inability to calculate the propagation delays, the adjustment is quite challenging.



1.3.3 DC-Synchronous

The SYNC0 or SYNC1 pins generate SYNC events cyclically based on the configuration in the DC related registers of ESC, this event will trigger the execution of the local application in μC. As shown in the diagram below, jitters and propagation delays still exist, and SM events are still triggered after receiving cyclic frames. However, in this DC-Synchronous method, SYNC0 events are triggered cyclically, which does not suffer from jitter or propagation delays.

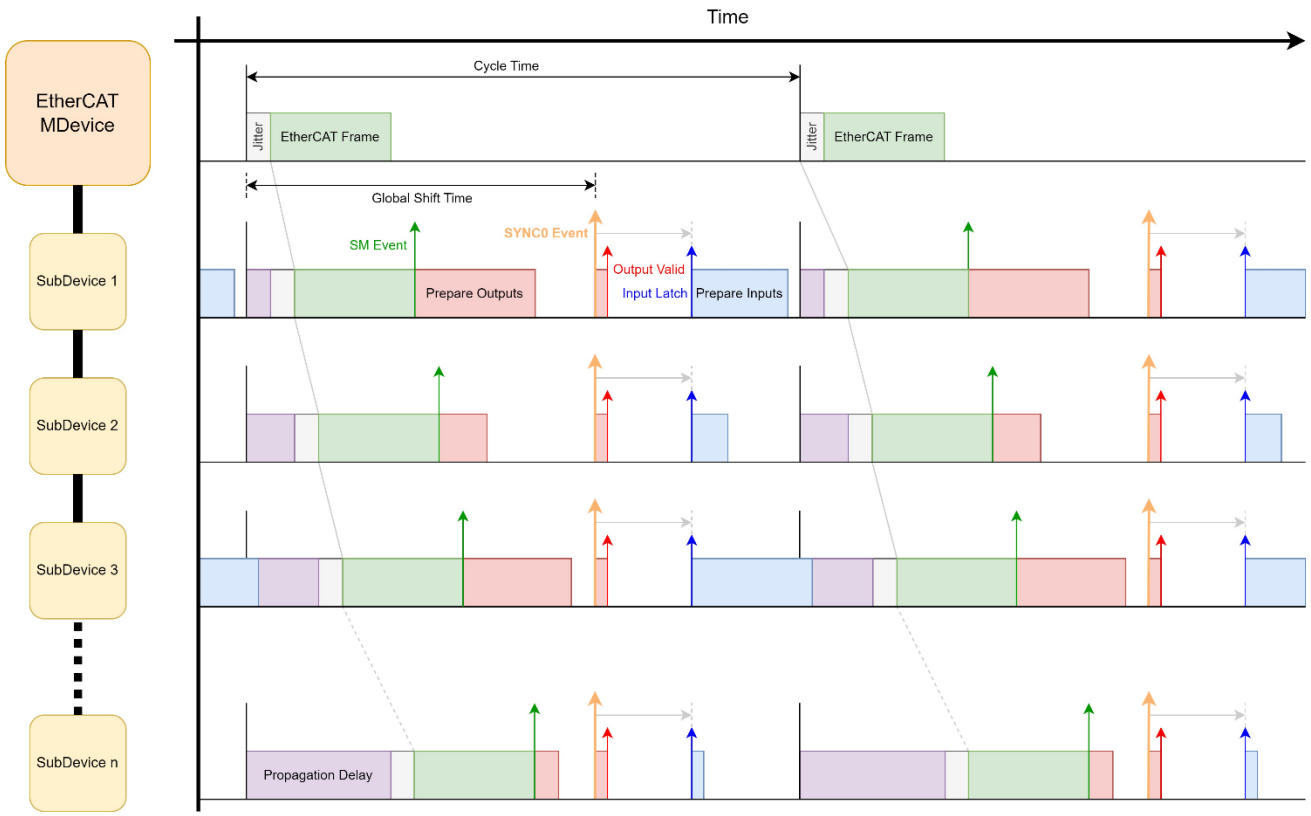
Because SYNC0 events are triggered by the DC unit and all SYNC0 events among the SubDevices have almost no offset, thanks to the periodic sending of APRW/FPRW commands to synchronize the system time of all SubDevices, the accuracy can reach the **nanosecond** level.

If the SubDevices support the Shift Time in SyncManager Parameter objects (0x1C32.3/0x1C33.3), it is possible to attempt to adjust the Output Valid and Input Latch timings of these SubDevices to the same time point.

However, the selection of the **Global Shift Time** in the diagram is crucial but must meet the following conditions:

- After the cyclic frames have been received by all SubDevices.
- Before sending the cyclic frames for the next cycle.
- According to different DC-Synchronous methods, it may need to be selected after executing Prepare Outputs:
    - Trigger μC to execute Prepare Outputs when the SM event occurs
    - Trigger μC to execute Output Valid when the SYNC event occurs.

The correct Global Shift Time is not unique; it can be chosen within the entire interval of the cycle time. To learn more about various DC-Synchronous methods, please refer to ETG.1020 EtherCAT Protocol Enhancements



1.4 Terminology

In line with the EtherCAT Technology Group (ETG) terminology, this manual uses the terms MDevice and SubDevice. Some labels in the 86EVA Configurator and some names in the 86Duino API still use the earlier terms Master and Slave.

Both terms refer to the same devices:

Term in this manual	**Term in 86EVA UI / API**
MDevice (QEC MDevice)	EtherCAT Master, Master (e.g., “EtherCAT Master”, “Cannot connect to EtherCAT Master”)
SubDevice	Slave, Slave Device (e.g., “Slave Device 0”, “Scan Slave Devices”, “Save to Slave”, “Set Servo CFG to Slave”)
Alias Address	Alias Address, alias ID
Object Name	Object Name (instance name in the generated code)
Virtual Arduino	EVA, EtherCATVirtualArduino (configuration file cfg.eva)
Master System / Slave System	The two CPU subsystems of the Vortex86EX2 (Ch. 1.2.3). These are not EtherCAT roles.

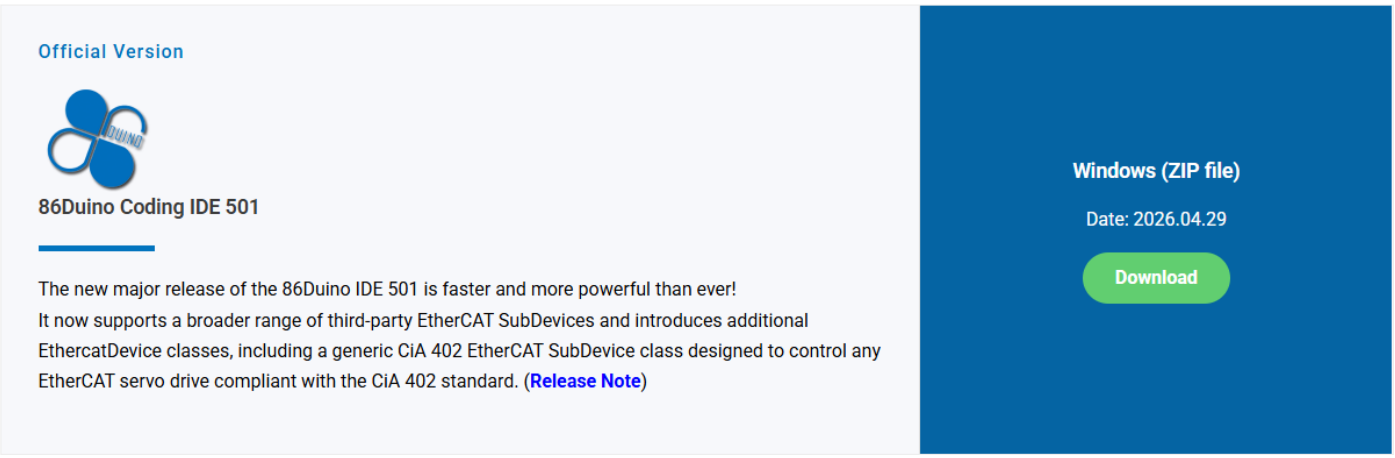
***Note:** UI labels are quoted exactly as they appear in 86EVA, so both terms may appear in the same section.

Ch. 2

Using 86EVA

2.1 Install 86Duino IDE

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 the "__Details__" then the "__Continue Run__" button once.

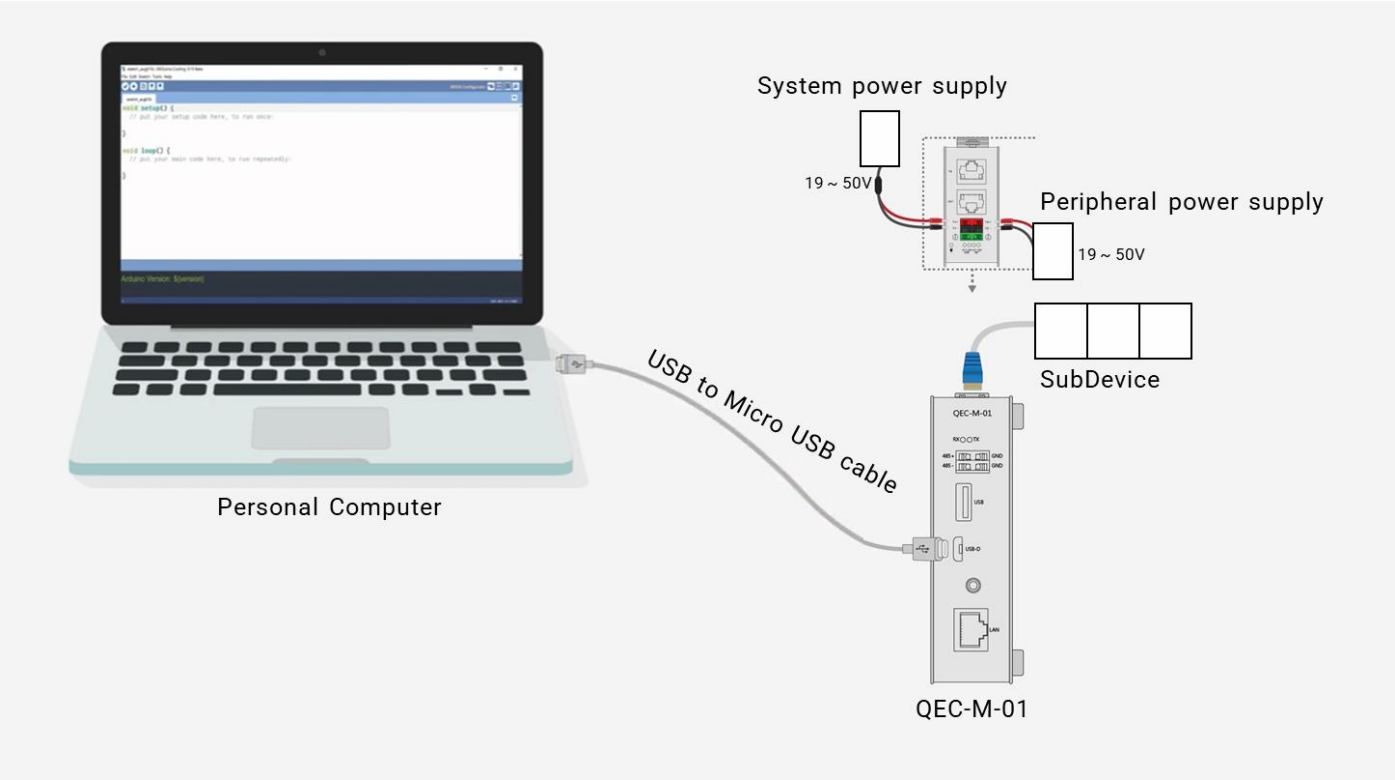
86Duino Coding IDE 501+ looks like below.



2.2 Connect to PC and set up the environment

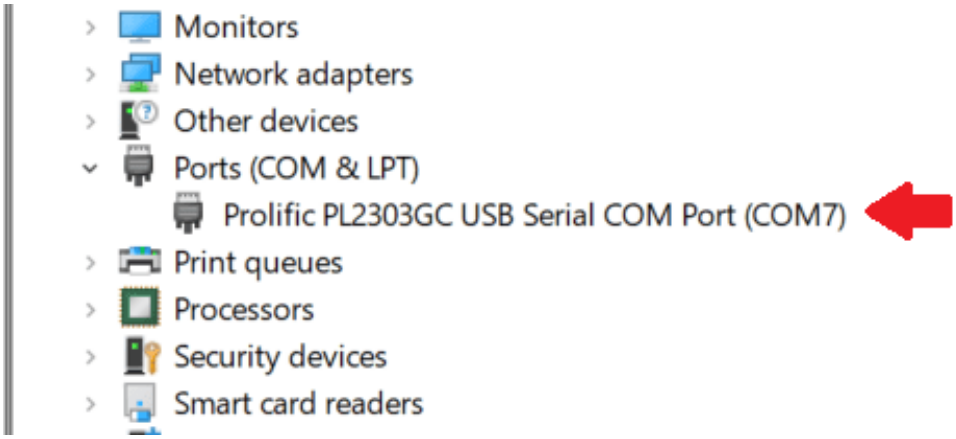
Follow the steps below to set up the environment:

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



2. Turn on the QEC power.

3. Open "**Device Manager**" (select in the menu after pressing Win+X) ->" **Ports (COM & LPT)**" in your PC and expand the ports; you should see that the "**Prolific PL2303GC USB Serial COM Port (COMx)**" is detected; if not, you will need to install the required drivers.
(The Windows PL2303 driver can be downloaded from [the Prolific website])

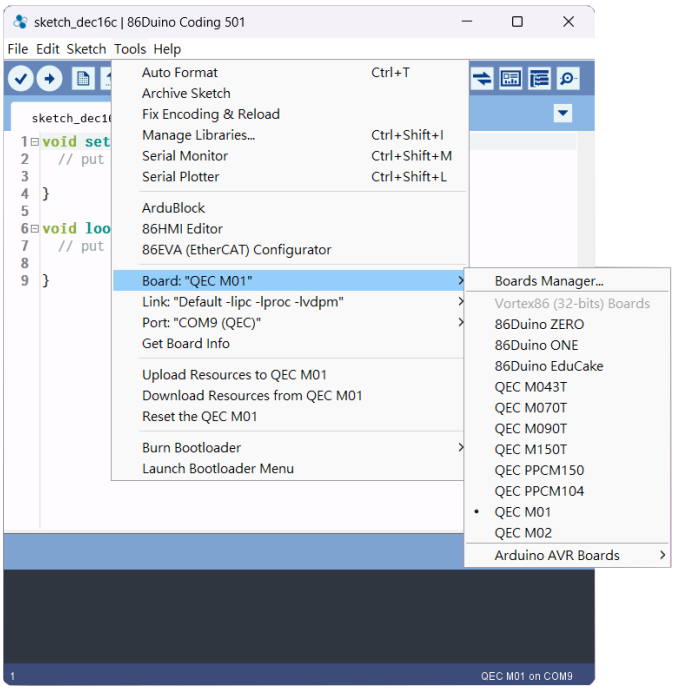


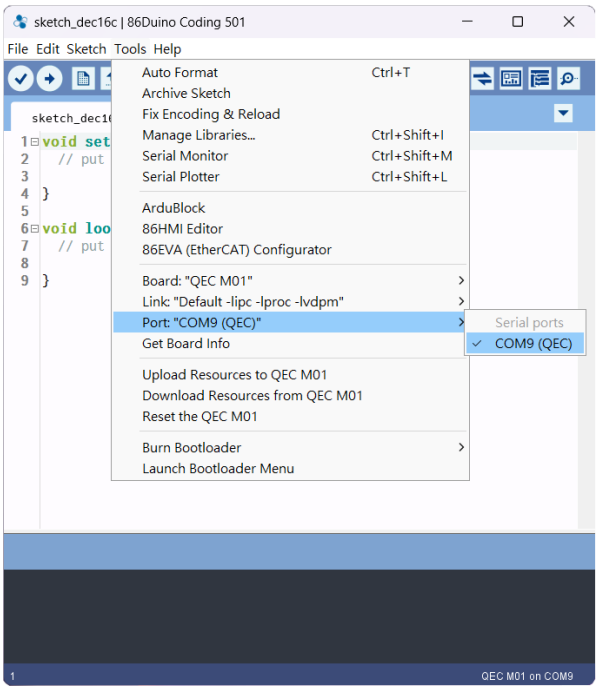
4. Open the 86Duino IDE.



5. Select the correct board: In the IDE's menu, select "**Tools**" -> "**Board**" -> "**QEC-M-01**" (or the QEC MDevice model you use).

6. Select Port: In the IDE's menu, select "**Tools**" -> "**Port**", and select the USB port to connect to the QEC MDevice (in this case, COM9 (QEC)).





2.3 Quick Start: Your First 86EVA Project

This section walks you through a complete 86EVA project in about 15 minutes. You will read a digital input of a QEC SubDevice and turn on a digital output with it: when DI00 is ON, DQ00 turns ON. After this exercise you will have used the complete 86EVA workflow; the following sections (Ch. 2.4 to Ch. 2.8) describe each function in detail.

2.3.1 Workflow Overview

Every 86EVA project follows the same six steps:

Step	What you do (details)
1. Open 86EVA	Save the sketch, then click the 86EVA icon in the IDE toolbar (Ch. 2.4).
2. Connect and scan	Select the COM port of the QEC MDevice and click “Connect” (Ch. 2.4).
3. Configure the MDevice	Object Name, Cycle Time, Redundancy, DC, ENI, Error Reactions, Max Virtual settings (Ch. 2.5).
4. Configure the SubDevices	Alias Address, Virtual Arduino mapping, device-specific settings (Ch. 2.6).
5. Code Generation	Click “Code Generation”; 86EVA writes myeva.h / myeva.cpp and the configuration files into the sketch (Ch. 2.7).
6. Write logic and Upload	Use the mapped virtual pins in loop(), then click “Upload” in the IDE (Ch. 2.8).

2.3.2 Example: Control a Digital Output with a Digital Input

Before you start

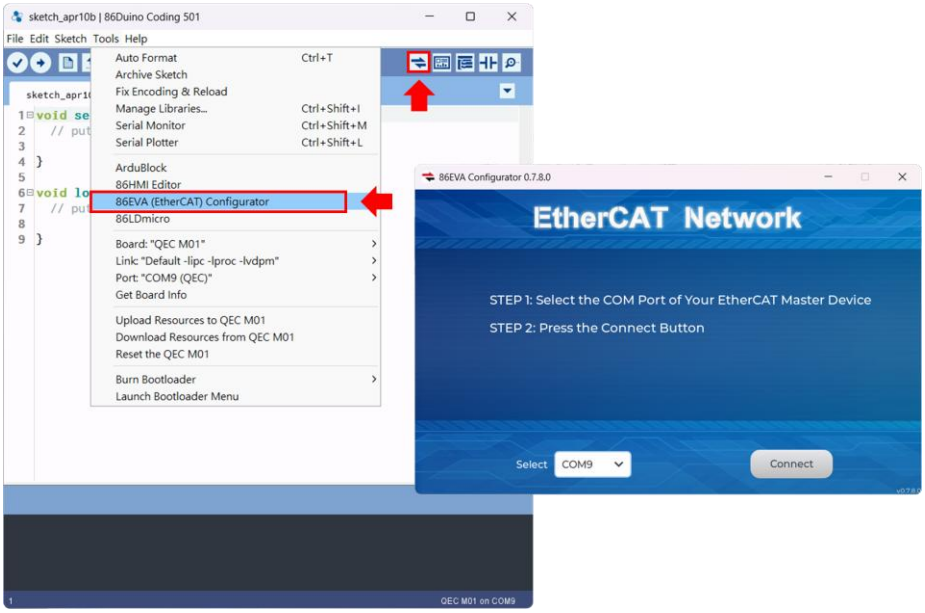
- Hardware: one QEC MDevice (this example uses QEC-M-01) and one QEC digital I/O SubDevice with 8 DI and 8 DO (this example uses QEC-R11D88D), a 24 V power supply, an Ethernet cable, and a Micro USB cable.
- Connect the EtherCAT port of the QEC MDevice to the IN port of the SubDevice, power on both devices, and connect the QEC MDevice to your PC with the Micro USB cable.
- Install the 86Duino IDE (Ch. 2.1), then select the correct Board and Port (Ch. 2.2).

Step 1 Create and save a new sketch

In the 86Duino IDE, select "**File**" -> "**New**", then "**File**" -> "**Save As**" and save the sketch, e.g., as EVA_QuickStart. 86EVA cannot be opened from an unsaved built-in example.

Step 2 Open 86EVA

Click the 86EVA icon in the IDE toolbar, or select "**Tools**" -> "**86EVA (EtherCAT) Configurator**".



- **Check:** The 86EVA start screen appears with "**STEP 1**" and "**STEP 2**". The version (e.g., v0.8.3.0) is shown in the lower-right corner.



Step 3 Connect and scan the EtherCAT network

Select the COM port of the QEC MDevice next to “**Select**” (e.g., COM9), then click “**Connect**”. 86EVA scans the network (“**Scan Slave Devices ... wait**”).



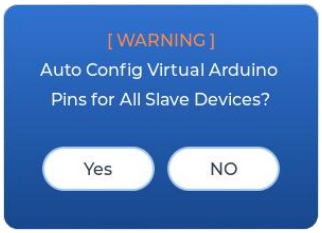
- **Check:** The “**EtherCAT Network**” screen shows “**EtherCAT Master QEC-M-01**” and **“Slave Device 0 QEC-R11D88D**”. If an error appears, see Appendix A.4.



Step 4 Map the digital pins automatically

Click “**Auto**” in the lower-left corner, then click “**Auto Set Digital Pin Mapping of the Virtual Arduino**”, and click “**Yes**” in the **[WARNING]** window.





- **Check:** 86EVA reports that the virtual Arduino pins have been set. Click the SubDevice image and scroll to “Virtual Arduino”: the digital outputs are mapped first, so DO0 → Virtual Digital Pin D0 and DI0 → Virtual Digital Pin D8.





*__Note:__ If your mapping differs (e.g., other SubDevices are connected), use the virtual pin numbers shown on your screen in Step 6.

Click “**Back**” to return to the “**EtherCAT Network**” screen. In this example the MDevice settings can stay at their defaults.

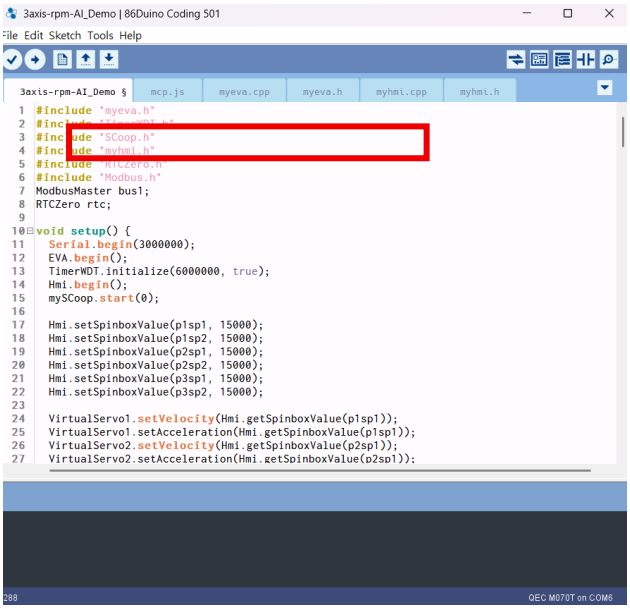


Step 5 Generate the code

Click “**Code Generation**” in the lower-right corner, then click “OK” (or wait 10 seconds) to close 86EVA.



- **Check:** The IDE now shows the tabs myeva.cpp, myeva.h, and mcp.js, and the main sketch contains `#include "myeva.h"` and `EVA.begin();` in `setup()`.



Step 6 Write the control logic

Edit `loop()` in the main sketch as follows:

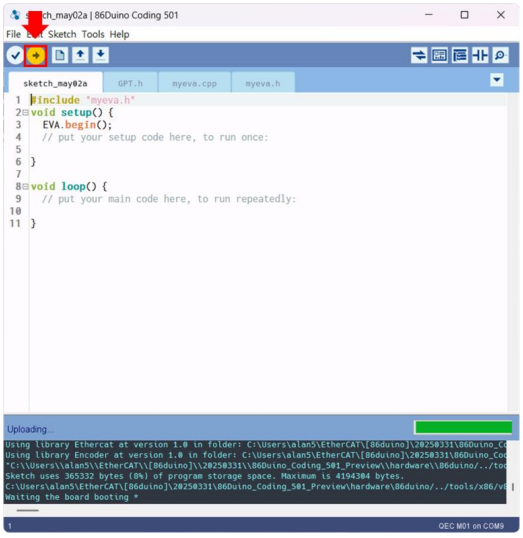
```
#include "myeva.h"

void setup() {
  EVA.begin();
}

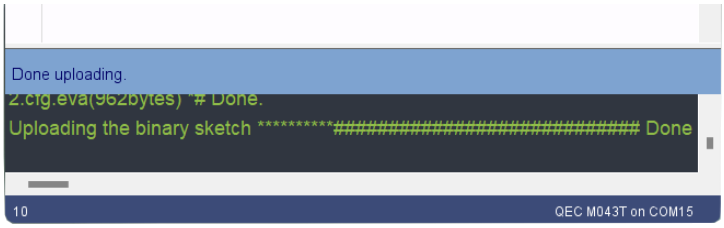
void loop() {
  if (EVA.digitalRead(8) == HIGH)   // DI0 -> Virtual Digital Pin D8
    EVA.digitalWrite(0, HIGH);      // DO0 -> Virtual Digital Pin D0
  else
    EVA.digitalWrite(0, LOW);
}
```

Step 7 Upload

Check “**Tools**” -> “**Board**” and “**Tools**” -> “**Port**”, then click “**Upload**”. The configuration files and the sketch are uploaded to the QEC MDevice together.



- **Check:** The IDE shows “**Done uploading**”.



Step 8 Verify

Turn on DI0 of the SubDevice. DO0 turns on; turn DI0 off and DO0 turns off.

You can also reopen 86EVA and scan the network to confirm that the SubDevice runs with the uploaded configuration.

Next steps

- Give each SubDevice a fixed Alias Address so that the program still works when SubDevices change positions (Ch. 2.6.1.2, Ch. 2.4 “Auto”).
- Change the cycle time, enable redundancy or DC, or import an ENI file (Ch. 2.5.1).
- Map analog, motion, serial, and HID resources (Ch. 2.6.3) and use them with the API in Ch. 3 or with ArduBlock in Ch. 4.

2.3.3 When Do Settings Take Effect?

86EVA writes settings to two different places. Settings written to a SubDevice take effect after the SubDevice restarts; settings stored in the project take effect only after Code Generation and Upload.

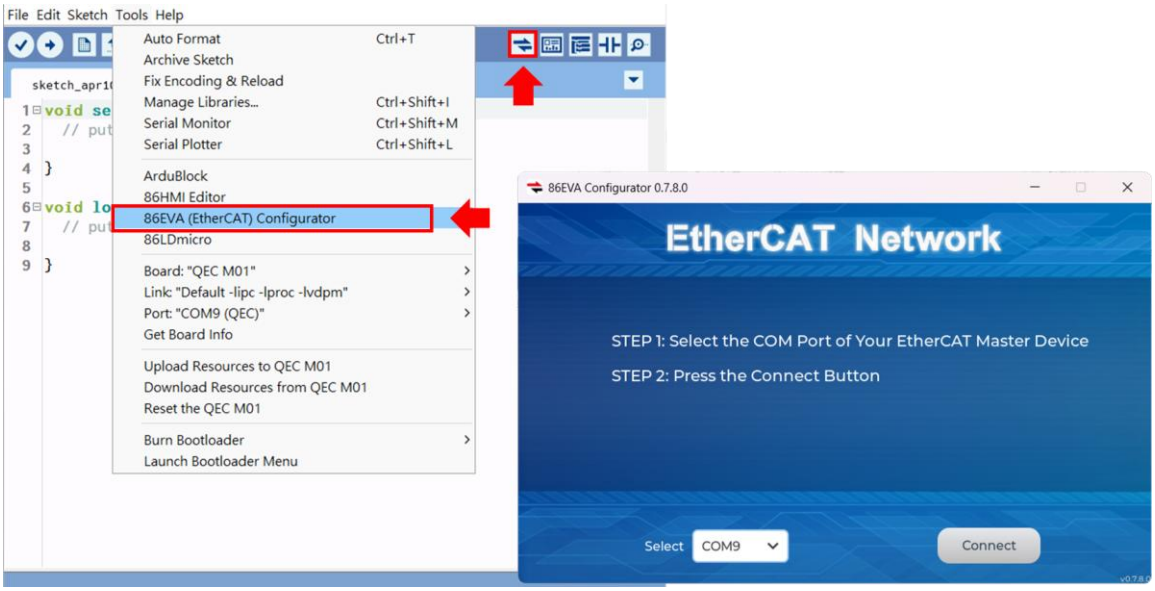
Setting / button	Where it is written	How to apply it
Alias Address “Set”, “Auto Set Alias Addresses”	SubDevice EEPROM (immediately)	Restart the SubDevice (“Please reset the slave device to apply the new alias ID.”)
Device Mode “Set”	SubDevice (immediately)	Restart the SubDevice
Firmware “Update”	SubDevice (immediately)	Restart the SubDevice and scan again
“Save to Slave”, “Set Machine / Servo / Encoder / Torque CFG to Slave”	SubDevice via SDO (immediately)	Restart the SubDevice (“Please reset the slave device to apply the new settings.”)
MDevice settings (Object Name, Apply BIOS Settings, Redundancy, Cycle Time, DC, ENI, Error Reactions, Max Virtual …)	Project (cfg.eva, myeva.h / myeva.cpp)	Code Generation, then Upload
SubDevice Object Name and Virtual Arduino mapping	Project (cfg.eva, myeva.h / myeva.cpp)	Code Generation, then Upload

***Note:** Closing 86EVA without clicking “Code Generation” discards the project settings. Generating the code without uploading leaves the QEC MDevice running the previous configuration.

2.4 Device Scan and User Interface Overview

After starting the 86Duino IDE, you can start to use the 86EVA.

Launch 86EVA by clicking the 86EVA icon in the 86Duino IDE toolbar, or by selecting “**Tools**” -> “**86EVA (EtherCAT) Configurator**” in the IDE menu.

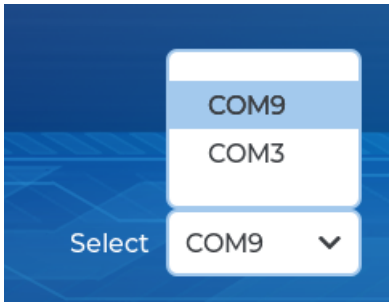


***Note:** 86EVA can be opened only when a QEC series board is selected in “**Tools**” -> “**Board**”. It cannot be opened from a built-in example sketch; save your sketch to your own folder first (“**File**” -> “**Save As**”). 86EVA stores its configuration in the data/Resource folder of that sketch.

After opening 86EVA, it automatically selects the COM port of the connected QEC MDevice station by default. In this example, the COM port is COM9. The start screen guides you through two steps: “STEP 1: Select the COM Port of Your EtherCAT Master Device” and “STEP 2: Press the Connect Button”.

If multiple QEC MDevice stations are connected, use the drop-down menu next to “**Select**” to choose the appropriate COM port.





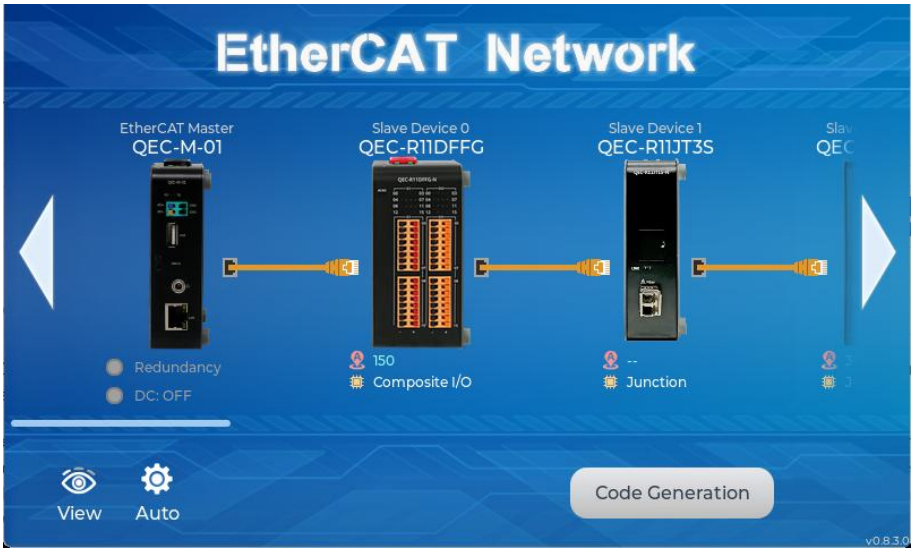
After selecting the correct COM port location, please click on the "**Connect**" button.



After clicking the "**Connect**" button, a scanning screen will appear.



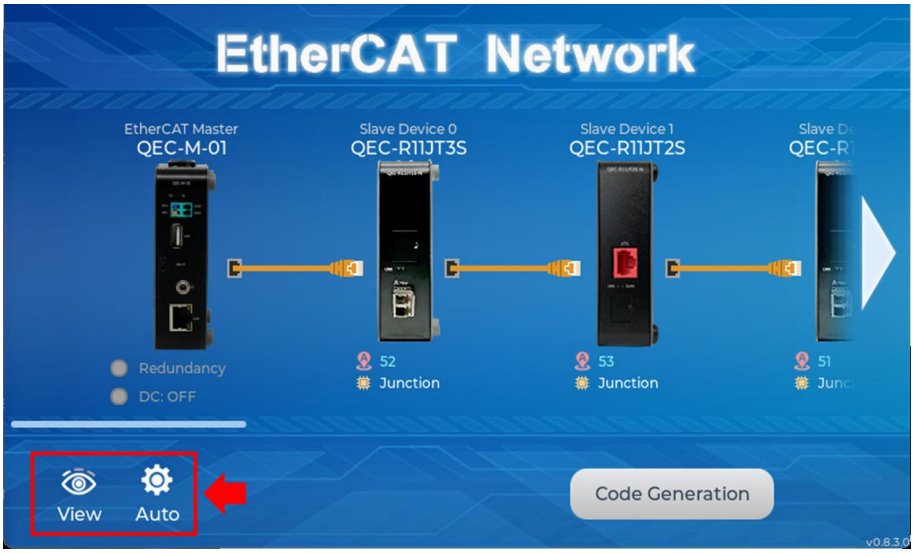
After scanning, you will see one QEC MDevice and all the EtherCAT SubDevices currently connected to the EtherCAT network.



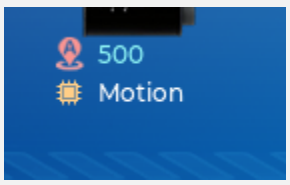
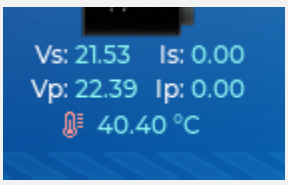
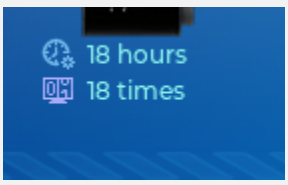
After the scan, 86EVA automatically detects all connected EtherCAT devices and displays them as a topology diagram. Verified QEC SubDevices will show their model image, Alias Address, and device type. If an unsupported third-party device is found, it will be displayed with a "**?**" icon and labeled as "**Unsupported**", as shown below:



Two buttons are available at the bottom-left corner: "**View**" and "**Auto**".



1. **View:**
   When View is clicked, users can cycle through three different layouts for the QEC devices display:

View 1: Class & Alias Address	View 2: Device Status	View 3: Startup time
		

2. **Auto:**

The "**Auto**" button provides two convenient automatic configuration tools to assist users in quickly preparing their EtherCAT topology.



a. The "**Auto Set Alias Address of All Slave Devices**" button will assign a specific Alias Address as follows to every detected EtherCAT SubDevice in the network.

Table: Auto Alias Address Number

Device / Module Type	Alias ID Range	Notes / Description
AC Power Vs	0090 – 0094	
AC Power Vp	0095 – 0099	
16 DQ (Digital Output)	0100 – 0199	
16 DI (Digital Input)	0200 – 0299	
8 DI + 8 DO	0250 – 0259	Mixed I/O module
8 DI Analog VM	0280 – 0299	Digital Input with Analog Voltage Monitoring (VM)
HID (Keypad, LCM, Buzzer, MPG...)	0300 – 0399	Human Interface Devices
Arduino Touch LCD	0400 – 0499	LCD Touch Panel
Step Motor	0500 – 0599	Stepper Motor Driver
MIC + Counter	0600 – 0699	Microphone + Encoder Counter
86 Servo Motor	0700 – 0799	
AI / AQ Analog A/D	0800 – 0899	Analog Input / Output Modules
32 DQ (Digital Output)	1100 – 1199	
32 DI (Digital Input)	1200 – 1299	
Environmental Monitoring Module	1500 – 1599	For temperature, humidity, PM2.5, etc.
Force Sensor	1600 – 1649	Supports force/torque reading (e.g., Fx, Fy, Fz, Mx, My, Mz)

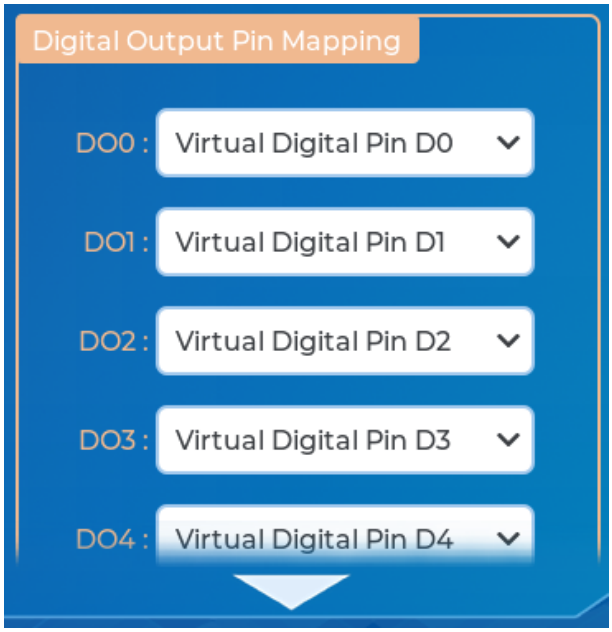
After clicking, 86EVA displays a **[WARNING]** window: "**Auto set Alias Address for all Slave Devices?**"



If confirmed, all SubDevices will receive a unique Alias Address. This is especially useful in systems involving multiple identical SubDevices, where manual alias configuration would be time-consuming.



b. The "**Auto Set Digital Pin Mapping of the Virtual Arduino**" button automatically maps all digital I/O pins that support Virtual Arduino to Virtual Digital Pins.



After clicking, 86EVA displays a **[WARNING]** window: "**Auto Config Virtual Arduino Pins for All Slave Devices?**"

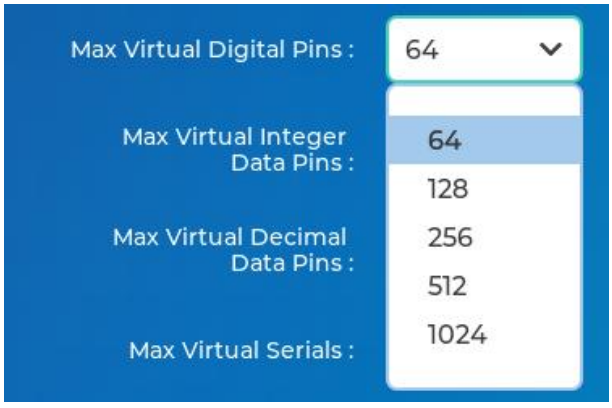


***Note:** Virtual digital pins are automatically mapped with priority given to Digital Output before Digital Input.





The "**Virtual Arduino**" area in the QEC MDevice configuration page allows you to set the maximum number of virtual digital pins.



These automation functions reduce setup time and minimize human error during device initialization.

***Note:** If 86EVA loses the connection to the QEC MDevice, an **[ERROR] window shows "Can not connect to EtherCAT Master, please restart 86EVA Configurator"**.



***Note:** If 86EVA cannot connect to the QEC MDevice, an **[ERROR]** window shows "**Connect Fail! Check COM port and try again.**" Check the COM port and the USB cable. If no SubDevice is detected, 86EVA shows "**No detect any EtherCAT slave device**." Check the EtherCAT cables and the power of the SubDevices.

2.5 MDevice Configuration

The 86Duino IDE is designed for the QEC-M series, including: QEC-M-01, QEC-M-02, QEC-M-043T, QEC-M-070T, QEC-M-090T, QEC-M-150T, QEC-PPC-M-090T, QEC-PPC-M-104T, and QEC-PPC-M-150T.
After completing the scan in 86EVA, users can view each EtherCAT MDevice in a graphical layout along with its product image.



This section introduces the main configuration functions available for QEC MDevices. The configuration page is divided into four sections:

1. General Settings
2. Error Reactions Function
3. Device Status
4. Virtual Arduino Settings



Each section in this chapter corresponds to a specific configuration tab in 86EVA for easier navigation and understanding.
Throughout this section, the **QEC-M-01** will be used as the example MDevice.

2.5.1 General

This section allows users to configure key parameters of the EtherCAT MDevice (QEC MDevice), including object name, BIOS settings, redundancy, cycle time, DC synchronization, and ENI file selection.



2.5.1.1 Object Name

Set the object name of the EtherCAT MDevice.
This name will be used as the instance name in the auto-generated 86Duino code, default is "**EcatMaster**".



The 86EVA displays a virtual keyboard if users click the input area.

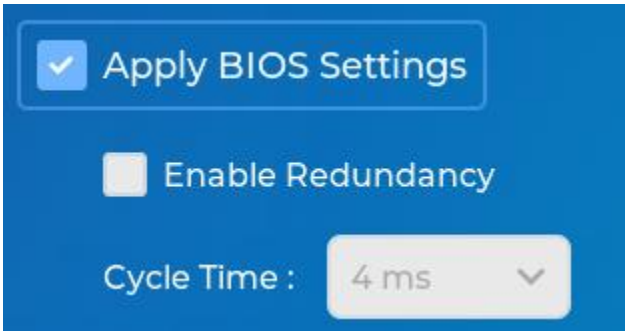


2.5.1.2 Apply BIOS Settings

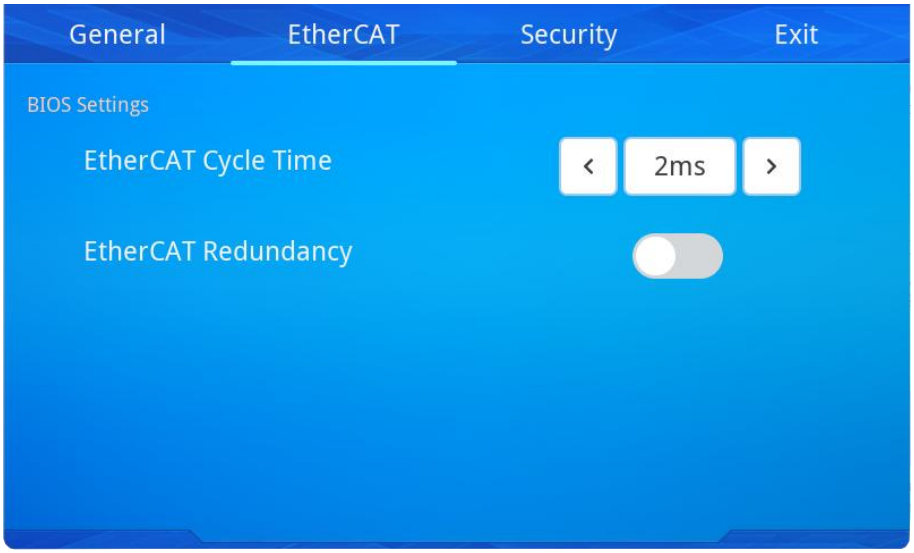
Choose whether to apply BIOS-predefined parameters from the QEC MDevice.



When this option is selected, the Enable Redundancy and Cycle Time settings in 86EVA are overridden by the BIOS values.



When BIOS settings are applied, the values configured in the Bootloader Menu of the QEC MDevice are used.



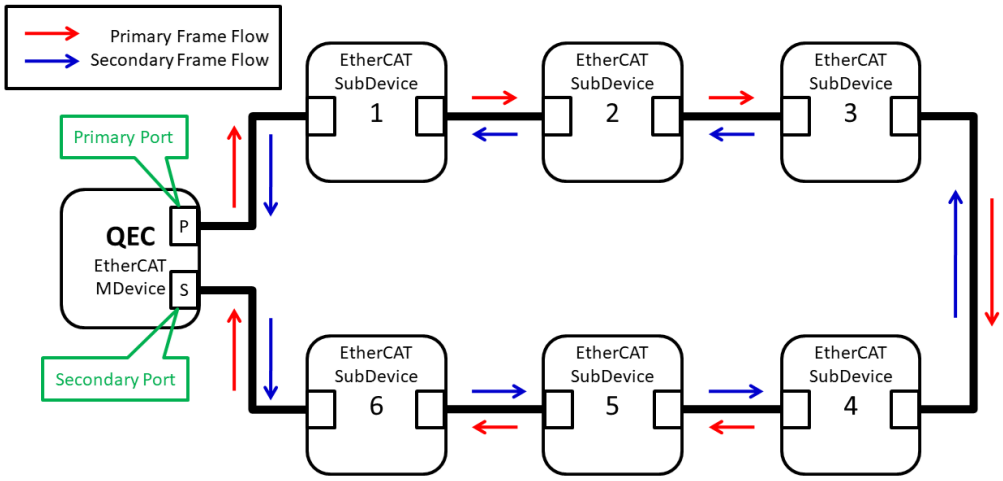
For more detailed information about the Bootloader Menu, please refer to each QEC MDevice's User Manual.

2.5.1.3 Enable Redundancy

Enable this option to activate EtherCAT cable redundancy.
This is recommended for systems requiring fault-tolerant communication.



***Note:** When this option is selected, your EtherCAT network must follow a physical cable redundancy topology, with both the Primary and Secondary ports connected.

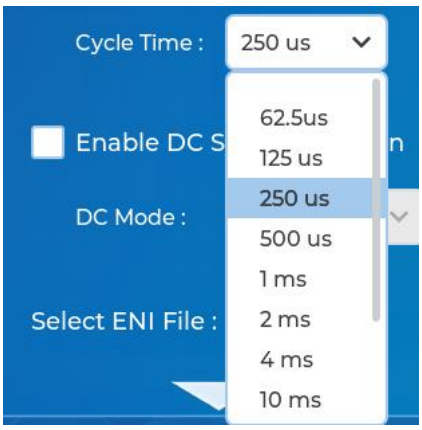


2.5.1.4 Cycle Time

Configure the control cycle time of the EtherCAT MDevice.



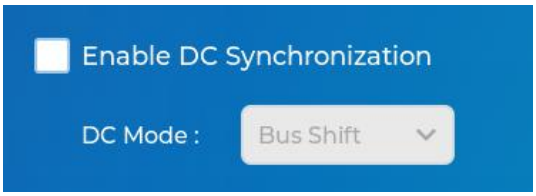
Available options: 62.5 µs, 125 µs, 250 µs, 500 µs, 1 ms, 2 ms, 4 ms, 10 ms, 20 ms, 50 ms, 100 ms.



***Note:** When configuring the cycle time, please refer to your EtherCAT network's maximum response time to avoid packet delays or synchronization issues.

2.5.1.5 Enable DC Synchronization

Enable this option to activate Distributed Clocks (DC) functionality in the EtherCAT network.



Distributed Clocks (DC) is an essential feature implemented in the EtherCAT SubDevice Controller (ESC). It is responsible for implementing a time synchronization mechanism across the EtherCAT network, ensuring that all SubDevices synchronize their clocks according to a unified time reference, thus ensuring consistency of time across the entire system.

For system synchronization all SubDevices are synchronized to one Reference Clock. Typically, the first ESC with Distributed Clocks capability after the MDevice within one segment holds the reference time (System Time). This System Time serves as the reference for synchronizing all DC-capable SubDevices and the MDevice itself. The propagation delays, local clock sources drift, and local clock offsets are taken into account for the clock synchronization.
For more detailed information, please refer to ESC Hardware Data Sheet Section I.

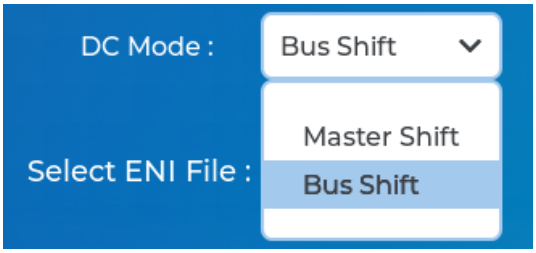
***Note:** The DC Synchronization function in 86EVA only enables the setting. It does not apply any time shift or offset calibration.

When this option is selected, the DC Mode drop-down list will be enabled.



***Note:** If DC Synchronization is disabled, the system will operate in Free Run or SM-synchronous mode, depending on the configuration. Some EtherCAT SubDevices must operate in DC mode; please follow the operating requirements of these SubDevices.

DC Mode
After enabling DC synchronization, the user can select the preferred DC operating mode.



***Note:** This function is still under development and not recommended for use at this time.

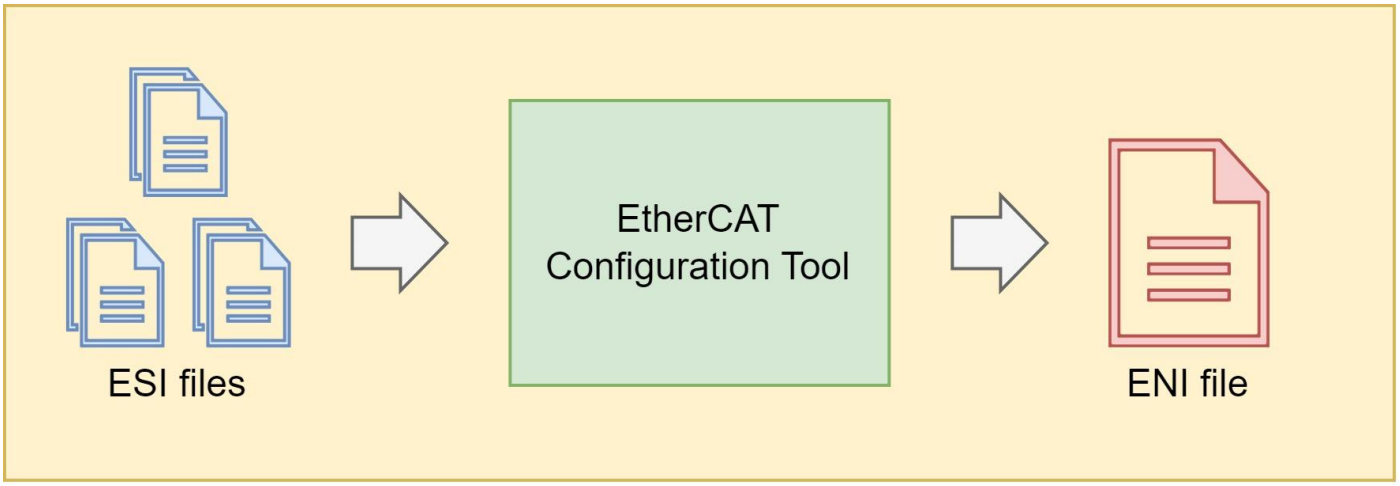
2.5.1.6 Select ENI File

This function allows you to manually import an EtherCAT Network Information (ENI) file to the QEC MDevice via 86EVA.



The ENI file is an XML-based configuration file that typically includes:

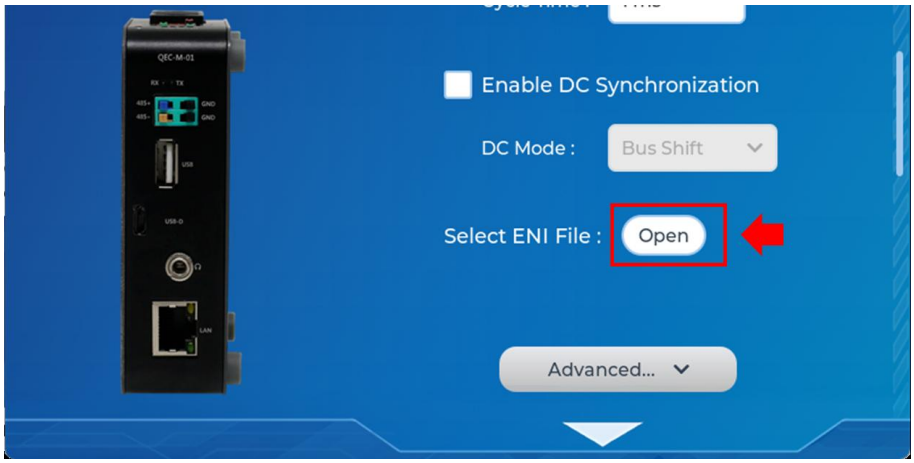
- MDevice and SubDevice layout
- PDO/SDO mapping
- DC synchronization settings
- Vendor ID & Product Code definitions



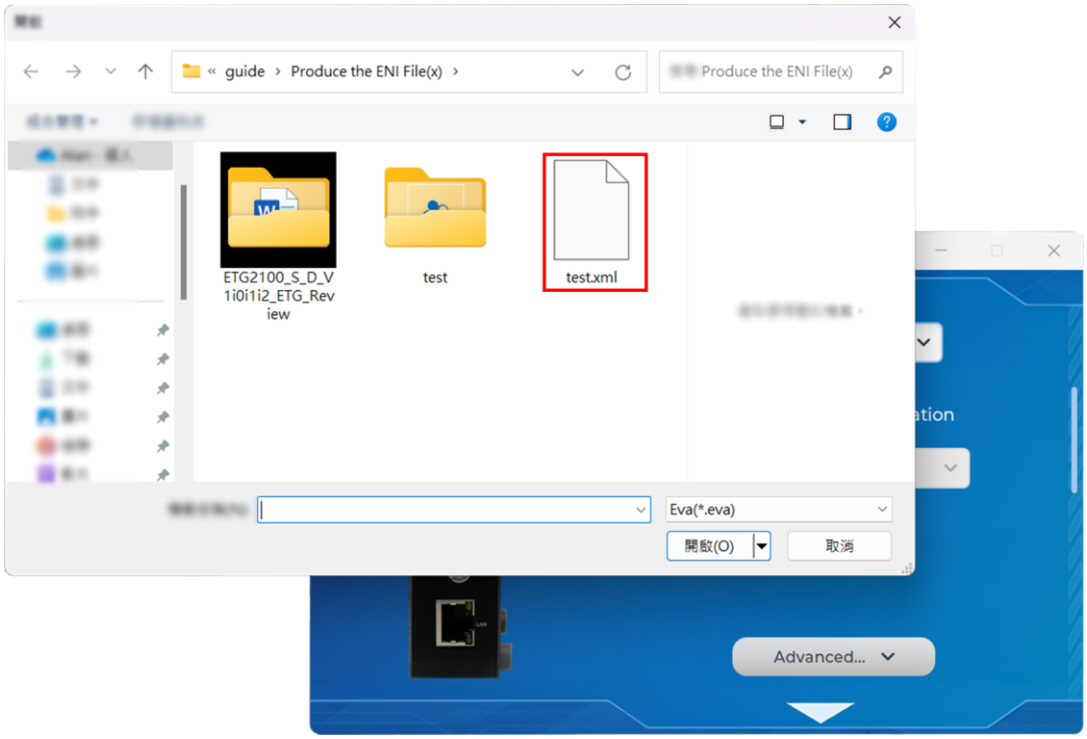
It can be generated from other EtherCAT configuration tools (e.g., TwinCAT, EC-Engineer) and imported into 86EVA for deployment.

2.5.1.6.1 How to Import ENI in 86EVA

1. Click the **"Open"** button next to Select ENI File.



2. Choose your ENI file (.xml) from the file browser.



3. After a successful import, the selected filename will appear in the ENI File field.



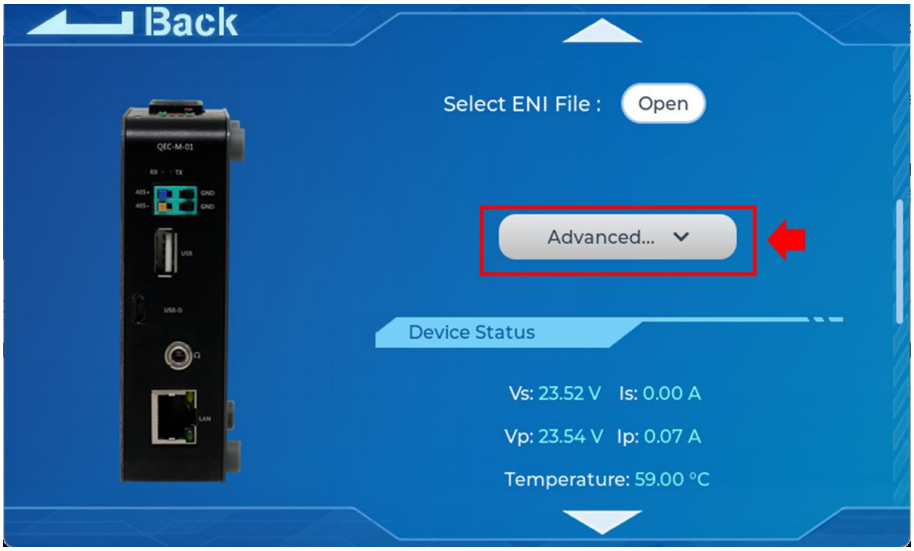
The ENI file will be automatically uploaded during the 86EVA code deployment process. There is no need to manually transfer the file to the QEC device via USB.

***Note: Limitation:** The EtherCAT library currently only supports partial parsing of ENI files, including Vendor ID, Product Code, Alias, CoE InitCmds, and DC settings.

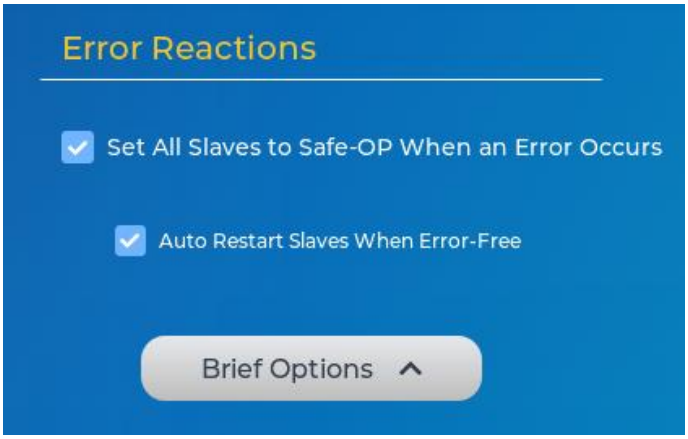
For best compatibility, please ensure your ENI file is generated using updated ESI files for each device in your EtherCAT network. For a full list of supported ENI fields, refer to Appendix A.3 About ENI Configuration in 86Duino IDE.

2.5.2 Error Reactions Function

This section explains how to configure the QEC MDevice's automatic error-handling responses during EtherCAT network faults. These mechanisms enhance system reliability and enable automatic fault recovery in industrial environments.



After clicking **Advanced**, the **Error Reactions** options will be displayed.



Function Overview:

Setting	Description
Set All Slaves to Safe-OP When an Error Occurs	Forces all SubDevices to enter Safe Operational state (Safe-OP) after a fault is detected
Auto Restart Slaves When Error-Free	Automatically attempts to resume normal operation (OP mode) once communication is restored

***Note:** Some SubDevices may not support automatic recovery from Safe-OP to OP.
Please refer to their datasheets if automatic restart fails.

2.5.3 Device Status

This section displays real-time system information from the QEC MDevice, including voltage, current, and internal temperature readings. These indicators help monitor the EtherCAT MDevice health and performance.



Parameters Displayed:

Label	Description
Vs	System (logic) supply voltage, Us
Is	System (logic) supply current
Vp	Peripheral (power) supply voltage, Up
Ip	Peripheral (power) supply current
Temperature	Internal system temperature (in degrees Celsius, °C)

While this information is visible in the 86EVA interface, it can also be accessed programmatically. Users may use ArduBlock or the 86Duino EtherCAT API to read and process these values in their own applications—for instance, triggering alarms, adjusting behavior dynamically, or logging data for analysis.

2.5.4 Virtual Arduino

Virtual resources are mapped from connected EtherCAT SubDevices and used for Arduino-style logic generation within 86EVA.



The MDevice “Virtual Arduino” page of 86EVA provides 16 “Max Virtual …” settings. Each setting can be configured within the following system-defined maximum limits:

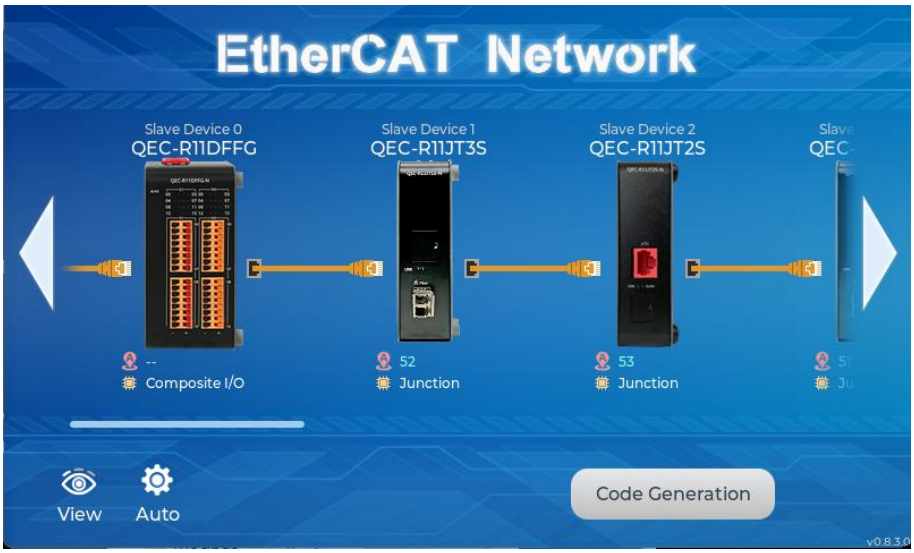
Category	Max Configurable
Virtual Digital Pins	1024
Virtual Integer Data Pins	1024
Virtual Decimal Data Pins	1024
Virtual Serials	64
Virtual Servos	384
Virtual Encoders	96
VirtualCNC Controllers	96
Virtual LCDs	64
Virtual LCMs	64
Virtual MPGs	64
Virtual Keypads	64
Virtual Buzzer Pins	64

***Note:** The practical limits may vary depending on the number of SubDevices in use and available memory on the QEC-MDevice.

These limits give developers tremendous flexibility when designing complex automation systems. All configurations are automatically mapped and translated into executable Arduino code by 86EVA.
For each Virtual Arduino class usage, please refer to Ch. 2.6.3 Virtual Arduino Mapping.

2.6 SubDevice Configuration

86EVA lets users access and configure each recognized SubDevice in the EtherCAT network through a graphical interface. Settings include object name, alias address, vendor/product identification, I/O mapping, and optional device status monitoring.



Each SubDevice can be:

- Identified by its Vendor ID and Product Code
- Assigned a custom Object Name and Alias Address

For verified SubDevices (such as QEC series), 86EVA additionally supports:

- Full product image display
- Virtual Arduino pin mapping for software integration
- Device-specific configuration (e.g., ADC range, servo settings, keypad layout)

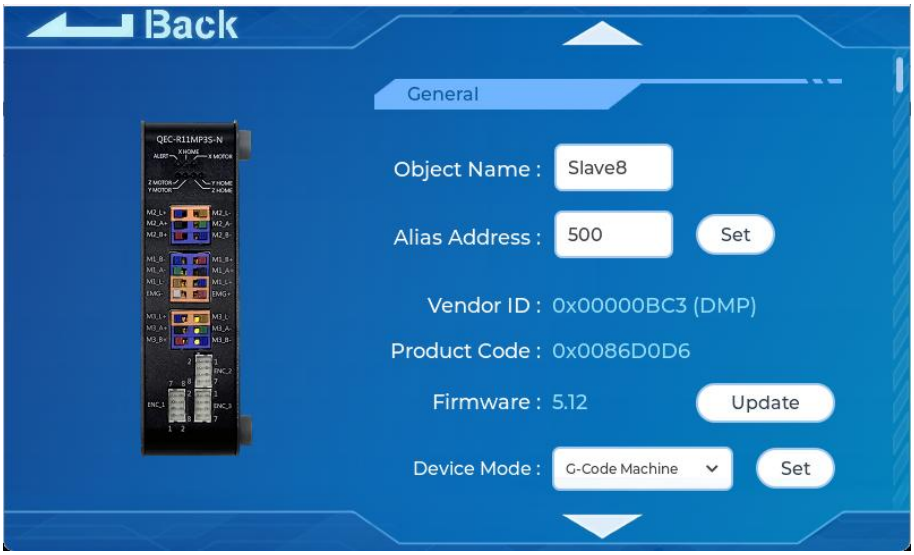
QEC SubDevices can also be monitored in real time for **voltage**, **current**, and **temperature**, aiding system diagnostics and energy tracking.

Unverified third-party devices will appear as "**Unsupported**" if not yet integrated. For the list of QEC and verified third-party SubDevices, refer to Ch. 2.6.5 Supported SubDevices.



2.6.1 General

The General tab displays essential information and configuration options for each EtherCAT SubDevice. In this section, we use the QEC-R11MP3S-N as an example to demonstrate all available functions.

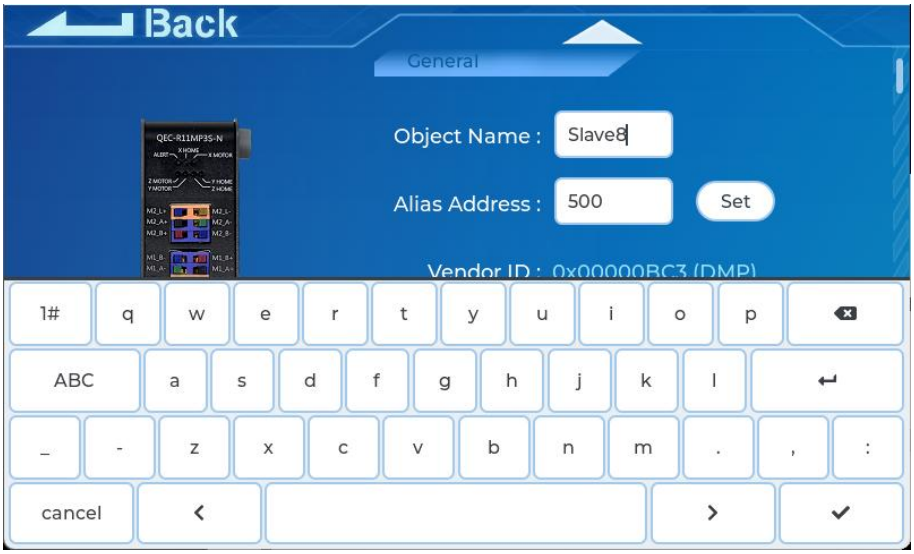


2.6.1.1 Object Name

Define a unique name for the SubDevice. This name is later used in the generated Arduino code to identify the device. Example: A SubDevice named ‘Slave8’ will appear in the code as `Slave8.attach(0, master)`.



The 86EVA displays a virtual keyboard if users click the input area.



2.6.1.2 Alias Address

Assign a station-specific Alias Address. This is written to the device's EEPROM and used to identify the device's position in the EtherCAT network.



After setting the Alias Address, the program will automatically identify the EtherCAT SubDevice using this number. When the user's EtherCAT SubDevice changes in the EtherCAT network due to system requirements, the program will automatically adapt and will not require setting up new configurations; if the Alias Address is not set, the program will not automatically adapt to the new configuration. The default Alias Address is 0; please set a number between 1 and 65535.

***Note:** Adapting to the new configuration refers to switching SubDevices locations (the total number of SubDevices will not increase or decrease). If a device is added or removed, the program must be reconfigured.

After the 86EVA scan of the SubDevice is successful, the Alias Address previously set by the SubDevice is automatically read and displayed in this field. If the value input process is canceled or interrupted, this field will return to its previous value.

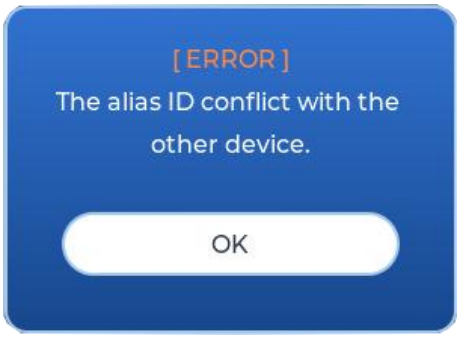
The 86EVA displays a virtual keyboard if users click the input area.



Pressing the "**Set**" button will write the Alias Address to the EtherCAT SubDevice's EEPROM, and 86EVA will display a **[Success]** window if it succeeds.



If there are two identical Alias Addresses, 86EVA will display an [ERROR] window.



If the Alias Address cannot be written, check the firmware version of the device.
Users can also use the "**Auto**" function on the main page settings. For detailed information, please refer to Ch. 2.4 Device Scan and User Interface Overview.

*__Note__: After closing 86EVA, please remember to power cycle the EtherCAT SubDevice.

2.6.1.3 Vendor ID

This displays the Vendor ID of the EtherCAT SubDevice (read-only).
It uniquely identifies the manufacturer of the device. For the list of registered Vendor IDs, refer to EtherCAT Technology Group | Vendor ID.



2.6.1.4 Product Code

This shows the Product Code of the EtherCAT SubDevice (read-only).
Used to distinguish different models or versions within the same vendor.



2.6.1.5 Firmware

Shows the current firmware version of the device.
Click the “**Update**” button to flash the latest firmware (QEC SubDevice only).



Pressing the "**Update**" button, 86EVA will display a **[FIRMWARE DOWNGRADE/UPGRADE]** window, asking whether you want to start updating the QEC SubDevice’s firmware.

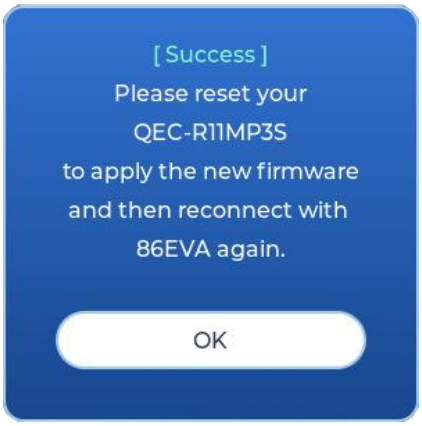


*__Note__: only the latest version of QEC SubDevice’s firmware is available in the 86EVA.

Click **“Yes”** to start updating the firmware, and 86EVA will display a **[Processing]** window showing the process.



After success, 86EVA will display a **[Success]** window.



After clicking **“OK”**, 86EVA returns to the main page, and the updated device shows “**[device to reset]**” in blue text.



If you click the device again, it will display a window showing **[Firmware Update OK]**.



After power cycle the EtherCAT SubDevice and scanning again, you can see that the device you updated has already been updated to the new firmware version, and without the update button.

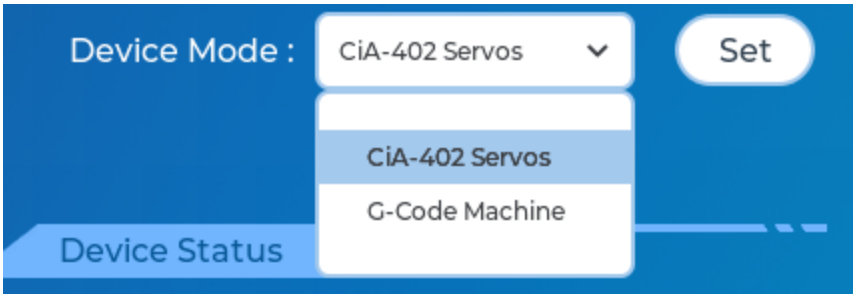


2.6.1.6 Device Mode

This option allows users to select the operating mode for the QEC Motor Driver specifically. Each mode corresponds to a different internal profile that the device supports.



The 86EVA displays a drop-down list for the device mode if users click.



After selecting the desired mode, click the **"Set"** button. If successful, 86EVA will display a **[Success]** window.



If an error occurs, 86EVA will display an [ERROR] window.



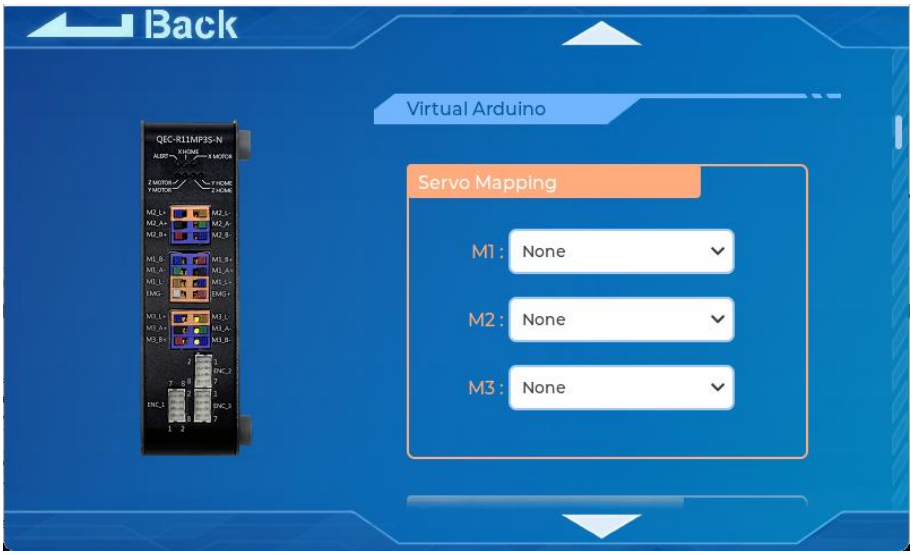
If the device mode cannot be written, check the firmware version of the device.
After applying the new device mode, restart the EtherCAT SubDevice to ensure the configuration takes effect properly.

2.6.1.6.1 Supported Device Modes

Depending on the SubDevice type and firmware, such as QEC-RXXMP3S and QEC-RXXMV3S, the following modes may be available:

Mode Name	Description
G-Code Machine	Executes G-code instructions, typically used in CNC systems.
CiA-402 Servos	Uses CiA 402 protocol, commonly adopted for servo control.

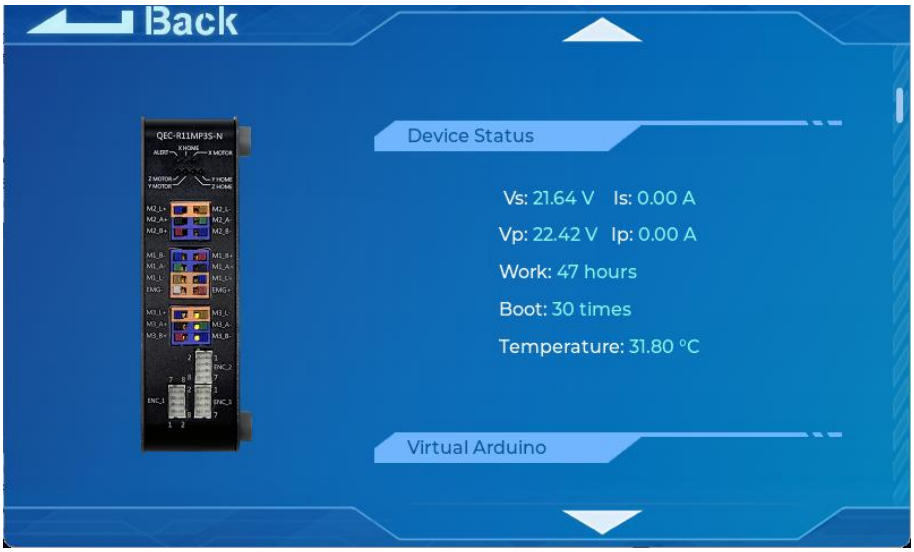
***Note**: The availability of modes depends on the specific model and firmware of the QEC SubDevice.



Refer to Ch. 2.6.3 Virtual Arduino Mapping and Ch. 2.6.4 Other Configuration for more about Virtual Arduino Mapping and related settings.

2.6.2 Device Status

This section displays real-time monitoring data from the QEC SubDevice, including voltage, current, operating hours, boot count, and temperature. These metrics are useful for diagnostics, maintenance tracking, and thermal management.



Displayed Parameters:

Label	Description
Vs / Is	System voltage and current.
Vp / Ip	Peripheral voltage and current.
Work	Total operating time since first power-on (in hours).
Boot	Number of times the SubDevice has been restarted.
Temperature	Real-time internal temperature of the device (in degrees Celsius, °C).

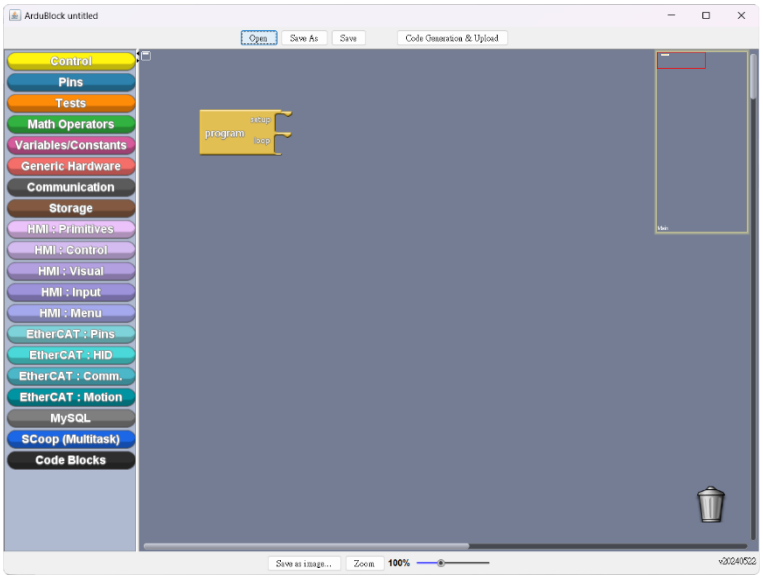
***Note:** This data is also accessible through ArduBlock or the 86Duino EtherCAT API, allowing applications to monitor health status, detect thermal risk, and optimize behavior dynamically. For related programming reference, see Ch. 3 API Overview.

2.6.3 Virtual Arduino Mapping

After defining the maximum number of virtual pins in the QEC MDevice, each SubDevice can be mapped to one or more of these virtual resources.



This mapping enables high-level application logic (via ArduBlock or code) to interact with hardware pins through a unified interface.



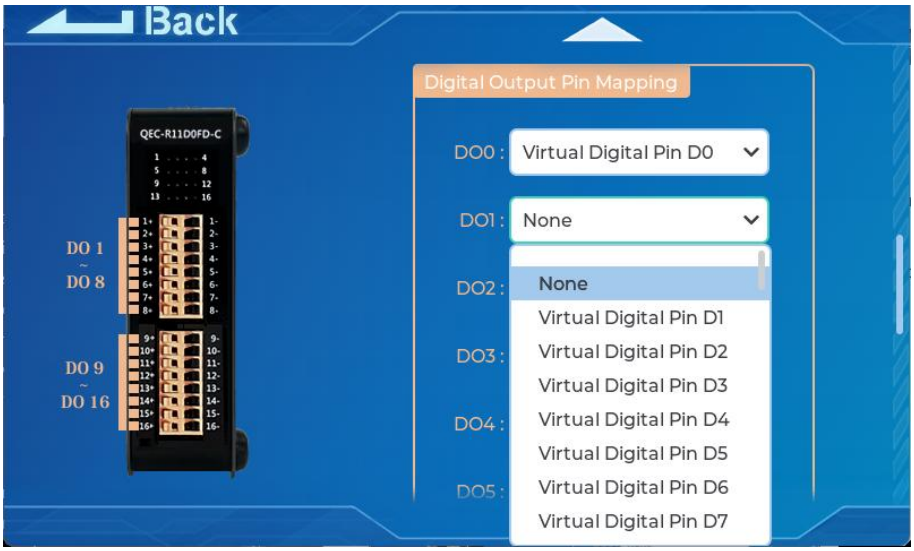
The following subsections introduce the available virtual pin types and how to configure them in 86EVA.
The number of virtual resources listed in each drop-down list (e.g., Virtual Servo1 to Virtual ServoN) depends on the corresponding “Max Virtual …” setting of the MDevice (see Ch. 2.5.4). The ranges shown in the examples below are for reference only.

***Note:** If the selected virtual pin is already mapped to another physical pin, 86EVA displays a **[WARNING]** window: “The virtual pin has already been mapped to other pin!”

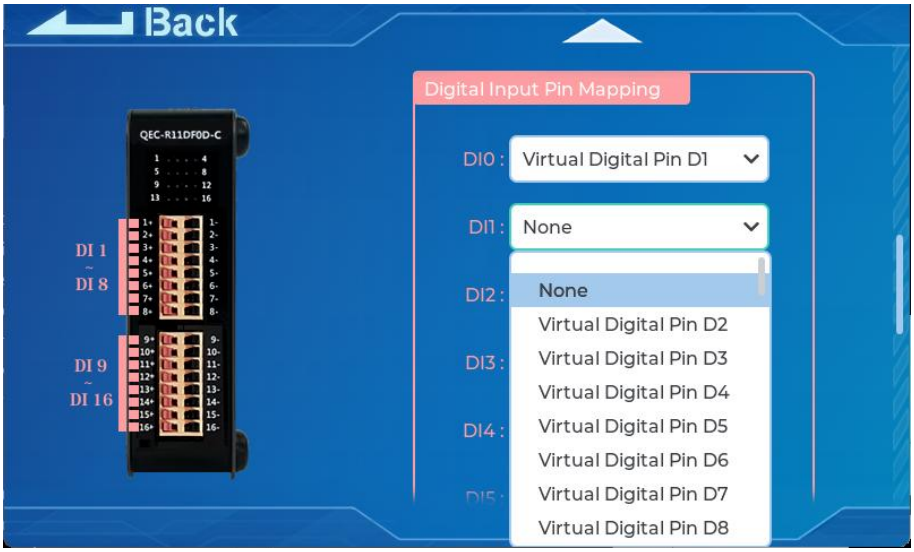
2.6.3.1 Virtual Digital Pin

Users can map available digital I/O pins on supported EtherCAT SubDevices to Virtual Digital Pins in the 86EVA environment.
As shown in the image below, 86EVA provides a drop-down list for each physical digital I/O pin. Users can assign a corresponding Virtual Digital Pin (e.g., D0 ~ D7) to control these pins via graphical or code-based methods.

Digital Output:



Digital Input:

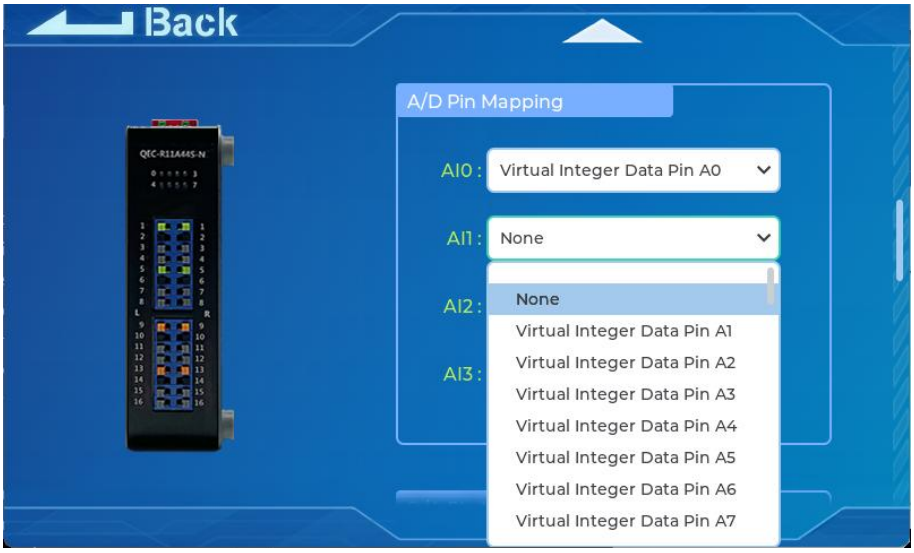


***Note:** This function is not limited to QEC DIO modules. Any DIO-type EtherCAT SubDevice that has passed compatibility testing with 86EVA can also support this mapping feature. Once assigned, Virtual Digital Pins can be controlled via graphical tools such as ArduBlock or through the 86Duino API.

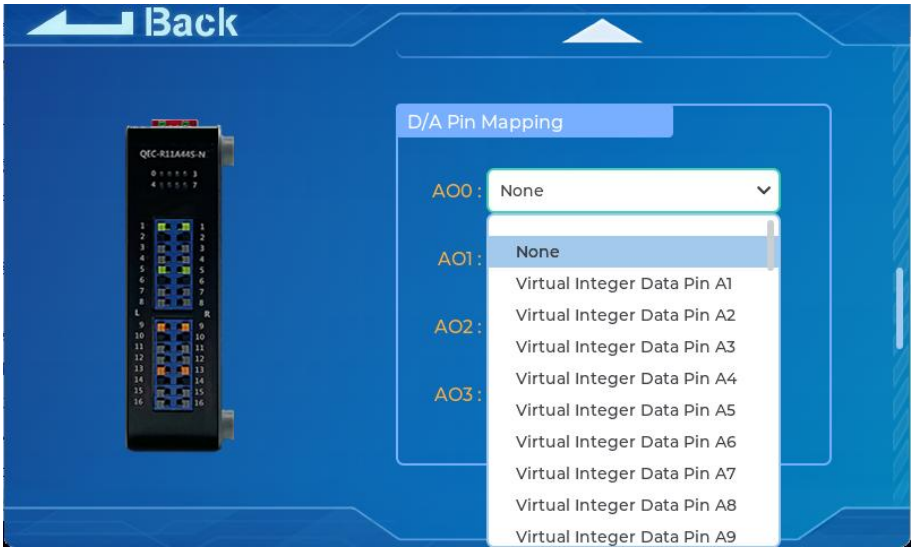
2.6.3.2 Virtual Integer Data Pin

Users can map available analog input/output (AIO) pins on supported EtherCAT SubDevices to Virtual Integer Data Pins in the 86EVA environment.
As shown in the image below, 86EVA offers a drop-down menu for each physical analog pin, allowing users to assign a Virtual Integer Data Pin (e.g., A0~A7). These mapped pins are accessed through the 32-bit integer data API; for QEC analog modules, the valid value range is 0 ~ 65535 (16-bit). They can be read or written through graphical or code-based methods.
The analog range configuration has to follow the device specification, and you can refer to Ch. 2.6.4.1 ADC/DAC Range Configuration.

Analog Input:



Analog Output:

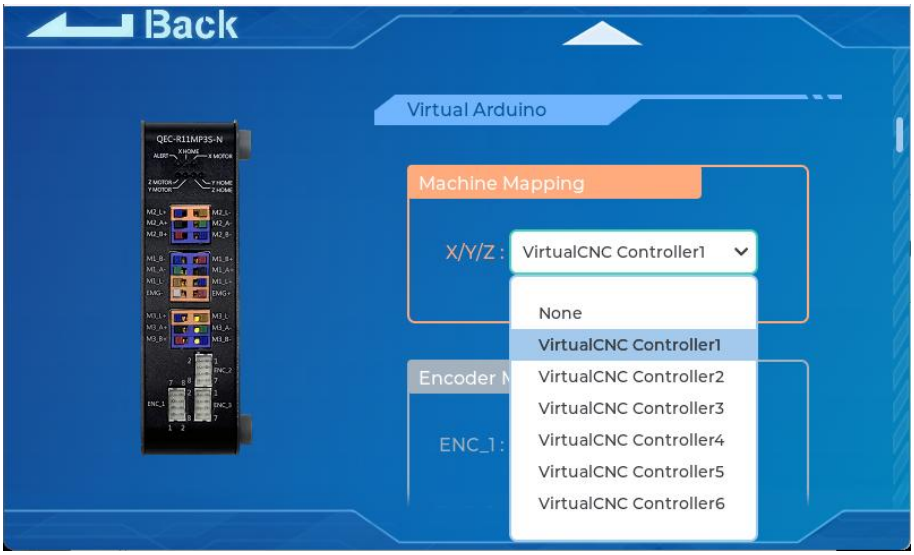


***Note:** Once assigned, the Virtual Integer Data Pin can be accessed through ArduBlock or the 86Duino API.

2.6.3.3 Virtual Machine

The Virtual Machine feature allows users to map a group of three physical motor axes (X, Y, Z) to a VirtualCNC Controller in the 86EVA environment. This function is specifically designed for QEC-RXXMP3S and QEC-RXXMV3S series three-axis motor drivers that support **G-code mode**.

As shown in the image below, 86EVA provides a simple drop-down menu for assigning a VirtualCNC Controller (e.g., VirtualCNC Controller1 to Controller6) to the corresponding three-axis machine.
This enables users to implement coordinated motion and CNC-style G-code control for industrial applications such as engraving machines, tool changers, or pick-and-place systems.



For the machine parameter settings, the user can refer to Ch. 2.6.4.2 Machine Configuration.

***Note:** Once mapped, the VirtualCNC Controller can be accessed through ArduBlock or the 86Duino API using G-code commands. It provides a high-level abstraction for motion control, simplifying development for G-code-based automation.

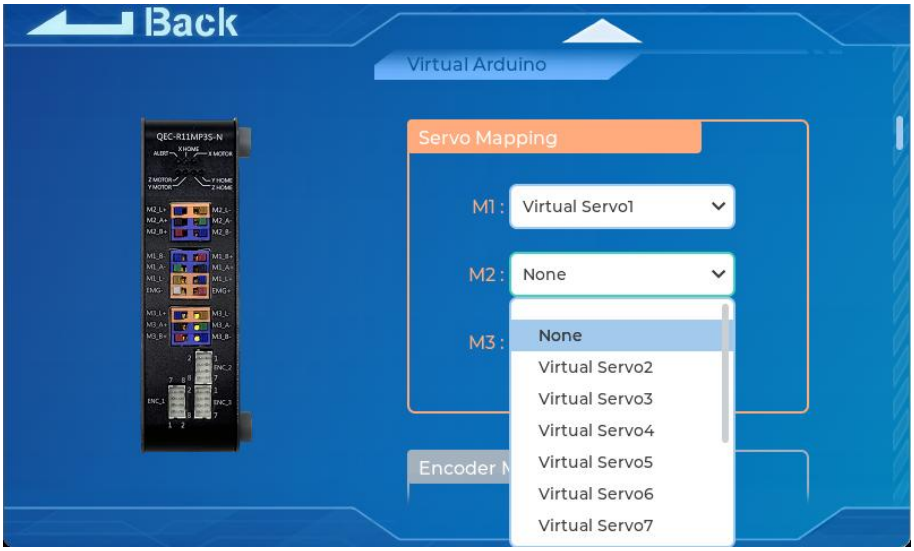
2.6.3.4 Virtual Servo

Users can map supported EtherCAT servo axes (e.g., from QEC-R11MP3S series) that support the CiA 402 profile to Virtual Servo instances in the 86EVA environment.

As shown in the image below, 86EVA provides a drop-down menu for each physical axis (M1, M2, M3). Users can assign a Virtual Servo (e.g., Virtual Servo1 to Virtual Servo7) to a selected axis, allowing motion control developers to abstract motor devices and manage them through Virtual Servo APIs or visual programming tools like ArduBlock.

This Virtual Servo will configure the CiA402 Profile Position (PP) mode by default. For the servo parameter settings, the user can refer to Ch. 2.6.4.3 Servo/Virtual Servo Configuration.

Servo:

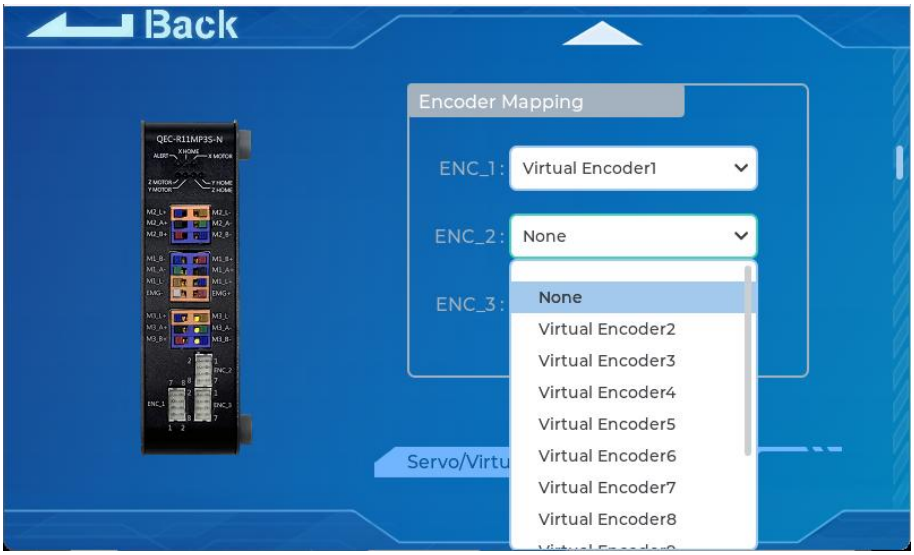


***Note:** This feature is not limited to the QEC-R11MP3S model. Any EtherCAT motor driver that supports CiA 402 and has passed 86EVA compatibility testing can be mapped to Virtual Servo instances. Once mapped, the Virtual Servos are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.5 Virtual Encoder

Users can map physical encoder inputs (ENC) on supported EtherCAT SubDevices (e.g., from QEC-R11MP3S series) to Virtual Encoder objects within the 86EVA environment. This mapping allows encoder signals to be accessed and interpreted programmatically via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

As shown in the image below, 86EVA provides a drop-down selection menu for each encoder input channel (ENC_1, ENC_2, ENC_3). Users can assign each input to a corresponding Virtual Encoder (e.g., Virtual Encoder1 to Virtual Encoder8).



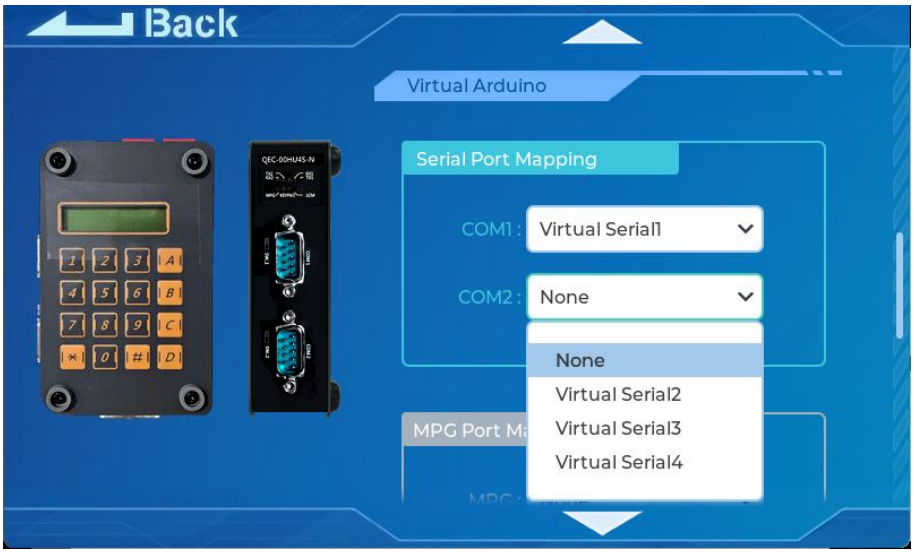
Once configured, Virtual Encoders can be used to read position values, direction, and other encoder-related data. For the encoder parameter settings, the user can refer to Ch. 2.6.4.4 Encoder Configuration.

***Note:** The Virtual Encoder feature is primarily designed for QEC-R11MP3S and other compatible EtherCAT motion modules. Once mapped, the encoders are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.6 Virtual Serial

The Virtual Serial function allows users to map physical RS-232/RS-485 ports on supported QEC HID SubDevices to Virtual Serial interfaces in the 86EVA environment. This mapping enables bi-directional serial communication with external devices such as barcode scanners, force sensors, RFID readers, and other serial-based peripherals.

As shown in the image below, 86EVA provides a drop-down configuration menu for each available serial port (e.g., COM1, COM2). Users can assign the ports to Virtual Serial interfaces (e.g., Virtual Serial1 ~ Virtual Serial4).



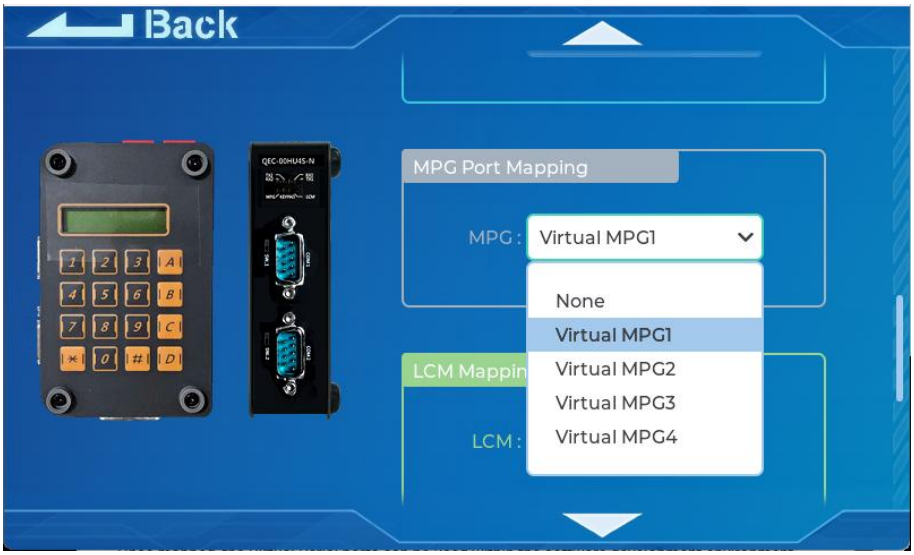
Once mapped, the Virtual Serial ports can be utilized in the development environment for:

- Sending and receiving serial commands
- Integrating third-party serial devices (e.g., Barcode, Force Sensors, RFID, 3D Joysticks)
- Building custom communication protocols
- Programming serial communication using ArduBlock blocks
- Accessing 86Duino Serial APIs for advanced control
- Accessing 86Duino Modbus APIs for advanced communication

*__Note:__ This feature is only available for QEC HID models with physical serial interfaces that have passed 86EVA compatibility testing. Once mapped, the Virtual Serial ports are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.7 Virtual MPG

The Virtual MPG (Manual Pulse Generator) function allows users to map a supported MPG port from the QEC HID series to a Virtual MPG interface in the 86EVA environment. This mapping enables rotary handwheel devices to be used in CNC or motion control applications via software. As shown in the image below, 86EVA provides a drop-down selection menu for MPG port mapping. Users can assign an MPG port (e.g., MPG:) to a Virtual MPG controller (Virtual MPG1 ~ Virtual MPG4).



This function supports integration with real MPG hardware for:

- Axis selection (X/Y/Z/A/B)
- Multiplier ratio control (1x, 10x, 100x)
- Position increment input
- Emergency stop and enable button reading

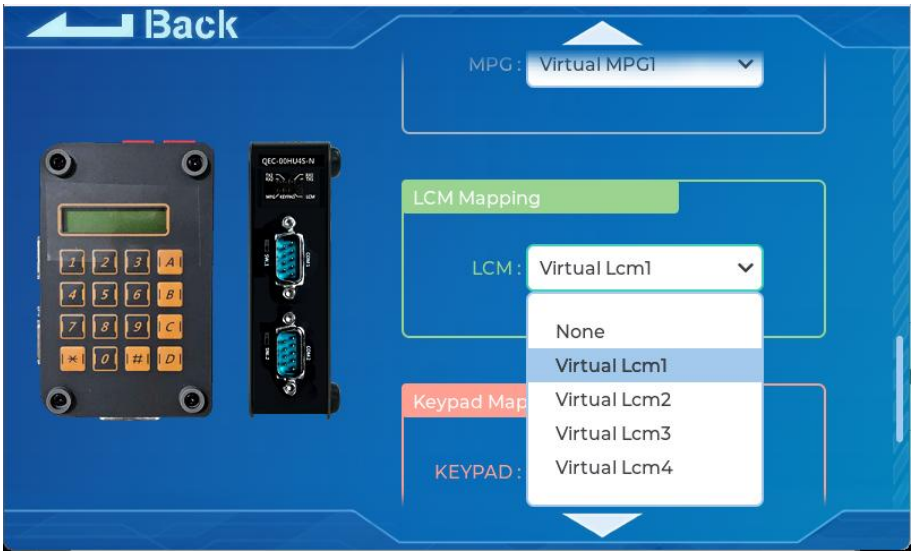
Once mapped, the Virtual MPG instance can be accessed via ArduBlock or the 86Duino API, allowing real-time control, feedrate tuning, or manual axis positioning.

***Note:** This feature is available only for QEC HID models with MPG ports that have passed compatibility validation in 86EVA. Once mapped, the Virtual MPGs are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.8 Virtual LCM

The Virtual LCM (Liquid Crystal Module) mapping feature in 86EVA allows users to bind an LCM port from a supported QEC HID module to a Virtual LCM interface. This is designed for 16x2 text-based LCD displays connected via EtherCAT-compatible HID ports.

As shown in the image below, users can select from Virtual Lcm1 to Virtual Lcm4 using a drop-down menu in 86EVA. Once assigned, these mapped virtual devices enable users to display text and interact with simple UI elements.

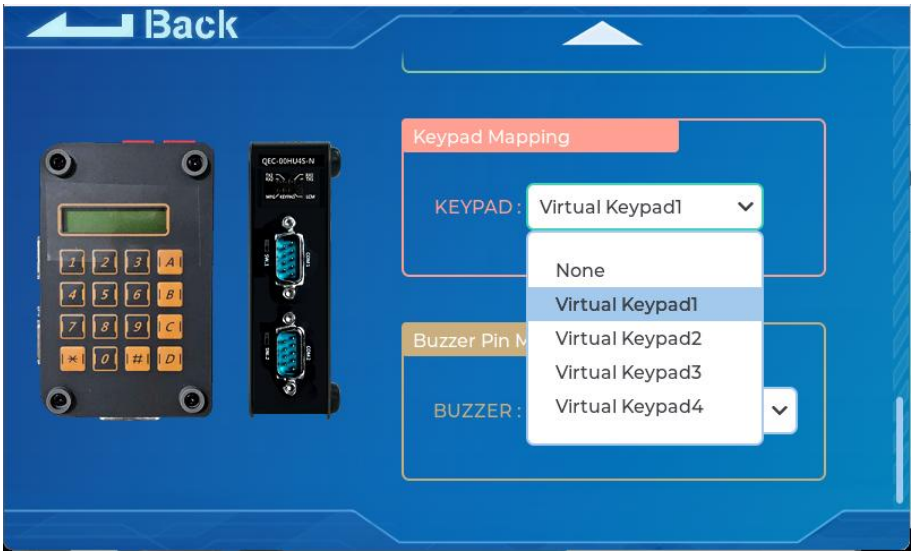


*__Note__: Only LCM-capable EtherCAT HID modules validated with 86EVA can utilize this feature. Once mapped, the Virtual LCMs are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.9 Virtual Keypad

The Virtual Keypad mapping function in 86EVA allows users to bind a physical matrix keypad from supported QEC HID modules to a virtual interface in the system. This function is essential for applications that require manual data entry, shortcut triggering, or menu navigation using tactile keys.

As shown in the image below, 86EVA offers a drop-down list for selecting from Virtual Keypad1 to Virtual Keypad4. Once mapped, key inputs from the hardware keypad can be captured by ArduBlock or the 86Duino API for further processing.

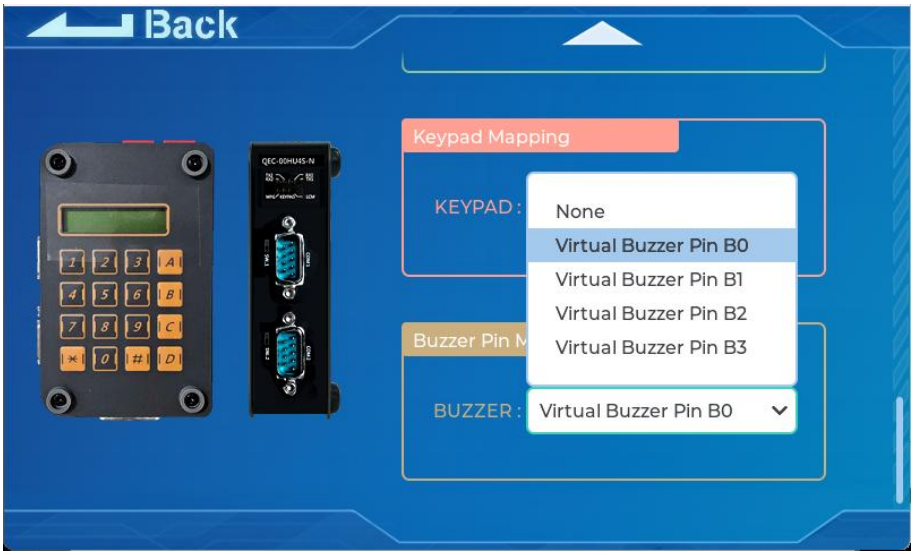


***Note:** This feature is only available on HID modules that support keypad input and have passed 86EVA compatibility testing. Once mapped, the Virtual Keypads are accessible via graphical interfaces (e.g., ArduBlock) or through the 86Duino API.

2.6.3.10 Virtual Buzzer Pin

The Virtual Buzzer Pin mapping function in 86EVA allows users to link a physical buzzer module—typically found in QEC HID devices—to a virtual pin (B0 ~ B3) that can be controlled in code or through a graphical interface.

As shown in the image below, users can select from Virtual Buzzer Pin B0 to B3 using the drop-down menu. Once mapped, the buzzer can be toggled via ArduBlock or the 86Duino API to emit audible alerts based on system events or user input.



*__Note__: This function is exclusive to HID-compatible devices that support buzzer output and have passed 86EVA validation. Virtual Buzzer Pins can be programmed for tone generation, pulse-based signaling, or simple on/off logic.

2.6.3.11 Other Mapping Types

Depending on the connected SubDevice, 86EVA provides the following additional mapping fields. They work in the same way as the mapping types above: select a virtual resource from the drop-down list of each physical channel, then use the mapped object in your code or in ArduBlock.

Mapping field in 86EVA	**Mapped to**	**Object / API in generated code**
Counter Pin Mapping / Digital Counter Pin Mapping	Virtual Integer Data Pin (A0 ~)	`EVA.integerDataRead()`
Detect Voltage Pin Mapping	Virtual Decimal Data Pin (F0 ~)	`EVA.decimalDataRead()`
VP Port Mapping / VS Port Mapping	Virtual Decimal Data Pin (F0 ~)	`EVA.decimalDataRead()`
2.4" LCD Mapping	Virtual LCD	MyVirtualLCD (VirtualLcdN)
Microphone Mapping	Virtual Mic	MyVirtualMicrophone (VirtualMicN)
Audio Mapping	Virtual Audio	MyVirtualSpeaker (VirtualSpeakerN)
Sensor Mapping	VirtualEM Sensor	MyVirtualEMSensor (VirtualEMSensorN)
Force Sensor	Virtual Force Sensor	MyVirtualForceSensor (VirtualForceSensorN)

***Note:** The available mapping fields depend on the SubDevice model and firmware. The maximum number of each virtual resource is set in Ch. 2.5.4 Virtual Arduino. For the API of these classes, see Ch. 3.10 to Ch. 3.14.

2.6.4 Other Configuration

In addition to Virtual Pin Mapping, 86EVA provides several configuration options to help users fine-tune SubDevice behavior, ensure compatibility, and optimize system performance. These settings are essential for advanced applications and typically include hardware-related parameters.



These configurations can vary by SubDevice type and are only visible if supported by the connected EtherCAT module.

2.6.4.1 ADC/DAC Range Configuration

In 86EVA, users can configure the analog input (AI) and analog output (AO) voltage or current ranges for QEC AIO modules. These settings ensure the connected devices operate within their required electrical specifications and improve measurement accuracy and signal safety.

Hardware Reference

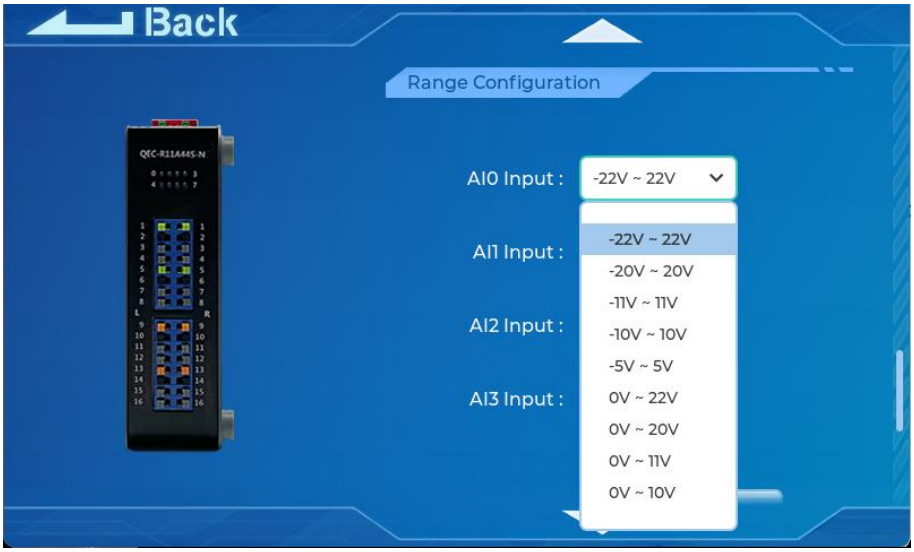
The QEC-RXXA44S-N module supports 4-channel differential analog input and 4-channel analog output. Refer to the specification below:

Analog Input	
Input Type	Voltage (V)
Input Range	±22 V; ±20 V; ±11 V; ±10 V; ±5 V; 0 V – 22 V; 0 V – 20 V; 0 V – 11 V; 0 V – 10 V
Input Impedance	≥ 1 MΩ
Analog Output	
Output Type	Voltage (V, mV), Current (mA)
Output Current	0 mA – 24 mA; 3.5 mA – 23.5 mA; 0 mA – 20 mA; 4 mA – 20 mA; ±24 mA
Output Voltage	0 V – 5 V; 0 V – 10 V; ±5 V; ±10 V; 0 V – 6 V; 0 V – 12 V; ±6 V; ±12 V

***Note:** The selection of voltage or current range must match your wiring method and load specification.

Analog Input Range Configuration

As shown in the image below, users can select an input voltage range for each analog input channel. The supported input types are voltage-based, with differential inputs and a high input impedance (≥1MΩ).



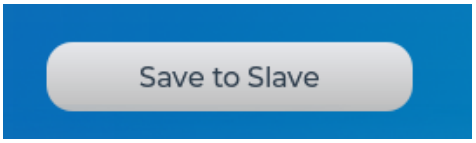
Analog Output Range Configuration

Depending on the application requirement, analog outputs can be configured to provide either voltage or current outputs.

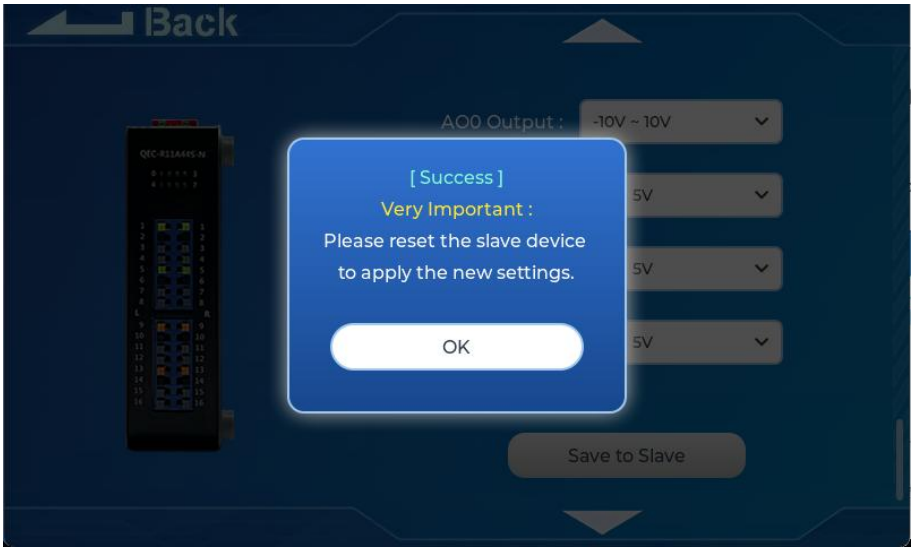


*__Note__: Ensure that the selected input/output range matches the sensor or device's output range to avoid damage or inaccurate readings.

After selecting the desired ranges, press the **“Save to Slave”** button to apply the settings to the EtherCAT SubDevice by SDO read/write.



86EVA will display a **[Success]** window if it succeeds.

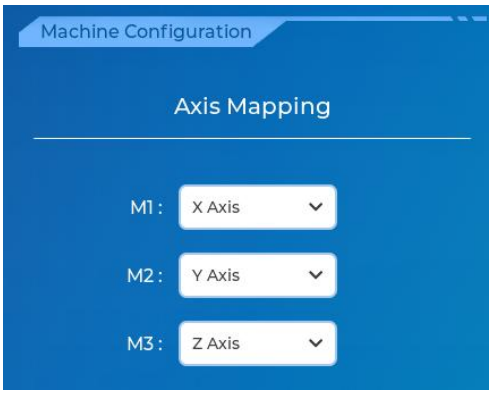


2.6.4.2 Machine Configuration

In 86EVA, the Machine Configuration interface allows users to define key mechanical parameters for QEC 3-axis motor controllers operating in G-code mode (e.g., QEC-RXXMP3S and QEC-RXXMV3S series). These settings are critical for precise motion control, including unit calibration, motion speed, homing behavior, and software limits.

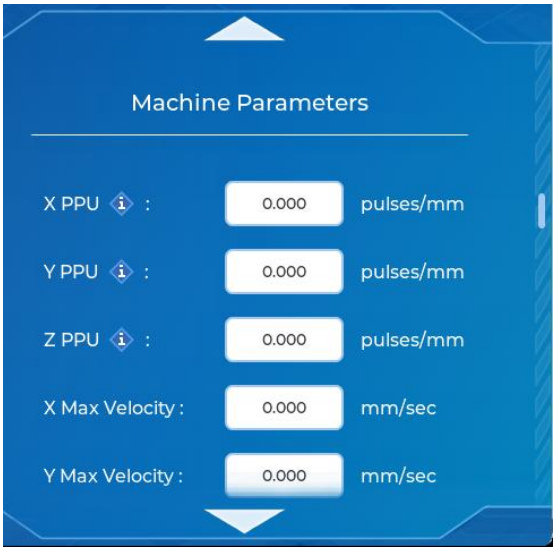
Axis Mapping

Users can assign each motor channel (M1–M3) to a specific axis (X, Y, or Z), which determines the motion logic in G-code execution.

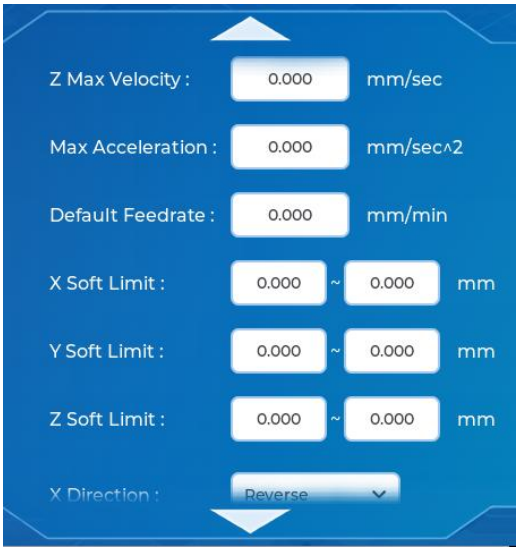


Machine Parameters

Users can configure the following parameters for each axis.



- PPU (Pulses Per Unit): Defines how many encoder pulses represent one unit of distance (mm), used for converting motor steps to physical travel.
- Max Velocity: The maximum speed (in mm/sec) each axis can move during operation.
- Max Acceleration: Sets the maximum acceleration (in mm/sec²) during movement.



- Default Feedrate: The default linear speed (in mm/min) when executing G-code commands without an explicit feedrate.
- Soft Limit Range: Define minimum and maximum allowable positions (in mm) to prevent mechanical overrun or collision.





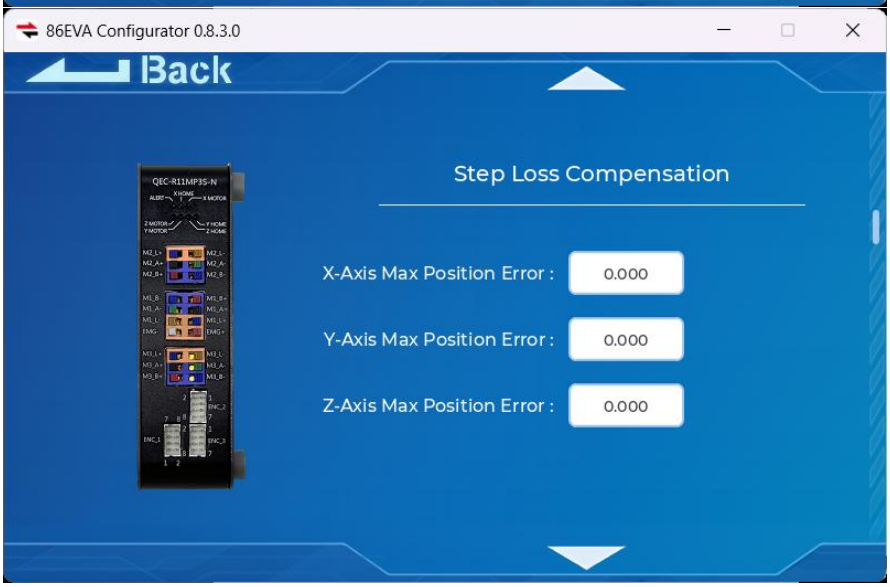
- Axis Direction: Set the positive motion direction for each axis (Normal or Reverse) depending on motor wiring and mechanical layout.
- Homing Speed: Speed (mm/min) at which each axis returns to its home position.
- Homing Direction: Direction (Positive or Negative) for homing motion.
- Homing Order: Defines the sequential order in which X, Y, and Z axes perform the homing procedure.

For QEC motor controllers with encoder feedback, the Machine Configuration page also provides the following items:

- **Encoder-to-Motor Mapping**: Assigns each encoder channel (ENC_1 ~ ENC_3) to a motor channel (M1 ~ M3) for position feedback.
- **Encoder-to-Position Transformation**: Sets the multiplier (ENC_n Counts x) that converts encoder counts into the motor position feedback (Mn Position Feedback).

- **Step Loss Compensation**: Sets the maximum allowable position error of each axis (X/Y/Z-Axis Max Position Error). When the difference between the command position and the encoder feedback exceeds this value, the position is compensated.

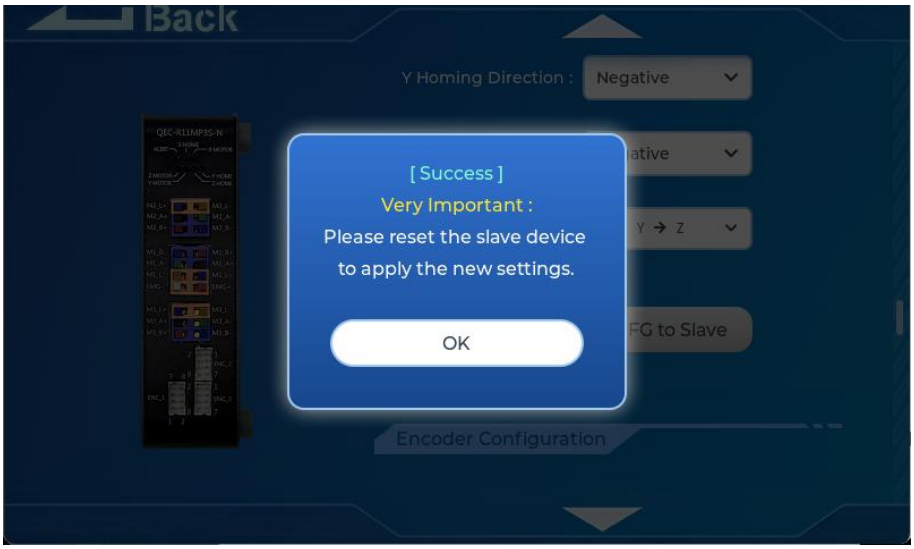




After completing the configuration, click the **"Set Machine CFG to Slave"** button to save the settings via SDO to the EtherCAT SubDevice.

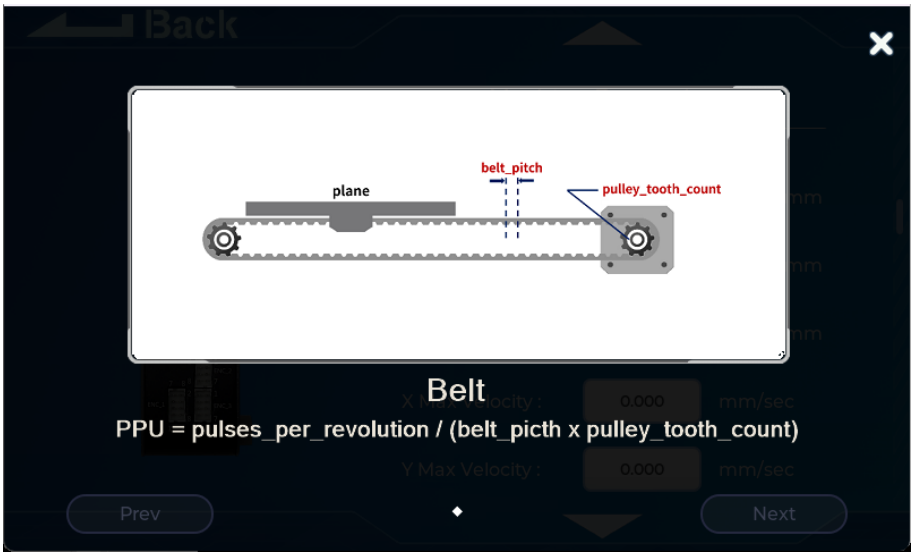


86EVA will display a **[Success]** window if it succeeds.



*__Note__: These machine parameters are essential for safe and reliable motion control. Always validate values according to your mechanical design and coordinate system before applying the configuration.

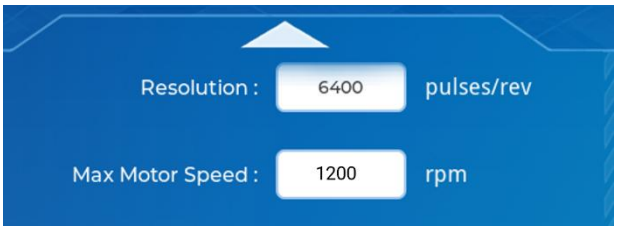
*__Note:__ Click the **<i>** button, 86EVA will display the PPU explanation.



2.6.4.3 Servo/Virtual Servo Configuration

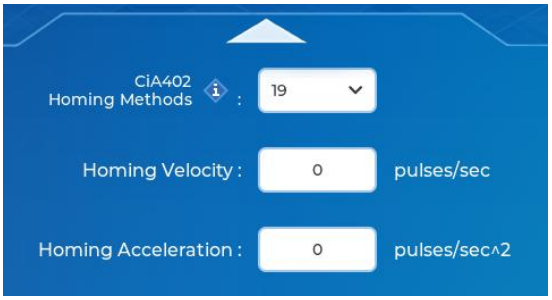
In 86EVA, the Servo/Virtual Servo Configuration interface allows users to define motion parameters for each axis (M1–M3), applicable to any EtherCAT SubDevice that supports the CiA402 standard and has passed 86EVA compatibility testing.
These parameters are critical for achieving precise and smooth motor control in standard motion applications.

Motion Parameters (per Axis):



- Resolution (pulses/rev): Sets the number of pulses per full revolution, based on the motor encoder or stepper driver configuration (e.g., 3200, 10000).
- Max Motor Speed (rpm): Sets the maximum motor speed, based on the motor specification.
- Max Acceleration / Max Deceleration (acc units): Sets the maximum acceleration and deceleration of the axis.

Homing Parameters (per Axis):



- CiA402 Homing Method: Selects the homing approach according to CiA402 standard.
- Homing Velocity (pulses/sec): Speed at which the axis moves during homing.
- Homing Acceleration (pulses/sec²): Acceleration applied during homing to avoid overshooting the switch.

Velocity Unit (per Axis):

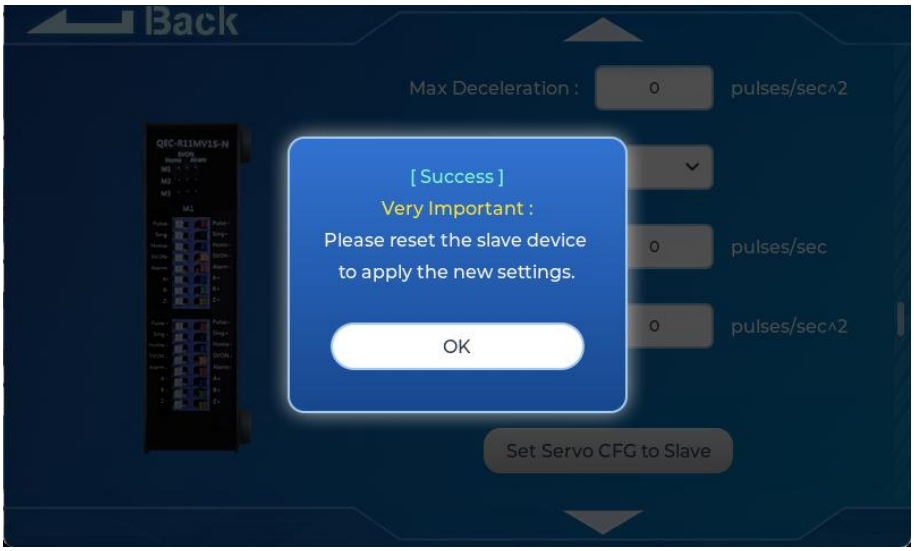


- Velocity Unit: Selects the velocity unit for each axis in velocity mode.

After completing the configuration, click the **“Set Servo CFG to Slave”** button to save the settings via SDO to the EtherCAT SubDevice.

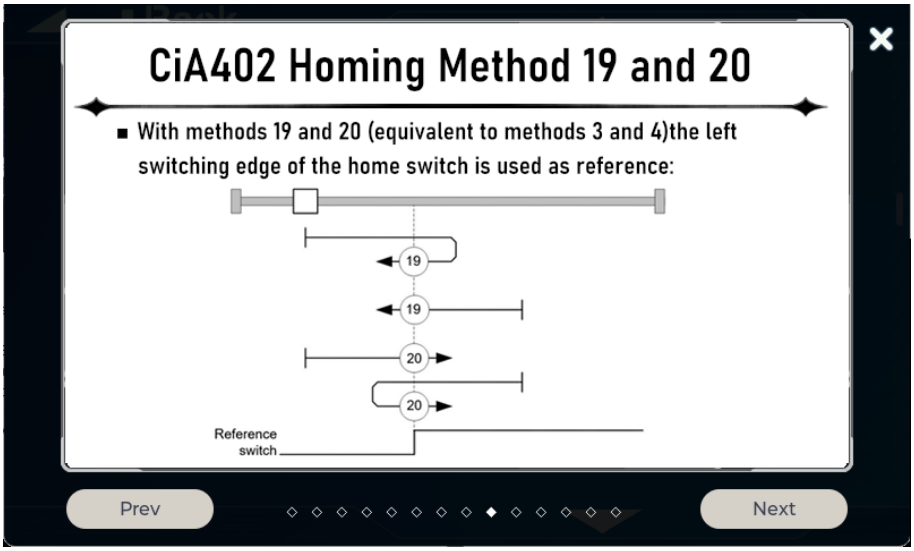


86EVA will display a **[Success]** window if it succeeds.



*__Note__: These servo parameters are essential for safe and reliable motion control. Always validate values according to your mechanical design and coordinate system before applying the configuration.
This feature works with both QEC-native and third-party CiA402-compatible devices. Configuration is per axis, and all three axes can be independently tuned. Always validate resolution and speed limits with your motor's rated specifications to ensure safe operation.

*__Note__: Click the **<i>** button, 86EVA will display the CiA 402 homing methods' explanation.

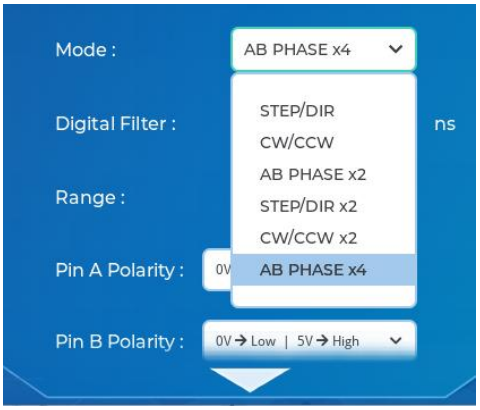


2.6.4.4 Encoder Configuration

In 86EVA, the Encoder Configuration interface allows users to define encoder parameters for EtherCAT encoder SubDevices or QEC motor controller modules (e.g., QEC-R11MV3S). This ensures accurate pulse counting, direction detection, and index reset functionality required for position feedback in motion control.

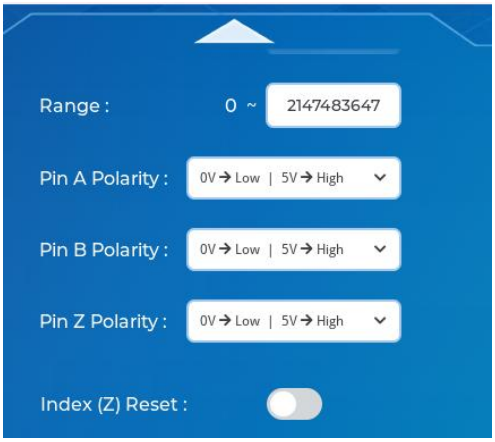


- Mode Selection
  Users can select the appropriate decoding mode based on their encoder type.
  Supported modes include:
  1. STEP/DIR
  2. CW/CCW
  3. AB PHASE x2
  4. STEP/DIR x2
  5. CW/CCW x2
  6. AB PHASE x4



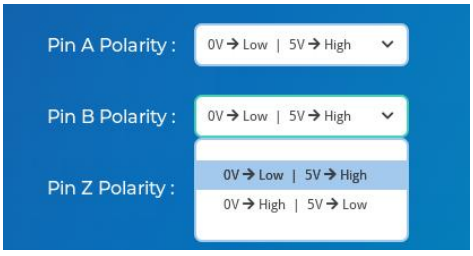


- Digital Filter (ns): Sets a hardware noise filter in nanoseconds to reject high-frequency interference.
- Range (counts): Specifies the maximum encoder count value before rollover. Default is 2147483647.



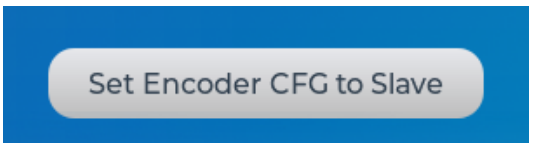
- Pin Polarity: Users can define the signal polarity for each encoder input:
    1. Pin A Polarity
    2. Pin B Polarity
    3. Pin Z (Index) Polarity

    Options:
    1. 0V → Low | 5V → High
    2. 0V → High | 5V → Low

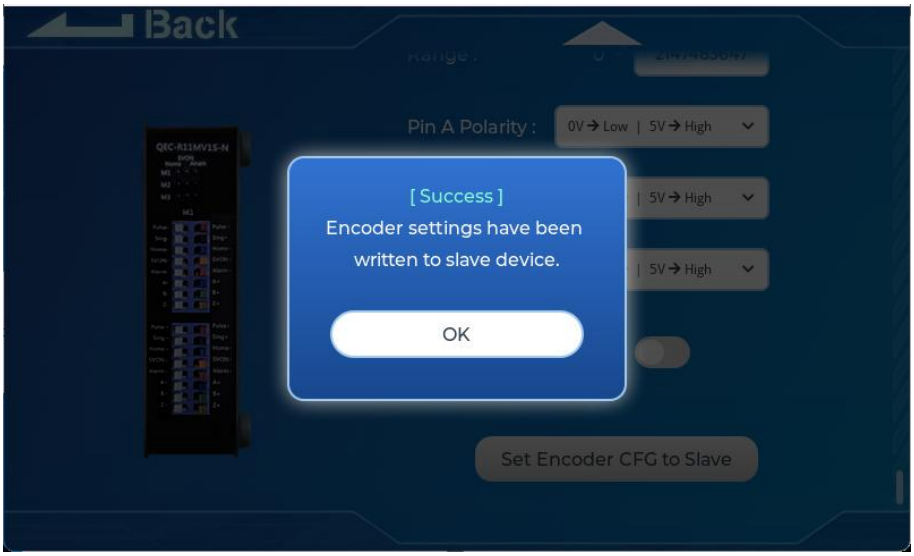


    This setting should match the electrical characteristics of your encoder signals.
- Index (Z) Reset: Enabling this option resets the encoder counter when the Z pulse is detected.

After completing the configuration, click the **“Set Encoder CFG to Slave”** button to save the settings via SDO to the EtherCAT SubDevice.



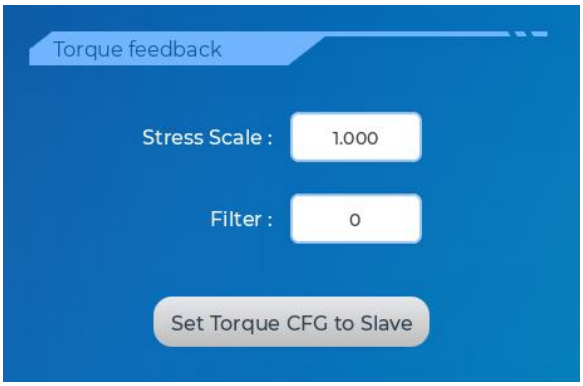
86EVA will display a **[Success]** window if it succeeds.



***Note:** Ensure that encoder mode and polarity settings are consistent with your encoder’s datasheet and wiring. Incorrect configurations may result in inaccurate position tracking.

2.6.4.5 Torque Feedback Configuration

For SubDevices that provide torque feedback, 86EVA provides the Torque feedback settings:



- **Stress Scale**: Sets the scale used to convert the sensor reading into torque (Nm).
- **Filter**: Sets the filter applied to the torque reading.

The Device Status shows the torque value (Nm) and the raw ADC value. After completing the configuration, click the “**Set Torque CFG to Slave**” button to save the settings via SDO to the EtherCAT SubDevice.

2.6.5 Supported SubDevices

86EVA identifies each SubDevice by its Vendor ID and Product Code. The following SubDevices are recognized by 86EVA Configurator 0.8.3.0 (86Duino Coding IDE 501) and are displayed with a product image and device-specific settings.

QEC SubDevices

Type	Model family	Models (QEC-R00 / QEC-R11 series)
Digital I/O	QEC-RxxD	D0FH, D0FS, D0FK, DF0D, DF0H, DF0S, DF0K, D88D, D88H, D88S, D88K, DC4D, D4CD, DT0H, DT0L, D0TH, D0TL
Composite I/O	QEC-RxxC	CFFG, CFFU, CFFL, CT0U, C0TU
Analog I/O	QEC-RxxA	A44S, A40S, A04S
HID (UART, MPG, Keypad, LCM, Buzzer)	QEC-RxxHU	HU1S, HU5S, HU8S, HU9S
3-axis motor driver	QEC-RxxM	MP3S, MV3S, MV6S
MIC + Counter	QEC-RxxMC	MC1S, MC4S, MC8S
Voltage monitoring	QEC-RxxVM	VM2S, VM4S, VM6S, VM8S
Environmental monitoring	QEC-R11EM	EM1S
AC power monitoring	QEC-R11AC	ACPS, ACSS
Junction	QEC-R11JT	JT2S, JT3S, JT6S
Others	—	QEC-RxxUN01, QEC-T120, QEC-T600, QEC-L360, QEC-Force, 86SENSE

***Note:** Availability of the QEC-R00 and QEC-R11 variants differs by model. Refer to the user manual of each QEC SubDevice for its specifications.

Verified third-party SubDevices

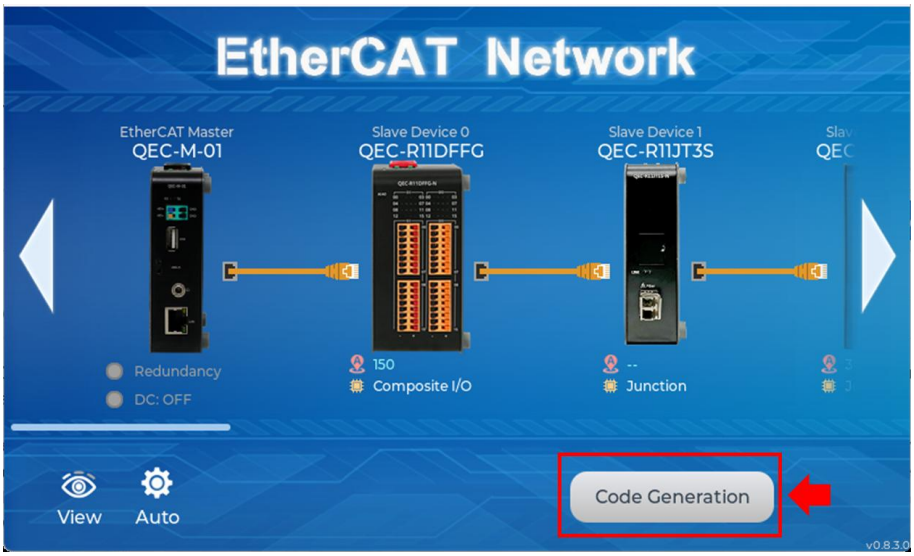
Vendor	Models
Beckhoff	EK1100, EK1814, EL1889, EL1904, EL2008, EL2889, EL2904, EL3008, EL4034
Delta	ASDA-A2, ASDA-B3, R1-EC5500D0E, R1-EC5621D0
Panasonic	MADLT05BF
Advantech	AMAX-4817
Oriental Motor	AZD-KRED, AZD4A-KED
FASTECH	Ezi-SERVO II, Ezi-STEP II, Ezi-IO
MOONS'	M56S
SANYO DENKI	GADSA01AH24
NPM	PULSERVO II, EC-AD1442A1, EC-AD1441A4
SENSODRIVE	SensoJoint 3000
Bore	CM-TE66NA3G-N, CM-TE66PA3G-P
Microchip	EVB-LAN9254

Other EtherCAT SubDevices are displayed with a “**?**” icon and labeled “**Unsupported**”. For SubDevices from the following vendors, 86EVA also displays the vendor’s name based on the Vendor ID: Beckhoff, ICP DAS, OMRON, Mitsubishi, Delta, Mirle, acontis, MOONS', HIWIN, ESTUN, Panasonic, Advantech, Nanotec, Shihlin, Oriental Motor, SANYO, Bosch Rexroth, Festo, Kollmorgen, Phoenix Contact, SEW-EURODRIVE, and others.

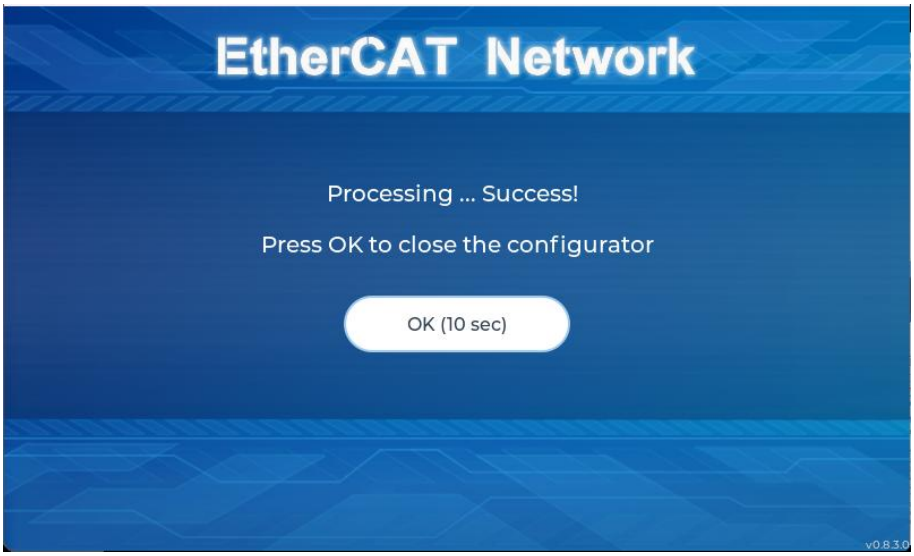
For an Unsupported SubDevice, you can still set the Object Name and Alias Address in 86EVA. Its process data can be accessed through the EtherCAT Library API, and its startup parameters can be configured with an ENI file (see Ch. 2.5.1.6 Select ENI File).

2.7 Code Generation

Once all device parameters are configured in 86EVA, users can return to the main EtherCAT Network screen and press the **“Code Generation”** button located at the bottom-right corner.

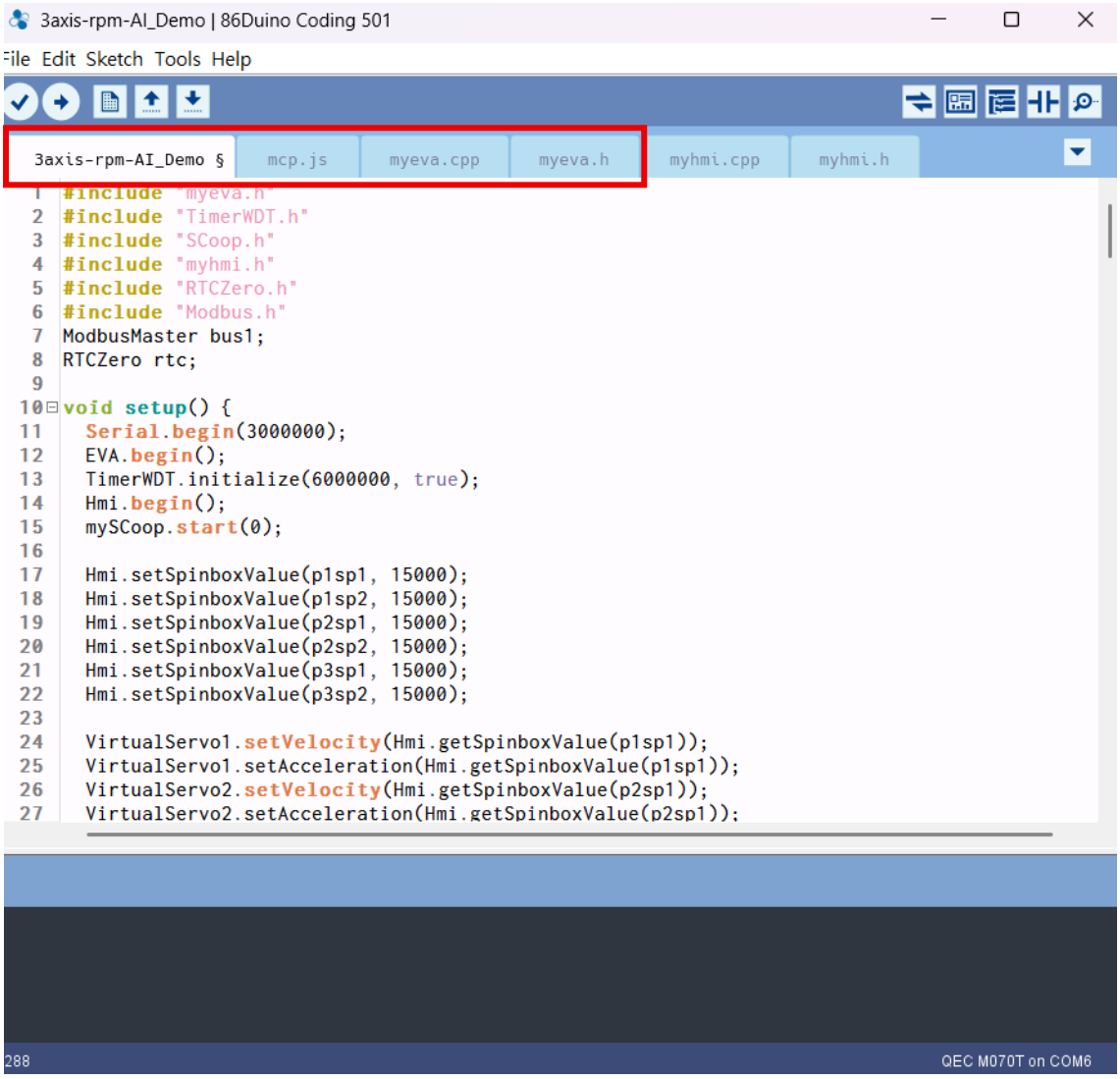


When code generation is successful, a confirmation window will appear. You may click the OK button to close 86EVA immediately or wait 10 seconds for it to close automatically.



The generated code and files are as follows:

- <Project name>.ino: Main sketch (e.g., 3axis-rpm-AI_Demo.ino). If the sketch is empty, 86EVA creates a basic setup() / loop() template.
- mcp.js: Model Context Protocol (MCP) server script that provides the 86EVA configuration and API information to an AI assistant for code generation.
- myeva.cpp: C++ program code of 86EVA
- myeva.h: Header file of 86EVA
- **data/Resource/MyEccfg/**: 86EVA configuration files (cfg.eva, AutoUpload.ini, and eni.xml if an ENI file is selected). These files are uploaded to the QEC MDevice together with the sketch.



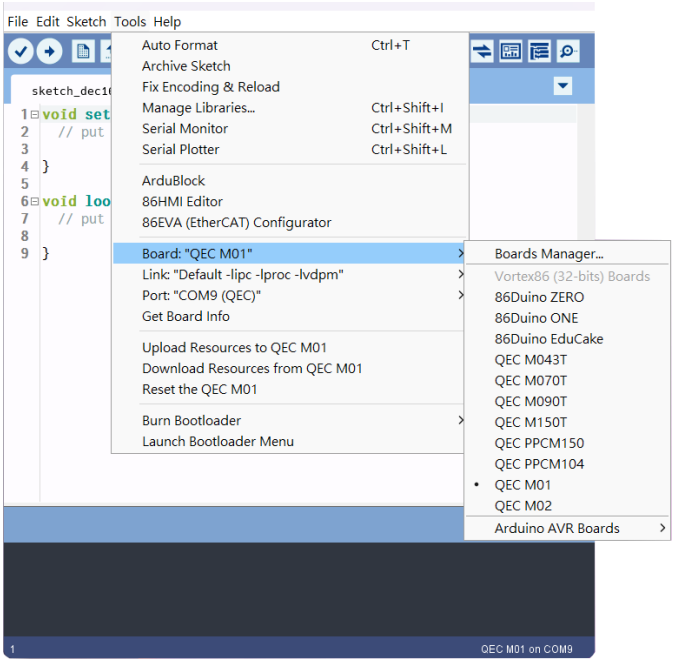
***Note:** After code generation, 86EVA will automatically include the following code in the .ino main file. Removing or modifying these lines may cause the program to fail to compile or run correctly:

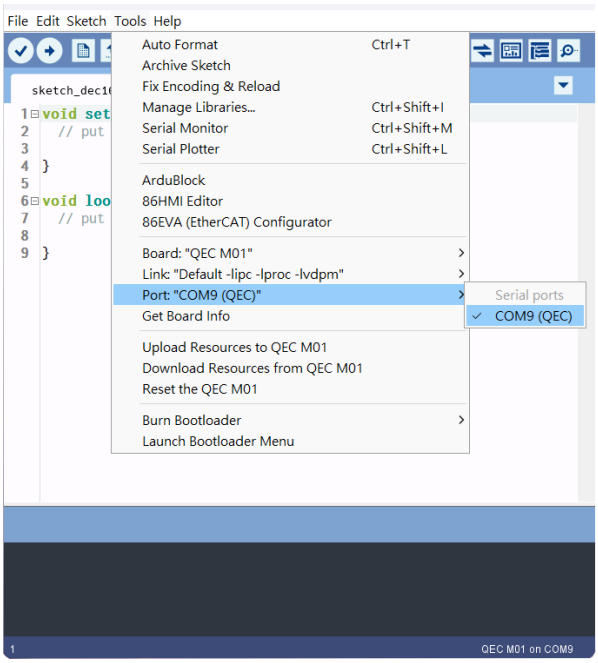
1. `#include "myeva.h"` : Include EVA Header file
2. `EVA.begin();` in `setup()` : Initialize the EVA function

2.8 Uploading to MDevice

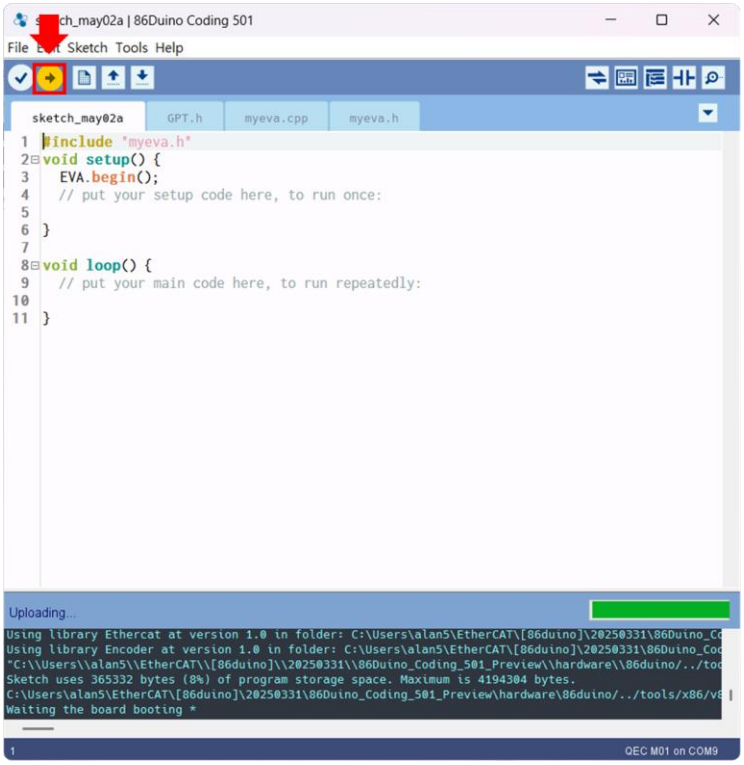
Please note that the generated configuration file and user program will be burned into the QEC MDevice using the Upload button. It is essential to press Upload to complete the configuration process—otherwise, the configuration created in 86EVA will be considered invalid.

Before uploading, make sure the "**Board**" and "**Port**" settings in the IDE are correctly selected for your QEC MDevice model.

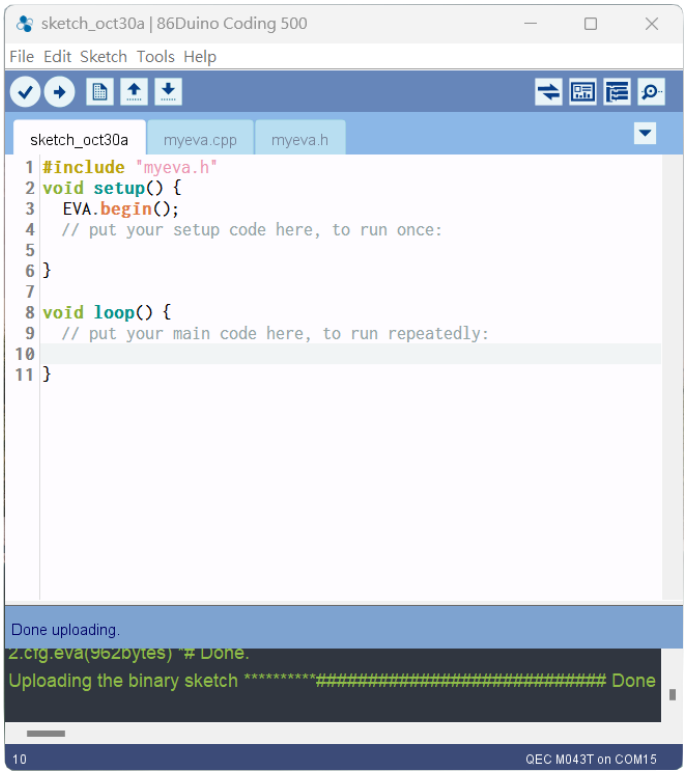




Click on the "**Upload**" button to begin the upload process.



Once the upload completes, the IDE will display a **Done uploading** message.



After uploading, you can reopen **86EVA** to scan the EtherCAT SubDevices and confirm whether they are running according to the configuration.

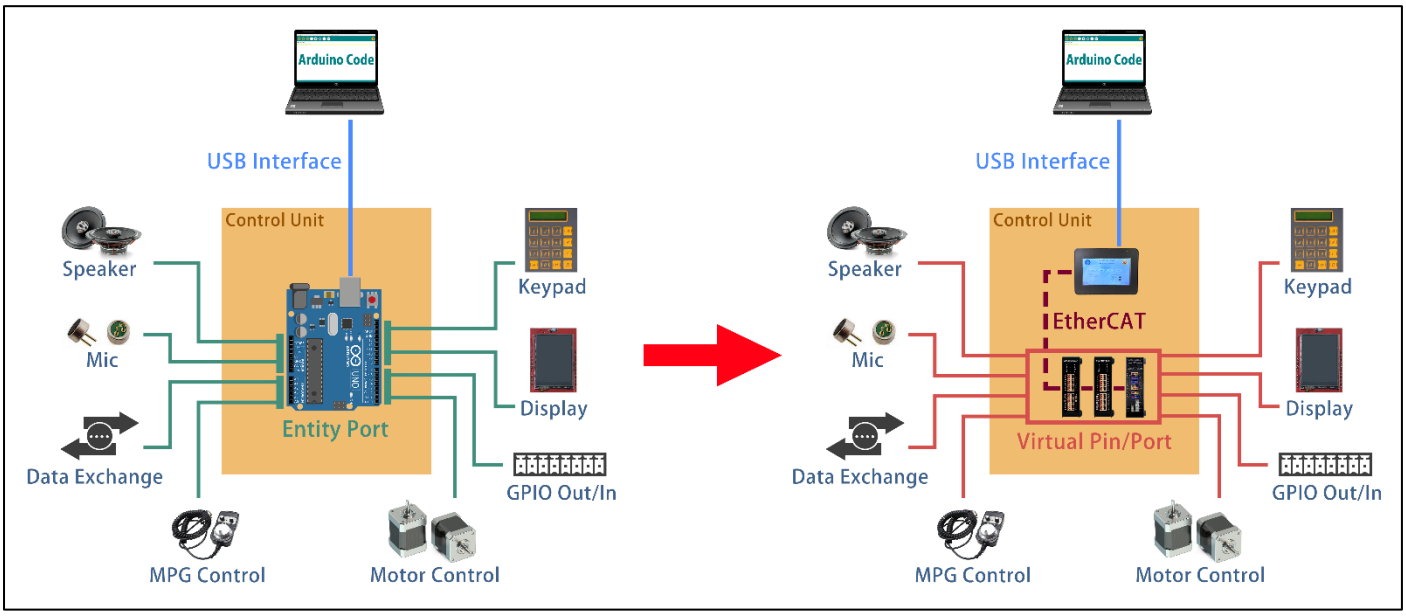
Ch. 3

API Overview

3.1 Description

86EVA provides a comprehensive set of APIs that allow real-time control and data access to EtherCAT SubDevices via the 86Duino IDE.

Built on the Virtual Arduino concept, these APIs enable developers to build industrial control systems with the simplicity of Arduino-style programming.



The available APIs are categorized into the following groups:

- Basic I/O control and data reading (e.g., `digitalRead()`, `analogRead()`, `voltageWrite()`)
- Virtual device operations (e.g., virtual serial ports, barcode readers, force sensors, MPG, LCD, EM sensors)
- System monitoring (e.g., device temperature, voltage, current, operating hours)

All functions follow C++ syntax and can be used directly within the 86Duino IDE.
With these APIs, developers can quickly integrate EtherCAT devices, manage I/O operations, and extend system functionality with graphical tools like ArduBlock.

3.2 EVA Class

The EVA class is the core interface for communicating with EtherCAT SubDevices through the Virtual Arduino environment in 86EVA. All APIs in the EVA class are designed to operate on virtual pin mappings.

Unless otherwise noted, all functions return an int value:

- Return < 0: Indicates an EtherCAT communication error. Refer to the A.1 Error list for details.
- Return ≥ 0: Indicates successful operation.

The following sections describe each EVA API in detail.

Functions:

Function Name	Description
`begin()`	Initializes the EtherCAT system.
`end()`	Terminates EtherCAT operation
`digitalDataRead()`	Reads digital input (0/1) from remote virtual pin
`digitalDataWrite()`	Writes digital output (0/1) to remote virtual pin
`integerDataRead()`	Reads 32-bit integer from remote virtual pin
`integerDataWrite()`	Writes 32-bit integer to remote virtual pin
`decimalDataRead()`	Reads a floating-point value (e.g., voltage) from remote virtual pin
`decimalDataWrite()`	Writes a floating-point value to remote virtual pin
`integerDataPinMode()`	Sets I/O mode for integer-type pins (e.g., analog/counter)
`digitalWrite()`	Equivalent to digitalDataWrite
`digitalRead()`	Equivalent to digitalDataRead
`analogPinMode()`	Sets analog pin input/output mode (range defined by hardware)
`analogWrite()`	Writes analog value (mapped to ±64V range)
`analogRead()`	Reads analog value (mapped from ±64V range)
`voltageWrite()`	Directly writes voltage to pin (in Volts)
`voltageRead()`	Reads voltage from analog input pin
`tone()`	Plays tone at specified frequency and duration on buzzer pin
`getUsVoltage()`	Reads 24V logic supply voltage from a device (DMP-only)
`getUpVoltage()`	Reads power supply voltage from a device (DMP-only)
`getIsCurrent()`	Reads logic supply current from a device (DMP-only)
`getIpCurrent()`	Reads power supply current from a device (DMP-only)
`getTemperature()`	Reads temperature of device (DMP-only)
`getWorkingHours()`	Reads total working hours from a device (DMP-only)
`getBootTimes()`	Reads number of power cycles of a device (DMP-only)

begin()

Description

Initializes the EtherCAT system configured by 86EVA.

Syntax

```
int begin(void (*userCallback)(void) = NULL, bool skipStart = false);
```

Parameters

- *[in] void (*userCallback)(void) = NULL*
  The cyclic callback function to be registered.
- *[in] bool skipStart = false*
  Set to true to initialize the EtherCAT network without starting the cyclic operation. Default is false.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

Example 1: No callback function.

```
#include "myeva.h"
void setup() {
  Serial.begin(3000000);

  if (!EVA.begin())
    Serial.println("EVA init OK");
}

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

Example 2: With a callback function.

```
#include "myeva.h"

void MyEthercatCallback() {
  // Callback function
}

void setup() {
  EVA.begin(MyEthercatCallback);
}

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

end()

Description

Terminates EtherCAT operation.

Syntax

`void end();`

Parameters

None

Return Value

None.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  EVA.end();
  while (true);
}
```

digitalDataRead()

Description

Reads digital input (0/1) from Virtual Digital Pin.

Syntax

```
int digitalDataRead(int pin);
```

Parameters

- *[in] int pin*
  Virtual Digital Pin number.

Return Value

Return the digital logic level of the specified digital pin, the value is HIGH or LOW.

Example

```
#include "myeva.h"
void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  int pin = 2;
  int state = EVA.digitalDataRead(pin);
  Serial.print("Digital pin state: ");
  Serial.println(state);
  delay(1000);
}
```

digitalDataWrite()

Description

Writes digital output (0/1) to Virtual Digital Pin.

Syntax

`int digitalDataWrite(int pin, uint8_t data);`

Parameters

- *`[in] int pin`*
  Virtual Digital Pin number.
- *`[in] uint8_t data`*
  Digital logic level, the value is HIGH or LOW.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  int pin = 2;
  EVA.digitalDataWrite(pin, HIGH);
  delay(1000);
  EVA.digitalDataWrite(pin, LOW);
  delay(1000);
}
```

integerDataRead()

Description

Reads 32-bit integer from the remote Virtual integer pin.

Syntax

`int integerDataRead(int pin);`

Parameters

- *`[in] int pin`*
  Virtual integer pin number.

Return Value

Returns the 32-bit integer value of the specified Virtual Integer Data Pin.

Example

```
#include "myeva.h"
void setup() {
    Serial.begin(3000000);
    EVA.begin();
}

void loop() {
    int pin = 3;
    int value = EVA.integerDataRead(pin);
    Serial.print("Integer pin value: ");
    Serial.println(value);
    delay(1000);
}
```

integerDataWrite()

Description

Writes 32-bit integer to the remote Virtual integer pin.

Syntax

`int integerDataWrite(int pin, int32_t data);`

Parameters

- *`[in] int pin`*
  Virtual Integer Data Pin number.
- *`[in] int32_t data`*
  The analog output value to be written. This parameter is a 32-bit signed integer whose value is linearly mapped to a physical output by the following rule.
  The output range of the virtual analog pin is fixed at 64V ~ -64V. Maps a value of 0 to voltage 0V, 2^{31} to 64V, and -2^{31} to -64V (i.e., 225 would get mapped to 1V).

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  int pin = 3;
  EVA.integerDataWrite(pin, 42);
  delay(1000);
}
```

decimalDataRead()

Description

Reads a floating-point value from the remote Virtual Decimal Data Pin, e.g., a voltage measured through Detect Voltage, VP, or VS mapping.

Syntax

```
double decimalDataRead(int pin);
```

Parameters

- *`[in] int pin`*
  Virtual Decimal Data Pin number (F0 = 0, F1 = 1, ...).

Return Value

Returns the value of the specified Virtual Decimal Data Pin (e.g., voltage in Volts).

Example

```
#include "myeva.h"
void setup() {
    Serial.begin(3000000);
    EVA.begin();
}

void loop() {
    int pin = 0;
    double value = EVA.decimalDataRead(pin);
    Serial.print("Decimal pin value: ");
    Serial.println(value);
    delay(1000);
}
```

decimalDataWrite()

Description

Writes a floating-point value to the remote Virtual Decimal Data Pin.

Syntax

`int decimalDataWrite(int pin, double data);`

Parameters

- *`[in] int pin`*
  Virtual Decimal Data Pin number.
- *`[in] double data`*
  The value to be written to the Virtual Decimal Data Pin.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  int pin = 0;
  EVA.decimalDataWrite(pin, 1.5);
  delay(1000);
}
```

integerDataPinMode()

Description

Set the configuration mode of the remote virtual integer pin, such as AIO.
It will automatically determine whether the pin is input or output, and then correctly set the specified mode to the pin.

Range Mode Mapping:

Mode Value	Description	Applicable To
0	±22V (default)	AI
1	±20V	AI
2	±11V	AI
3	±10V	AI
4	±5V	AI
5	0 ~ 22V	AI
6	0 ~ 20V	AI
7	0 ~ 11V	AI
8	0 ~ 10V	AI
0	0 ~ +5V (default)	AO
1	0 ~ +10V	AO
2	±5V	AO
3	±10V	AO
4	3.5 mA ~ 23.5 mA	AO
5	0 ~ 20 mA	AO
6	0 ~ 24 mA	AO
7	±24 mA	AO
8	0 ~ +6V	AO
9	0 ~ +12V	AO
10	±6V	AO
11	±12V	AO
12	4 mA ~ 20 mA	AO

Syntax

```
int integerDataPinMode(int pin, uint32_t range);
```

Parameters

- *`[in] int pin`*
  Virtual integer pin number.
- *`[in] uint32_t range`*
  The Analog Range mode.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
  int pin = 5;
  EVA.integerDataPinMode(pin, 1);
}

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

digitalWrite()

Description

This function behaves the same as `digitalDataWrite()`.

Please refer to that section for the detailed description, syntax, parameters, and return values.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  int pin = 2;
  EVA.digitalWrite(pin, HIGH);
  delay(1000);
  EVA.digitalWrite(pin, LOW);
  delay(1000);
}
```

digitalRead()

Description

This function behaves the same as `digitalDataRead()`.

Please refer to that section for the detailed description, syntax, parameters, and return values.

Example

```
#include "myeva.h"
void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  int pin = 2;
  int state = EVA.digitalRead(pin);
  Serial.print("Digital pin state: ");
  Serial.println(state);
  delay(1000);
}
```

analogPinMode()

Description

This function behaves the same as `integerDataPinMode()`.

Please refer to that section for the detailed description, syntax, parameters, and return values.

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
  int pin = 0;  // Virtual Integer Data Pin A0
  EVA.analogPinMode(pin, 1);
}

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

analogWrite()

Description

This function behaves the same as `integerDataWrite()`.
Please refer to that section for the detailed description, syntax, parameters, and return values.

The output range of the virtual analog pin is fixed at 64V ~ -64V. Maps a value of 0 to voltage 0V, 2^{31} to 64V, and -2^{31} to -64V (i.e., 225 would get mapped to 1V).

Example

```
#include "myeva.h"
void setup() {
  EVA.begin();
}

void loop() {
  int pin = 0;  // Virtual Integer Data Pin A0
  EVA.analogWrite(pin, 0);
  delay(1000);
  EVA.analogWrite(pin, 1 << 25); // Set analog output is 1V (2^25);
  delay(1000);
}
```

analogRead()

Description

This function behaves the same as `integerDataRead()`.
Please refer to that section for the detailed description, syntax, parameters, and return values.

The input range of the virtual analog pin is fixed at 64V ~ -64V. Maps a value of 0 to voltage 0V, 2^{31} to 64V, and -2^{31} to -64V (i.e., 225 would get mapped to 1V).

Example

```
#include "myeva.h"
void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  int pin = 0;  // Virtual Integer Data Pin A0
  unsigned long value = EVA.analogRead(pin);
  Serial.print("Analog pin value: ");
  Serial.println(value >> 25); // 1V (2^25);
  Serial.print("V");
  delay(1000);
}
```

voltageWrite()

Description

Sets the output voltage of a remote virtual analog pin.

Unlike `analogWrite()`, which requires converting the value to an internal representation (based on ±64V scaling), this function allows users to directly input the desired voltage value for more intuitive control.

Syntax

```
int voltageWrite(int pin, double value);
```

Parameters

- *`[in] int pin`*
  Virtual Integer Data Pin number.
- *`[in] double value`*
  The voltage to output (in Volts).

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();
}

void loop() {
  int pin = 0;  // Virtual Integer Data Pin A0
  EVA.voltageWrite(pin, 5.0);       // Output 5.0V to the pin
  delay(1000);
}
```

voltageRead()

Description

Reads the voltage value from a remote virtual analog input pin.

This function directly returns the voltage in volts, eliminating the need to manually convert the raw analog value.

Syntax

```
double voltageRead(int pin);
```

Parameters

- *`[in] int pin`*
  Virtual Integer Data Pin number.

Return Value

Returns the voltage value read from the specified analog input pin (in Volts).

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  int pin = 0;  // Virtual Integer Data Pin A0
  double voltage = EVA.voltageRead(pin); // Read voltage value
  Serial.print("Voltage: ");
  Serial.println(voltage);
  delay(1000);
}
```

tone()

Description

Plays a tone on the specified remote virtual buzzer pin at a defined frequency and duration. This API is typically used with DMP HID devices such as the QEC-RXXHU9S series.

Syntax

```
int tone(int pin, uint32_t freq, uint32_t duration);
```

Parameters

- *[in] int pin*
  Virtual buzzer pin number.
- *[in] uint32_t freq*
  Tone frequency in Hertz (Hz).
- *[in] uint32_t duration*
  Duration to play the tone, in milliseconds (ms).

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();
}

void loop() {
  int pin = 0;  // Virtual Buzzer Pin B0
  EVA.tone(pin, 1000, 500); // Play 1000 Hz tone for 500 ms
  delay(1000);
}
```

getUsVoltage()

Description

Reads the *Us* voltage of a remote DMP EtherCAT device. This API does **not** support third-party EtherCAT SubDevice.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
double getUsVoltage(short alias);
```

Parameters

- *`[in] short alias`*
  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the Us voltage (in Volts) of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = -1; // Read from the master device
  double voltage = EVA.getUsVoltage(alias);
  Serial.print("Us Voltage: ");
  Serial.println(voltage);
  delay(1000);
}
```

getUpVoltage()

Description

Reads the *Up* voltage of a remote DMP EtherCAT device. This API does **not** support third-party EtherCAT SubDevice.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
double getUpVoltage(short alias);
```

Parameters

- *[in] short alias*

  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the Up voltage (in Volts) of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = -1; // Read from the master device
  double voltage = EVA.getUpVoltage(alias);
  Serial.print("Up Voltage: ");
  Serial.println(voltage);
  delay(1000);
}
```

getIsCurrent()

Description

Reads the *Is* current value of a remote DMP EtherCAT device. This API does **not** support third-party EtherCAT SubDevice.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
double getIsCurrent(short alias);
```

Parameters

- *`[in] short alias`*
  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the Is current value (in Amps) of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = 2000; // Targeting device with alias address 2000
  double current = EVA.getIsCurrent(alias);
  Serial.print("Is Current: ");
  Serial.println(current);
  delay(1000);
}
```

getIpCurrent()

Description

Reads the *Ip* current value of a remote DMP EtherCAT device. This API does **not** support third-party EtherCAT SubDevice.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
double getIpCurrent(short alias);
```

Parameters

- *`[in] short alias`*
  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the Ip current value (in Amps) of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = 2000; // Targeting device with alias address 2000
  double current = EVA.getIpCurrent(alias);
  Serial.print("Ip Current: ");
  Serial.println(current);
  delay(1000);
}
```

getTemperature()

Description

Reads the internal temperature of a remote DMP EtherCAT device.

This API does not support third-party EtherCAT SubDevices.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
double getTemperature(short alias);
```

Parameters

- *[in] short alias*

  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the internal temperature (in degrees Celsius) of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = 3000; // Targeting the device with alias address 3000
  double temperature = EVA.getTemperature(alias);
  Serial.print("Temperature: ");
  Serial.println(temperature);
  delay(1000);
}
```

getWorkingHours()

Description

Retrieves the total working hours of a remote DMP EtherCAT device.

This API does not support third-party EtherCAT SubDevices.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
int getWorkingHours(short alias);
```

Parameters

- *[in] short alias*

  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the accumulated working hours of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = -1; // Refers to QEC MDevice
  int hours = EVA.getWorkingHours(alias);
  Serial.print("Working Hours: ");
  Serial.println(hours);
  delay(1000);
}
```

getBootTimes()

Description

Retrieves the total number of times a remote DMP EtherCAT device has booted.

This API does not support third-party EtherCAT SubDevices.

The target device is identified by its alias address, which must be pre-configured in 86EVA.

Use -1 to refer to the QEC MDevice.

Syntax

```
int getBootTimes(short alias);
```

Parameters

- *[in] short alias*

  The alias address of the target EtherCAT device. Use -1 for the QEC MDevice.

Return Value

Returns the total boot count of the specified device.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();
}

void loop() {
  short alias = -1; // Refers to QEC MDevice
  int bootTimes = EVA.getBootTimes(alias);
  Serial.print("Boot Times: ");
  Serial.println(bootTimes);
  delay(1000);
}
```

3.3 MyVirtualSerial Class

This class is used to control a remote Virtual Serial Port device over EtherCAT. It allows developers to send and receive serial data through a virtual channel mapped to a remote EtherCAT device. It is typically applied in DMP HID modules (e.g., QEC-R00HU9S series). You can use it to initialize the port, send data, and receive bytes from the remote target.

Functions:

Function Name	Description
`begin()`	Initializes the remote Virtual Serial port with the specified baudrate (e.g., 9600, 115200). Default mode is 8N1.
`write()`	Sends one byte to the remote Virtual Serial port.
`print()`	Sends a string (null-terminated or formatted) to the remote Virtual Serial port.
`available()`	Returns the number of bytes available for reading from the remote port.
`read()`	Reads one byte of incoming data from the remote Virtual Serial port.

begin()

Description

Initializes the remote Virtual Serial port on the specified EtherCAT SubDevice. This must be called before any read or write operations. The serial communication defaults to 8 data bits, no parity, and 1 stop bit (8N1).

Syntax

```
int begin(int baudrate, int config = SERIAL_8N1);
```

Parameters

- *[in] int baudrate*
  The desired baud rate for the virtual serial communication (e.g., 9600, 38400, 115200).
- *[in] int config = SERIAL_8N1*
  Data bits, parity, and stop bits of the serial port. Default is SERIAL_8N1.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();                  // Initialize EVA
  VirtualSerial1.begin(115200);   // Initialize virtual serial port at 115200
baud
}

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

write()

Description

Writes a single 1-byte value (character or raw data) to the remote Virtual Serial port.

Syntax

```
size_t write(uint8_t value);
```

Parameters

- *[in] uint8_t value*
  The byte to send to the Virtual Serial port.

Return Value

Returns the number of bytes written._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();                  // Initialize EVA
  VirtualSerial1.begin(115200);   // Initialize virtual serial port at 115200
baud
}

void loop() {
  // Write a character 'A' to the virtual serial port
  VirtualSerial1.write('A');

  delay(1000); // Send once per second
}
```

print()

Description

Write a string to the remote Virtual Serial port.

Syntax

`size_t print(const char* str);`

Parameters

- *`[in] const char* str`*
  A null-terminated string to be sent through the virtual serial port.

Return Value

Returns the number of bytes written._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();                  // Initialize EVA
  VirtualSerial1.begin(115200);   // Initialize virtual serial port at 115200
baud
}

void loop() {
  // Write a string to the virtual serial port
  VirtualSerial1.print("Hello from Virtual Serial!");

  delay(1000); // Send once per second
}
```

available()

Description

Checks the number of bytes (characters) available for reading from the remote Virtual Serial port buffer. This is useful to avoid blocking operations and to confirm if incoming data is ready.

Syntax

`int available();`

Parameters

None.

Return Value

Returns the number of bytes available to read from the Virtual Serial port.

If 0, no data is available.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);         // Initialize the standard serial port
  EVA.begin();                  // Initialize EVA
  VirtualSerial1.begin(115200);   // Initialize virtual serial port at 115200
baud
}

void loop() {
  // Write a single character
  VirtualSerial1.write('A');

  // Write a string
  VirtualSerial1.print("Hello from Virtual Serial!");

  // Check for incoming data
  int bytesAvailable = VirtualSerial1.available();
  if (bytesAvailable > 0) {
    int incomingByte = VirtualSerial1.read();
    Serial.print("Received: ");
    Serial.println((char)incomingByte); // Print to the standard serial port
  }

  delay(1000); // Repeat every second
}
```

read()

Description

Reads one byte of data from the remote Virtual Serial port. It is recommended to call `available()` first to check if data is present before calling `read()`, to avoid undefined results.

Syntax

`int read();`

Parameters

None.

Return Value

Returns the first byte of incoming data (as an int) if available.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);         // Initialize the standard serial port
  EVA.begin();                   // Initialize EVA
  VirtualSerial1.begin(115200);   // Initialize virtual serial port at 115200
baud
}

void loop() {
  // Write a single character
  VirtualSerial1.write('A');

  // Write a string
  VirtualSerial1.print("Hello from Virtual Serial!");

  // Check for incoming data
  int bytesAvailable = VirtualSerial1.available();
  if (bytesAvailable > 0) {
    int incomingByte = VirtualSerial1.read();
    Serial.print("Received: ");
    Serial.println((char)incomingByte); // Print to the standard serial port
  }

  delay(1000); // Repeat every second
}
```

Other member functions

The following member functions are also available in the IDE 501 library:

Function	Syntax	Description
`write()`	size_t write(const uint8_t *buf, size_t size);	Writes size bytes from buf to the Virtual Serial port. Returns the number of bytes written.
`peek()`	int peek(void);	Returns the next received byte without removing it from the receive buffer, or -1 if no data is available.
`flush()`	void flush(int w = 0);	Clears the transmit buffer (w = 0) or the receive buffer (w ≠ 0).

3.3.1 BarcodeScanner Class

The BarcodeScanner class provides a high-level interface for reading barcode data through a remote Virtual Serial port. It is typically used on a DMP HID module where a barcode scanner (e.g., Honeywell N4680 or HF521) is connected via RS232/RS485. Communication is handled over EtherCAT using the VirtualSerial object.

Functions:

Function Name	Description
`begin()`	Initializes the remote barcode scanner using the specified model and VirtualSerial.
`available()`	Checks if barcode data is available to read.
`readLine()`	Reads a complete line of barcode data from the scanner.

begin()

Description

Initializes a remote barcode scanner connected to a DMP HID module via RS232/RS485.
This function should be called in the `setup()` phase. The function links a specific barcode device model (e.g., HONEYWELL_N4680 or HONEYWELL_HF521) with a Virtual Serial port (e.g., VirtualSerial1).

Syntax

```
bool begin(BarcodeDevice dev, SerialInterface &serialport);
```

Parameters

- *`[in] BarcodeDevice dev`*
  The model of the barcode scanner. Supported values: HONEYWELL_N4680, HONEYWELL_HF521.
- *`[in] SerialInterface &serialport`*
  The Virtual Serial port interface used for communication. Typically, one of VirtualSerial1, VirtualSerial2, etc.

Return Value

Returns true if the initialization is successful.
Returns false if the specified barcode model is not supported or communication fails.

Example

```
#include "myeva.h"

BarcodeScanner Barcode1;

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

  VirtualSerial1.begin(115200);
  bool ok = Barcode1.begin(HONEYWELL_N4680, VirtualSerial1);

  if (!ok) {
    Serial.println("Failed to initialize barcode scanner.");
  }
}

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

available()

Description

Checks whether the remote barcode scanner has received a complete line of data.
This function is typically used inside the `loop()` function to determine if scanned barcode data is ready to be read.

Syntax

`int available();`

Parameters

None.

Return Value

Returns a positive integer if a complete line of barcode data is available.
Returns 0 if no data is available.

Example

```
#include "myeva.h"

BarcodeScanner Barcode1; // Declare BarcodeScanner object

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

  VirtualSerial1.begin(115200); // Initialize Virtual Serial port
  Barcode1.begin(HONEYWELL_N4680 , VirtualSerial1); // Use Honeywell N4680
scanner
}

void loop() {
  if (Barcode1.available()) {
    Serial.print(Barcode1.readLine()); // Print scanned barcode data
    Serial.println();
  }
}
```

readLine()

Description

Reads a full line of data from the remote barcode scanner device.

Syntax

`char* readLine();`

Parameters

None.

Return Value

Returns a pointer to a null-terminated C-style string containing the barcode data.

Returns 0 if no data is available.

Example

```
#include "myeva.h"

BarcodeScanner Barcode1; // Declare BarcodeScanner object

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

  VirtualSerial1.begin(115200); // Initialize Virtual Serial port
  Barcode1.begin(HONEYWELL_N4680 , VirtualSerial1); // Use Honeywell N4680
scanner
}

void loop() {
  if (Barcode1.available()) {
    Serial.print(Barcode1.readLine()); // Print scanned barcode data
    Serial.println();
  }
}
```

3.3.2 ForceSensor Class

The ForceSensor class is used to interface with remote force sensor devices connected to one of the RS232/RS485 ports on a DMP HID module.
Communication between the QEC MDevice and the force sensor is established through EtherCAT and a VirtualSerial port.

This class currently supports the DMP DM320 model and provides functions to read force values along multiple axes (Mx, My, Fz) as well as raw sensor data (F0, F1, F2).

Functions:

Function Name	Description
`begin()`	Initialize the remote force sensor device.
`getMx()`	Read the Mx axis force value.
`getMy()`	Read the My axis force value.
`getFz()`	Read the Fz axis force value.
`getF0Raw()`	Read the raw data from channel F0.
`getF1Raw()`	Read the raw data from channel F1.
`getF2Raw()`	Read the raw data from channel F2.

begin()

Description

Initializes the remote force sensor device connected through one of the RS232/RS485 ports on a DMP HID module. This function must be called in the `setup()` function before using any of the reading APIs.

Currently, only the DMP DM320 model is supported. The connection is established via a user-specified VirtualSerial object.

Syntax

```
bool begin(ForceSensorDevice dev, SerialInterface &serialport);
```

Parameters

- *`[in] ForceSensorDevice dev`*
  The sensor model. Currently supports DM320.
- *`[in] SerialInterface &serialport`*
  The Virtual Serial port interface used for communication. Typically, one of VirtualSerial1, VirtualSerial2, etc.

Return Value

Returns true if initialization is successful, or false if it fails.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getMx()

Description

Reads the force value along the Mx axis from the remote force sensor device.

Syntax

```
double getMx(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the force value on the Mx axis as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getMy()

Description

Reads the force value along the My axis from the remote force sensor device.

Syntax

```
double getMy(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the force value on the My axis as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getFz()

Description

Reads the force value along the Fz axis from the remote force sensor device.

Syntax

```
double getFz(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the force value on the Fz axis as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getF0Raw()

Description

Reads the raw force value along the F0 from the remote force sensor device.

Syntax

```
double getF0Raw(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the raw force value on the F0 as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getF1Raw()

Description

Reads the raw force value along the F1 from the remote force sensor device.

Syntax

```
double getF1Raw(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the raw force value on the F1 as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

getF2Raw()

Description

Reads the raw force value along the F2 from the remote force sensor device.

Syntax

```
double getF2Raw(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns the raw force value on the F2 as a double.

Example

```
#include "myeva.h"

_ForceSensor ForceSensor1;

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

  VirtualSerial1.begin(115200);
  ForceSensor1.begin(DM320, VirtualSerial1);
}

void loop() {
  Serial.println(ForceSensor1.getMx());
  Serial.println(ForceSensor1.getMy());
  Serial.println(ForceSensor1.getFz());
  Serial.println(ForceSensor1.getF0Raw());
  Serial.println(ForceSensor1.getF1Raw());
  Serial.println(ForceSensor1.getF2Raw());

  delay(1000);
}
```

3.3.3 3DJoystick Class

The _3DJoystick class is designed for interfacing with a 3-axis industrial joystick (3D joystick) connected via one of the RS232/RS485 ports on a DMP HID module. Communication is handled through EtherCAT using a virtual serial port defined by VirtualSerial. This class supports reading X, Y, Z axis positions as well as the push-button state of the joystick.

Currently, only the SMC30D1 joystick model is supported. This device is ideal for motion control, HMI interfaces, and robotic navigation where multidimensional input is required.

Functions:

Function Name	Description
`begin()`	Initialize the 3D joystick device.
`getX()`	Read the current position on the X-axis.
`getY()`	Read the current position on the Y-axis.
`getZ()`	Read the current position on the Z-axis.
`btnPressed()`	Check if the joystick button is currently pressed.

begin()

Description

Initializes the remote 3D joystick device connected to a DMP HID RS232/RS485 port via EtherCAT. The first parameter specifies the device model (currently only SMC30D1 is supported), and the second parameter is the user-defined VirtualSerial.

Syntax

`bool begin(JoystickDevice dev, SerialInterface &serialport);`

Parameters

- *`[in] JoystickDevice dev`*
  JoystickDevice enum specifying the model (e.g., SMC30D1).
- *`[in] SerialInterface &serialport`*
  The Virtual Serial port interface used for communication. Typically, one of VirtualSerial1, VirtualSerial2, etc.

Return Value

Returns true if initialization is successful, or false if it fails.

Example

```
#include "myeva.h"

_3DJoystick Joystick1;

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

  VirtualSerial1.begin(115200);
  Joystick1.begin(SMC30D1, VirtualSerial1); // Initialize 3D Joystick on
VirtualSerial1
}

void loop() {
  if (Joystick1.btnPressed()) {
    int x = Joystick1.getX();
    int y = Joystick1.getY();
    int z = Joystick1.getZ();

    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Z: "); Serial.println(z);
  }
}
```

getX()

Description

Reads the X axis value from the connected 3D joystick.

Syntax

```
int getX(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns an int representing the current X axis value of the joystick.

Example

```
#include "myeva.h"

_3DJoystick Joystick1;

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

  VirtualSerial1.begin(115200);
  Joystick1.begin(SMC30D1, VirtualSerial1); // Initialize 3D Joystick on
VirtualSerial1
}

void loop() {
  if (Joystick1.btnPressed()) {
    int x = Joystick1.getX();
    int y = Joystick1.getY();
    int z = Joystick1.getZ();

    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Z: "); Serial.println(z);
  }
}
```

getY()

Description

Reads the Y axis value from the connected 3D joystick.

Syntax

`int getY(bool block = false);`

Parameters

- *`[in] bool block`*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns an int representing the current Y axis value of the joystick.

Example

```
#include "myeva.h"

_3DJoystick Joystick1;

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

  VirtualSerial1.begin(115200);
  Joystick1.begin(SMC30D1, VirtualSerial1); // Initialize 3D Joystick on
VirtualSerial1
}

void loop() {
  if (Joystick1.btnPressed()) {
    int x = Joystick1.getX();
    int y = Joystick1.getY();
    int z = Joystick1.getZ();

    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Z: "); Serial.println(z);
  }
}
```

getZ()

Description

Reads the Z axis value from the connected 3D joystick.

Syntax

```
int getZ(bool block = false);
```

Parameters

- *[in] bool block*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns an int representing the current Z axis value of the joystick.

Example

```
#include "myeva.h"

_3DJoystick Joystick1;

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

  VirtualSerial1.begin(115200);
  Joystick1.begin(SMC30D1, VirtualSerial1); // Initialize 3D Joystick on
VirtualSerial1
}

void loop() {
  if (Joystick1.btnPressed()) {
    int x = Joystick1.getX();
    int y = Joystick1.getY();
    int z = Joystick1.getZ();

    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Z: "); Serial.println(z);
  }
}
```

btnPressed()

Description

Checks whether the button on the 3D joystick is currently pressed.

Syntax

`bool btnPressed(bool block = false);`

Parameters

- *`[in] bool block`*
  Default is false. If true, blocks execution until new data is available. If false, returns the latest value.

Return Value

Returns true if the button is pressed, otherwise returns false.

Example

```
#include "myeva.h"

_3DJoystick Joystick1;

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

  VirtualSerial1.begin(115200);
  Joystick1.begin(SMC30D1, VirtualSerial1); // Initialize 3D Joystick on
VirtualSerial1
}

void loop() {
  if (Joystick1.btnPressed()) {
    int x = Joystick1.getX();
    int y = Joystick1.getY();
    int z = Joystick1.getZ();

    Serial.print("X: "); Serial.print(x);
    Serial.print(" Y: "); Serial.print(y);
    Serial.print(" Z: "); Serial.println(z);
  }
}
```

3.3.4 ECRFID Class

The ECRFID class is designed for communicating with remote RFID readers connected to a DMP HID device. The host communicates with the RFID reader through an RS232/RS485 port via EtherCAT.

This class supports commonly used RFID modules such as RC522 and PN532, enabling initialization, tag scanning, UID reading, and basic peripheral control (beeper and LED).

This class requires a pre-configured VirtualSerial interface and is ideal for applications such as access control, automation, and inventory systems where RFID tags are involved.

Functions:

Function Name	Description
`begin()`	Initializes the RFID device via the specified virtual serial port.
`scan()`	Performs one scan operation to detect an RFID card.
`getType()`	Returns the card type of the detected RFID tag.
`getUIDByte()`	Returns the UID byte at the specified index.
`Beep()`	Triggers the device buzzer n times.
`Led()`	Turns the specified LED on or off.
`available()`	Checks if valid RFID data is available after a scan.

begin()

Description

Initializes a remote RFID reader connected via EtherCAT HID serial port.
This function must be called inside `setup()` to specify the target RFID model and the corresponding virtual serial port.

Syntax

```
bool begin(RFIDDevice dev, SerialInterface &serialport);
```

Parameters

- *`[in] RFIDDevice dev`*
  The RFID device model to initialize. Currently supported: RC522, PN532.
- *`[in] SerialInterface &serialport`*
  The Virtual Serial port interface used for communication. Typically, one of VirtualSerial1, VirtualSerial2, etc.

Return Value

Returns true if initialization is successful, or false if it fails.

Example

```
#include "myeva.h"

ECRFID rfid1;

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

  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}

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

scan()

Description

Triggers a single scan operation on the remote RFID device to detect the presence of an RFID card.

Syntax

`bool scan();`

Parameters

None.

Return Value

Returns true if an RFID card is detected, otherwise false.

Example

```
#include "myeva.h"

ECRFID rfid1;

void setup() {
  Serial.begin(3000000);
  EVA.begin();
  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}
void loop() {
  rfid1.scan(); // Scan for RFID card
  delay(300);

  if (rfid1.available()) {
    int type = rfid1.getType();
    if (type > 0) {
      Serial.print("Type: "); Serial.print(type);
      Serial.print(", UID: ");
      for (int i = 0; i < 4; i++) {
        Serial.print(rfid1.getUIDByte(i), HEX);
        if (i < 3) Serial.print(":");
      }
      Serial.println();
    } else {
      Serial.println("unknown/no RFID Card");
    }
  }
  delay(1000);
}
```

getType()

Description

Retrieves the type of the RFID card that was scanned by the remote RFID device.

Syntax

```
int getType();
```

Parameters

None.

Return Value

Returns an integer representing the detected RFID card type.

Example

```
#include "myeva.h"

ECRFID rfid1;

void setup() {
  Serial.begin(3000000);
  EVA.begin();
  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}
void loop() {
  rfid1.scan(); // Scan for RFID card
  delay(300);

  if (rfid1.available()) {
    int type = rfid1.getType();
    if (type > 0) {
      Serial.print("Type: "); Serial.print(type);
      Serial.print(", UID: ");
      for (int i = 0; i < 4; i++) {
        Serial.print(rfid1.getUIDByte(i), HEX);
        if (i < 3) Serial.print(":");
      }
      Serial.println();
    } else {
      Serial.println("unknown/no RFID Card");
    }
  }
  delay(1000);
}
```

getUIDByte()

Description

Retrieves a specific byte of the UID (Unique Identifier) from the RFID card scanned by the remote device.

The typical UID length is 4 to 7 bytes depending on the card type.

Syntax

```
unsigned char getUIDByte(int index);
```

Parameters

- *`[in] int index`*
  The byte position to retrieve from the UID (starting from 0).

Return Value

Returns the byte value (as unsigned char) at the specified index of the UID.

Example

```
#include "myeva.h"

ECRFID rfid1;

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

  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}

void loop() {
  rfid1.scan(); // Scan for RFID card
  delay(300);

  if (rfid1.available()) {
    int type = rfid1.getType();
    if (type > 0) {
      Serial.print("Type: "); Serial.print(type);
      Serial.print(", UID: ");
      for (int i = 0; i < 4; i++) {
        Serial.print(rfid1.getUIDByte(i), HEX);
        if (i < 3) Serial.print(":");
      }
      Serial.println();
    } else {
      Serial.println("unknown/no RFID Card");
    }
```

```
  }
  delay(1000);
}
```

Beep()

Description

Triggers the buzzer on the remote RFID device to beep a specified number of times.

Syntax

```
bool Beep(char n);
```

Parameters

- *[in] char n*
  The number of times the RFID device should beep. Valid range: 1–255.

Return Value

Returns true if the beep command was successfully sent to the device, otherwise false.

Example

```
#include "myeva.h"

ECRFID rfid1;

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

  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}
void loop() {
  if (rfid1.scan()) {
    delay(300);
    if (rfid1.available()) {
      int type = rfid1.getType();
      if (type == 4) {
        // Beep once for recognized card type 4
        rfid1.Beep(1);
      } else {
        // Beep twice for other types
        rfid1.Beep(2);
      }
    }
  }

  delay(2000);
}
```

Led()

Description

Turns on or off the LED indicator on the remote RFID device.

Syntax

`bool Led(char which, bool onoff);`

Parameters

- *`[in] char which`*
  LED selection: 0 = Green LED; 1 = Red LED.
- *`[in] bool onoff`*
  LED status.

Return Value

Returns true if the command was successfully sent to the RFID device, otherwise false.

Example

```
#include "myeva.h"

ECRFID rfid1;

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

  VirtualSerial1.begin(115200);
  rfid1.begin(RC522, VirtualSerial1);
}

void loop() {
  if (rfid1.scan()) {
    delay(300);
    if (rfid1.available()) {
      int type = rfid1.getType();
      if (type == 4) {
        // Turn on green LED for 2 seconds
        rfid1.Led(0, true);
        delay(2000);
        rfid1.Led(0, false);
      } else {
        // Turn on red LED for 2 seconds
        rfid1.Led(1, true);
        delay(2000);
        rfid1.Led(1, false);
      }
    }
```

```
  }
  delay(2000);
}
```

available()

Description

Checks whether there is available RFID card data after a scan.

This function should be called after invoking `scan()` to confirm whether a card was successfully detected and read.

Syntax

```
bool available();
```

Parameters

None.

Return Value

true: RFID data is available (a card has been successfully scanned).

false: No valid data available.

Example

```
#include "myeva.h"
ECRFID rfid1;

void setup() {
  Serial.begin(3000000);
  EVA.begin();
  VirtualSerial1.begin(115200);
  rfid1.begin(PN532, VirtualSerial1);
}
void loop() {
  rfid1.scan(); // Scan for RFID card
  delay(300);

  if (rfid1.available()) {
    int type = rfid1.getType();
    if (type > 0) {
      Serial.print("Type: "); Serial.print(type);
      Serial.print(", UID: ");
      for (int i = 0; i < 4; i++) {
        Serial.print(rfid1.getUIDByte(i), HEX);
        if (i < 3) Serial.print(":");
      }
      Serial.println();
    } else {
      Serial.println("unknown/no RFID Card");
    }
  }
  delay(1000);
}
```

3.4 MyVirtualMpg Class

The MyVirtualMpg class provides an interface for communicating with remote MPG (Manual Pulse Generator) devices connected to the dedicated MPG port on DMP HID modules. The communication between the host and the MPG is carried over EtherCAT, enabling high-speed, real-time feedback for industrial control applications such as CNC or motion systems.

This class allows users to read movement increments, axis selections, ratio settings, and button states (enable/emergency), as well as apply signal filtering and direction inversion—all crucial for implementing precise manual jogging or fine-tuning operations in automation systems.

Functions:

Function Name	Description
`begin()`	Initialize the remote Virtual MPG (Manual Pulse Generator).
`write()`	Write a value to the remote Virtual MPG.
`read()`	Read the current raw value from the remote Virtual MPG.
`readVelocity()`	Get the current rotational velocity of the Virtual MPG.
`readIncrement()`	Read the incremental value of the Virtual MPG. May be positive or negative based on rotation direction.
`currentAxis()`	Read the currently selected axis (0 to 4 representing X, Y, Z, A, B).
`currentRatio()`	Read the current feed rate ratio (returns 1, 10, or 100).
`isEnabled()`	Check if the enable button on the Virtual MPG is pressed (true/false).
`isEmergencyStop()`	Check if the emergency stop button is triggered (true/false).
`setDigitalFilter()`	Set the digital filter width in 10 ns units. For example, 250 ns = setDigitalFilter(25).
`invertDirection()`	Invert the counting direction of the Virtual MPG.

begin()

Description

Initializes the remote Virtual MPG device. This function should be called in `setup()` to establish communication with the MPG module via EtherCAT through the HID port.

Syntax

`void begin();`

Parameters

None.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();         // Initialize the EtherCAT Virtual Arduino system
  VirtualMpg1.begin();  // Initialize the Virtual MPG device
}

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

write()

Description

Sends a numerical value to the remote Virtual MPG. This API is typically used for testing or manually updating the internal MPG counter from the QEC MDevice side.

Syntax

`int write(unsigned long data);`

Parameters

- *`[in] unsigned long data`*
  The value to be written to the Virtual MPG.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();           // Initialize EVA
  VirtualMpg1.begin();    // Initialize Virtual MPG

  // Write value to Virtual MPG
  unsigned long valueToWrite = 1000;
  VirtualMpg1.write(valueToWrite);
  Serial.print("Written value: ");
  Serial.println(valueToWrite);
}

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

read()

Description

Reads the current counter value from the remote Virtual MPG.

Syntax

`int read();`

Parameters

None.

Return Value

Returns the current counter value of the Virtual MPG.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();           // Initialize EVA
  VirtualMpg1.begin();    // Initialize Virtual MPG
}

void loop() {
  // Read current MPG value
  int readValue = VirtualMpg1.read();
  Serial.print("Read value: ");
  Serial.println(readValue);

  delay(100); // Print every 100 ms
}
```

readVelocity()

Description

Reads the current rotational speed (velocity) of the remote Virtual MPG.

Syntax

`double readVelocity();`

Parameters

None.

Return Value

Returns the velocity value of the Virtual MPG as a double.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();           // Initialize EVA
  VirtualMpg1.begin();    // Initialize Virtual MPG
}

void loop() {
  // Read the rotational velocity of the MPG
  double velocity = VirtualMpg1.readVelocity();
  Serial.print("Velocity: ");
  Serial.println(velocity);

  delay(100); // Read every 100 ms
}
```

readIncrement()

Description

Reads the incremental value from the remote Virtual MPG (Manual Pulse Generator). The result may be positive or negative depending on the direction of rotation.

Syntax

`int readIncrement();`

Parameters

None.

Return Value

Returns an integer representing the incremental movement. Positive for clockwise rotation, negative for counterclockwise.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();           // Initialize EVA
  VirtualMpg1.begin();    // Initialize Virtual MPG
}

void loop() {
  // Read the incremental rotation of the MPG
  int increment = VirtualMpg1.readIncrement();
  Serial.print("Increment: ");
  Serial.println(increment);

  delay(100); // Read every 100 ms
}
```

currentAxis()

Description

Reads the current axis selected on the remote Virtual MPG (Manual Pulse Generator) rotary selector.

Returns an integer value representing the axis:

- 0 → X Axis
- 1 → Y Axis
- 2 → Z Axis
- 3 → A Axis
- 4 → B Axis

This allows manual operations to be directed to the appropriate machine axis.

Syntax

```
int currentAxis();
```

Parameters

None.

Return Value

Returns an integer (0 to 4) corresponding to the selected axis.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Read the currently selected axis on the MPG
  int currentAxis = VirtualMpg1.currentAxis();
  Serial.print("Current Axis: ");
  Serial.println(currentAxis);
}

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

currentRatio()

Description

Reads the multiplier ratio selected on the remote Virtual MPG (Manual Pulse Generator).
Returns an integer value representing the feed rate multiplier:

- 1 → ×1
- 10 → ×10
- 100 → ×100

This function is used to determine how much movement is applied per encoder pulse based on the operator's selection.

Syntax

```
int currentRatio();
```

Parameters

None.

Return Value

Returns the current multiplier ratio (1, 10, or 100).

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Read the current multiplier ratio
  int currentRatio = VirtualMpg1.currentRatio();
  Serial.print("Current Ratio: ");
  Serial.println(currentRatio);
}

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

isEnabled()

Description

Checks whether the "Enable" button on the remote Virtual MPG is currently pressed.

Syntax

`bool isEnabled();`

Parameters

None.

Return Value

Returns true if the Enable button is pressed, otherwise returns false.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Check if the Enable button is pressed
  bool enabled = VirtualMpg1.isEnabled();
  Serial.print("Is Enabled: ");
  Serial.println(enabled ? "true" : "false");
}

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

isEmergencyStop()

Description

Checks whether the emergency stop button on the remote Virtual MPG (Manual Pulse Generator) has been pressed.

Syntax

```
bool isEmergencyStop();
```

Parameters

None.

Return Value

Returns true if the emergency stop button is currently pressed, otherwise returns false.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Check if the emergency stop button is pressed
  bool emergencyStop = VirtualMpg1.isEmergencyStop();
  Serial.print("Is Emergency Stop: ");
  Serial.println(emergencyStop ? "true" : "false");
}

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

setDigitalFilter()

Description

Sets the digital filter width for the remote Virtual MPG (Manual Pulse Generator). This filter helps to eliminate electrical noise or signal glitches shorter than the specified duration.
The unit is in 10 nanoseconds (ns). For example, passing 25 sets the filter width to 250ns, which means any pulse shorter than 250ns will be filtered out.

Syntax

```
void setDigitalFilter(unsigned long time_us);
```

Parameters

- *`[in] unsigned long time_us`*
  The desired filter width in units of 10ns.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Set digital filter to 250ns
  unsigned long filterTime = 25;
  VirtualMpg1.setDigitalFilter(filterTime);
  Serial.print("Digital filter set to: ");
  Serial.println(filterTime);
}

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

invertDirection()

Description

Reverses the counting direction of the remote Virtual MPG (Manual Pulse Generator).
For example, if rotating clockwise originally increases the value, after calling this function, clockwise rotation will decrease it.

Syntax

`int invertDirection(void);`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(9600);
  EVA.begin();
  VirtualMpg1.begin();

  // Invert counting direction of the MPG
  int invertResult = VirtualMpg1.invertDirection();
  Serial.print("Invert Direction Result: ");
  Serial.println(invertResult);
}

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

Other member functions

The following member functions are also available in the IDE 501 library:

Function	Syntax	Description
`writeRaw()`	`int writeRaw(unsigned long data);`	Writes the raw counter value of the Virtual MPG.
`readRaw()`	`int readRaw();`	Reads the raw counter value of the Virtual MPG.
`attachInterrupt()`	`void attachInterrupt(void (*isr)(int));`	Registers a callback function that is called when the MPG input changes.
`detachInterrupt()`	`void detachInterrupt();`	Removes the callback function registered by `attachInterrupt()`.

3.5 MyVirtualLcm Class

The MyVirtualLcm class provides control over the remote Virtual LCM (Liquid Crystal Module) device on DMP HID modules. This device features a 16x2 character display and communicates via EtherCAT. It enables users to initialize the display, write characters or strings, control the cursor position, and clear the screen — all remotely.

This class is ideal for status displays, user prompts, or simple HMI feedback in automation applications.

Functions:

Function Name	Description
`begin()`	Initializes the remote Virtual LCM.
`home()`	Sets the LCM cursor to the home position (row 1, column 1).
`setCursor()`	Moves the cursor to the specified row (0 or 1) and column (0–15).
`write()`	Writes a single character to the LCM.
`print()`	Prints a string starting from the current cursor position.
`clear()`	Clears all contents on the LCM display.

begin()

Description

Initializes the remote Virtual LCM device on the DMP HID module.

Syntax

`void begin();`

Parameters

None.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();
  VirtualLcm1.begin();    // Initialize remote Virtual LCM1
}

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

home()

Description

Moves the cursor of the remote Virtual LCM (on DMP HID) back to the home position at the top-left of the screen (row 1, column 1).

Syntax

`void home();`

Parameters

None.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);  // Initialize host serial port
  EVA.begin();
  VirtualLcm1.begin();    // Initialize remote Virtual LCM1

  // Move cursor to the top-left corner (1,1)
  VirtualLcm1.home();
}

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

setCursor()

Description

Moves the cursor of the remote Virtual LCM (on DMP HID) to the specified position.

The display supports 2 rows and 16 columns (indexed from 0).

Syntax

```
void setCursor(uint8_t row, uint8_t col);
```

Parameters

- *[in] uint8_t row*
  Target row index. Valid range is 0 (first row) or 1 (second row).
- *[in] uint8_t col*
  Target column index. Valid range is 0 to 15.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000); // Initialize host serial port
  EVA.begin();
  VirtualLcm1.begin();   // Initialize remote Virtual LCM1

  // Move cursor to row 1, column 2 (second row, third column)
  VirtualLcm1.setCursor(1, 2);
}

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

write()

Description

Writes a single character to the remote Virtual LCM screen.

Syntax

`virtual size_t write(uint8_t ch);`

Parameters

- *`[in] uint8_t ch`*
  A single character to display on the LCM.

Return Value

Returns the number of bytes written (1)._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000); // Initialize host serial port
  EVA.begin();
  VirtualLcm1.begin();   // Initialize remote Virtual LCM1
}

void loop() {
  // Write a single character to the LCM
  VirtualLcm1.write('A');
  delay(2000); // Wait for 2 seconds before repeating
}
```

print()

Description

Writes a string to the remote Virtual LCM screen starting from the current cursor position.

Syntax

`size_t print(const char* str);`

Parameters

- *`[in] const char* str`*
  A null-terminated string to be displayed on the LCM.

Return Value

Returns the number of bytes written._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);   // Initialize host serial port
  EVA.begin();
  VirtualLcm1.begin();     // Initialize remote Virtual LCM1

  VirtualLcm1.home();      // Move cursor to starting position
  VirtualLcm1.print("Hello, LCM!"); // Display a message
}

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

clear()

Description

Clears all characters displayed on the remote Virtual LCM (16x2 LCD screen).

Syntax

`void clear();`

Parameters

None.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000); // Initialize serial port
  EVA.begin();
  VirtualLcm1.begin();   // Initialize remote Virtual LCM1
}

void loop() {
  // Display text
  VirtualLcm1.print("Hello, LCM!");

  delay(500); // Wait for a moment

  // Clear the screen
  VirtualLcm1.clear();

  delay(500); // Wait again
}
```

3.6 MyVirtualKeypad Class

The MyVirtualKeypad class provides access to a remote 16-key physical keypad device, which is part of the DMP HID module and connected via EtherCAT.
The keypad includes the following 16 keys: 0–9, A–D, *, #.

This class enables developers to initialize the keypad, detect real-time key presses, or poll the latest pressed key without blocking program execution.

Functions:

Function Name	Description
`begin()`	Initializes the remote Virtual Keypad.
`waitForKey()`	Blocks and waits until a key is pressed, then returns the key value.
`getKey()`	Non-blocking version. Returns the latest pressed key if available.

begin()

Description

Initializes the remote Virtual Keypad device on the DMP HID module.

Syntax

`void begin();`

Parameters

None.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();
  VirtualKeypad1.begin();    // Initialize remote Virtual Keypad1
}

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

waitForKey()

Description

Waits until a key on the remote Virtual Keypad is pressed, then returns the corresponding key value. This is a blocking function, meaning it will halt program execution until a key is pressed.

Syntax

`char waitForKey();`

Parameters

None.

Return Value

Returns the character of the key that was pressed ('0'–'9', 'A'–'D', '*', '#').

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);       // Initialize Serial
  EVA.begin();                // Initialize EVA
  VirtualKeypad1.begin();      // Initialize remote Virtual Keypad
}

void loop() {
  // Wait until a key is pressed
  char key = VirtualKeypad1.waitForKey();

  // Print the key to Serial
  Serial.print("Key Pressed: ");
  Serial.println(key);
}
```

getKey()

Description

Retrieves the current key value from the remote Virtual Keypad without blocking. If no key is currently pressed, it returns a null character ('\0').

Syntax

`char getKey();`

Parameters

None.

Return Value

Returns the character of the key currently pressed ('0'–'9', 'A'–'D', '*', '#').

If no key is pressed, returns '\0'.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);  // Initialize Serial
  EVA.begin();           // Initialize EVA
  VirtualKeypad1.begin(); // Initialize remote Virtual Keypad
}

void loop() {
  // Non-blocking read of current key
  char currentKey = VirtualKeypad1.getKey();

  // Print if a key is pressed
  if (currentKey != '\0') {
    Serial.print("Key Pressed: ");
    Serial.println(currentKey);
  }

  delay(500); // Check every 500ms
}
```

3.7 MyVirtualEncoder Class

The MyVirtualEncoder class provides access to a virtual quadrature encoder interface designed specifically for the **DMP 3-axis motor controller** (e.g., MP3S and MV3S series). Through EtherCAT communication, this virtual encoder simulates a hardware encoder signal and allows developers to both read and write encoder values in real time, configure modes, reset behavior, polarity, and apply digital filtering.

This class is particularly useful in applications where precise positioning and motion simulation are required, such as virtual homing, multi-axis synchronization, or encoder emulation for closed-loop systems.

Supported Features

- Read/write encoder position
- Direction detection
- Index pulse configuration
- Digital filtering for signal stability
- Input signal polarity control
- Range boundary setup for overflow/underflow handling

Functions:

Function Name	Description
`begin()`	Initializes the virtual encoder device.
`write()`	Writes a value to the encoder (sets the current position).
`read()`	Reads the current encoder position value.
`directionRead()`	Reads the current rotation direction (1 = forward, -1 = reverse, 0 = idle).
`setMode()`	Sets the encoder counting mode.
`setIndexReset()`	Enables or disables index pulse reset behavior.
`setDigitalFilter()`	Sets the digital filter width (in 10ns units) for signal noise suppression.
`setInputPolarity()`	Sets polarity inversion for each input pin A, B, and Z.
`setRange()`	Sets the encoder range and optional reset-on-overflow condition.

begin()

Description

Initializes the remote virtual encoder on the QEC 3-axis motion controller (MP3S or MV3S series). This must be called in `setup()` before using other VirtualEncoder functions.

Syntax

`int begin();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  VirtualEncoder1.begin(); // Init Virtual Encoder
}

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

write()

Description

Sets the current value of the remote Virtual Encoder.

This is useful for resetting or initializing the encoder's position count.

Default encoder value after boot is 0.

Syntax

`void write(unsigned long value);`

Parameters

- *`[in] unsigned long value`*
  The new position value to set for the encoder.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

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

  // Start virtual encoder
  VirtualEncoder1.begin();

  // Set encoder value to 1000
  // Default value after boot is 0
  VirtualEncoder1.write(1000);
}

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

read()

Description

Reads the current position value of the remote Virtual Encoder.

The value returned will not exceed the range previously set by `setRange()`.

Syntax

`unsigned long read();`

Parameters

None.

Return Value

Returns the current encoder count as an unsigned long.

Example

```
#include "myeva.h"

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

  // Start the virtual encoder
  VirtualEncoder1.begin();
}

void loop() {
  // Read encoder value
  unsigned long value = VirtualEncoder1.read();
  // Value won't exceed range set earlier

  Serial.print("Encoder Value: ");
  Serial.println(value);
}
```

directionRead()

Description

Reads the current rotation direction of the remote Virtual Encoder.

Syntax

`int directionRead();`

Parameters

None.

Return Value

Returns an int indicating the direction of rotation.

Example

```
#include "myeva.h"

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

  VirtualEncoder1.begin();   // Start the virtual encoder
}

void loop() {
  // Read encoder direction
  int direction = VirtualEncoder1.directionRead();

  Serial.print("Direction: ");
  Serial.println(direction);
}
```

setMode()

Description

Sets the operation mode of the remote Virtual Encoder.

This determines how input signals are interpreted and counted.

Mode List

- 0: STEP_DIR
- 1: CWCCW
- 2: AB_PHASE
- 5: STEP_DIR x2
- 6: CWCCW x2
- 7: AB_PHASE x2

Syntax

```
void setMode(uint8_t mode);
```

Parameters

- *[in] uint8_t mode*
  A number indicating the encoder input mode.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();               // Initialize EVA
  VirtualEncoder1.begin();   // Start the virtual encoder

  // Set encoder mode to STEP_DIR
  VirtualEncoder1.setMode(0);
}

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

setIndexReset()

Description

Enables or disables the Index (Z) reset function on the remote Virtual Encoder.

When enabled, the encoder value will automatically reset to 0 upon detecting an index signal (Z).

Syntax

```
void setIndexReset(bool _on_off);
```

Parameters

- *`[in] bool _on_off`*
  true – Enable index reset.
  false – Disable index reset.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();                // Initialize EVA
  VirtualEncoder1.begin();     // Start the virtual encoder

  // Enable index reset
  VirtualEncoder1.setIndexReset(true);
}

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

setDigitalFilter()

Description

Sets the digital filter width for the remote Virtual Encoder.

This filter suppresses input pulses that are shorter than the specified width.

Syntax

`void setDigitalFilter(unsigned long width);`

Parameters

- *`[in] unsigned long width`*
  The filter width in units of 10 nanoseconds. For example, passing 10 sets the filter to 100ns (10 × 10ns).

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();                // Initialize EVA
  VirtualEncoder1.begin();     // Start the virtual encoder

  // Set digital filter width to 100ns
  VirtualEncoder1.setDigitalFilter(10);
}

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

setInputPolarity()

Description

Sets the input polarity for the A, B, and Z signal lines of the remote Virtual Encoder.

Syntax

```
void setInputPolarity(bool pinA, bool pinB, bool pinZ);
```

Parameters

- *[in] bool pinA*
  true = inverted; false = normal.
- *[in] bool pinB*
  true = inverted; false = normal.
- *[in] bool pinZ*
  true = inverted; false = normal.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();                // Initialize EVA
  VirtualEncoder1.begin();     // Start the virtual encoder

  // Set polarity: A = inverted, B = normal, Z = normal
  VirtualEncoder1.setInputPolarity(true, false, false);
}

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

setRange()

Description

Sets the counting range for the remote Virtual Encoder. This defines the maximum value the encoder can reach. By default, values start from 0 up to the range you define. If condition is set to true, the encoder will allow signed (±) values.

Syntax

```
void setRange(unsigned long val, bool condition = false);
```

Parameters

- *[in] unsigned long val*
  Maximum encoder value.
- *[in] bool condition = false*
  Set to true to enable signed (±) range mode. Default is false.

Return Value

None._A.1__Error

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);
  EVA.begin();                // Initialize EVA
  VirtualEncoder1.begin();     // Start the virtual encoder

  // Set encoder range from 0 to 1000 (unsigned)
  VirtualEncoder1.setRange(1000);
}

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

Other member functions

The following member functions are also available in the IDE 501 library:

Function	Syntax	Description
`getRange()`	`uint32_t getRange();`	Returns the counter range set by `setRange()`.

3.8 MyVirtualCNC Class

MyVirtualCNC is a virtual CNC control class designed specifically for DMP's 3-axis EtherCAT motor controllers (such as the QEC-RXXMP3S or QEC-RXXMV3S series).

It provides a simplified interface to send and execute G-code instructions directly from your sketch via EVA, enabling motion control commands in a CNC-like format.

Functions:

Function Name	Description
`begin()`	Initializes the VirtualCNC.
`gcode()`	Sends a formatted G-code command string to the virtual CNC controller.

begin()

Description

Initializes the VirtualCNC system and prepares the device to receive G-code commands.

Syntax

`int begin();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Initialize Virtual CNC
  VirtualCNC1.begin();

  // Send G-code command
  VirtualCNC1.gcode("G0 X10 Y10 Z10");
}

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

gcode()

Description

Sends a formatted G-code command to the VirtualCNC device for motion control or other CNC operations. This function supports C-style format strings and variable arguments, similar to `printf()`.

Syntax

```
int gcode(char *format, ...);
```

Parameters

- *`[in] char *format`*
  The G-code command string.
- *`[in] ...`*
  Optional variable arguments to be substituted into the format string.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Initialize Virtual CNC
  VirtualCNC1.begin();

  // Send G-code command
  VirtualCNC1.gcode("G0 X10 Y10 Z10");
}

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

Other member functions

The following member functions are also available in the IDE 501 library:

Function	Syntax	Description
`gcode()`	`int gcode(const String &s);`	Sends a G-code command given as a String object.
`machineOn()`	`int machineOn(void);`	Turns on (servo on) the motors of the machine and waits up to 1 second for confirmation. Returns 0 on success, or a negative value on failure.
`disableSoftLimit()`	`int disableSoftLimit(void);`	Turns on the machine and disables the software limits set in Machine Configuration. Returns 0 on success, or a negative value on failure.
`getLimitInputs()`	`unsigned char getLimitInputs();`	Returns the state of the limit switch inputs as a bit mask.
`getHomeInputs()`	`unsigned char getHomeInputs();`	Returns the state of the home switch inputs as a bit mask.
`getAlarmInputs()`	`unsigned char getAlarmInputs();`	Returns the state of the driver alarm inputs as a bit mask.
`motorIsMoving()`	`uint8_t motorIsMoving();`	Returns 1 while the machine is moving, otherwise 0.
`setDACVoltage()`	`void setDACVoltage(double outV);`	Sets the analog (DAC) output voltage of the motor controller (QEC-R11MV6S).

***Note:** `getLimitInputs()`, `getHomeInputs()`, and `getAlarmInputs()` read the input states through SDO; avoid calling them in the cyclic callback function.

3.9 MyVirtualServo Class

The MyVirtualServo class provides a high-level interface for controlling servo motors that conform to the CiA402 standard. It is designed to be compatible with all EtherCAT CiA402-compliant motion devices, including DMP's own MP3S and MV3S 3-axis controllers.

Depending on the specific device used, some API behaviors may vary—for example, homing or movement profiles may differ slightly between third-party drives and DMP-native hardware. This class allows for intuitive control of servo position, speed, and acceleration in real-time EtherCAT systems.

The typical use case includes:

- Writing target angles for motion
- Reading the current position
- Controlling velocity and acceleration
- Executing motion commands such as homing, pausing, or stopping
- Monitoring movement status in real time

Functions:

Function Name	Description
`write()`	Set the target angle or target position for the servo motor.
`read()`	Read the current position or angle of the servo.
`setVelocity()`	Set the target velocity.
`setAcceleration()`	Set the target acceleration.
`setPositionType()`	Set the motion profile type (e.g., trapezoidal, S-curve).
`home()`	Execute a homing operation.
`pause()`	Pause current motion.
`resume()`	Resume motion after pause.
`stop()`	Immediately stop the servo motor.
`release()`	Release the servo to allow free movement.
`isMoving()`	Check if the servo is currently in motion.

write()

Description

Sets the target position (or angle) for the virtual servo motor.

The actual meaning of the angle value depends on the currently selected position mode via `setPositionType()`.

Syntax

```
int write(double angle);
```

Parameters

- *`[in] double angle`*
  The target position, which can be interpreted as either a number of pulses or an angle in degrees, depending on the set mode.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  // Example 1: Absolute position in pulse units
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);
  VirtualServo1.write(10000); // Move to 10,000 pulses from origin

  // Example 2: Absolute position in angle units
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_ANGLE);
  VirtualServo1.write(360); // Rotate to 360 degrees absolute

  // Example 3: Relative move in pulse units
  VirtualServo1.setPositionType(VIRTUALSERVO_RELATIVE_POSITION_STEP);
  VirtualServo1.write(10000); // Move +10,000 pulses relative to current

  // Example 4: Relative move in angle units
  VirtualServo1.setPositionType(VIRTUALSERVO_RELATIVE_POSITION_ANGLE);
  VirtualServo1.write(100); // Rotate +100 degrees from current angle
}
void loop() {
  //...
}
```

read()

Description

Reads the current target position of the virtual servo motor.
This position is calculated internally by the controller, not from the encoder.

The return unit depends on the position mode set via `setPositionType()`:

- If set to angle mode, the return value is in degrees.
- If set to step mode, the return value is in pulses.

Syntax

```
double read(void);
```

Parameters

None.

Return Value

Returns the current position (angle or pulse) based on the configured mode.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA and VirtualServo
  EVA.begin();

  // Configure velocity, acceleration, and position type
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Read current servo position (unit: pulse)
  Serial.println(VirtualServo1.read());
}

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

setVelocity()

Description

Sets the target velocity of the virtual servo motor.

This function must be called before issuing a movement command using `write()`.

Syntax

```
int setVelocity(double velocity);
```

Parameters

- *`[in] double velocity`*
  The desired velocity for the servo motor.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  VirtualServo1.write(10000);
}

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

setAcceleration()

Description

Sets the acceleration value for the virtual servo motor.
This value determines how quickly the motor ramps up to its target velocity when a movement command is issued.

Syntax

```
int setAcceleration(double acc);
```

Parameters

- *[in] double acc*
  The acceleration value for the servo motor.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  VirtualServo1.write(10000);
}

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

setPositionType()

Description

This determines how the input to `write()` will be interpreted — as absolute or relative, and in units of angle or pulse.

Modes

Enum Constant	Description
VIRTUALSERVO_ABSOLUTE_POSITION_ANGLE	Absolute position in angle units
VIRTUALSERVO_RELATIVE_POSITION_ANGLE	Relative position in angle units
VIRTUALSERVO_ABSOLUTE_POSITION_STEP	Absolute position in pulse counts
VIRTUALSERVO_RELATIVE_POSITION_STEP	Relative position in pulse counts

Syntax

```
int setPositionType(VirtualServoPositionTypes type);
```

Parameters

- *`[in] VirtualServoPositionTypes type`*
  One of the 4 predefined VirtualServoPositionTypes.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  // Example 1: Absolute position in pulse units
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);
  VirtualServo1.write(10000); // Move to 10,000 pulses from origin

  // Example 2: Absolute position in angle units
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_ANGLE);
  VirtualServo1.write(360); // Rotate to 360 degrees absolute

  // Example 3: Relative move in pulse units
  VirtualServo1.setPositionType(VIRTUALSERVO_RELATIVE_POSITION_STEP);
  VirtualServo1.write(10000); // Move +10,000 pulses relative to current

  // Example 4: Relative move in angle units
  VirtualServo1.setPositionType(VIRTUALSERVO_RELATIVE_POSITION_ANGLE);
  VirtualServo1.write(100); // Rotate +100 degrees from current angle
}
```

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

home()

Description

Commands the virtual servo motor to perform a homing operation.
The homing behavior (mode, speed, acceleration, etc.) follows the configuration set in the 86EVA interface.
This function internally switches to homing mode. After homing is completed, it automatically returns to Profile Position (PP) mode.
This function is blocking — it will not return until the homing procedure is complete.

Syntax

`int home();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Set motion profile
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Perform homing
  VirtualServo1.home();
}

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

pause()

Description

Pauses the servo motion.
This function applies a controlled deceleration rather than an immediate stop.

Not all EtherCAT servo controllers support this function. Some third-party devices may not respond to this command due to firmware limitations.

Syntax

`int pause();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Configure motion
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Start movement
  VirtualServo1.write(10000);

  delay(1000); // Wait for 1 second before pausing
  VirtualServo1.pause(); // Issue pause command
}

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

resume()

Description

Resumes servo motion from a previously paused state.
The servo will continue toward the original target position using the configured velocity and acceleration.

Not all EtherCAT servo controllers support this function. Some controllers may ignore the resume command due to firmware limitations.

Syntax

`int resume();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Configure motion
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Start movement
  VirtualServo1.write(10000);
  delay(1000);
  VirtualServo1.pause();   // Pause after 1 second
  delay(1000);
  VirtualServo1.resume();  // Resume after another 1 second
}

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

stop()

Description

Stops the motion of the servo motor.
The stop action is not immediate; it decelerates the motor according to the configured acceleration profile.
After stopping, the target position is cleared — meaning you cannot resume the motion.
To move the servo again, a new target position must be set using `write()`.

Not all EtherCAT servo controllers support this function. Some controllers may not respond to the stop command due to internal design.

Syntax

```
int stop();
```

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(3000000);

  // Initialize EVA
  EVA.begin();

  // Configure motion parameters
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Move to position
  VirtualServo1.write(10000);
  delay(1000);

  // Stop the servo
  VirtualServo1.stop();
  delay(1000);

  // Resume won't work here
  VirtualServo1.resume();  // This will have no effect
```

```
  delay(1000);

  // Must send new target to move again
  VirtualServo1.write(0);
}

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

release()

Description

Releases the servo motor, cutting off the motor output so it no longer holds position (i.e., enters free-running mode).
To re-enable output and move again, you must issue a new `write()` command.
Calling `stop()` or `resume()` after `release()` has no effect.

The actual behavior may depend on the specific EtherCAT servo controller in use. Some may ignore the release command if unsupported.

Syntax

`int release();`

Parameters

None.

Return Value

Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Example

```
#include "myeva.h"

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

  // Configure motion parameters
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);

  // Move to position
  VirtualServo1.write(10000);
  delay(1000);
  // Release the motor (disable holding force)
  VirtualServo1.release();
  delay(1000);

  // Resume/stop has no effect here
  VirtualServo1.write(0);
}
void loop() {
}
```

isMoving()

Description

Checks whether the servo motor is currently moving toward its target.
Returns true if the motor is in motion, and false if it has stopped, paused, or completed its motion.
This function is useful for synchronizing operations or sequencing commands after motion completes.
Use this function in a loop to block execution until motion finishes.

Syntax

`bool isMoving();`

Parameters

None.

Return Value

Returns true if the servo is moving, otherwise false._A.1__Error

Example

```
#include "myeva.h"

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

  // Set motion parameters
  VirtualServo1.setVelocity(3200.0);
  VirtualServo1.setAcceleration(3200.0);
  VirtualServo1.setPositionType(VIRTUALSERVO_ABSOLUTE_POSITION_STEP);
}

void loop() {
  // Move to target position 1
  VirtualServo1.write(10000);

  // Wait until the motor stops moving
  while (VirtualServo1.isMoving());

  // Move to target position 2
  VirtualServo1.write(0);

  // Wait again until the motion is done
  while (VirtualServo1.isMoving());
}
```

Other member functions

The following member functions are also available in the IDE 501 library:

Function	Syntax	Description
setOffset()	int setOffset(double offset);	Sets a position offset in angle or step units, according to the position type set by `setPositionType()`.
angleMap()	void angleMap(double angle1, double angle2, double motorStep1, double motorStep2);	Defines the conversion between angle and motor steps. By default, 0–360 degrees is mapped to 0 – motor resolution (pulses/rev).
getT600Voltage()	double getT600Voltage(void);	Returns the supply voltage of a QEC-T600 servo (V).
getT600Temperature()	double getT600Temperature(void);	Returns the coil temperature of a QEC-T600 servo (°C).
getT600ServoCurrent()	double getT600ServoCurrent(void);	Returns the servo current of a QEC-T600 servo (A).

3.10 MyVirtualLCD Class

The MyVirtualLCD class draws graphics and text on a touch screen LCD connected through EtherCAT (2.4" LCD Mapping, see Ch. 2.6.3.11).
The objects are declared in myeva.h as VirtualLcd1, VirtualLcd2, and so on. The class is derived from Print, so `print()` and `println()` can be used to display text. Colors are 16-bit RGB565 values; constants such as TFT_BLACK, TFT_WHITE, TFT_RED, TFT_GREEN, and TFT_BLUE are defined in VirtualLcd.h.
Functions:

Function	Syntax	Description
`begin()`	`void begin(uint16_t lcdID);`	Initializes the Virtual LCD. lcdID: 0 = 2.4" LCD, 2 = 3.5" LCD. Must be called in setup().
`width() / height()`	`int16_t width(); int16_t height();`	Returns the width / height of the display in pixels.
`getRotation() / setRotation()`	`uint8_t getRotation(); int setRotation(uint8_t x);`	Gets / sets the display rotation (0–3).
`fillScreen()`	`int fillScreen(uint16_t color);`	Fills the entire screen with a color.
`drawPixel()`	`int drawPixel(int16_t x, int16_t y, uint16_t color);`	Draws one pixel.
`drawLine() / drawFastHLine() / drawFastVLine()`	`int drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color);`	Draws a line, or a horizontal / vertical line with drawFastHLine(x, y, w, color) / drawFastVLine(x, y, h, color).
`drawRect() / fillRect()`	`int drawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);`	Draws an outlined / filled rectangle.
`drawRoundRect() / fillRoundRect()`	`int drawRoundRect(int16_t x, int16_t y, int16_t w, int16_t h, int16_t r, uint16_t color);`	Draws an outlined / filled rectangle with rounded corners of radius r.
`drawCircle() / fillCircle()`	`int drawCircle(int16_t x, int16_t y, int16_t r, uint16_t color);`	Draws an outlined / filled circle.
`drawTriangle() / fillTriangle()`	`int drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color);`	Draws an outlined / filled triangle.
`setCursor()`	`int setCursor(int16_t x,`	Sets the text cursor position in

	int16_t y);	pixels.
setTextColor()	int setTextColor(uint16_t color, uint16_t background);	Sets the text color, and optionally the background color.
setTextSize()	int setTextSize(uint8_t s);	Sets the text size multiplier (setTextSize(w, h) sets width and height separately).
setTextWrap()	int setTextWrap(bool wrap);	Enables or disables automatic text wrapping.
color565()	uint16_t color565(uint8_t red, uint8_t green, uint8_t blue);	Converts 8-bit red, green, and blue values to an RGB565 color.
isTouched()	int isTouched();	Returns a non-zero value while the touch screen is pressed.
touchX() / touchY()	int touchX(size_t point = 0); int touchY(size_t point = 0);	Returns the X / Y coordinate of the touch point.
touchCalibration()	int touchCalibration(int flag = 0);	Starts the touch screen calibration.
flush()	int flush(int32_t timeout_ms = -1);	Sends pending drawing commands to the LCD. timeout_ms = -1 waits until finished.

Example

```
#include "myeva.h"

void setup() {
  EVA.begin();
  VirtualLcd1.begin(0);             // 2.4" LCD
  VirtualLcd1.fillScreen(TFT_BLACK);
  VirtualLcd1.setTextColor(TFT_WHITE);
  VirtualLcd1.setTextSize(2);
  VirtualLcd1.setCursor(10, 10);
  VirtualLcd1.print("Hello 86EVA");
}

void loop() {
  if (VirtualLcd1.isTouched())
    VirtualLcd1.fillCircle(VirtualLcd1.touchX(), VirtualLcd1.touchY(), 5,
TFT_RED);
}
```

3.11 MyVirtualMicrophone Class

The MyVirtualMicrophone class reads audio data from a microphone connected through EtherCAT (Microphone Mapping, see Ch. 2.6.3.11).
The objects are declared in myeva.h as VirtualMic1, VirtualMic2, and so on. Audio data is transferred in blocks of 256 bytes.
Functions:

Function	Syntax	Description
`begin()`	`int begin(void);`	Initializes the Virtual Microphone. Must be called in `setup()`. Returns an error code.
`setGain()`	`int setGain(uint16_t gain0, uint16_t gain1);`	Sets the gain of the two microphone amplifier stages (Gain1 and Gain2). Returns an error code.
`record()`	`int record(char* buf, int size);`	Copies one block of audio data into buf. size is normally 256. Returns the number of bytes copied, or a negative value on error.

For an example, see Ch. 3.12 MyVirtualSpeaker Class (microphone-to-speaker loopback).

3.12 MyVirtualSpeaker Class

The MyVirtualSpeaker class plays audio data on a speaker connected through EtherCAT (Audio Mapping, see Ch. 2.6.3.11).
The objects are declared in myeva.h as VirtualSpeaker1, VirtualSpeaker2, and so on. Audio data is transferred in blocks of 256 bytes.
Functions:

Function	Syntax	Description
`begin()`	`int begin(void);`	Initializes the Virtual Speaker. Must be called in `setup()`. Returns an error code.
`setVolume()`	`int setVolume(uint16_t volume);`	Sets the speaker volume. Returns an error code.
`play()`	`int play(char* buf, int size);`	Plays one block of audio data. size is normally 256; shorter data is padded with zeros. Returns an error code.

Example

```
#include "myeva.h"

char buf[256];

void setup() {
  EVA.begin();
  VirtualMic1.begin();
  VirtualSpeaker1.begin();
  VirtualSpeaker1.setVolume(50);
}

void loop() {
  int n = VirtualMic1.record(buf, 256);   // microphone -> speaker loopback
  if (n > 0)
    VirtualSpeaker1.play(buf, n);
}
```

3.13 MyVirtualEMSensor Class

The MyVirtualEMSensor class reads an environmental monitoring SubDevice (e.g., QEC-R11EM1S, Sensor Mapping, see Ch. 2.6.3.11).
The objects are declared in myeva.h as VirtualEMSensor1, VirtualEMSensor2, and so on. Each read function returns the latest value reported by the sensor.
Functions:

Function	Syntax	Description
`begin()`	`int begin(void);`	Initializes the Virtual Environment Sensor. Must be called in `setup()`.
`readTemperature()`	`double readTemperature(void);`	Returns the temperature (°C).
`readHumidity()`	`double readHumidity(void);`	Returns the relative humidity (%RH).
`readPressure()`	`int readPressure(void);`	Returns the air pressure.
`readPM25()`	`int readPM25(void);`	Returns the PM2.5 concentration.
`readCO2()` / `readCO()` / `readO2()`	`int readCO2(void); int readCO(void); int readO2(void);`	Returns the CO2 / CO / O2 concentration.
`readFlammableGas()`	`int readFlammableGas(void);`	Returns the flammable gas concentration.
`readIllumination()`	`int readIllumination(void);`	Returns the illumination.
`readPIR()`	`int readPIR(void);`	Returns the passive infrared (PIR) motion sensor state.
`readBatteryStatus()`	`int readBatteryStatus(void);`	Returns the battery status.
`light()`	`int light(int value);`	Turns the LED on (1) or off (0).
`digitalWrite()` / `digitalRead()`	`int digitalWrite(uint8_t pin, uint8_t value); int digitalRead(uint8_t pin);`	Writes / reads the HIGH or LOW state of a digital pin on the sensor module.
`setVolume()` / `play()`	`int setVolume(uint16_t volume); int play(char* buf, int size);`	Sets the volume / plays one 256-byte block of audio data on the sensor module.

Example

```
#include "myeva.h"

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

```
  VirtualEMSensor1.begin();
}

void loop() {
  Serial.print("Temperature: ");
  Serial.println(VirtualEMSensor1.readTemperature());
  Serial.print("Humidity: ");
  Serial.println(VirtualEMSensor1.readHumidity());
  Serial.print("PM2.5: ");
  Serial.println(VirtualEMSensor1.readPM25());
  delay(1000);
}
```

3.14 MyVirtualForceSensor Class

The MyVirtualForceSensor class reads an EtherCAT force sensor SubDevice (e.g., QEC-Force). The objects are declared in myeva.h as VirtualForceSensor1, VirtualForceSensor2, and so on. This class is different from the ForceSensor class in Ch. 3.3.2, which reads an RS-232 force sensor through a Virtual Serial port.

Functions:

Function	Syntax	Description
`begin()`	`int begin(void);`	Initializes the Virtual Force Sensor. Must be called in setup().
`getFz() / getMx() / getMy()`	`float getFz(); float getMx(); float getMy();`	Returns the force Fz and the moments Mx / My.
`getStress()`	`float getStress(int ch);`	Returns the scaled stress value of channel ch.
`getRaw()`	`uint32_t getRaw(int ch);`	Returns the raw ADC value of channel ch.
`setFootLength() / setFootWidth()`	`int setFootLength(uint32_t length); int setFootWidth(uint32_t width);`	Sets the foot (plate) length / width used to calculate the moments. getFootLength() / getFootWidth() return the current values.
`setSensorScale() / getSensorScale()`	`int setSensorScale(int ch, float scale); float getSensorScale(int ch);`	Sets / gets the scale factor of channel ch.
`setSensorFilter() / getSensorFilter()`	`int setSensorFilter(int ch, float filter); float getSensorFilter(int ch);`	Sets / gets the filter of channel ch.
`userSoftReset()`	`int userSoftReset();`	Performs a soft reset of the sensor.

Example

```
#include "myeva.h"

void setup() {
  Serial.begin(115200);
  EVA.begin();
  VirtualForceSensor1.begin();
}

void loop() {
  Serial.print("Fz: ");
```

```
  Serial.print(VirtualForceSensor1.getFz());
  Serial.print("  Mx: ");
  Serial.print(VirtualForceSensor1.getMx());
  Serial.print("  My: ");
  Serial.println(VirtualForceSensor1.getMy());
  delay(100);
}
```

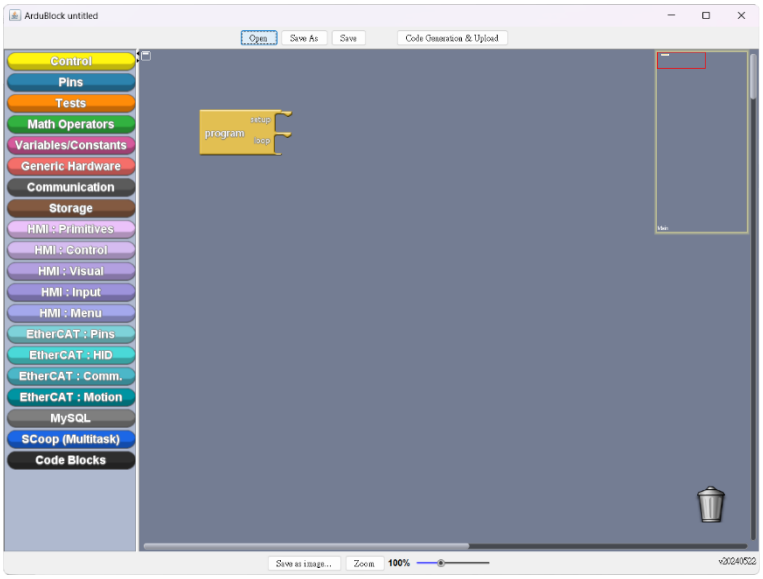
Ch. 4

ArduBlock Programming

4.1 About ArduBlock

ArduBlock is a visual programming language that lowers the barrier to entry for writing Arduino programs by dragging and dropping building blocks. Even if you have never learned to code, you can easily write powerful Arduino programs. ArduBlock was developed by third-party developer David Li and other creators as a plug-in for the Arduino IDE, and was developed from 2014 to October 2017. Its source code is open source for download and use on GitHub. (GitHub download: https://github.com/taweili/ardublock).

ArduBlock was developed with the goal of making it possible for users with no programming background to get started with Arduino quickly. With an intuitive block interface, users can simply drag and splice blocks with different functions to complete the program, each corresponding to a specific function or instruction in Arduino, so that users can easily understand and use it.



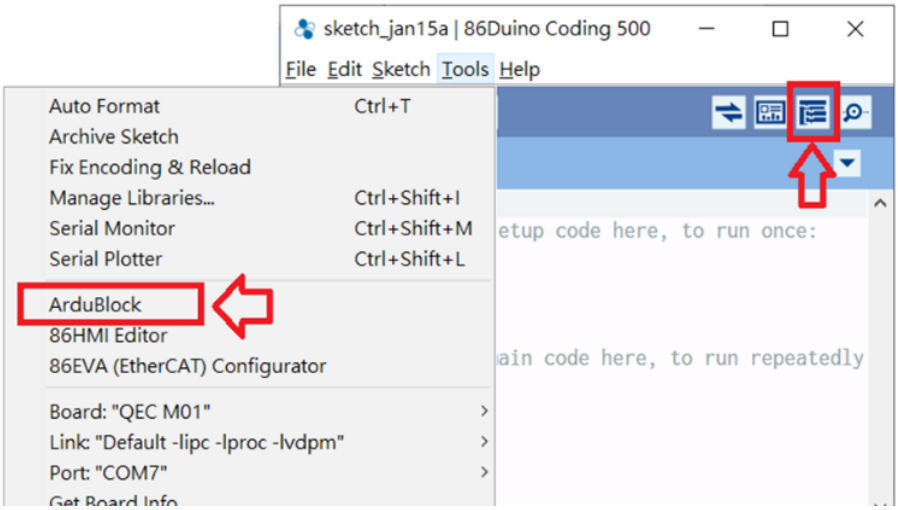
This document will provide an in-depth look at the EtherCAT function of this powerful tool so that you can understand its various functions and features.

4.2 Using ArduBlock

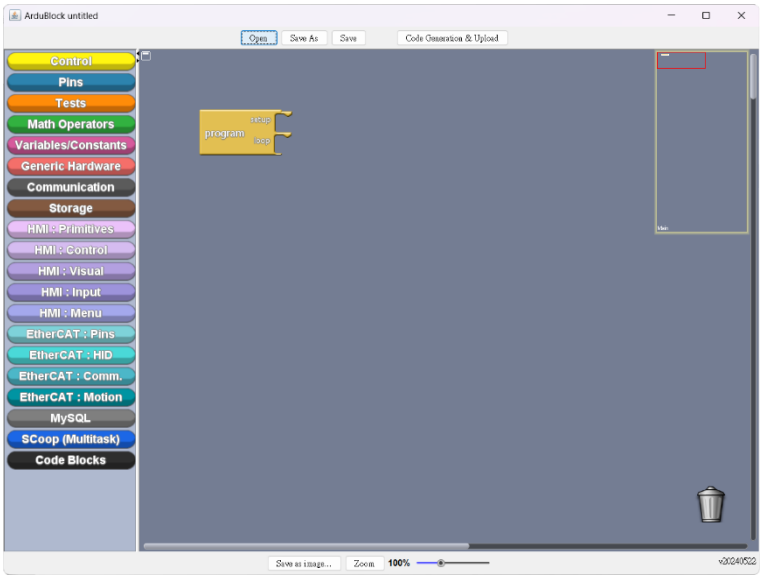
In this section, we will quickly introduce how to use the ArduBlock tool.

4.2.1 Start the ArduBlock

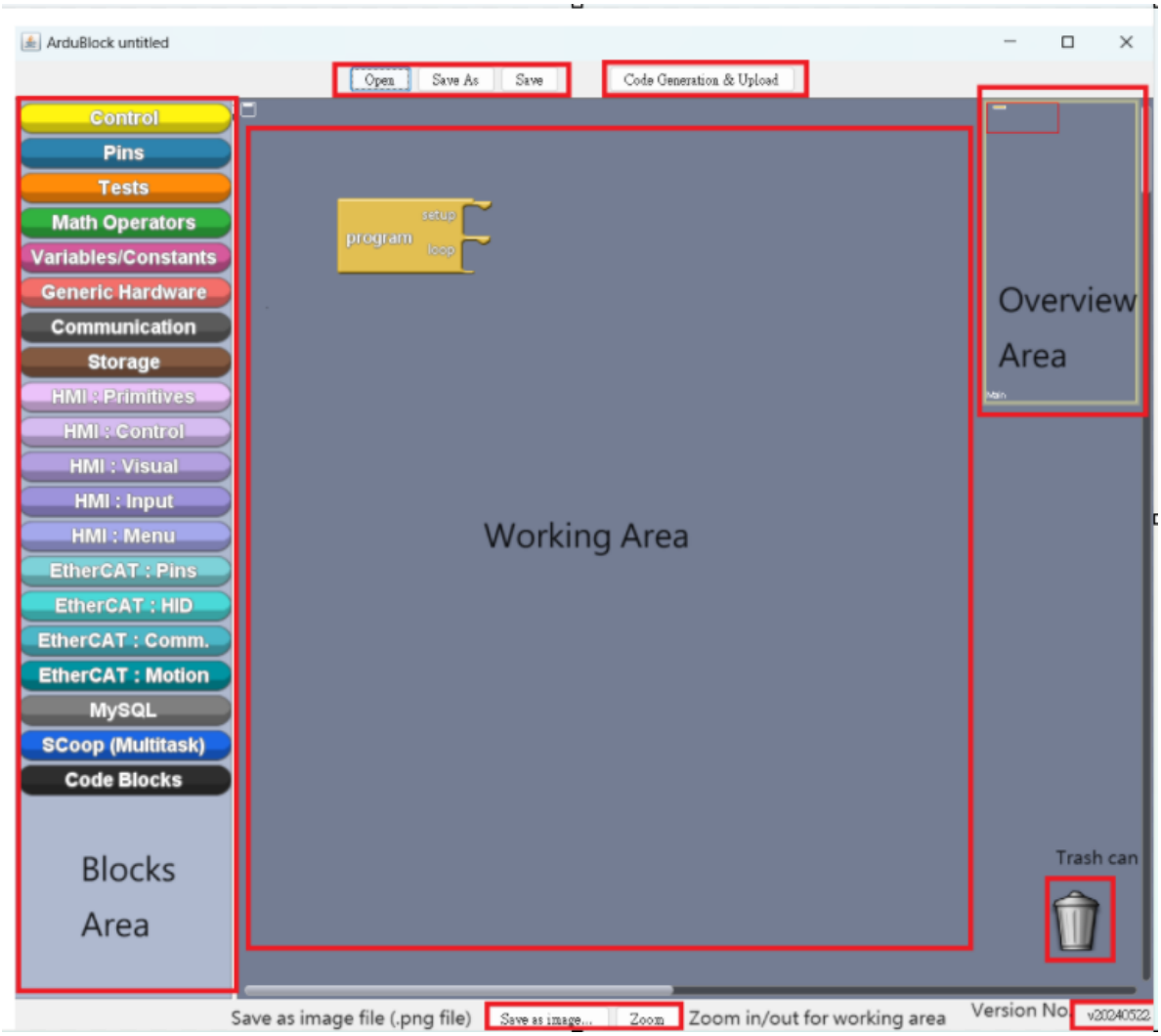
There are two ways to start ArduBlock:
Go to **Tools** -> **ArduBlock** or Click the icon as shown below.



Once clicked, the ArduBlock will start.



4.2.2 Interface introduction



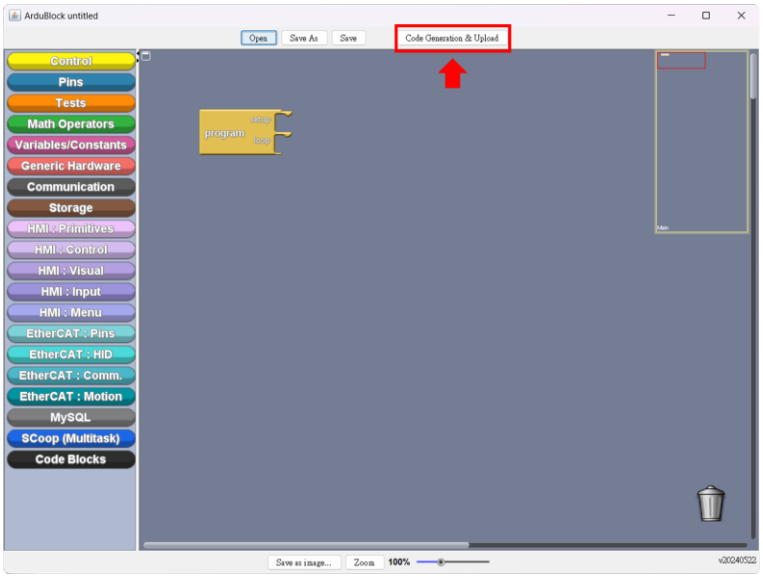
In Blocks Area, the building blocks including the types as below:

1. Control
2. Pins
3. Test
4. Math Operators
5. Variables/Constants
6. Generic Hardware
7. Communication
8. Storage
9. 86HMI
10. EtherCAT
11. MySQL
12. Scoop
13. Code Blocks

In this manual, we'll focus on **EtherCAT** Classes.

4.2.3 Finished, code generation and upload

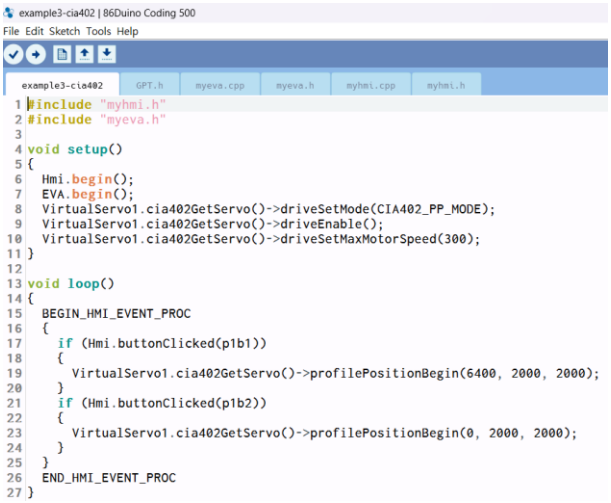
After finishing the ArduBlock program, use the "**Code Generation & Upload**" button to generate and upload the code to the QEC MDevice.



Example: CiA402 control example for EtherCAT:

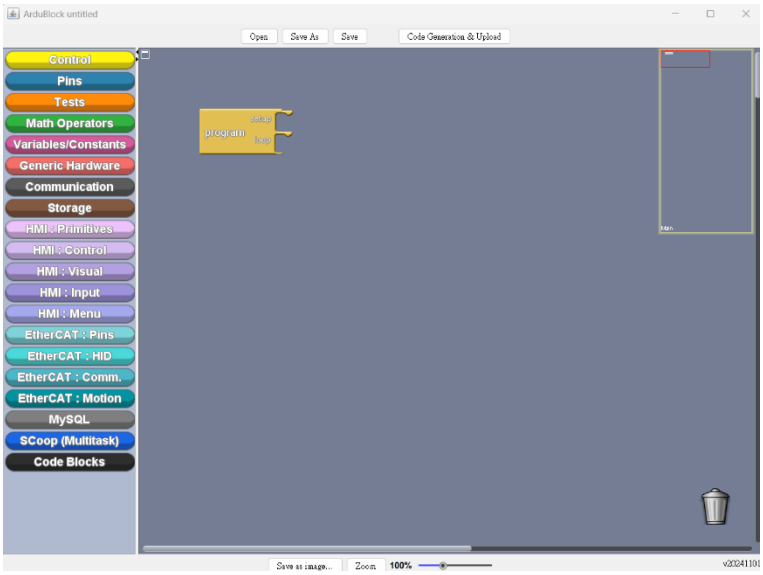


The generated code is as follows:

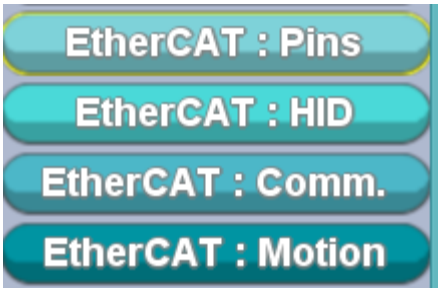


4.3 EtherCAT Class Blocks

The EtherCAT Class Blocks in ArduBlock provide a user-friendly, graphical way to access and control EtherCAT devices through the QEC platform.

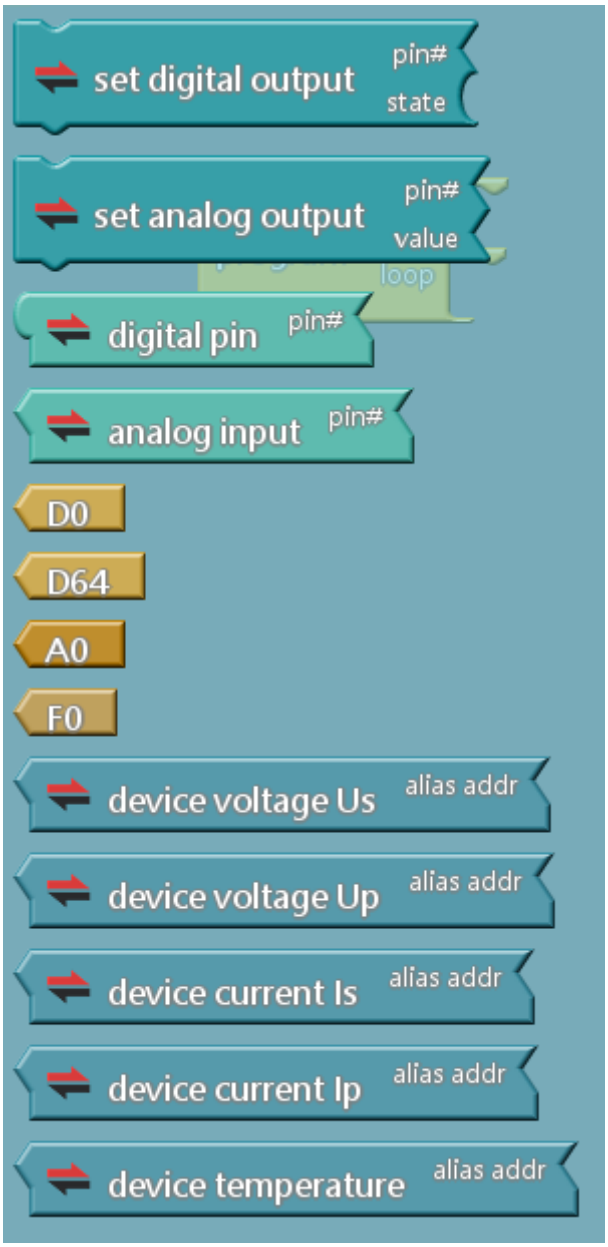


These blocks are categorized into four main groups—Pins, HID, Comm., and Motion—each designed to handle specific types of I/O or device communication.



This chapter will give you a comparison table of the blocks and the APIs.

4.3.1 Pins



API Reference Table:

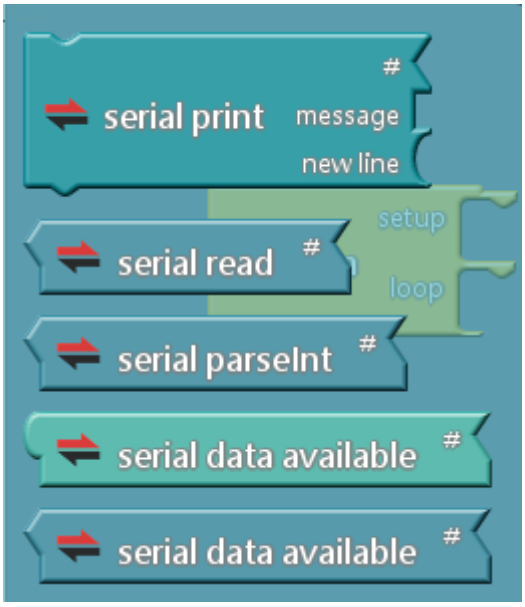
Block Name	API
set digital output	EVA.digitalWrite()
set analog output	EVA.analogWrite()
digital pin	EVA.digitalRead()
analog input	EVA.analogRead()
device voltage Us	EVA.getUsVoltage()
device voltage Up	EVA.getUpVoltage()
device current Is	EVA.getIsCurrent()
device current Ip	EVA.getIpCurrent()
device temperature	EVA.getTemperature()

4.3.2 HID



Block Name	API
LCM print	MyVirtualLcm.print()
clear LCM screen	MyVirtualLcm.clear()
get Keypad key	MyVirtualKeypad.waitForKey()
Tone	EVA.tone()
MPG read	MyVirtualMpg.read()
MPG write	MyVirtualMpg.write()
MPG axis	MyVirtualMpg.currentAxis()
MPG ratio	MyVirtualMpg.currentRatio()
if MPG enable switch is pressed	MyVirtualMpg.isEnabled()
if MPG emergency stop button is pressed	MyVirtualMpg.isEmergencyStop()
invert MPG counter direction	MyVirtualMpg.invertDirection()
set MPG digital filter	MyVirtualMpg.setDigitalFilter()
Barcode scanner setup	BarcodeScanner.begin()
Barcode read line	BarcodeScanner.readLine()
if Barcode scanner detects Barcode	BarcodeScanner.available()

4.3.3 Comm.



Block Name	API
serial print	`MyVirtualSerial.print()`
serial read	`MyVirtualSerial.read()`
serial parseInt	`MyVirtualSerial.read()` + manual `toInt()`
serial data available	`MyVirtualSerial.available()`

4.3.4 Motion



Block Name	API
send G-code	MyVirtualCNC.gcode()
servo setup	MyVirtualServo.setVelocity() MyVirtualServo.setAcceleration() MyVirtualServo.setPositionType()
move to	MyVirtualServo.write()
stop	MyVirtualServo.stop()
is moving	MyVirtualServo.isMoving()
go home	MyVirtualServo.home()
read (encoder)	MyVirtualEncoder.read()
write (encoder)	MyVirtualEncoder.write()
read direction	MyVirtualEncoder.directionRead()
set mode (encoder)	MyVirtualEncoder.setMode()
enable index reset	MyVirtualEncoder.setIndexReset()
set digital filter of input pins	MyVirtualEncoder.setDigitalFilter()
set polarity of input pins	MyVirtualEncoder.setInputPolarity()
set counter range	MyVirtualEncoder.setRange()

Appendix

A.1 Error List

For most functions, a returned value less than zero indicates an error, and the value represents an error code. If there is an error code, you can find the error cause and corrective actions below.

Definition	Code
ECAT_SUCCESS	0
ECAT_ERR_MODULE_INIT_FAIL	-100
ECAT_ERR_MODULE_GET_VERSION_FAIL	-101
ECAT_ERR_MODULE_VERSION_MISMATCH	-102
ECAT_ERR_MODULE_GENERIC_TRANSFER_INIT_FAIL	-103
ECAT_ERR_MASTER_DOWNLOAD_SETTINGS_FAIL	-200
ECAT_ERR_MASTER_SET_DEVICE_SETTINGS_FAIL	-201
ECAT_ERR_MASTER_GET_GROUP_INFO_FAIL	-202
ECAT_ERR_MASTER_GET_MASTER_INFO_FAIL	-203
ECAT_ERR_MASTER_GET_DEVICE_INFO_FAIL	-204
ECAT_ERR_MASTER_SET_GROUP_SETTINGS_FAIL	-205
ECAT_ERR_MASTER_MAPPING_INIT_FAIL	-206
ECAT_ERR_MASTER_INTERRUPT_INIT_FAIL	-207
ECAT_ERR_MASTER_ACTIVE_FAIL	-208
ECAT_ERR_MASTER_ENI_INITCMDS_FAIL	-209
ECAT_ERR_MASTER_NO_DEVICE	-210
ECAT_ERR_MASTER_ACYCLIC_INIT_FAIL	-300
ECAT_ERR_MASTER_ACYCLIC_REQUEST_FAIL	-301
ECAT_ERR_MASTER_ACYCLIC_BUSY	-302
ECAT_ERR_MASTER_ACYCLIC_TIMEOUT	-303
ECAT_ERR_MASTER_ACYCLIC_ERROR	-304
ECAT_ERR_MASTER_ACYCLIC_WRONG_STATUS	-405
ECAT_ERR_MASTER_GENERIC_SEND_FAIL	-400
ECAT_ERR_MASTER_GENERIC_RECV_FAIL	-401
ECAT_ERR_MASTER_NOT_BEGIN	-1000
ECAT_ERR_MASTER_WRONG_BUFFER_SIZE	-1001
ECAT_ERR_MASTER_REDUNDANCY_NO_DC	-1002
ECAT_ERR_MASTER_MEMORY_ALLOCATION_FAIL	-1003
ECAT_ERR_MASTER_OSLAYER_INIT_FAIL	-1004
ECAT_ERR_MASTER_NIC_INIT_FAIL	-1005
ECAT_ERR_MASTER_BASE_INIT_FAIL	-1006
ECAT_ERR_MASTER_CIA402_INIT_FAIL	-1007
ECAT_ERR_MASTER_SETUP_PDO_FAIL	-1008

ECAT_ERR_MASTER_SCAN_NETWORK_FAIL	-1009
ECAT_ERR_MASTER_START_MASTER_FAIL	-1010
ECAT_ERR_MASTER_CYCLETIME_TOO_SMALL	-1011
ECAT_ERR_MASTER_DUMP_OUTPUT_PDO_FAIL	-1012
ECAT_ERR_MASTER_CONFIG_DEVICE_FAIL	-1013
ECAT_ERR_MASTER_CONFIG_MAPPING_FAIL	-1014
ECAT_ERR_MASTER_WAIT_BUS_SYNC_TIMEOUT	-1015
ECAT_ERR_MASTER_WAIT_MASTER_SYNC_TIMEOUT	-1016
ECAT_ERR_MASTER_CYCLIC_START_FAIL	-1017
ECAT_ERR_MASTER_WRONG_BUFFER_POINTER	-1018
ECAT_ERR_MASTER_ENI_INIT_FAIL	-1050
ECAT_ERR_MASTER_ENI_MISMATCH	-1051
ECAT_ERR_MASTER_STOPPED	-1100
ECAT_ERR_MASTER_STARTED	-1101
ECAT_ERR_MASTER_NOT_IN_PREOP	-1102
ECAT_ERR_MASTER_NOT_IN_SAFEOP	-1103
ECAT_ERR_MASTER_NOT_IN_OP	-1104
ECAT_ERR_MASTER_II_TRANSITION_FAIL	-1200
ECAT_ERR_MASTER_IP_TRANSITION_FAIL	-1201
ECAT_ERR_MASTER_PS_TRANSITION_FAIL	-1202
ECAT_ERR_MASTER_SO_TRANSITION_FAIL	-1203
ECAT_ERR_DEVICE_NOT_EXIST	-2000
ECAT_ERR_DEVICE_NOT_ATTACH	-2001
ECAT_ERR_DEVICE_NO_MAILBOX	-2002
ECAT_ERR_DEVICE_NO_DC	-2003
ECAT_ERR_DEVICE_WRONG_INPUT	-2004
ECAT_ERR_DEVICE_MEMORY_ALLOCATION_FAIL	-2005
ECAT_ERR_DEVICE_VENDOR_ID_MISMATCH	-2006
ECAT_ERR_DEVICE_PRODUCT_CODE_MISMATCH	-2007
ECAT_ERR_DEVICE_NO_SUCH_FUNCTION	-2008
ECAT_ERR_DEVICE_FUNCTION_NOT_INIT	-2009
ECAT_ERR_DEVICE_BUSY	-2010
ECAT_ERR_DEVICE_TIMEOUT	-2011
ECAT_ERR_DEVICE_NO_DATA	-2012
ECAT_ERR_DEVICE_CANNOT_BE_CALLED_IN_CALLBACK	-2013
ECAT_ERR_DEVICE_ENTER_BOOT_FAIL	-2014
ECAT_ERR_DEVICE_EXIT_BOOT_FAIL	-2015
ECAT_ERR_DEVICE_BOOT_STATE_NOT_ENTERED	-2016
ECAT_ERR_DEVICE_BOOT_STATE_ENTERED	-2017
ECAT_ERR_DEVICE_SII_READ_FAIL	-2100
ECAT_ERR_DEVICE_SII_WRITE_FAIL	-2101

ECAT_ERR_DEVICE_SII_BUSY	-2102
ECAT_ERR_DEVICE_SII_REQUEST_FAIL	-2103
ECAT_ERR_DEVICE_SII_TIMEOUT	-2104
ECAT_ERR_DEVICE_SII_ERROR	-2105
ECAT_ERR_DEVICE_ESC_BUSY	-2150
ECAT_ERR_DEVICE_ESC_REQUEST_FAIL	-2151
ECAT_ERR_DEVICE_ESC_TIMEOUT	-2152
ECAT_ERR_DEVICE_ESC_ERROR	-2153
ECAT_ERR_DEVICE_PDO_NOT_EXIST	-2200
ECAT_ERR_DEVICE_PDO_OUT_OF_RANGE	-2201
ECAT_ERR_DEVICE_FOE_NOT_SUPPORT	-2300
ECAT_ERR_DEVICE_FOE_REQUEST_FAIL	-2310
ECAT_ERR_DEVICE_FOE_TIMEOUT	-2311
ECAT_ERR_DEVICE_FOE_ERROR	-2312
ECAT_ERR_DEVICE_FOE_BUFFER_TOO_SMALL	-2313
ECAT_ERR_DEVICE_FOE_READ_FAIL	-2314
ECAT_ERR_DEVICE_FOE_WRITE_FAIL	-2315
ECAT_ERR_DEVICE_FOE_BUSY	-2316
ECAT_ERR_DEVICE_FOE_PROTOCOL_ERROR	-2317
ECAT_ERR_DEVICE_COE_SDO_NOT_SUPPORT	-2400
ECAT_ERR_DEVICE_COE_SDO_INFO_NOT_SUPPORT	-2401
ECAT_ERR_DEVICE_COE_BUSY	-2410
ECAT_ERR_DEVICE_COE_REQUEST_FAIL	-2411
ECAT_ERR_DEVICE_COE_TIMEOUT	-2412
ECAT_ERR_DEVICE_COE_ERROR	-2413
ECAT_ERR_DEVICE_CIA402_NOT_EXIST	-2500
ECAT_ERR_DEVICE_CIA402_ADD_FAIL	-2501
ECAT_ERR_DEVICE_CIA402_TYPE_MISMATCH	-2502
ECAT_ERR_DEVICE_CIA402_NO_MODE_SUPPORT	-2503
ECAT_ERR_DEVICE_CIA402_WRONG_MODE	-2504
ECAT_ERR_DEVICE_CIA402_MODE_NOT_SUPPORT	-2505
ECAT_ERR_DEVICE_CIA402_CHANGE_WRONG_STATE	-2506
ECAT_ERR_DEVICE_CIA402_WRITE_OBJECT_FAIL	-2507
ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE	-2580
ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE_SOURCE	-2581
ECAT_ERR_DEVICE_EOE_NOT_SUPPORT	-2600
ECAT_ERR_DEVICE_EOE_NO_SUCH_PORT	-2601
ECAT_ERR_DEVICE_EOE_TOO_MUCH_CONTENT	-2602
ECAT_ERR_DEVICE_EOE_BUSY	-2610
ECAT_ERR_DEVICE_EOE_REQUEST_FAIL	-2611
ECAT_ERR_DEVICE_EOE_TIMEOUT	-2612

ECAT_ERR_GROUP_WRONG_INPUT	-3000
ECAT_ERR_GROUP_NOT_ATTACH	-3001

***Note:** EVA.begin() may also return EVAERR_NO_CONFIG_FILE (-1000) when the 86EVA configuration file is not found (e.g., Code Generation or Upload has not been performed), and EVAERR_NO_ENOUGH_MEMORY (-1001) when memory allocation fails.

A.2 Error Description and Corrective Actions

0: ECAT_SUCCESS

Description

Function calls successful.

Corrective Actions

Nothing to do.

-100: ECAT_ERR_MODULE_INIT_FAIL

Description

EtherCAT firmware initialization error.

Corrective Actions

Please contact the manufacturer.

-101: ECAT_ERR_MODULE_GET_VERSION_FAIL

Description

The command to obtain the EtherCAT firmware version encountered an error.

Corrective Actions

Please contact the manufacturer.

-102: ECAT_ERR_MODULE_VERSION_MISMATCH

Description

The EtherCAT firmware version does not match the EtherCAT library version.

Corrective Actions

Please update the EtherCAT firmware. If this issue persists, please contact the manufacturer.

-103: ECAT_ERR_MODULE_GENERIC_TRANSFER_INIT_FAIL

Description

General transfer interface initialization between dual systems failed.

Corrective Actions

Please contact the manufacturer.

-200: ECAT_ERR_MASTER_DOWNLOAD_SETTINGS_FAIL

Description

The command to set the configuration of the EtherCAT MDevice encountered an error.

Corrective Actions

Please contact the manufacturer.

-201: ECAT_ERR_MASTER_SET_DEVICE_SETTINGS_FAIL

Description

The command to set the configuration of the EtherCAT SubDevices encountered an error.

Corrective Actions

Please contact the manufacturer.

-202: ECAT_ERR_MASTER_GET_GROUP_INFO_FAIL

Description

The command to get the configuration of the EtherCAT group encountered an error.

Corrective Actions

Please contact the manufacturer.

-203: ECAT_ERR_MASTER_GET_MASTER_INFO_FAIL

Description

The command to get the configuration of the EtherCAT MDevice encountered an error.

Corrective Actions

Please contact the manufacturer.

-204: ECAT_ERR_MASTER_GET_DEVICE_INFO_FAIL

Description

The command to get the configuration of the EtherCAT SubDevices encountered an error.

Corrective Actions

Please contact the manufacturer.

-205: ECAT_ERR_MASTER_SET_GROUP_SETTINGS_FAIL

Description

The command to set the configuration of the EtherCAT group encountered an error.

Corrective Actions

Please contact the manufacturer.

-206: ECAT_ERR_MASTER_MAPPING_INIT_FAIL

Description

Failed to allocate memory for PDO mapping.

Corrective Actions

Please contact the manufacturer.

-207: ECAT_ERR_MASTER_INTERRUPT_INIT_FAIL

Description

Initialization of the interrupt function failed.

Corrective Actions

Please contact the manufacturer.

-208: ECAT_ERR_MASTER_ACTIVE_FAIL

Description

The command to activate the EtherCAT MDevice failed.

Corrective Actions

Please contact the manufacturer.

-209: ECAT_ERR_MASTER_ENI_INITCMDS_FAIL

Description

The execution of the SDO Download command in the ENI file failed.

Corrective Actions

Please verify the correctness of the ENI file content and ensure that all SubDevices are functioning properly.

-210: ECAT_ERR_MASTER_NO_DEVICE

Description

EtherCAT network has no SubDevices.

Corrective Actions

Please connect at least one EtherCAT SubDevice.

-300: ECAT_ERR_MASTER_ACYCLIC_INIT_FAIL

Description

Acyclic transfer interface initialization between dual systems failed.

Corrective Actions

Please contact the manufacturer.

-301: ECAT_ERR_MASTER_ACYCLIC_REQUEST_FAIL

Description

Failed to request acyclic transfer.

Corrective Actions

Please try again later.

-302: ECAT_ERR_MASTER_ACYCLIC_BUSY

Description

Acyclic transfer is busy.

Corrective Actions

Please try again later.

-303: ECAT_ERR_MASTER_ACYCLIC_TIMEOUT

Description

Acyclic transfer timed out.

Corrective Actions

Please try again later or confirm that the SubDevice is functioning properly.

-304: ECAT_ERR_MASTER_ACYCLIC_ERROR

Description

Acyclic transfer encountered an error.

Corrective Actions

Please check the error code. Such as the CoE communication, you can check Abort Code.

-405: ECAT_ERR_MASTER_ACYCLIC_WRONG_STATUS

Description

Acyclic transfer obtained an invalid status.

Corrective Actions

Please contact the manufacturer.

-400: ECAT_ERR_MASTER_GENERIC_SEND_FAIL

Description

Failed to send data during generic transmission.

Corrective Actions

Please contact the manufacturer.

-401: ECAT_ERR_MASTER_GENERIC_RECV_FAIL

Description

Failed to receive data during generic transmission.

Corrective Actions

Please contact the manufacturer.

-1000: ECAT_ERR_MASTER_NOT_BEGIN

Description

The EtherCAT MDevice has not been initialized.

Corrective Actions

Please call `EthercatMaster::begin()`.

-1001: ECAT_ERR_MASTER_WRONG_BUFFER_SIZE

Description

The size of the input buffer is incorrect.

Corrective Actions

Please input a buffer with correct size.

-1002: ECAT_ERR_MASTER_REDUNDANCY_NO_DC

Description

Cable Redundancy mode does not support DC (Distributed Clocks).

Corrective Actions

Do not call this function or avoid Cable Redundancy mode.

-1003: ECAT_ERR_MASTER_MEMORY_ALLOCATION_FAIL

Description

Failed to allocate memory.

Corrective Actions

Please contact the manufacturer.

-1004: ECAT_ERR_MASTER_OSLAYER_INIT_FAIL

Description

Operating system layer initialization failed.

Corrective Actions

Please contact the manufacturer.

-1005: ECAT_ERR_MASTER_NIC_INIT_FAIL

Description

Network controller initialization failed.

Corrective Actions

Please ensure the network controller is present or contact the manufacturer.

-1006: ECAT_ERR_MASTER_BASE_INIT_FAIL

Description

Initialization of EtherCAT MDevice command interface failed.

Corrective Actions

Please contact the manufacturer.

-1007: ECAT_ERR_MASTER_CIA402_INIT_FAIL

Description

Initialization of EtherCAT MDevice CiA 402 framework failed.

Corrective Actions

Please contact the manufacturer.

-1008: ECAT_ERR_MASTER_SETUP_PDO_FAIL

Description

Configuration of PDO mapping for the SubDevice using SDO Download failed.

Corrective Actions

Please ensure that all SubDevices are functioning properly. If the issue is persisted, please contact the manufacturer.

-1009: ECAT_ERR_MASTER_SCAN_NETWORK_FAIL

Description

Failed to scan EtherCAT network.

Corrective Actions

Please connect the network cable properly and ensure that EtherCAT SubDevices are powered on, or verify the presence of the network controller.

-1010: ECAT_ERR_MASTER_START_MASTER_FAIL

Description

Failed to start EtherCAT MDevice.

Corrective Actions

Please ensure that the configuration of PDO mapping for all SubDevices is correct. If the issue is persisted, please contact the manufacturer.

-1011: ECAT_ERR_MASTER_CYCLETIME_TOO_SMALL

Description

The input cycle time is too small.

Corrective Actions

Please try increasing the cycle time.

-1012: ECAT_ERR_MASTER_DUMP_OUTPUT_PDO_FAIL

Description

Failed to update output process data using SDO Upload.

Corrective Actions

Please ensure that all SubDevices are functioning correctly, and verify that the configuration of PDO mapping is correct.

-1013: ECAT_ERR_MASTER_CONFIG_DEVICE_FAIL

Description

EtherCAT SubDevice initialization failed.

Corrective Actions

Please contact the manufacturer.

-1014: ECAT_ERR_MASTER_CONFIG_MAPPING_FAIL

Description

Failed to create PDO mapping.

Corrective Actions

Please contact the manufacturer.

-1015: ECAT_ERR_MASTER_WAIT_BUS_SYNC_TIMEOUT

Description

Synchronization timeout for all SubDevices.

Corrective Actions

Please contact the manufacturer.

-1016: ECAT_ERR_MASTER_WAIT_MASTER_SYNC_TIMEOUT

Description

Synchronization timeout between MDevice and SubDevices.

Corrective Actions

Please contact the manufacturer.

-1017: ECAT_ERR_MASTER_CYCLIC_START_FAIL

Description

Failed to start cyclic transmission.

Corrective Actions
Please contact the manufacturer.

-1018: ECAT_ERR_MASTER_WRONG_BUFFER_POINTER

Description
Incorrect data buffer pointer.
Corrective Actions
Please input the correct data buffer pointer.

-1050: ECAT_ERR_MASTER_ENI_INIT_FAIL

Description
ENI initialization failed.
Corrective Actions
Please confirm that the specified ENI file exists. If this issue persists, please contact the manufacturer.

-1051: ECAT_ERR_MASTER_ENI_MISMATCH

Description
The SubDevice information in the ENI file does not match the scanned EtherCAT network SubDevices.
Corrective Actions
Please adjust the order of the SubDevices on the EtherCAT network or configure the SubDevices identifications.

-1100: ECAT_ERR_MASTER_STOPPED

Description
EtherCAT MDevice has not been started.
Corrective Actions
Please call `EthercatMaster::start()` or avoid calling this function.

-1101: ECAT_ERR_MASTER_STARTED

Description
EtherCAT MDevice has already been started.
Corrective Actions
Please call `EthercatMaster::stop()` or avoid calling this function.

-1102: ECAT_ERR_MASTER_NOT_IN_PREOP

Description

EtherCAT MDevice is not in PRE-OP state.

Corrective Actions

Please ensure that all SubDevices are in PRE-OP state before calling this function.

-1103: ECAT_ERR_MASTER_NOT_IN_SAFEOP

Description

EtherCAT MDevice is not in SAFE-OP state.

Corrective Actions

Please ensure that all SubDevices are in SAFE-OP state before calling this function.

-1104: ECAT_ERR_MASTER_NOT_IN_OP

Description

EtherCAT MDevice is not in OP state.

Corrective Actions

Please ensure that all SubDevices are in OP state before calling this function.

-1200: ECAT_ERR_MASTER_II_TRANSITION_FAIL

Description

Switching all SubDevices to INIT state during EtherCAT MDevice initialization failed.

Corrective Actions

Please power cycle all SubDevices and then run the EtherCAT MDevice program again.

-1201: ECAT_ERR_MASTER_IP_TRANSITION_FAIL

Description

Failed to transition EtherCAT state from INIT to PRE-OP.

Corrective Actions

Please ensure that all SubDevices are functioning properly, or check the quality of the network connections.

-1202: ECAT_ERR_MASTER_PS_TRANSITION_FAIL

Description

Failed to transition EtherCAT state from PRE-OP to SAFE-OP.

Corrective Actions

Please check the correctness of the PDO mapping configuration for all SubDevices. If DC synchronization is enabled, try adjusting the related parameters for DC synchronization.

-1203: ECAT_ERR_MASTER_SO_TRANSITION_FAIL

Description

Failed to transition EtherCAT state from SAFE-OP to OP.

Corrective Actions

Please try adjusting the related parameters for DC synchronization and test again.

-2000: ECAT_ERR_DEVICE_NOT_EXIST

Description

The specified SubDevice does not exist.

Corrective Actions

Please do not access the specified SubDevice.

-2001: ECAT_ERR_DEVICE_NOT_ATTACH

Description

The object of such SubDevice has not been initialized.

Corrective Actions

Please call `attach()` to initialize the SubDevice object.

-2002: ECAT_ERR_DEVICE_NO_MAILBOX

Description

The SubDevice does not support Mailbox.

Corrective Actions

Please do not call Mailbox-related functions for the SubDevice.

-2003: ECAT_ERR_DEVICE_NO_DC

Description

The SubDevice does not support DC synchronization.

Corrective Actions

Please do not call DC-related functions for the SubDevice.

-2004: ECAT_ERR_DEVICE_WRONG_INPUT

Description

Incorrect input parameter.

Corrective Actions

Please input the correct parameter.

-2005: ECAT_ERR_DEVICE_MEMORY_ALLOCATION_FAIL

Description

Failed to allocate memory for the SubDevice object.

Corrective Actions

Please contact the manufacturer.

-2006: ECAT_ERR_DEVICE_VENDOR_ID_MISMATCH

Description

The Vendor ID of the SubDevice does not match that of the SubDevice object.

Corrective Actions

Please use the correct SubDevice object.

-2007: ECAT_ERR_DEVICE_PRODUCT_CODE_MISMATCH

Description

The Product Code of the SubDevice does not match that of the SubDevice object.

Corrective Actions

Please use the correct SubDevice object.

-2008: ECAT_ERR_DEVICE_NO_SUCH_FUNCTION

Description

The SubDevice does not support this feature.

Corrective Actions

Please do not call this function.

-2009: ECAT_ERR_DEVICE_FUNCTION_NOT_INIT

Description

The specific function of the SubDevice has not been initialized.

Corrective Actions

Please initialize that function.

-2010: ECAT_ERR_DEVICE_BUSY

Description

The SubDevice is busy.

Corrective Actions

Please try again later.

-2011: ECAT_ERR_DEVICE_TIMEOUT

Description

The SubDevice has encountered a timeout.

Corrective Actions

Please try again later or confirm that the SubDevice is functioning properly.

-2012: ECAT_ERR_DEVICE_NO_DATA

Description

The SubDevice has no available data.

Corrective Actions

Please try again later.

-2013: ECAT_ERR_DEVICE_CANNOT_BE_CALLED_IN_CALLBACK

Description

This function cannot be called in the cyclic callback function.

Corrective Actions

Please call this function in setup() or loop() instead of the cyclic callback function.

-2014: ECAT_ERR_DEVICE_ENTER_BOOT_FAIL

Description

Failed to switch the SubDevice to the BOOT state.

Corrective Actions

Please confirm that the SubDevice supports the BOOT state and try again.

-2015: ECAT_ERR_DEVICE_EXIT_BOOT_FAIL

Description

Failed to switch the SubDevice out of the BOOT state.

Corrective Actions

Please restart the SubDevice and try again.

-2016: ECAT_ERR_DEVICE_BOOT_STATE_NOT_ENTERED

Description

The SubDevice is not in the BOOT state.

Corrective Actions

Please switch the SubDevice to the BOOT state before calling this function.

-2017: ECAT_ERR_DEVICE_BOOT_STATE_ENTERED

Description

The SubDevice is already in the BOOT state.

Corrective Actions

Please exit the BOOT state before calling this function.

-2100: ECAT_ERR_DEVICE_SII_READ_FAIL

Description

Failed to read the SII EEPROM of the SubDevice.

Corrective Actions

Please ensure that this function is called when the EtherCAT state of the SubDevice is INIT or PRE-OP.

-2101: ECAT_ERR_DEVICE_SII_WRITE_FAIL

Description

Failed to write the SII EEPROM of the SubDevice.

Corrective Actions

Please ensure that this function is called when the EtherCAT state of the SubDevice is INIT or PRE-OP.

-2102: ECAT_ERR_DEVICE_SII_BUSY

Description

SII EEPROM access is busy.

Corrective Actions

Please try again later.

-2103: ECAT_ERR_DEVICE_SII_REQUEST_FAIL

Description

Failed to request SII EEPROM access.

Corrective Actions

Please try again later.

-2104: ECAT_ERR_DEVICE_SII_TIMEOUT

Description

SII EEPROM access timeout.

Corrective Actions

Please try again later or confirm that the SubDevice is functioning properly.

-2105: ECAT_ERR_DEVICE_SII_ERROR

Description

SII EEPROM access encountered an error.

Corrective Actions

Please confirm that the SubDevice is functioning properly.

-2150: ECAT_ERR_DEVICE_ESC_BUSY

Description

ESC register access is busy.

Corrective Actions

Please try again later.

-2151: ECAT_ERR_DEVICE_ESC_REQUEST_FAIL

Description

Failed to request ESC register access.

Corrective Actions

Please try again later.

-2152: ECAT_ERR_DEVICE_ESC_TIMEOUT

Description

ESC register access timeout.

Corrective Actions

Please try again later or confirm that the SubDevice is functioning properly.

-2153: ECAT_ERR_DEVICE_ESC_ERROR

Description

ESC register access encountered an error.

Corrective Actions

Please confirm that the SubDevice is functioning properly.

-2200: ECAT_ERR_DEVICE_PDO_NOT_EXIST

Description

The SubDevice has no output process data.

Corrective Actions

Please do not call this function.

-2201: ECAT_ERR_DEVICE_PDO_OUT_OF_RANGE

Description

Accessing beyond the process data range of the SubDevice.

Corrective Actions

Please access the correct range of process data for the SubDevice.

-2300: ECAT_ERR_DEVICE_FOE_NOT_SUPPORT

Description

The SubDevice does not support FoE.

Corrective Actions

Please do not call FoE-related functions for the SubDevice.

-2310: ECAT_ERR_DEVICE_FOE_REQUEST_FAIL

Description

Failed to request FoE communication.

Corrective Actions

Please try again later.

-2311: ECAT_ERR_DEVICE_FOE_TIMEOUT

Description

FoE communication timeout.

Corrective Actions

Please verify that the SubDevice supports FoE and is functioning properly.

-2312: ECAT_ERR_DEVICE_FOE_ERROR

Description

FoE communication encountered an error.

Corrective Actions

Please verify that the SubDevice supports FoE and is functioning properly.

-2313: ECAT_ERR_DEVICE_FOE_BUFFER_TOO_SMALL

Description

The size of the input buffer for FoE communication is too small.

Corrective Actions

Please input a buffer with suitable size.

-2314: ECAT_ERR_DEVICE_FOE_READ_FAIL

Description

Failed to read the file via FoE communication.

Corrective Actions

Please verify that the SubDevice supports reading files via FoE communication.

-2315: ECAT_ERR_DEVICE_FOE_WRITE_FAIL

Description

Failed to write the file via FoE communication.

Corrective Actions

Please verify that the SubDevice supports writing files via FoE communication.

-2316: ECAT_ERR_DEVICE_FOE_BUSY

Description

FoE communication is busy.

Corrective Actions

Please try again later.

-2317: ECAT_ERR_DEVICE_FOE_PROTOCOL_ERROR

Description

A FoE protocol error occurred.

Corrective Actions

Please verify that the SubDevice supports FoE and that the file name and password are correct.

-2400: ECAT_ERR_DEVICE_COE_SDO_NOT_SUPPORT

Description

The SubDevice does not support SDO commands of CoE communication.

Corrective Actions

Please do not call functions related to SDO commands of CoE communication.

-2401: ECAT_ERR_DEVICE_COE_SDO_INFO_NOT_SUPPORT

Description

The SubDevice does not support SDO Information commands of CoE communication.

Corrective Actions

Please do not call functions related to SDO Information commands of CoE communication.

-2410: ECAT_ERR_DEVICE_COE_BUSY

Description

CoE communication is busy.

Corrective Actions

Please try again later.

-2411: ECAT_ERR_DEVICE_COE_REQUEST_FAIL

Description

Failed to request CoE communication.

Corrective Actions

Please try again later.

-2412: ECAT_ERR_DEVICE_COE_TIMEOUT

Description

CoE communication timeout.

Corrective Actions

Please try again later or confirm that the SubDevice is functioning properly.

-2413: ECAT_ERR_DEVICE_COE_ERROR

Description

CoE communication encountered an error.

Corrective Actions

Please check the abort code for CoE communication.

-2500: ECAT_ERR_DEVICE_CIA402_NOT_EXIST

Description

This SubDevice does not have objects related to CiA 402.

Corrective Actions

Please do not call functions related to CiA 402 for the SubDevice.

-2501: ECAT_ERR_DEVICE_CIA402_ADD_FAIL

Description

Failed to insert the CiA 402 SubDevice object to the CiA 402 framework of EtherCAT MDevice.

Corrective Actions

Please confirm whether the CiA 402 SubDevice object has been successfully inserted into the CiA 402 framework of the EtherCAT MDevice.

-2502: ECAT_ERR_DEVICE_CIA402_TYPE_MISMATCH

Description

The content of the object of Device Type (Index 1000h) for the SubDevice is not CiA 402 type.

Corrective Actions

Please do not call functions related to CiA 402 for the SubDevice.

-2503: ECAT_ERR_DEVICE_CIA402_NO_MODE_SUPPORT

Description

This CiA 402 SubDevice does not support any operation modes defined by CiA 402.

Corrective Actions

Please do not call functions related to CiA 402 for the SubDevice. Also, ensure that the CiA 402 SubDevice supports CiA 402.

-2504: ECAT_ERR_DEVICE_CIA402_WRONG_MODE

Description

This function cannot be called in the current CiA 402 operation mode for the SubDevice.

Corrective Actions

Please change the CiA 402 operation mode of this SubDevice to the correct mode before calling this function.

-2505: ECAT_ERR_DEVICE_CIA402_MODE_NOT_SUPPORT

Description

The SubDevice does not support the specified CiA 402 operation mode.

Corrective Actions

Please do not change the CiA 402 operation mode of this SubDevice to an unsupported mode.

-2506: ECAT_ERR_DEVICE_CIA402_CHANGE_WRONG_STATE

Description

The current CiA 402 state does not allow switching to the specified CiA 402 state.

Corrective Actions

Please check the current CiA 402 state. If it is in FAULT state, switch to SWITCH_ON_DISABLED state first, and then switch to the target state.

-2507: ECAT_ERR_DEVICE_CIA402_WRITE_OBJECT_FAIL

Description

Accessing CiA 402 objects is not allowed using SDO Upload or SDO Download in Cyclic Callback.

Corrective Actions

Please avoid using SDO Upload/Download to access CiA 402 objects in Cyclic Callback. If needed, map the CiA 402 objects to process data.

-2580: ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE

Description

The input number for the Touch Probe functionality is incorrect.

Corrective Actions

Please input the correct Touch Probe number, ranging from 0 to 1.

-2581: ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE_SOURCE

Description

The input signal source for the Touch Probe functionality is incorrect.

Corrective Actions

Please input the correct signal source, ranging from 0 to 2.

-2600: ECAT_ERR_DEVICE_EOE_NOT_SUPPORT

Description

The SubDevice does not support EoE.

Corrective Actions

Please do not call EoE-related functions for the SubDevice.

-2601: ECAT_ERR_DEVICE_EOE_NO_SUCH_PORT

Description

Incorrect EoE port number.

Corrective Actions

Please input the correct EoE port number.

-2602: ECAT_ERR_DEVICE_EOE_TOO_MUCH_CONTENT

Description

The input content is too much.

Corrective Actions

Please provide the correct content.

-2610: ECAT_ERR_DEVICE_EOE_BUSY

Description

EoE communication is busy.

Corrective Actions

Please try again later.

-2611: ECAT_ERR_DEVICE_EOE_REQUEST_FAIL

Description

Failed to request EoE communication.

Corrective Actions

Please try again later.

-2612: ECAT_ERR_DEVICE_EOE_TIMEOUT

Description

EoE communication timeout.

Corrective Actions

Please verify that the SubDevice supports EoE and is functioning properly.

-3000: ECAT_ERR_GROUP_WRONG_INPUT

Description

The input parameter is incorrect.

Corrective Actions

Please input the correct parameter.

-3001: ECAT_ERR_GROUP_NOT_ATTACH

Description

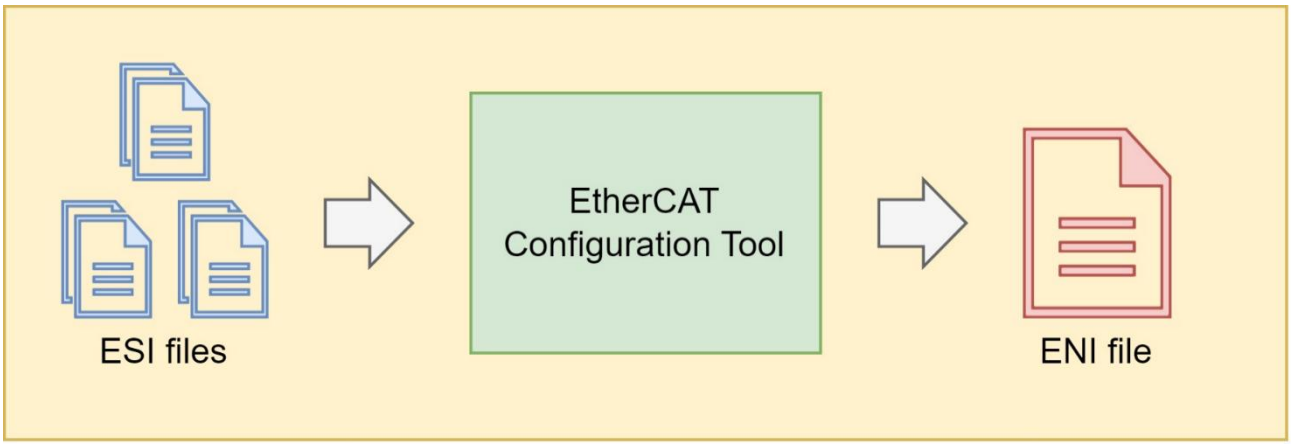
The object of such group has not been initialized.

Corrective Actions

Please initialize the group object.

A.3 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 : SubDevice : 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 : SubDevice : Mailbox

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

EtherCATConfig : Config : SubDevice : 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 : SubDevice : 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 : SubDevice : 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 : SubDevice : DC

- Elements
  - CycleTime0
  - CycleTime1
  - ShiftTime
- Purpose
  Used to configure the DC parameters of an EtherCAT SubDevice.

A.4 86EVA Troubleshooting

The following table lists the messages shown by 86EVA and the 86Duino IDE, and how to solve them. For error codes returned by the EVA API, see A.1 Error List.

Message / symptom	Possible cause	Action
IDE status bar: “Selected boards should be QEC series”	A non-QEC board is selected.	Select your QEC MDevice in “Tools” -> “Board”.
IDE status bar: “You can't open EtherCAT Configurator program under … example project”	The sketch is a built-in example.	Save the sketch to your own folder (“File” -> “Save As”) and open 86EVA again.
[ERROR] “Connect Fail! Check COM port and try again.”	Wrong COM port, USB cable not connected, QEC MDevice not powered, or driver not installed.	Check the port in Device Manager and the PL2303 driver (Ch. 2.2), then select the correct port.
[ERROR] “Can not connect to EtherCAT Master, please restart 86EVA Configurator”	The connection to the QEC MDevice was lost (USB unplugged, MDevice reset or powered off).	Reconnect the cable and power, close 86EVA, and open it again.
“No detect any EtherCAT slave device.”	No SubDevice answers: cable in the wrong port, IN/OUT reversed, or SubDevice not powered.	Connect the MDevice EtherCAT port to the IN port of the first SubDevice; check the 24 V supply and LEDs.
[ERROR] “Exist the same Alias Address … on Slave Device … and …” / “The alias ID conflict with the other device.”	Two SubDevices use the same Alias Address.	Set a unique Alias Address (Ch. 2.6.1.2) or use “Auto Set Alias Addresses” (Ch. 2.4).
“The alias ID is empty.” / “The alias ID must be number.” / “The alias ID value must be < 65536.”	Invalid Alias Address input.	Enter a number from 1 to 65535.
“The object name is empty.” / “Incorrect object name.” / “The object name conflict with …”	The Object Name is empty, is not a valid C++ identifier, or is already used.	Use a unique name with letters, digits, and underscores, starting with a letter.
[WARNING] “The virtual pin has already been mapped to other pin!”	The selected virtual pin is already used by another physical pin.	Select another virtual pin or set the other pin to “None”.
[ERROR] “… exists invalid value. (max value should be larger then min value)”	A minimum value (e.g., Soft Limit min) is larger than the maximum value.	Correct the range in Machine Configuration (Ch. 2.6.4.2).
“Can not generate code to	The IDE window was	Keep the IDE open, restart

86Duino IDE, please restart 86EVA Configurator”	closed or the IDE connection was lost.	86EVA, and click “Code Generation” again.
“A new 86EVA firmware be detected. Please update it by IDE.” / “Please update QEC bootloader.”	The 86EVA firmware or bootloader on the QEC MDevice is older than the IDE.	Update the QEC MDevice from the 86Duino IDE (refer to the QEC MDevice user manual).
Settings are not applied after closing 86EVA	Code Generation or Upload was not performed, or the SubDevice was not restarted.	See Ch. 2.3.3 When Do Settings Take Effect?
A SubDevice is shown as “Unsupported”	The SubDevice is not in the 86EVA device list.	See Ch. 2.6.5; use the EtherCAT Library API or an ENI file.

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.

All Trademarks appearing in this manuscript are registered trademark of their respective owners. All Specifications are subject to change without notice.

©ICOP Technology Inc. 2026