User Manual

EtherCAT Master API

86Duino Coding IDE 500

EtherCAT Library

(Version2.2r1)

REVISION

DATE	VERSION	DESCRIPTION
2023/5/5	**Version1.0**	**New Release.**
2023/10/24	**Version2.0**	**Updated API.**
2024/8/1	**Version2.2**	**Updated API.**

COPYRIGHT

Ver.2.2r1 August, 2024

TRADEMARKS ACKNOWLEDGMENT

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

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

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

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

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

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

This Manual is for the QEC series.

SAFETY INFORMATION

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

WARNING!

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

Content

Ch. 1

Introduction

EtherCAT (Ethernet for Control Automation Technology) is a communication protocol designed specifically for industrial control applications. It is a high-performance, real-time communication technology with extremely low communication latency and high bandwidth, suitable for various industrial automation and control systems.

Key features of EtherCAT include:

- **High-speed Real-time Performance**: EtherCAT can transmit data at very high speeds while maintaining extremely low communication latency, enabling high levels of real-time performance suitable for demanding control applications.
- **Flexible Scalability**: EtherCAT networks can easily scale to support large numbers of nodes and complex network topologies, making them suitable for various scales and application scenarios.
- **Open Standard**: EtherCAT is an open standard with widespread industrial support and application. It has rich libraries and tools that can be easily integrated into existing industrial automation systems.
- **Cost-effectiveness**: EtherCAT uses standard Ethernet hardware without the need for additional specialized hardware, reducing costs and deployment complexity.

EtherCAT is commonly used in industrial automation, robotics control, motion control, embedded systems, and real-time data acquisition, providing high-performance and reliable communication solutions for these applications.

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.

1.1 About QEC Master

QEC Master is an EtherCAT Master System compatible with 86Duino Coding IDE 500+. It offers real-time EtherCAT communication between EtherCAT Master and EtherCAT slave devices. Except for the EtherCAT Library of 86Duino IDE, QEC Master 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.

1.1.1 What is 86Duino IDE?

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

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

- EtherCAT slave devices scanning
- Import ENI file
- Setting EtherCAT Master
- Configure EtherCAT slave devices

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

1.1.2 QEC EtherCAT Master Architecture

The EtherCAT Master 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 Master 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.
    - Slave Information Interface (SII) access interface.
    - Distributed Clocks (DC) access interface.
- **EtherCAT Master Firmware**
  - Executes the EtherCAT Master Core.
  - Controls the Primary/Secondary Ethernet Driver, sending EtherCAT frames

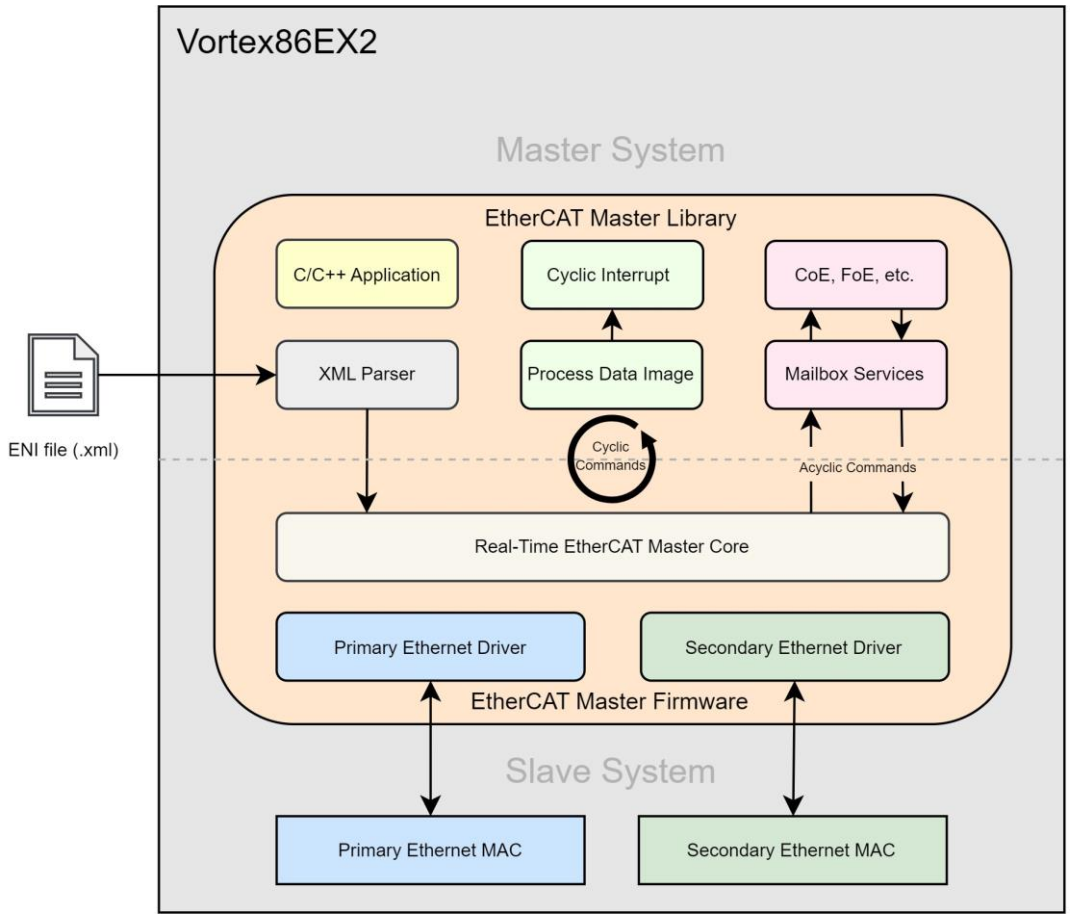
The programs are designed to run on the FreeDOS operating system and have been compiled using the GCC compiler provided by the DJGPP environment.

1.1.3 Hardware Platform

The EtherCAT Master 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 Master Firmware.

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



1.1.4 Dual-System Synchronization

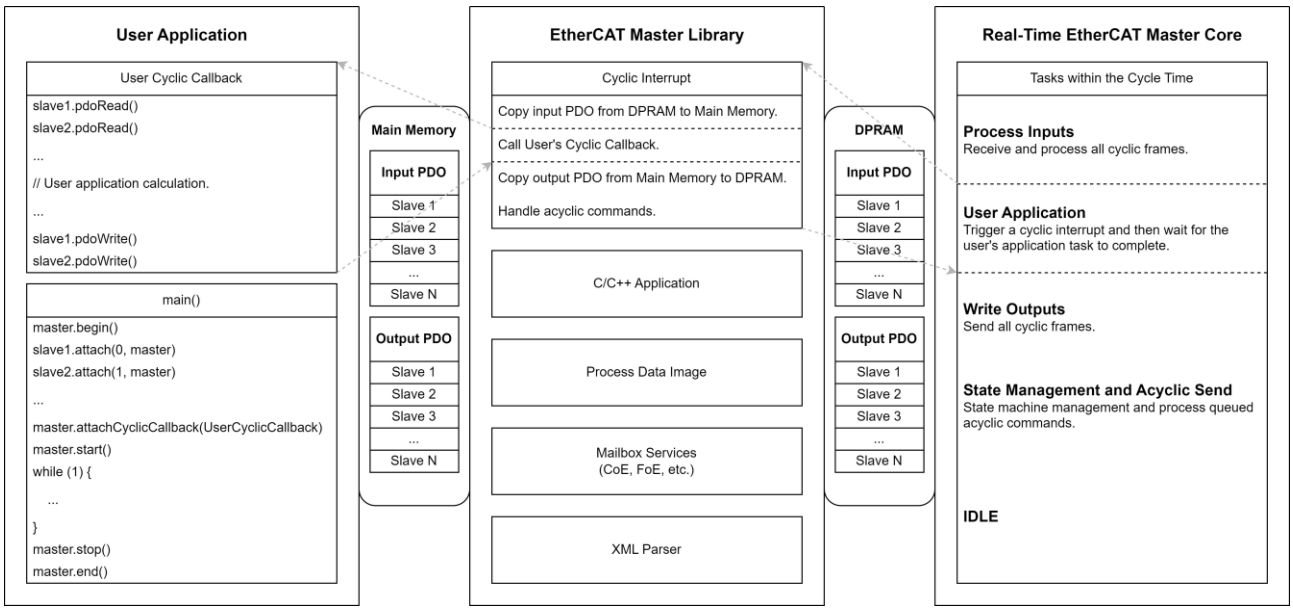
The primary focus of this section is the synchronization of dual-system PDO data. As illustrated in the diagram below, the User Application and EtherCAT Master Library blocks run on the Master System, while the Real-Time EtherCAT Master Core runs on the Slave System.

When the EtherCAT Master 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 Master 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 Master 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 Master Core's **User Application** and the interrupt handling procedure are completed simultaneously.

When the EtherCAT Master 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.



1.2 Features

The EtherCAT Technology Group defined two classes of EtherCAT Master software implementation in ETG.1500. This specification defines Master Classes with a well-defined set of Master functionalities. In order to keep things simple only 2 Master Classes are defined:

- Class A: Standard EtherCAT Master Device
- Class B: Minimum EtherCAT Master Device

You will see the comparison among Class A, Class B, and our QEC Master as follow.

1.2.1 Feature Table

Word Usage:

- **shall** equals is required to.
- **should** equals is recommended that.
- **may** equals is permitted to.
- **O** equals supported.

Feature Name	Short Description	QEC Master
Basic Features		
Service Commands	Support of all commands	O
Slaves with Device Emulation	Support Slaves with and without application controller	O
EtherCAT State Machine	Support of ESM special behavior	O
Error Handling	Checking of network or slave errors, e.g. Working Counter	O
EtherCAT Frame Types	Support EtherCAT Frames	O
Process Data Exchange		
Cyclic PDO	Cyclic process data exchange	O
Network Configuration		
Online scanning	Network configuration functionality included in EtherCAT Master	O
Reading ENI	Network Configuration taken from ENI file	O
Compare Network configuration	Compare configured and existing network configuration during boot-up	O

Explicit Device Identification	Identification used for Hot Connect and prevention against cable swapping	O
Access to EEPROM	Support routines to access EEPROM via ESC register	O
Mailbox Support		
Support Mailbox	Main functionality for mailbox transfer	O
Mailbox Resilient Layer	Support underlying resilient layer	O
CAN application layer over EtherCAT (CoE)		
SDO Up/Download	Normal and expedited transfer	O
Segmented Transfer	Segmented transfer	O
Complete Access	Transfer the entire object (with all sub-indices) at once	O
SDO Info service	Services to read object dictionary	O
FoE		
FoE Protocol	Support FoE Protocol	O
Firmware Up/Download	Password, FileName should be given by the application	O
Synchronization with Distributed Clock (DC)		
DC support	Support of Distributed Clock	O

1.3 Feature Packs

1.3.1 Cable Redundancy

EtherCAT Cable Redundancy refers to the capability of the EtherCAT communication system to maintain continuous and reliable communication even in the event of a cable failure. Cable Redundancy employs a ring topology, which is operated in both directions. If one cable fails or is disconnected, another cable path still works to ensure uninterrupted communication. Cable Redundancy enhances the fault tolerance of the EtherCAT network, minimizing downtime and improving overall system reliability.

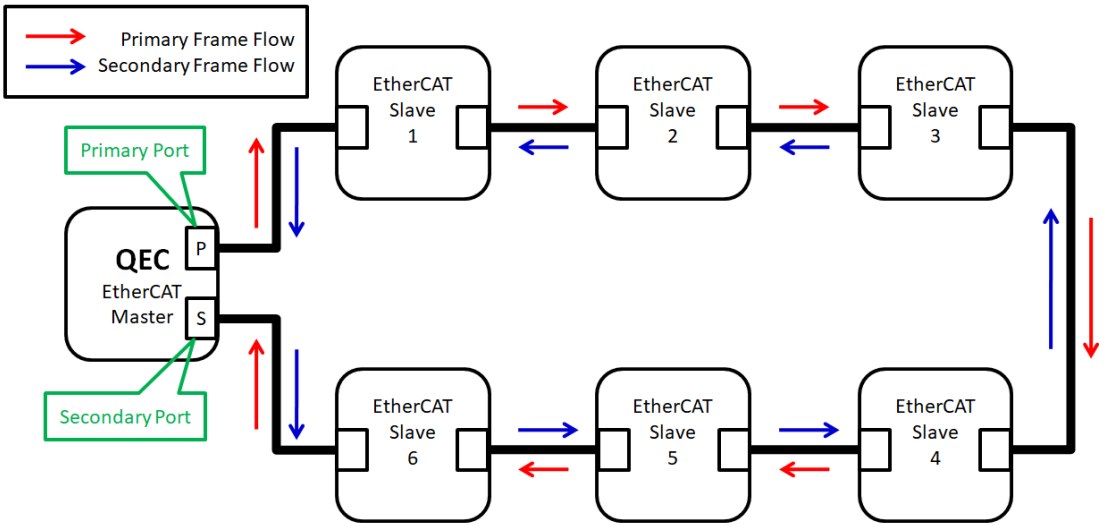
Three scenarios are listed as below regarding whether the cable is broken or not in Cable Redundancy. The following you will see how to work for Cable Redundancy, and the differences for the EtherCAT Master Controller between these scenarios.

- Without Cable Broken
- Cable Broken between two slaves
- Cable broken between master and slave

For ease of explanation, some assumptions will be made here:

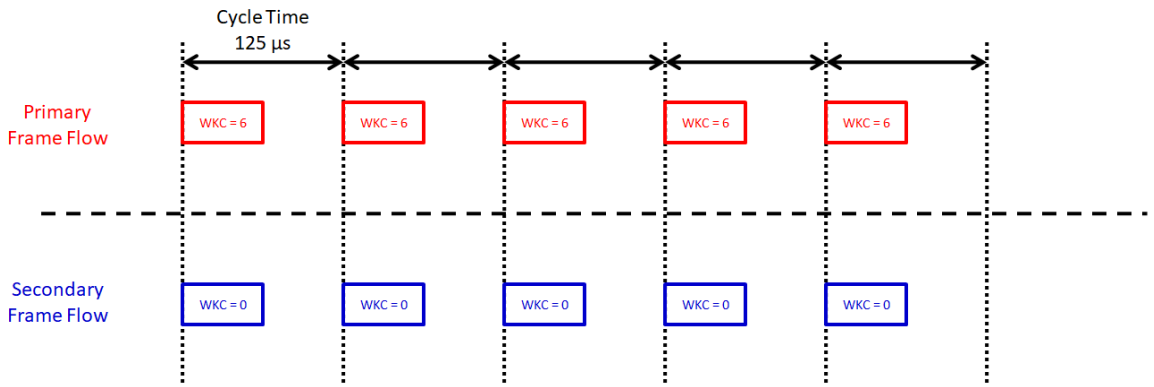
- Assume Cycle Time is set to 125 μs.
- Assume all slaves only with input PDO (without output PDO), the working counter (WKC) in process data frame will increase 1 when passing through every slaves.
- Assume only have 6 slaves on EtherCAT network, and the expected working counter (EWKC) is 6.
- Primary Port and Secondary Port send process data frame in every cycle at the same time.

Without Cable Broken



Without Cable Broken

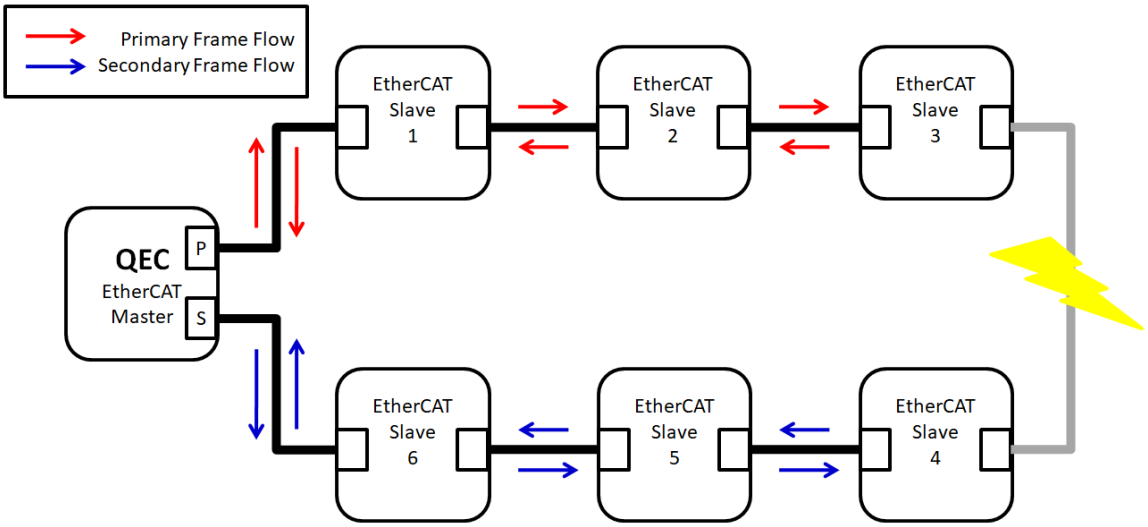
- Primary frame from primary port will be received by secondary port, which frame with WKC = 6.
- Secondary frame from secondary port will be received by primary port, which frame with WKC = 0.
- Master will discard the secondary frame because primary frame's WKC equals to EWKC and secondary frame's WKC equals to 0, it means that without cable broken.



Cable Broken between two slaves

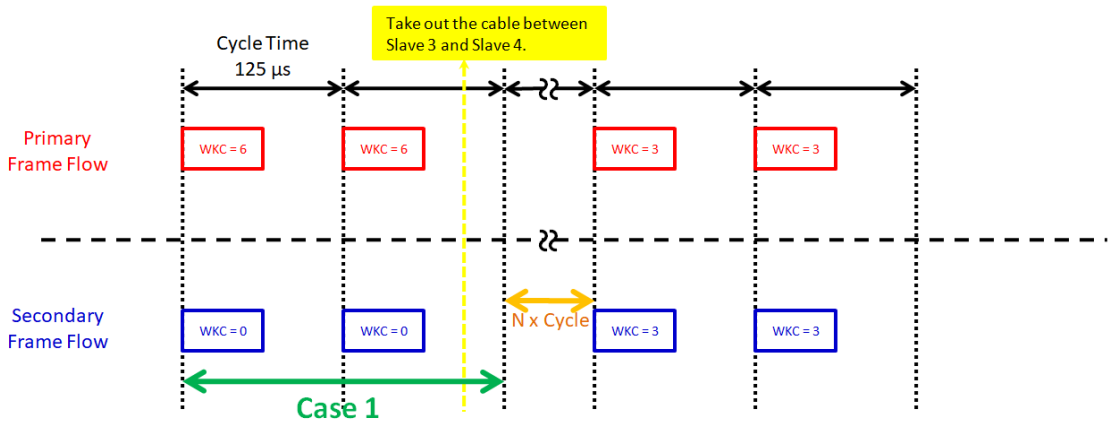
Unplug the cable between Slave 3 and Slave 4

- If you unplug the cable manually, it might with some interference which last several cycles. (N = 0 ~ more)
- If the interference continues for a period of time, some slaves will enter the SAFEOP state due to SyncManager Watchdog.



Cable broken between Slave 3 and Slave 4

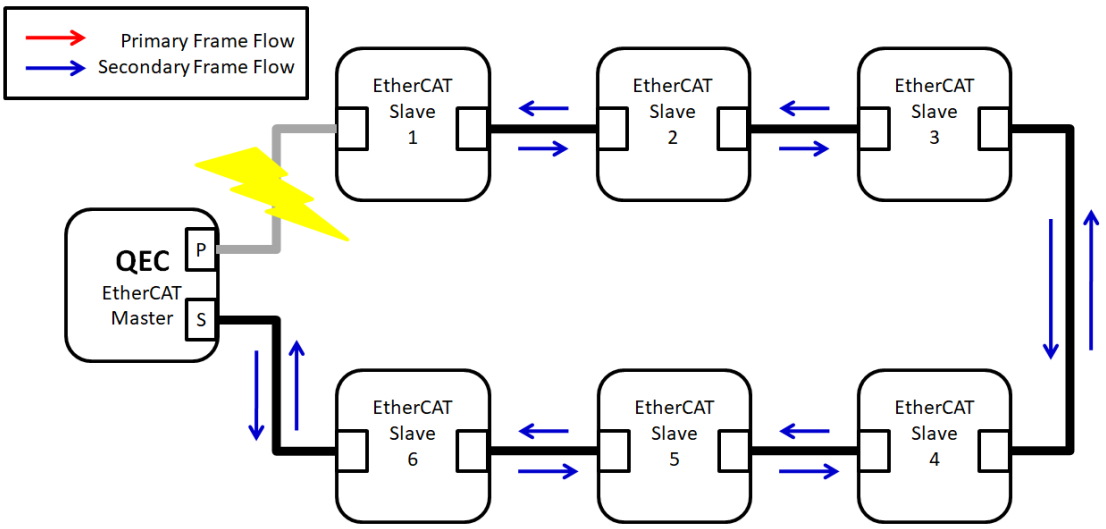
- Primary frame from primary port will be received by primary port, which frame with WKC = 3.
- Secondary frame from secondary port will be received by secondary port, which frame with WKC = 3.
- Primary frame and secondary frame will be combined by Master, because primary frame's WKC is less than EWKC and secondary frame's WKC is greater than 0, it means that the cable broken event is happened between two slaves.



Cable broken between master and slave

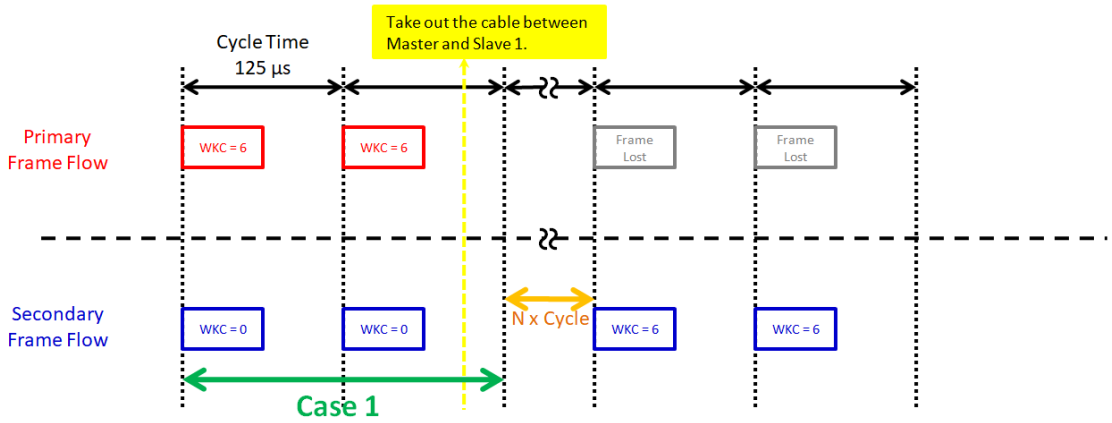
Unplug the cable between Master and Slave 1

- If you unplug the cable manually, it might with some interference which last several cycles. (N = 0 ~ more)
- If the interference continues for a period of time, some slaves will enter the SAFEOP state due to SyncManager Watchdog.



Cable broken between Master and Slave 1

- Primary frame from primary port will lost.
- Secondary frame from secondary port will be received by secondary port, which frame with WKC = 6.
- Master will ignore the primary frame because primary frame was lost and secondary frame's WKC equals to EWKC, it means that the cable broken event is happened between Master and Slave 1.



1.4 Benchmark

EtherCAT is a fieldbus technology known for its high synchronization capabilities. In applications requiring high synchronization, there is often a demand for real-time performance and high control frequencies. Users in these scenarios typically consider specifications such as:

- Support for shorter cycle times
- Support for more process data
- Support for more EtherCAT slave devices

However, assessing whether an EtherCAT master meets the user's application requirements often involves benchmark measurements as a primary consideration.

1.4.1 System Variables

The following factors and variables determine the achievable performance and the choice of EtherCAT cycle time:

- **Network Cable Length**

  Affects the transmission delay of network packets.
- **Number of Slaves**

  Influences the transmission delay of network packets; each slave device may contribute approximately 0.3 to 1 µs of transmission delay.
- **Slave Process Data Bytes**

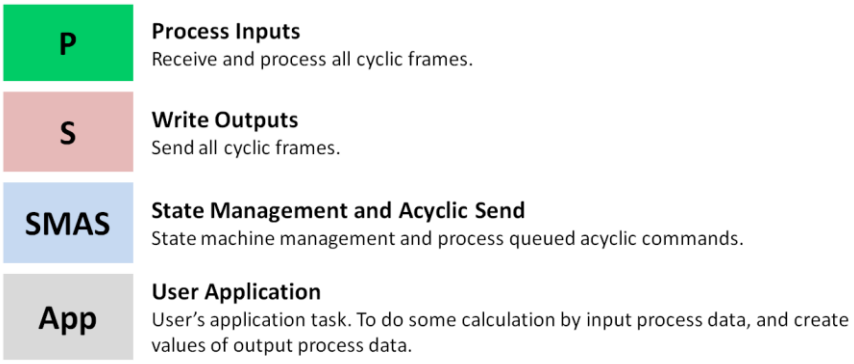
  Determines the length of network packets.
- **Slave Synchronous Mode**

  Constrained by the EtherCAT slave's processing performance in the application.
- **Master Computational Efficiency**

  The efficiency of the EtherCAT master not only affects the cycle time but also influences synchronization accuracy. Factors determining the efficiency of the EtherCAT master include CPU processing speed, software architecture and efficiency, memory transfer speed, etc.

1.4.2 Measurement Functions

For EtherCAT applications with real-time requirements, precise task scheduling within the cycle time is crucial to minimize cycle time jitter. In the QEC-Master software, the cycle time is divided into four phases, each with its respective tasks, as follows:



In the default cycle mode, the timing diagram for tasks within the cycle time is illustrated as follows:

- **Process Inputs**

  The first task at the beginning of the cycle is performed by the EtherCAT master firmware. It is responsible for receiving and processing cyclic process data, transferring Input Process Data to the shared memory of the dual system.

- **User Application**

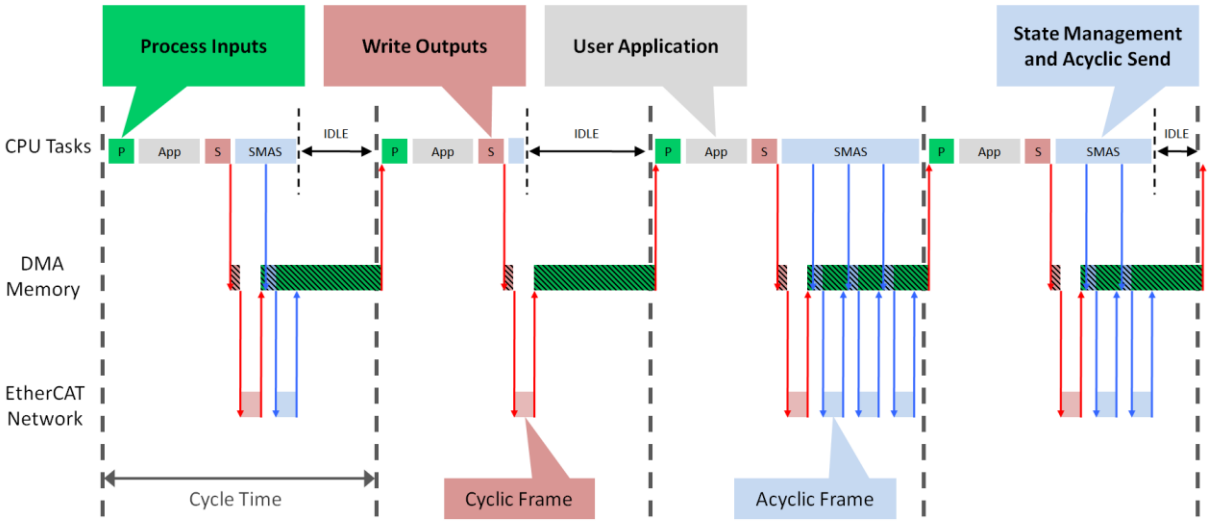
  The EtherCAT master firmware sends a Cyclic Interrupt to the main system. The main system transfers Input Process Data from shared memory to main memory for user reading of the previous cycle's process data. Subsequently, the user's registered Cyclic Callback is invoked, allowing the user to operate EtherCAT slaves at the correct time in control system. The user writes the Output Process Data of current cycle to the main memory. The main system moves Output Process Data to shared memory after completion and responds to the EtherCAT master firmware that the User Application has finished execution.

- **Write Outputs**

  The EtherCAT master firmware performs the Write Outputs task after waiting for the main system's response or timeout. This task involves packaging Output Process Data from shared memory into EtherCAT cyclic process data frames and sending them to all slaves.

- **State Management and Acyclic Send**

The final task within the cycle time for the EtherCAT master firmware. This task involves sending, receiving, and processing the EtherCAT state machine, as well as handling acyclic data transmission (e.g., Mailbox, CoE).



1.4.3 Measurement Result

The following measurement results were performed with 5 slaves on the same controllers with different cycle times.

The 5 slaves are as follows:

- QEC-R00DC4D: 12 digital inputs, and 4 digital outputs.
- QEC-R00D4CD: 4 digital inputs, and 12 digital outputs.
- QEC-R00D88D: 8 digital inputs, and 8 digital outputs.
- QEC-R00D0FS: 16 digital outputs.
- QEC-R00DF0D: 16 digital inputs.

The platform conditions are as follows:

- Processor: Vortex86EX2 600/400 MHz
- OS: FreeDOS
- Compiler: GCC 8.3.0
- Payload: 60 Bytes

A. Configured Cycle Time equals 125 µs

With Acyclic Transfer.

Master Function	min.	avg.	max.
Measured cycle time.	124.90	124.92	125.09
Process Inputs.	6.36	7.26	10.81
Write Outputs.	7.22	7.35	11.89
State Management and Acyclic Send.	2.46	2.48	34.89

Without Acyclic Transfer.

The maximum throughput of SDO commands per second is **769.48**.

Master Function	min.	avg.	max.
Measured cycle time.	124.85	124.86	125.12
Process Inputs.	4.79	4.88	9.80
Write Outputs.	7.91	8.08	14.06
State Management and Acyclic Send.	2.47	60.55	96.30

B. Configured Cycle Time equals 250 µs

With Acyclic Transfer.

Master Function	min.	avg.	max.
Measured cycle time.	249.92	249.93	250.11
Process Inputs.	5.99	7.83	10.42
Write Outputs.	7.90	7.92	12.29
State Management and Acyclic Send.	2.33	2.33	26.66

Without Acyclic Transfer.

The maximum throughput of SDO commands per second is **1409.11**.

Master Function	min.	avg.	max.
Measured cycle time.	249.86	249.87	250.19
Process Inputs.	5.42	5.50	13.03
Write Outputs.	8.39	8.66	13.27
State Management and Acyclic Send.	2.90	12.27	227.15

C. Configured Cycle Time equals 500 µs

With Acyclic Transfer.

Master Function	min.	avg.	max.
Measured cycle time.	499.93	499.93	500.08
Process Inputs.	6.13	7.91	10.26
Write Outputs.	7.91	7.98	12.50
State Management and Acyclic Send.	2.28	2.34	35.17

Without Acyclic Transfer.
The maximum throughput of SDO commands per second is **1657.79**.

Master Function	min.	avg.	max.
Measured cycle time.	499.81	499.83	500.19
Process Inputs.	5.71	6.01	14.03
Write Outputs.	8.84	9.04	14.14
State Management and Acyclic Send.	2.77	123.89	474.36

D. Configured Cycle Time equals 1000 µs

With Acyclic Transfer.

Master Function	min.	avg.	max.
Measured cycle time.	999.95	999.96	1000.07
Process Inputs.	6.57	8.29	10.11
Write Outputs.	7.99	8.01	11.72
State Management and Acyclic Send.	2.32	2.33	34.71

Without Acyclic Transfer.
The maximum throughput of SDO commands per second is **999.99**.

Master Function	min.	avg.	max.
Measured cycle time.	999.93	999.94	1000.08
Process Inputs.	6.42	6.60	9.18
Write Outputs.	10.32	11.57	14.03
State Management and Acyclic Send.	9.15	345.94	593.68

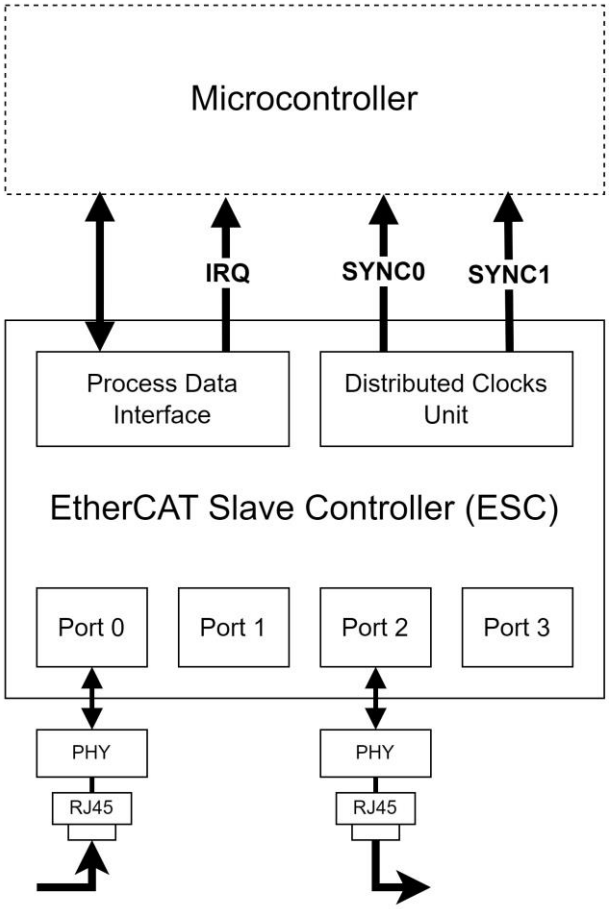
1.4.4 EtherCAT Master Cyclic Frame Jitter

The jitter in the transmission of cyclic frame from the EtherCAT master, which is referenced to DC SYNC0.

- Video: https://youtu.be/O888jD4XUsY?si=NaI9gsafyA1D2DlK

1.5 Synchronization

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



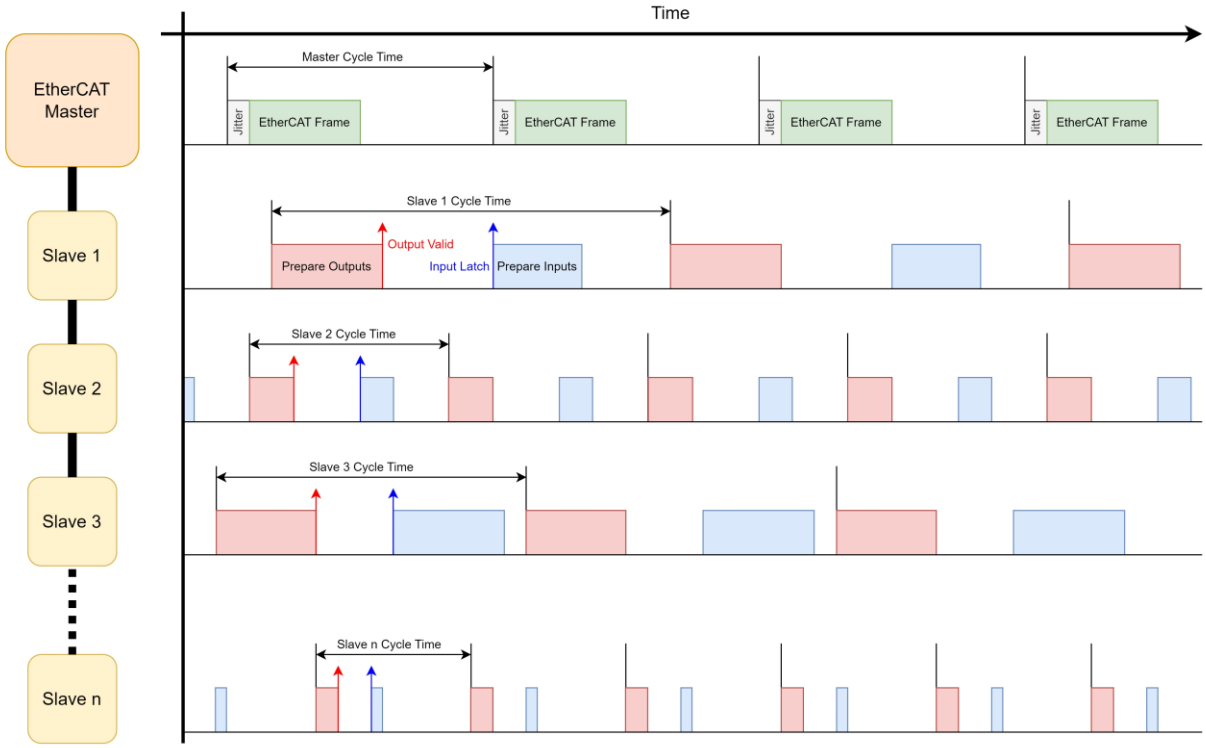
The ESC has three synchronization output pins: ***IRQ***, ***SYNC0***, and ***SYNC1***. The IRQ pin generates a signal to the upper-layer microcontroller (µC) after the ESC receives EtherCAT Cyclic Frames. SYNC0 and SYNC1 pins cyclically generate signals to the µC based on the configuration in the DC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

There are three synchronization modes in EtherCAT:

- Free Run
- SM-Synchronous
- DC-Synchronous

1.5.1 Free Run

The EtherCAT master and all EtherCAT slaves 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 Master and Slave 1, Slave 2, Slave 3 to Slave 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.

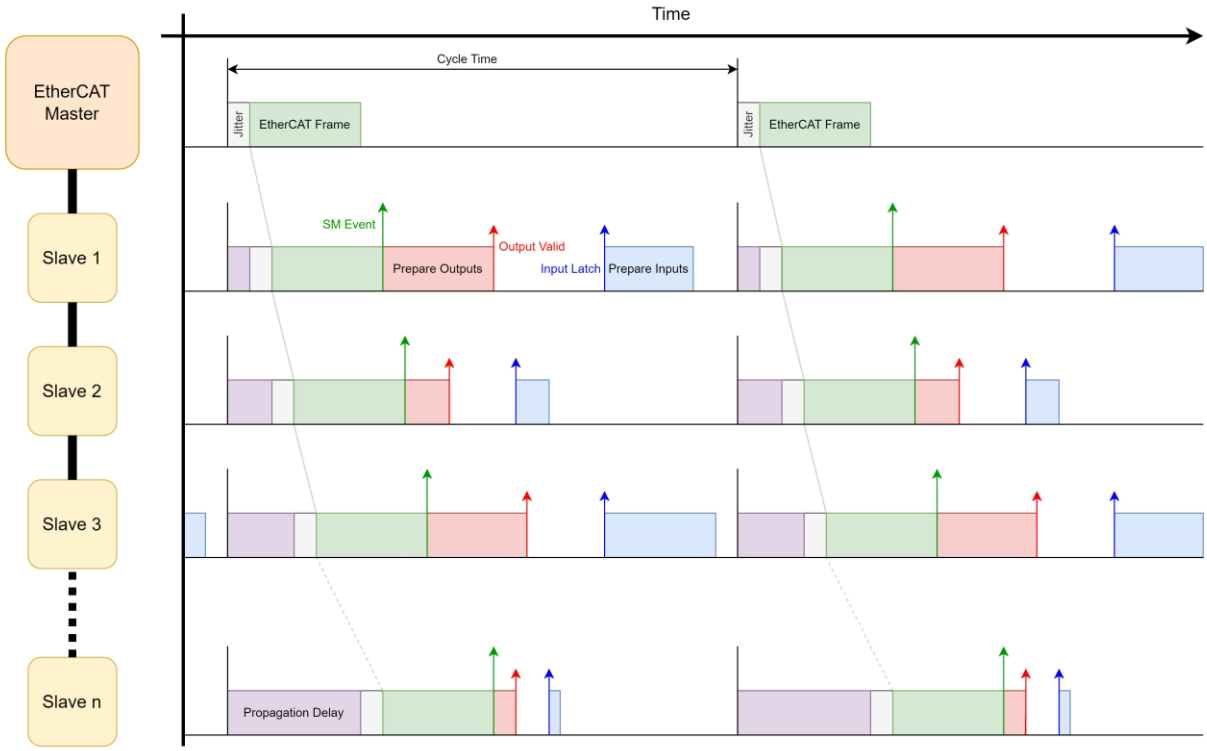


1.5.2 SM-Synchronous

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 slaves with the same jitter of master in sending them. Even assuming zero jitter, due to finite hardware ***Propagation Delay*** the last slaves 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 slaves, resulting in an accuracy of SM-Synchronous at the ***microsecond*** level.

If each slave 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 slaves to be close to each other. However, due to the inability to calculate the propagation delays, the adjustment is quite challenging.



1.5.3 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.

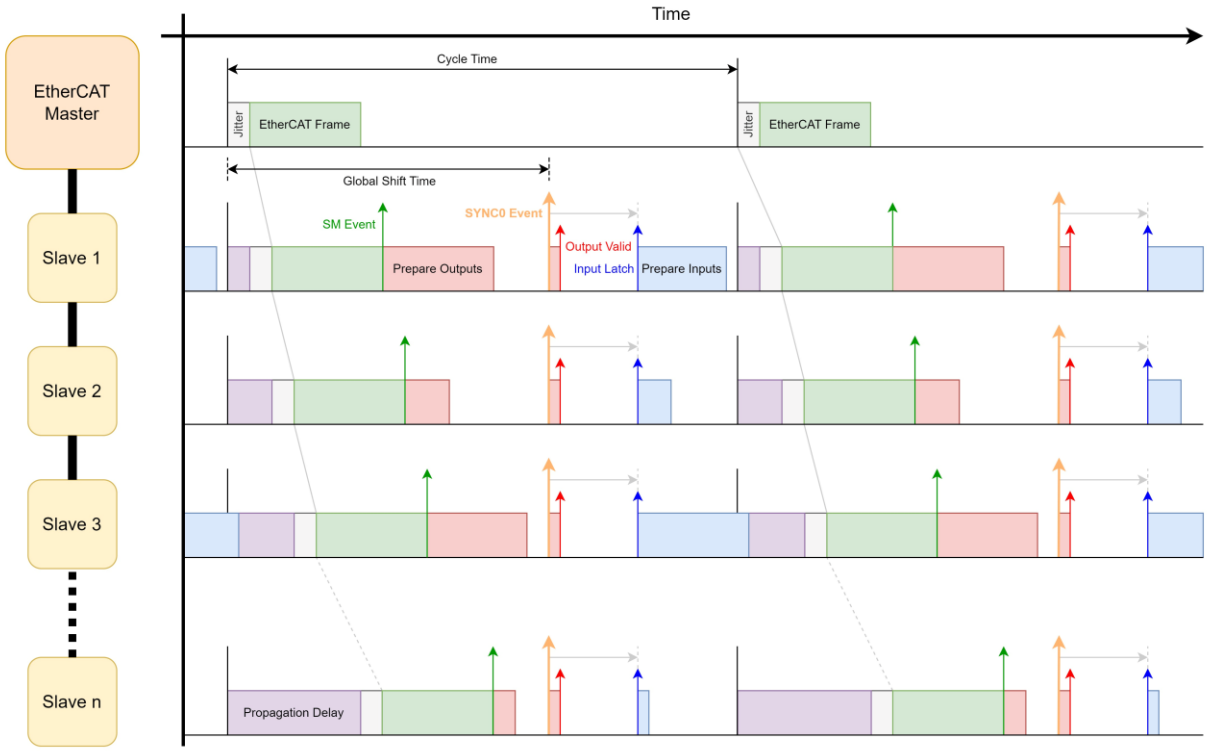
Because SYNC0 events are triggered by the DC unit and all SYNC0 events among the slaves have almost no offset, thanks to the periodic sending of APRW/FPRW commands to synchronize the system time of all slaves, the accuracy can reach the ***nanosecond*** level.

If the slaves 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 slaves 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 slaves.
- 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*.

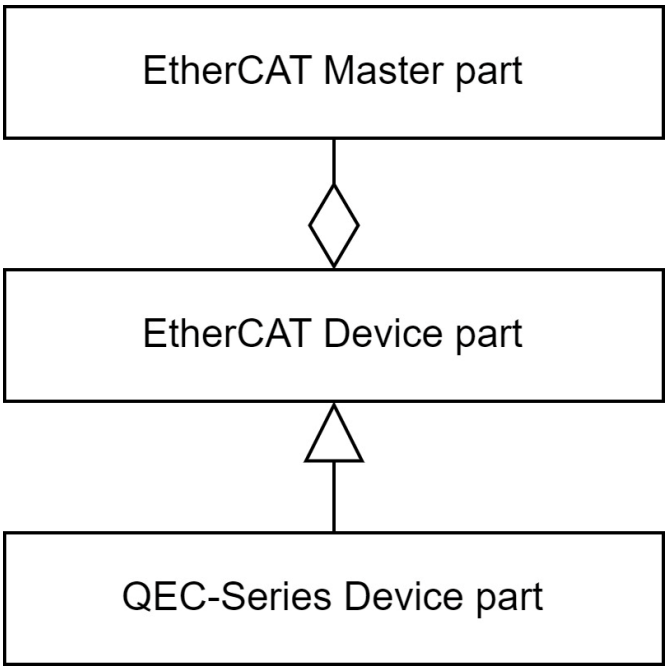


Ch. 2

Functions

QEC EtherCAT Master Library.

EtherCAT is a real-time industrial Ethernet communication protocol widely used in automation and control systems. QEC-Master is an EtherCAT master library implemented in C/C++, which includes classes for the master, generic slave, CiA 402 slave, and dedicated classes for QEC series slaves. These classes not only have clearly defined responsibilities but also consider future extensibility.



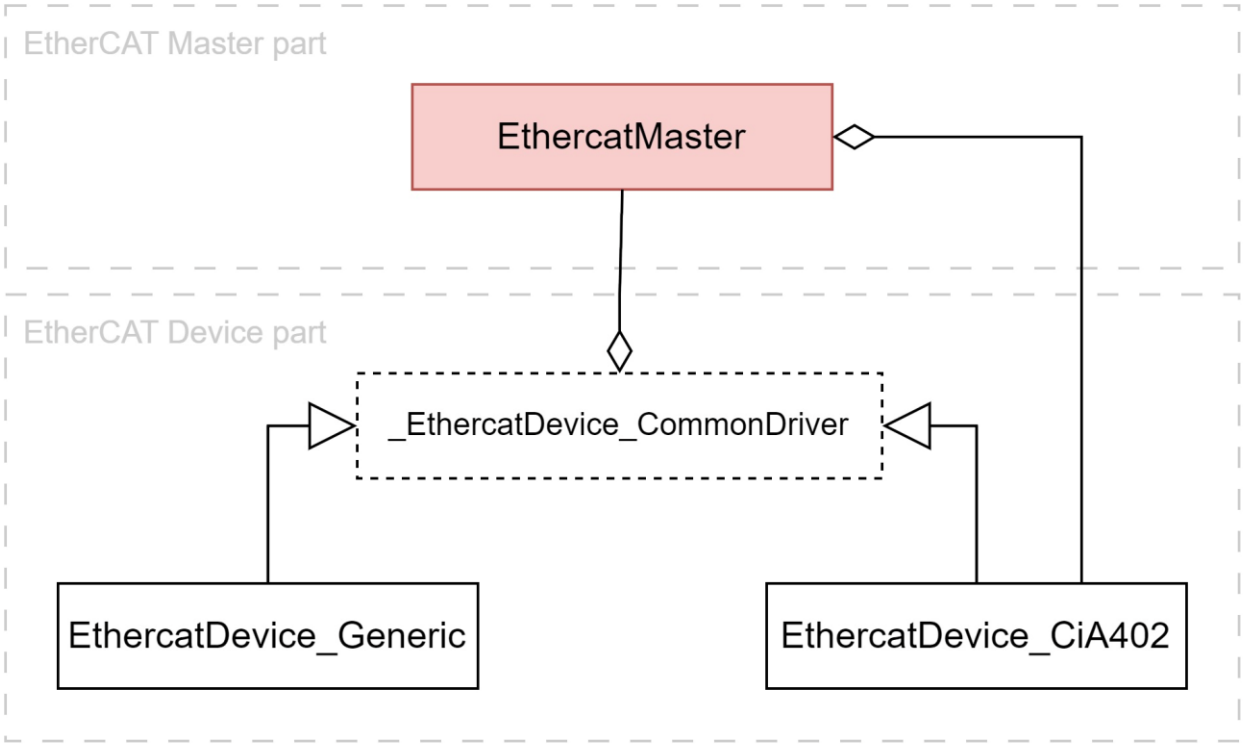
These classes can be divided into three parts as follows:

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

2.1 EtherCAT Master

The *EtherCAT Master part* not only provides various and flexible master configuration and operation functions but also offers diverse EtherCAT slave operation functions for invocation by the EtherCAT Device part.
EthercatMaster is the only class in the *EtherCAT Master part*, it serves as a crucial communication bridge with the EtherCAT firmware. In the Dual-System communication aspect, its responsibilities include communication interface initialization, process data exchange cyclically, handling acyclic transfer interfaces, and managing interrupt events. In the API aspect, it provides functions related to master initialization, master control, and access to slave information.

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



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

Function Groups:

- Initialization
- Control
- Callback
- Slave Information

EtherCAT Master Settings

This library offers a variety of configuration parameters for users to choose from, aiming to meet the diverse application needs of users. Below are the configuration parameters provided by this library.

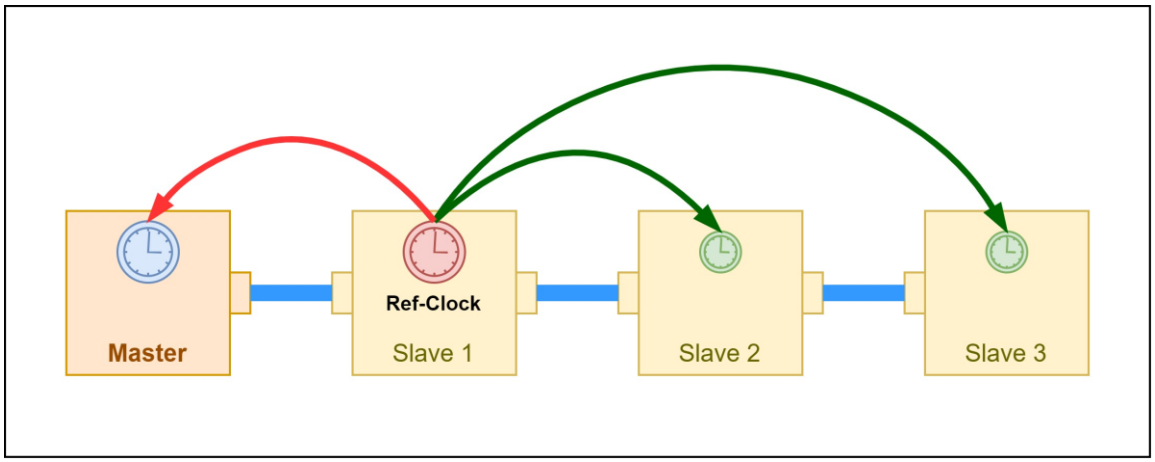
```
typedef struct {

    EthercatDcSyncMode DcSyncMode;
    uint32_t StaticDriftCompensationFrames;

    uint32_t StateMachineTimeoutI2P;
    uint32_t StateMachineTimeoutP2S;
    uint32_t StateMachineTimeoutS2O;
    uint32_t ScanNetworkTimeout;
    uint32_t StartMasterTimeout;
    uint32_t StartDeviceTimeout;

    uint32_t ErrorDetectWkcMultipleFaultsThreshold;
    uint32_t ErrorDetectMultipleLostFramesThreshold;
    uint32_t EnableErrorBusReactionSyncUnitToSafeOp:1,
             EnableErrorBusReactionSyncUnitToSafeOpAutoRestart:1,
             IgnoreBiosOverride:1;

} EthercatMasterSettings;
```

DcSyncMode

Default: `ECAT_MASTER_SHIFT`
In DC-Synchronous mode, the first slave with DC serves as the system reference clock to synchronize other slaves with DC. However, this only involves synchronizing the system time of all slaves on the network and does not include the EtherCAT master. In applications with DC-Synchronous mode enabled, the master usually needs to precisely and periodically control the slaves, so the master must also synchronize its system time with all slaves on the network.

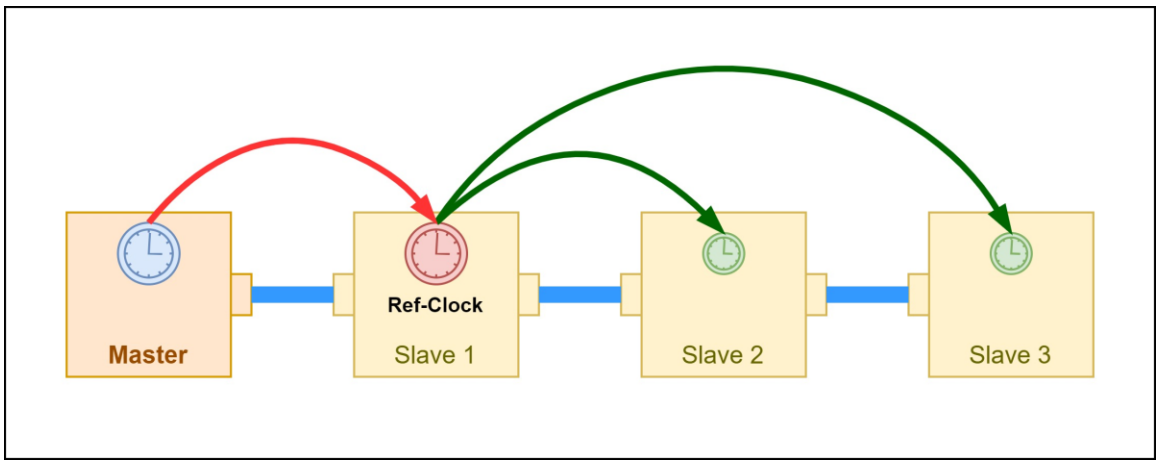
There are two methods for this synchronization:

- **Master Shift Mode** - `ECAT_MASTER_SHIFT`



  - The master system time is synchronized to the reference clock.
  - All DC slaves are synchronized to the reference clock.

- **Bus Shift Mode** - `ECAT_BUS_SHIFT`



  - The reference clock is synchronized to the master system time.
  - All DC slaves are synchronized to the reference clock.

StaticDriftCompensationFrames

Default: 30000
The EtherCAT master sends many separate ARMW or FRMW drift compensation frames to distribute the System Time of the reference clock to all DC slaves.

StateMachineTimeoutI2P

Default: 3000 Unit: milliseconds
Timeout for the transition from the Init state to the Pre-Operational state.

StateMachineTimeoutP2S

Default: 10000 Unit: milliseconds
Timeout for the transition from the Pre-Operational state to the Safe-Operational state.

StateMachineTimeoutS2O

Default: 10000 Unit: milliseconds
Timeout for the transition from the Safe-Operational state to the Operational state.

ScanNetworkTimeout

Default: 5000 Unit: milliseconds
Timeout for scan network. The scan network operation is executed within `EthercatMaster::begin()`.

StartMasterTimeout

Default: 3000 Unit: milliseconds
Base timeout for start master, $T\sim base\sim$.
In EthercatMaster::start(), the firmware is requested to start EtherCAT, and the timeout for this request is referred to as startup timeout, $\$T\sim startup\sim$.
The startup timeout for EtherCAT is calculated as follows:
$\$Tstartup = Tbase + (Tslave \times Nslaves)\$$
Here, $N\sim slaves\sim$ is the number of slaves on the network.

StartDeviceTimeout

Default: 500 Unit: milliseconds
Timeout per slave for start master, $T\sim slave\sim$.

ErrorDetectWkcMultipleFaultsThreshold

Default: 3
The master should check the Working Counter of a received EtherCAT datagram. If the Working Counter does not match with the expected value an error is detected. When the number of consecutive errors exceeds this parameter,
an ECAT_ERR_WKC_MULTIPLE_FAULTS error interrupt will be triggered.

ErrorDetectMultipleLostFramesThreshold

Default: 3
The master may use the index of the EtherCAT datagram header to check if all sent EtherCAT datagrams will be received. If EtherCAT datagrams are lost an error is detected. When the number of consecutive errors exceeds this parameter,
an ECAT_ERR_MULTIPLE_LOST_FRAMES error interrupt will be triggered.

EnableErrorBusReactionSyncUnitToSafeOp

Default: 0
If this parameter is set to 1, the master will change the EtherCAT state of the slaves with an application controller and will disable the Sync Manager channels of the slaves which only support the EtherCAT state machine emulation.

EnableErrorBusReactionSyncUnitToSafeOpAutoRestart

Default: 1
This parameter only takes effect if `EnableErrorBusReactionSyncUnitToSafeOp` is set to 1. If this parameter is set to 1, the master will automatically attempt to restart the Sync Unit according to the Restart Behavior of a Sync Unit in the Master as defined in ETG.1020 EtherCAT Protocol Enhancements, switching the EtherCAT state machine back to the Operational state.

IgnoreBiosOverride

Default: 0
QEC-Master has some EtherCAT configuration parameters in the BIOS. Setting this parameter to 1 means ignoring the EtherCAT configuration parameters in the BIOS; otherwise, they are not ignored.

2.1.1 Initialization Functions

Before starting the EtherCAT master, it must be initialized. This library offers a variety of configuration parameters for users to choose from, aiming to meet the diverse application needs of users.

Functions:

- `begin()`
- `end()`
- `isRedundancy()`
- `libraryVersion()`
- `firmwareVersion()`
- `readSettings()`
- `saveSettings()`

`begin()`

Description

Initialize the EtherCAT master, scan all EtherCAT slave devices on the network, and switch the EtherCAT state machine to the Pre-Operational state.

Syntax

```
int begin(EthernetPort eth = ECAT_ETH_0, const char *eni_filename = NULL);
```

Parameters

- `[in] EthernetPort eth`
  Selection of the Ethernet interface for EtherCAT communication.
  - `ECAT_ETH_0`: Only eth0 is used as the EtherCAT communication interface.
  - `ECAT_ETH_1`: Only eth1 is used as the EtherCAT communication interface.
  - `ECAT_ETH_REDUNDANCY`: Enable EtherCAT Cable Redundancy with eth0 as the primary port and eth1 as the secondary port.

  The default is `ECAT_ETH_0`.
- `[in] const char *eni_filename`
  The file name of the EtherCAT Network Information (ENI). For details about the ENI content supported by this library, please refer to EtherCAT Network Information. The default is NULL.

Return Value

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

Comment

This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    master.begin();
    master.start(1000000); // cycle time set as 1 millisecond.
}
void loop() {

}
```

end()

Description

Deinitialize the EtherCAT master.

Syntax

```
void end();
```

Parameters

None.

Return Value

None.

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    master.begin();
    master.end();
}
void loop() {

}
```

isRedundancy()

Description

Check if the EtherCAT master has Cable Redundancy enabled.

Syntax

`bool isRedundancy();`

Parameters

None.

Return Value

Return whether Cable Redundancy is enabled for the EtherCAT master. `true` indicates it is enabled, while `false` indicates it is not.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
  Serial.begin(115200);
  master.begin();
  Serial.println(master.isRedundancy());
}
void loop() {

}
```

libraryVersion()

Description

Get the EtherCAT master library version.

Syntax

```
char *libraryVersion();
```

Parameters

None.

Return Value

Return a pointer to the EtherCAT master library version string.

Comment

This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
  Serial.begin(115200);
  Serial.println(master.libraryVersion());
}
void loop() {

}
```

firmwareVersion()

Description

Get the EtherCAT firmware version.

Syntax

```
char *firmwareVersion();
```

Parameters

None.

Return Value

Return a pointer to the EtherCAT firmware version string.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
  Serial.begin(115200);
  master.begin();
  Serial.println(master.firmwareVersion());
}
void loop() {

}
```

readSettings()

Description

Read the current EtherCAT master settings.

Syntax

```
int readSettings(EthercatMasterSettings *settings);
```

Parameters

- `[in] EthercatMasterSettings *settings`
  A pointer to the `EthercatMasterSettings` data structure. For more details about the `EthercatMasterSettings` parameters, please refer to EtherCAT Master Settings.

Return Value

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

Comment

This function must be called before `EthercatMaster::begin()` or after `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup(void) {
  EthercatMasterSettings settings;

  Serial.begin(115200);
  while (!Serial);

  master.readSettings(&settings);
  settings.DcSyncMode = ECAT_BUS_SHIFT;
  settings.StateMachineTimeoutI2P = 3000;
  settings.StateMachineTimeoutP2S = 10000;
  settings.StateMachineTimeoutS2O = 1000;
  settings.ScanNetworkTimeout = 5000;
  settings.StartMasterTimeout = 3000;
  settings.StartDeviceTimeout = 2000;
  master.saveSettings(&settings);
```

```
  Serial.println(master.begin());
  slave.attach(0, master);

  Serial.println(master.start(1000000, ECAT_SYNC));
}

void loop() {

}
```

saveSettings()

Description

Save the EtherCAT master settings.

Syntax

```
int saveSettings(EthercatMasterSettings *settings);
```

Parameters

- `[in] EthercatMasterSettings *settings`
  A pointer to the `EthercatMasterSettings` data structure. For more details about the `EthercatMasterSettings` parameters, please refer to EtherCAT Master Settings.

Return Value

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

Comment

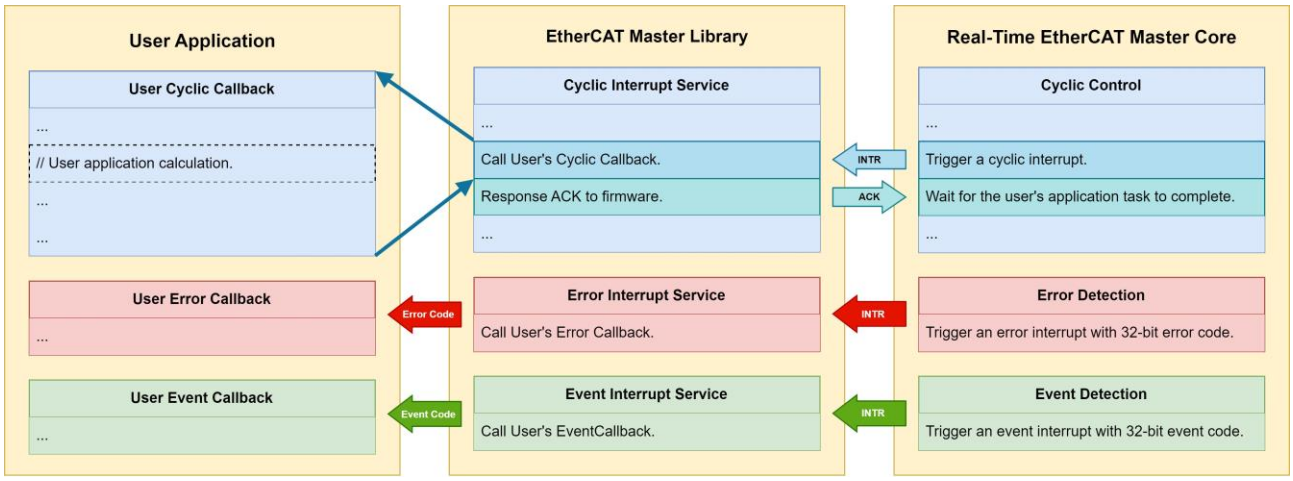
This function must be called before `EthercatMaster::begin()` or after `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup(void) {
  EthercatMasterSettings settings;

  Serial.begin(115200);
  while (!Serial);

  master.readSettings(&settings);
  settings.DcSyncMode = ECAT_BUS_SHIFT;
  settings.StateMachineTimeoutI2P = 3000;
  settings.StateMachineTimeoutP2S = 10000;
  settings.StateMachineTimeoutS2O = 1000;
  settings.ScanNetworkTimeout = 5000;
  settings.StartMasterTimeout = 3000;
  settings.StartDeviceTimeout = 2000;
  master.saveSettings(&settings);
```

```
  Serial.println(master.begin());
  slave.attach(0, master);

  Serial.println(master.start(1000000, ECAT_SYNC));
}

void loop() {

}
```

2.1.2 Callback Functions



This library provides three types of callbacks as follows:

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

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

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

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

- **Event Callback**
  When the Real-Time EtherCAT Master Core detects an event, it triggers an event interrupt and passes a 32-bit event code to the EtherCAT Master Library. If the user has registered an event callback, the system will invoke that callback to inform the user of the specific event.

  The event codes supported by the Event Callback are as follows:

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

Functions:

- attachCyclicCallback()
- detachCyclicCallback()
- attachErrorCallback()
- detachErrorCallback()
- attachEventCallback()
- detachEventCallback()
- errGetCableBrokenLocation1()
- errGetCableBrokenLocation2()
- evtGetMasterState()

attachCyclicCallback()

Description

Register Cyclic Callback Function.

Syntax

```
int attachCyclicCallback(void (*callback)(void));
```

Parameters

- `[in] void (*callback)(void)`
  The cyclic callback function to be registered, which has no parameters and no return value.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()` and before `EthercatMaster::start()`, or after a successful execution of `EthercatMaster::stop()` and before `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void CyclicCallback() {
    // put your cyclic Callback function here.
}
void setup() {
    master.begin();
    slave.attach(0, master);
    master.attachCyclicCallback(CyclicCallback);
    master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms
}
void loop() {
}
```

detachCyclicCallback()

Description

Unregister Cyclic Callback Function.

Syntax

```
int detachCyclicCallback();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()` and before `EthercatMaster::start()`, or after a successful execution of `EthercatMaster::stop()` and before `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"
EthercatMaster master;
EthercatDevice_Generic slave;

void CyclicCallback() {}
void setup() {
    master.begin();
    slave.attach(0, master);
    master.attachCyclicCallback(CyclicCallback);
    master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms

    delay(5000);
    master.stop();
    master.detachCyclicCallback();
    master.end();
}
void loop() {
}
```

attachEventCallback()

Description

Register Event Callback Function.

Syntax

```
void attachEventCallback(void (*callback)(uint32_t));
```

Parameters

- `[in] void (*callback)(uint32_t)`
  The event callback function to be registered, which only has one parameter, a 32-bit event code.

Return Value

None.

Comment

This function must be called before `EthercatMaster::begin()` or after `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t EventCode = 0;

void EventCallback(uint32_t eventcode) {
  EventCode = eventcode;
}

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

  master.attachEventCallback(EventCallback); // EventCallback Function.

  master.begin();
  slave.attach(0, master);
  master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms
}
```

```
void loop() {
  /* Event Code */
  uint32_t event = EventCode;
  EventCode = 0;
  if (event) {
    Serial.print("EventCode:");
    Serial.println(event);
  }
}
```

detachEventCallback()

Description

Unregister Event Callback Function.

Syntax

```
void detachEventCallback();
```

Parameters

None.

Return Value

None.

Comment

This function must be called before EthercatMaster::begin() or after EthercatMaster::end().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void EventCallback(uint32_t eventcode){

}

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

  delay(5000);
  master.stop();
  master.detachEventCallback();
  master.end();
}

void loop() {

}
```

attachErrorCallback()

Description

Register Error Callback Function.

Syntax

```
void attachErrorCallback(void (*callback)(uint32_t));
```

Parameters

- `[in] void (*callback)(uint32_t)`
  The error callback function to be registered, which only has one parameter, a 32-bit error code.

Return Value

None.

Comment

This function must be called before `EthercatMaster::begin()` or after `EthercatMaster::end()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint32_t ErrorCode = 0;

void ErrorCallback(uint32_t errorcode) {
    ErrorCode = errorcode;
}

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

    master.attachErrorCallback(ErrorCallback); // ErrorCallback
Function.

    master.begin();
```

```
    slave.attach(0, master);
    master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms
}

void loop() {
    /* Error Code */
    uint32_t error = ErrorCode;
    ErrorCode = 0;
    if (error) {
        Serial.print("ErrorCode:");
        Serial.println(error);
    }
}
```

detachErrorCallback()

Description

Unregister Error Callback Function.

Syntax

```
void detachErrorCallback();
```

Parameters

None.

Returns

None.

Comment

This function must be called before EthercatMaster::begin() or after EthercatMaster::end().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void ErrorCallback(uint32_t errorcode) {

}

void setup() {
  master.attachErrorCallback(ErrorCallback); // ErrorCallback Function.

  master.begin();
  slave.attach(0, master);
  master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms

  delay(5000);
  master.stop();
  master.detachEventCallback();
  master.end();
```

```
}

void loop() {

}
```

errGetCableBrokenLocation1()

Description

Get the content of the `ECAT_ERR_CABLE_BROKEN` error, which represents the cable broken location 1.

Syntax

```
int errGetCableBrokenLocation1();
```

Parameters

None.

Return Value

Return the cable broken location 1.

Comment

This function must be called in error callback.

Example Code

```
#include <Ethercat.h>

EthercatMaster master;

bool CableBrokenLatched = false;
int CableBrokenLocation1;
int CableBrokenLocation2;

void ErrorCallback(uint32_t errorcode)
{
  if (errorcode == ECAT_ERR_CABLE_BROKEN) {
    if (CableBrokenLatched == false) {
      CableBrokenLatched = true;
      CableBrokenLocation1 = master.errGetCableBrokenLocation1();
      CableBrokenLocation2 = master.errGetCableBrokenLocation2();
    }
  }
}

void setup() {
  Serial.begin(115200);
  master.attachErrorCallback(ErrorCallback);
  master.begin();
  master.start();
}
```

```
void loop() {
  if (CableBrokenLatched == true) {
    Serial.print("Cable broken between ");
    if (CableBrokenLocation1 < 0) Serial.print("Primary Port");
    else Serial.print("Slave "); Serial.print(CableBrokenLocation1);
    Serial.print(" and ");
    if (CableBrokenLocation2 < 0) Serial.println("Secondary Port");
    else Serial.println("Slave ");
Serial.println(CableBrokenLocation2);
    CableBrokenLatched = false;
  }
}
```

errGetCableBrokenLocation2()

Description

Get the content of the `ECAT_ERR_CABLE_BROKEN` error, which represents the cable broken location 2.

Syntax

```
int errGetCableBrokenLocation2();
```

Parameters

None.

Return Value

Return the cable broken location 2.

Comment

This function must be called in error callback.

Example Code

```
#include <Ethercat.h>

EthercatMaster master;

bool CableBrokenLatched = false;
int CableBrokenLocation1;
int CableBrokenLocation2;

void ErrorCallback(uint32_t errorcode)
{
  if (errorcode == ECAT_ERR_CABLE_BROKEN) {
    if (CableBrokenLatched == false) {
      CableBrokenLatched = true;
      CableBrokenLocation1 = master.errGetCableBrokenLocation1();
      CableBrokenLocation2 = master.errGetCableBrokenLocation2();
    }
  }
}

void setup() {
  Serial.begin(115200);
  master.attachErrorCallback(ErrorCallback);
  master.begin();
  master.start();
}
```

```
void loop() {
  if (CableBrokenLatched == true) {
    Serial.print("Cable broken between ");
    if (CableBrokenLocation1 < 0) Serial.print("Primary Port");
    else Serial.print("Slave "); Serial.print(CableBrokenLocation1);
    Serial.print(" and ");
    if (CableBrokenLocation2 < 0) Serial.println("Secondary Port");
    else Serial.println("Slave ");
Serial.println(CableBrokenLocation2);
    CableBrokenLatched = false;
  }
}
```

evtGetMasterState()

Description

Get the content of the `ECAT_EVT_STATE_CHANGED` event, which represents the EtherCAT master state.

Syntax

```
int evtGetMasterState();
```

Parameters

None.

Return Value

Return the EtherCAT master state.

Comment

This function must be called in event callback.

Example Code

```
#include <Ethercat.h>

EthercatMaster master;
int CurrentMasterState;

void EventCallback(uint32_t eventcode)
{
  if (eventcode == ECAT_EVT_STATE_CHANGED)
    CurrentMasterState = master.evtGetMasterState();
}

void setup() {
  Serial.begin(115200);
  master.attachEventCallback(EventCallback);
  master.begin();
  master.start();
}

void loop() {
  Serial.print("CurrentMasterState: ");
  Serial.println(CurrentMasterState);
  delay(1000);
}
```

2.1.3 Slave Information Functions

This library provides functions to obtain information about EtherCAT slave devices on the network. It includes querying the number of slaves on the network, retrieving a slave's Vendor ID, Product Code, Alias Address by its sequence number, and reverse querying the slave number using the aforementioned information.

This is used to identify the type of slave device and to choose the appropriate EtherCAT slave device class to attach.

Functions:

- `getSlaveCount()`
- `getVendorID()`
- `getProductCode()`
- `getRevisionNumber()`
- `getSerialNumber()`
- `getAliasAddress()`
- `getSlaveNo()`

getSlaveCount()

Description

Get EtherCAT Slave Count Number on EtherCAT bus.

Syntax

```
uint16_t getSlaveCount();
```

Return Value

Return the number of slaves on the network. If an error occurs, it will return 0.

Comment

This function must be called after a successful execution of EthercatMaster::begin().

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
  uint16_t slavecount, i;
  Serial.begin(115200);
  master.begin();
  slavecount = master.getSlaveCount();
  for (i = 0; i < slavecount; i++) {
    Serial.print("Slave");
    Serial.print(i);

  }
}
void loop() {

}
```

getVendorID()

Description

Get the EtherCAT Slave Vendor ID on EtherCAT bus.

Syntax

```
uint32_t getVendorID(uint16_t slave_no);
```

Parameters

- `[in] unit16_t slave_no`
  The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.

Return Value

Return the Vendor ID of the specified slave. If an error occurs, it will return `UINT32_MAX`.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
  uint32_t slavevendorID;
  uint16_t slavecount = 1;
  Serial.begin(115200);
  while (!Serial);
  master.begin();
  slavevendorID = master.getVendorID(slavecount);
  Serial.println(slavevendorID);
}
void loop() {

}
```

getProductCode()

Description

Get the EtherCAT Slave Product Code on EtherCAT bus.

Syntax

```
uint32_t getProductCode(uint16_t slave_no);
```

Parameters

- `[in] unit16_t slave_no`
  The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.

Return Value

Return the Product Code of the specified slave. If an error occurs, it will return `UINT32_MAX`.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    uint32_t slaveproductcode;
    uint16_t slavecount=1;
    Serial.begin(115200);
    while (!Serial);
    master.begin();
    slaveproductcode = master.getProductCode(slavecount);
    Serial.println(slaveproductcode);
}
void loop() {

}
```

getRevisionNumber()

Description

Get the EtherCAT Slave revision Number on EtherCAT bus.

Syntax

```
uint32_t getRevisionNumber(uint16_t slave_no);
```

Parameters

- `[in] unit16_t slave_no`
  The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.

Return Value

Return the Revision Number of the specified slave. If an error occurs, it will return `UINT32_MAX`.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    uint32_t slaverevisionnumber;
    uint16_t slavecount=1;
    Serial.begin(115200);
    while (!Serial);
    master.begin();
    slaverevisionnumber = master.getRevisionNumber(slavecount);
    Serial.println(slaverevisionnumber);
}
void loop() {

}
```

getSerialNumber()

Description

Get the EtherCAT Slave Serial Number on EtherCAT bus.

Syntax

```
uint32_t getSerialNumber(uint16_t slave_no);
```

Parameters

- `[in] unit16_t slave_no`
  The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.

Return Value

Return the Serial Number of the specified slave. If an error occurs, it will return `UINT32_MAX`.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    uint32_t slaveserialnum;
    uint16_t slavecount=1;
    Serial.begin(115200);
    while (!Serial);
    master.begin();
    slaveserialnum = master.getSerialNumber(slavecount);
    Serial.println(slaveserialnum);
}
void loop() {

}
```

getAliasAddress()

Description

Get the EtherCAT Slave Alias Address on EtherCAT bus.

Syntax

```
int getAliasAddress(uint16_t slave_no);
```

Parameters

- `[in] unit16_t slave_no`
  The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.

Return Value

Return the Alias Address of the specified slave. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;

void setup() {
    uint32_t slavealiasaddress;
    uint16_t slavecount=1;
    Serial.begin(115200);
    while (!Serial);
    master.begin();
    slavealiasaddress = master.getAliasAddress(slavecount);
    Serial.println(slavealiasaddress);
}
void loop() {

}
```

getSlaveNo()

Description

Obtain the sequential ID of the specified slave based on the alias address, supplier number, product number, amendment number, serial number.

Syntax

```
int getSlaveNo(uint16_t alias_addr);
int getSlaveNo(uint32_t vendor, uint32_t product);
int getSlaveNo(uint32_t vendor, uint32_t product, uint32_t revision,
uint32_t serial_num);
int getSlaveNo(uint16_t alias_addr, uint32_t vendor, uint32_t product);
int getSlaveNo(uint16_t alias_addr, uint32_t vendor, uint32_t product,
uint32_t revision, uint32_t serial_num);
```

Parameters

- `[in] uint16_t alias_addr`
  The Alias Address of the EtherCAT slave device you want to find. If the input value is 0, it indicates that this parameter should not be used for the search.
- `[in] uint32_t vendor`
  The Vendor ID of the EtherCAT slave device you want to find. If the input value is 0, it indicates that this parameter should not be used for the search.
- `[in] uint32_t product`
  The Product Code of the EtherCAT slave device you want to find. If the input value is 0, it indicates that this parameter should not be used for the search.
- `[in] uint32_t revision`
  The Revision Number of the EtherCAT slave device you want to find. If the input value is 0, it indicates that this parameter should not be used for the search.
- `[in] uint32_t serial_num`
  The Serial Number of the EtherCAT slave device you want to find. If the input value is 0, it indicates that this parameter should not be used for the search.

Return Value

Return the sequence number of the matching EtherCAT slave device on the network. If the returned value is -1, it indicates that no matching EtherCAT slave device was found. Any other value less than 0 indicates an error code.

Comment

This function must be called after a successful execution of EthercatMaster::begin().

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;

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

  Serial.print("Search Test 1 => ");
  Serial.println(master.getSlaveNo(1200));

  Serial.print("Search Test 2 => ");
  Serial.println(master.getSlaveNo(0x00000BC3, 0x0086D324));

  Serial.print("Search Test 3 => ");
  Serial.println(master.getSlaveNo(0x00000BC3, 0x0086D304, 0x20220316,
0x00000000));

  Serial.print("Search Test 4 => ");
  Serial.println(master.getSlaveNo(251, 0x00000BC3, 0x0086D302));

  Serial.print("Search Test 5 => ");
  Serial.println(master.getSlaveNo(252, 0x00000BC3, 0x0086D301,
0x20220316, 0x00000000));
}

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

2.1.4 Control Functions

The control functions provided by the EtherCAT master library are crucial for managing the state and operation of the EtherCAT network.

By using these functions, users can ensure precise control over the network, achieving reliable and synchronized communication between the master and slave devices.

Functions:

- start()
- stop()
- update()
- setShiftTime()
- getShiftTime()
- getSystemTime()
- getWorkingCounter()
- getExpectedWorkingCounter()

`start()`

Description

Start the EtherCAT master, configure the SM, FMMU, and DC registers of all slaves, and switch the EtherCAT state machine to the Operational state.

Syntax

```
int start(uint64_t cycletime_ns = 0, EthercatMasterMode mode =
ECAT_FREERUN);
```

Parameters

- `[in] uint64_t cycletime_ns`
  EtherCAT cycle time. The EtherCAT firmware will periodically generate interrupts according to this cycle time to update process data and handle acyclic transmissions. If the user has registered a Cyclic Callback and the EtherCAT master control mode is not set to `ECAT_FREERUN_MANUAL`, the callback will be called during these periodic interrupts.
- `[in] EthercatMasterMode mode`
  Selection of the EtherCAT control mode. For detailed descriptions of each mode, please refer to the examples below.
    a. `ECAT_SYNC`
    b. `ECAT_FREERUN`
    c. `ECAT_FREERUN_MANUAL`

Return Value

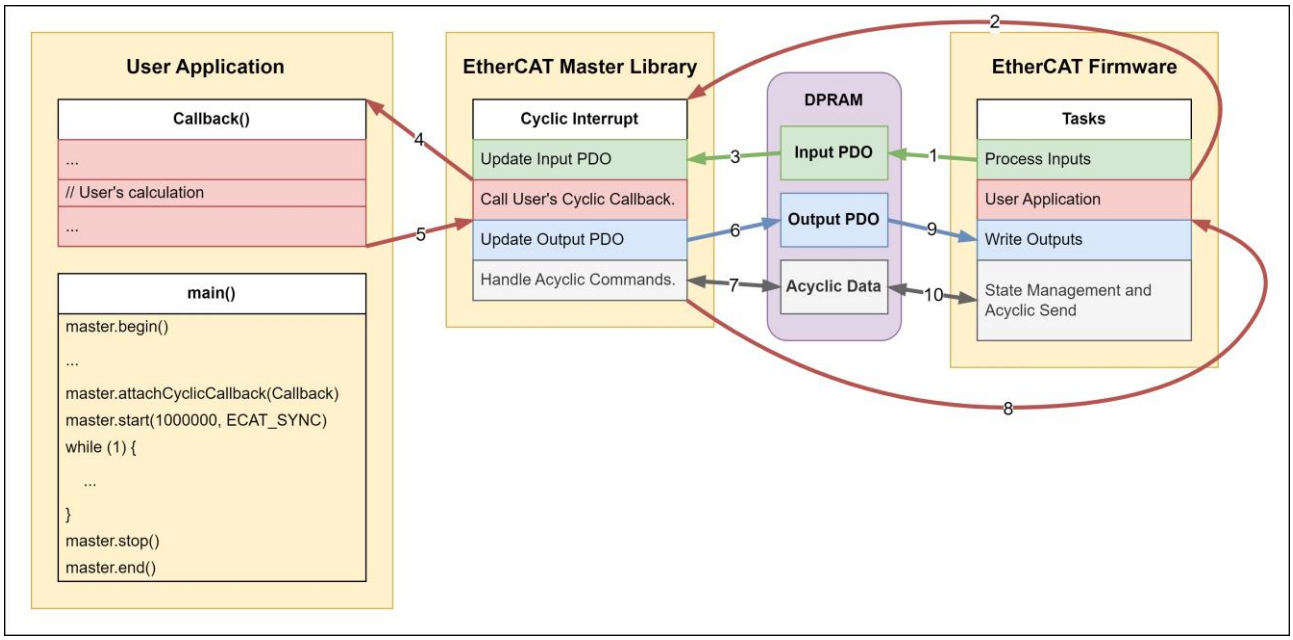
Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Comment

This function is blocking and cannot be called within the callback functions.

Example

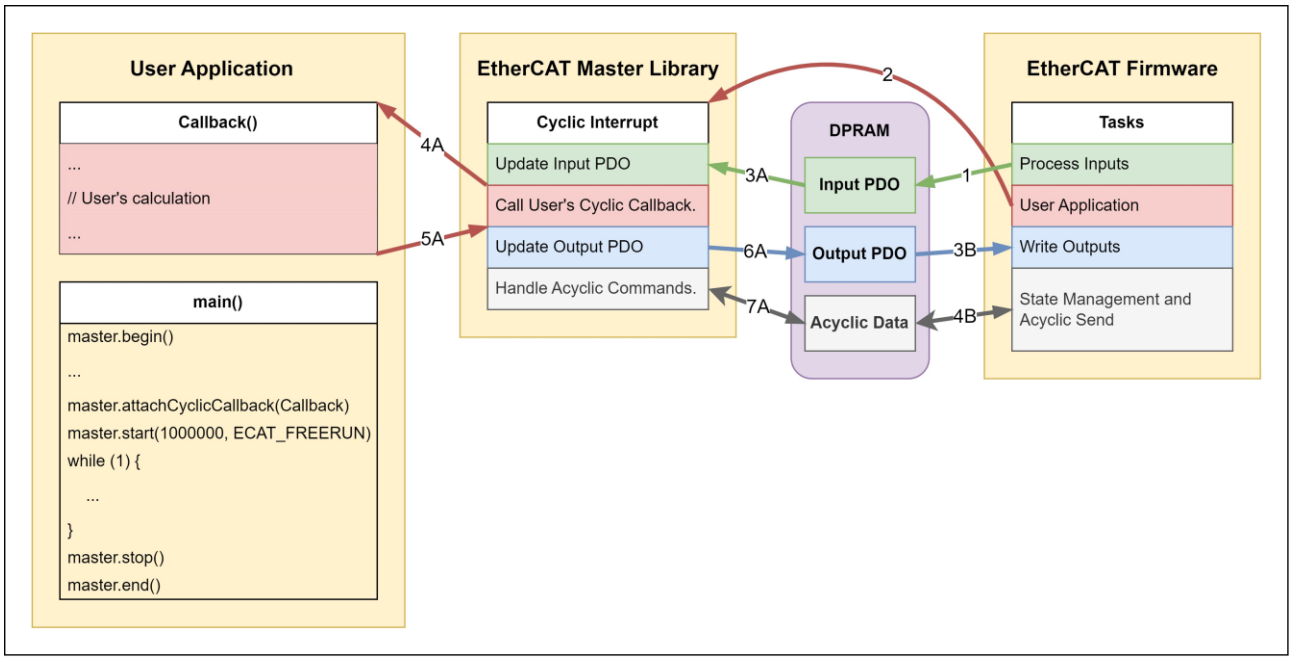
1. ECAT_SYNC



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

```
#include "Ethercat.h"
EthercatMaster master;

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

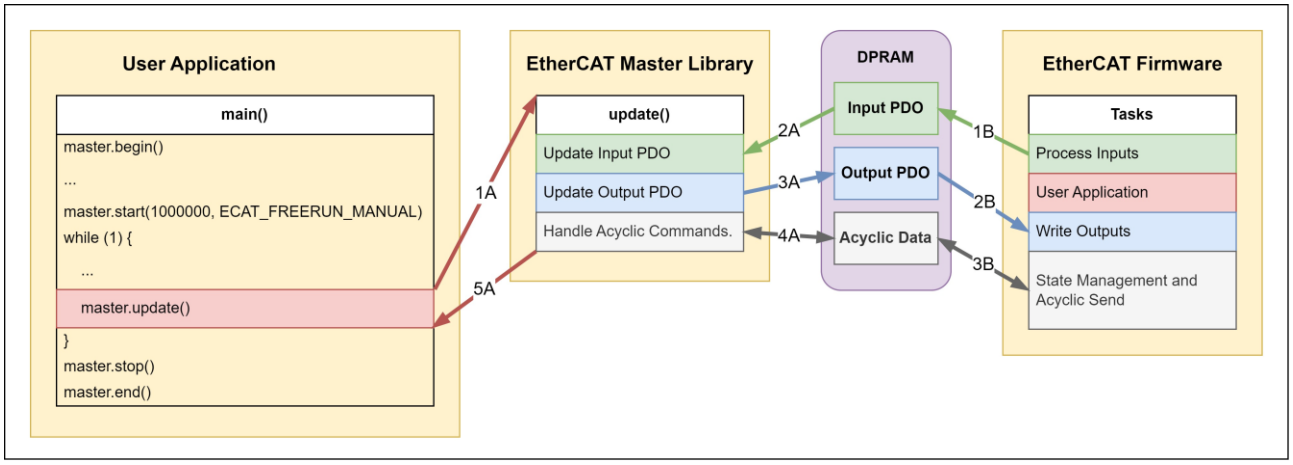
2. ECAT_FREERUN



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

```
#include "Ethercat.h"

EthercatMaster master;

void CyclicCallback() {
  // ...
}

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

void loop() {
}
```

3. ECAT_FREERUN_MANUAL



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

```
#include "Ethercat.h"

EthercatMaster master;

void setup() {
  master.begin();
  master.start(1000000, ECAT_FREERUN_MANUAL);
}
void loop() {
  // ...
  master.update();
}
```

stop()

Description

Stop the EtherCAT master, and switch the EtherCAT state machine to the Pre-Operational state.

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.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;

void setup() {
    master.begin();
    master.start();
    master.stop();
    master.end();
}
void loop() {

}
```

getSystemTime()

Description

Get the system time for the current cycle. It is recommended to use this within the cyclic callback to ensure the system time is retrieved in the correct cycle.

Syntax

```
uint64_t getSystemTime();
```

Parameters

None.

Return Value

Return the system time for the current cycle.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
uint64_t CurrentSystemTime;

void CyclicCallback() {
  CurrentSystemTime = master.getSystemTime();
}

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

  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
```

```
  Serial.print("Current System Time: ");
  Serial.print((uint32_t)(CurrentSystemTime >> 32));
  Serial.print(" ");
  Serial.println((uint32_t)(CurrentSystemTime & 0xFFFFFFFF));

  delay(1000);
}
```

getWorkingCounter()

Description

Get the working counter for the current cycle. It is recommended to use this within the cyclic callback to ensure the working counter is retrieved in the correct cycle.

Syntax

```
int getWorkingCounter();
```

Parameters

None.

Return Value

Return the working counter for the current cycle.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
int CurrentWorkingCounter;

void CyclicCallback() {
  CurrentWorkingCounter = master.getWorkingCounter();
}

void setup() {
  Serial.begin(115200);
  master.begin();
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  Serial.print("Current Working Counter: ");
```

```
  Serial.println(CurrentWorkingCounter);
  delay(1000);
}
```

getExpectedWorkingCounter()

Description

Get the expected working counter.

Syntax

```
int getExpectedWorkingCounter();
```

Parameters

None.

Return Value

Return the expected working counter.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;

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

  Serial.print("Expected Working Counter: ");
  Serial.println(master.getExpectedWorkingCounter());
  // ...
}

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

update()

Description

Update process data and handle acyclic commands. This can only operate under the ECAT_FREERUN_MANUAL control mode.

Syntax

```
void update();
```

Parameters

None.

Return Value

None.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;

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

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

getShiftTime()

Description

Get the Global Shift Time for DC-Synchronous mode. For the definition of Global Shift Time, please refer to Synchronization.

Syntax

```
int getShiftTime();
```

Parameters

None.

Returns

Return the *Global Shift Time*, in nanoseconds.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  slave.setDc(1000000);
  master.start(1000000);

  Serial.print("Global Shift Time: ");
  Serial.println(master.getShiftTime());
}

void loop() {
```

```
  // ...
}
```

setShiftTime()

Description

Set the Global Shift Time for DC-Synchronous mode. If the user does not set the Global Shift Time or sets it to `INT32_MAX`, the EtherCAT firmware will automatically calculate an appropriate value. For the definition of Global Shift Time, please refer to Synchronization.

Syntax

```
int setShiftTime(int32_t nanoseconds);
```

Parameters

- `[in] int32_t nanoseconds`
  The value of the Global Shift Time to be set, in nanoseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()` and before `EthercatMaster::start()`, or after a successful execution of `EthercatMaster::stop()` and before `EthercatMaster::end()`. This function is non-blocking and can be called within the callback functions, but it is not recommended.

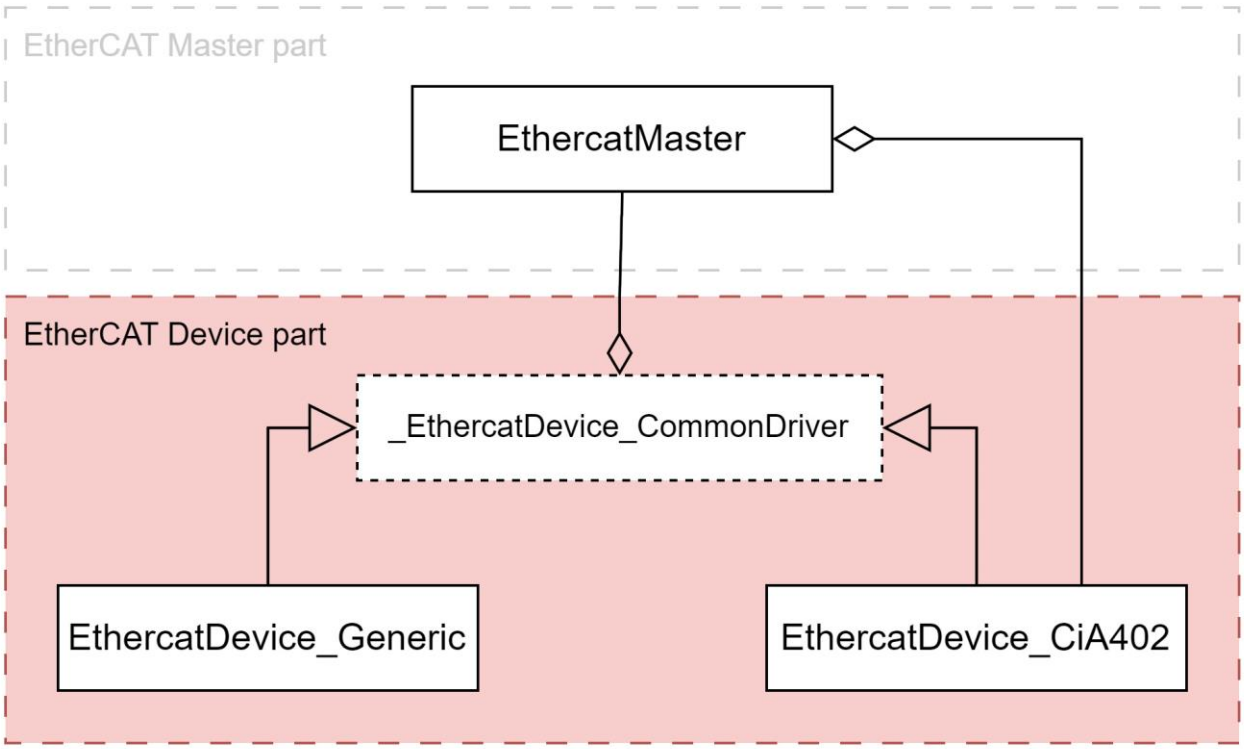
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    master.begin();
    slave.attach(0, master);
    slave.setDc(1000000);
    master.setShiftTime(250000); // 250000 ns
    master.start(1000000);
}
void loop() {
}
```

2.2 EtherCAT Device

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

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



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

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

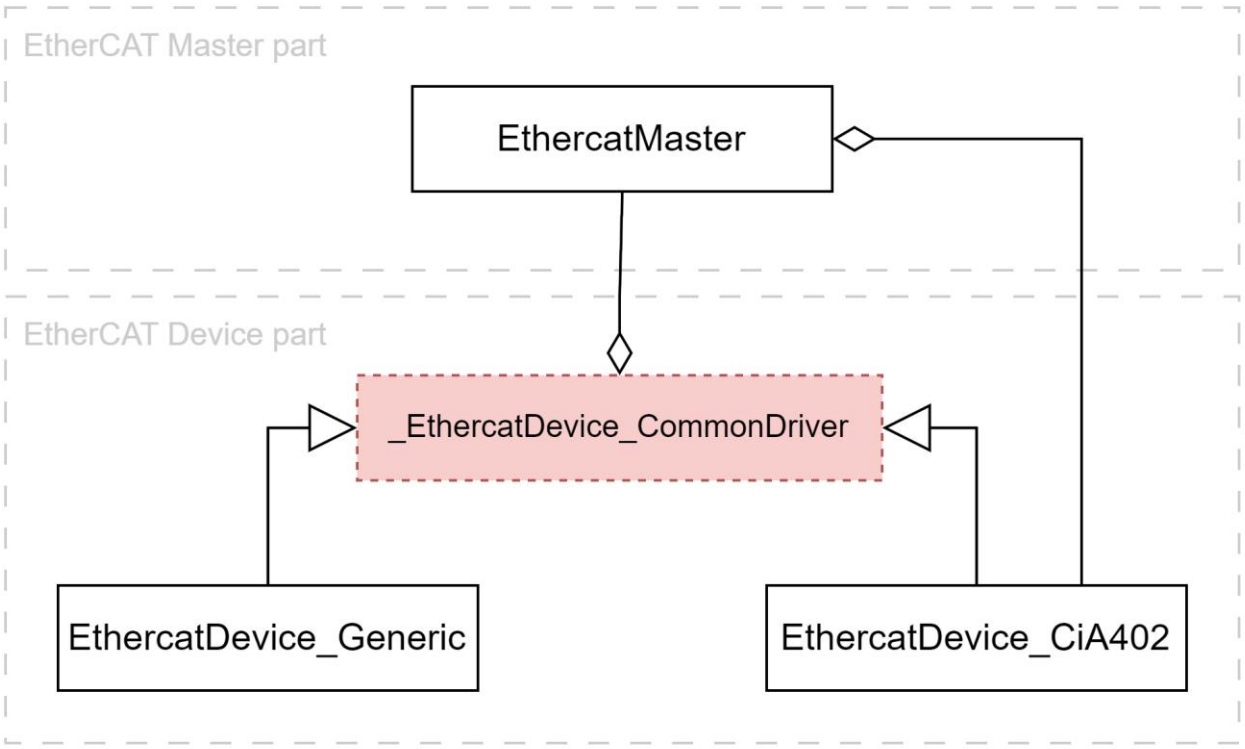
Classes:

- _EthercatDevice_CommonDriver
- EthercatDevice_Generic
- EthercatDevice_CiA402

2.2.1 _EthercatDevice_CommonDriver

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

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



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

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

Function Groups:

- Information
- PDO
- CoE
- FoE
- DC
- SII EEPROM

Information Functions

The library provides functions for obtaining information about EtherCAT slave devices on the network. This includes essential details such as Vendor ID, Product Code, Alias Address, and Device Name, used for device identification.

Moreover, it offers information on whether the EtherCAT slave device supports specific features like CoE, FoE, DC, etc. This slave information enables users to understand the characteristics and capabilities of devices within the network and perform corresponding configuration and control tasks.

Functions:

- getVendorID()
- getProductCode()
- getRevisionNumber()
- getSerialNumber()
- getAliasAddress()
- getSlaveNo()
- getDeviceName()
- getMailboxProtocol()
- getCoEDetails()
- getFoEDetails()
- getEoEDetails()
- getSoEChannels()
- isSupportDC()

getVendorID()

Description

Get the vendor ID of the EtherCAT slave device.

Syntax

`uint32_t getVendorID();`

Parameters

None.

Return Value

Return the vendor ID of the EtherCAT slave device. If there is an error, return `UINT32_MAX` (2^{32} - 1).

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getVendorID());
}

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

}
```

getProductCode()

Description

Get the product code of the EtherCAT slave device.

Syntax

`uint32_t getProductCode();`

Parameters

None.

Return Value

Return the product code of the EtherCAT slave device. If there is an error, return `UINT32_MAX` (2^{32} - 1).

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getProductCode());
}

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

}
```

getRevisionNumber()

Description

Get the revision number of the EtherCAT slave device.

Syntax

```
uint32_t getRevisionNumber();
```

Parameters

None.

Return Value

Return the revision number of the EtherCAT slave device. If there is an error, return `UINT32_MAX` (2^{32} - 1).

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getRevisionNumber());
}

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

}
```

getSerialNumber()

Description

Get the serial number of the EtherCAT slave device.

Syntax

`uint32_t getSerialNumber();`

Parameters

None.

Return Value

Return the serial number of the EtherCAT slave device. If there is an error, return `UINT32_MAX` (2^{32} - 1).

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getSerialNumber());
}

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

}
```

getAliasAddress()

Description

Get the alias address of the EtherCAT slave device.

Syntax

```
int getAliasAddress();
```

Parameters

None.

Return Value

Return the alias address of the EtherCAT slave device. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getAliasAddress());
}

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

}
```

getSlaveNo()

Description

Get the sequence ID of the EtherCAT slave device on the EtherCAT network.

Syntax

```
int getSlaveNo();
```

Parameters

None.

Return Value

Return the sequence ID of the EtherCAT slave device on the EtherCAT network. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getSlaveNo());
}

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

}
```

getDeviceName()

Description

Get the device name of the EtherCAT slave device.

Syntax

```
char *getDeviceName(char *name, size_t len);
```

Parameters

- `[out] char *name`
  The buffer used to store the device name.
- `[in] size_t len`
  The size of the buffer used to store the device name.

Return Value

Return the pointer to the buffer used to store the device name. If the returned value is `NULL`, it indicates an error.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  char name[64];
  Serial.println(slave.getDeviceName(name, 64));
}

void loop() {}
```

getMailboxProtocol()

Description

Get the supported mailbox protocol types by the EtherCAT slave device.

Syntax

```
int getMailboxProtocol();
```

Parameters

None.

Return Value

Return the supported mailbox protocol types.

- Bit 0: ADS over EtherCAT.
- Bit 1: Ethernet over EtherCAT.
- Bit 2: CAN application protocol over EtherCAT.
- Bit 3: File Access over EtherCAT.
- Bit 4: Servo Drive Profile over EtherCAT.
- Bit 5: Vendor specific protocol over EtherCAT.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.print("Mailbox Protocols: ");
  Serial.println(slave.getMailboxProtocol());

}

void loop() {}
```

getCoEDetails()

Description

Get the details about the CAN application protocol over EtherCAT (CoE) supported by the EtherCAT slave device.

Syntax

```
int getCoEDetails();
```

Return Value

Return the details about CoE supported of the EtherCAT slave device.

- Bit 0: Enable SDO.
- Bit 1: Enable SDO Info.
- Bit 2: Enable PDO Assign.
- Bit 3: Enable PDO Configuration.
- Bit 4: Enable PDO Upload at startup.
- Bit 5: Enable SDO complete access.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.print("CoE Details: ");
  Serial.println(slave.getCoEDetails());
}

void loop() {
```

```
  // ...

}
```

getFoEDetails()

Description

Get the details about the File Access over EtherCAT (FoE) supported by the EtherCAT slave device.

Syntax

```
int getFoEDetails();
```

Return Value

Return the details about FoE supported of the EtherCAT slave device.

- Bit 0: Enable FoE.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getFoEDetails());
}

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

}
```

getEoEDetails()

Description

Get the details about the Ethernet over EtherCAT (EoE) supported by the EtherCAT slave device.

Syntax

```
int getEoEDetails();
```

Return Value

Return the details about EoE supported of the EtherCAT slave device.

- Bit 0: Enable EoE.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getEoEDetails());
}

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

}
```

getSoEChannels()

Description

Get the number of Servo Drive Profile over EtherCAT (SoE) channels supported by the EtherCAT slave device.

Syntax

```
int getSoEChannels();
```

Return Value

Return the number of SoE channels supported by the EtherCAT slave device. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.getSoEChannels());
}

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

}
```

isSupportDC()

Description

Determines whether the EtherCAT slave device supports Distributed Clocks (DC).

Syntax

```
int isSupportDC();
```

Parameters

None.

Return Value

Return whether the EtherCAT slave device supports DC. A positive value indicates support, 0 indicates no support, and a negative value indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is non-blocking and can be called within the Cyclic Callback.

Example

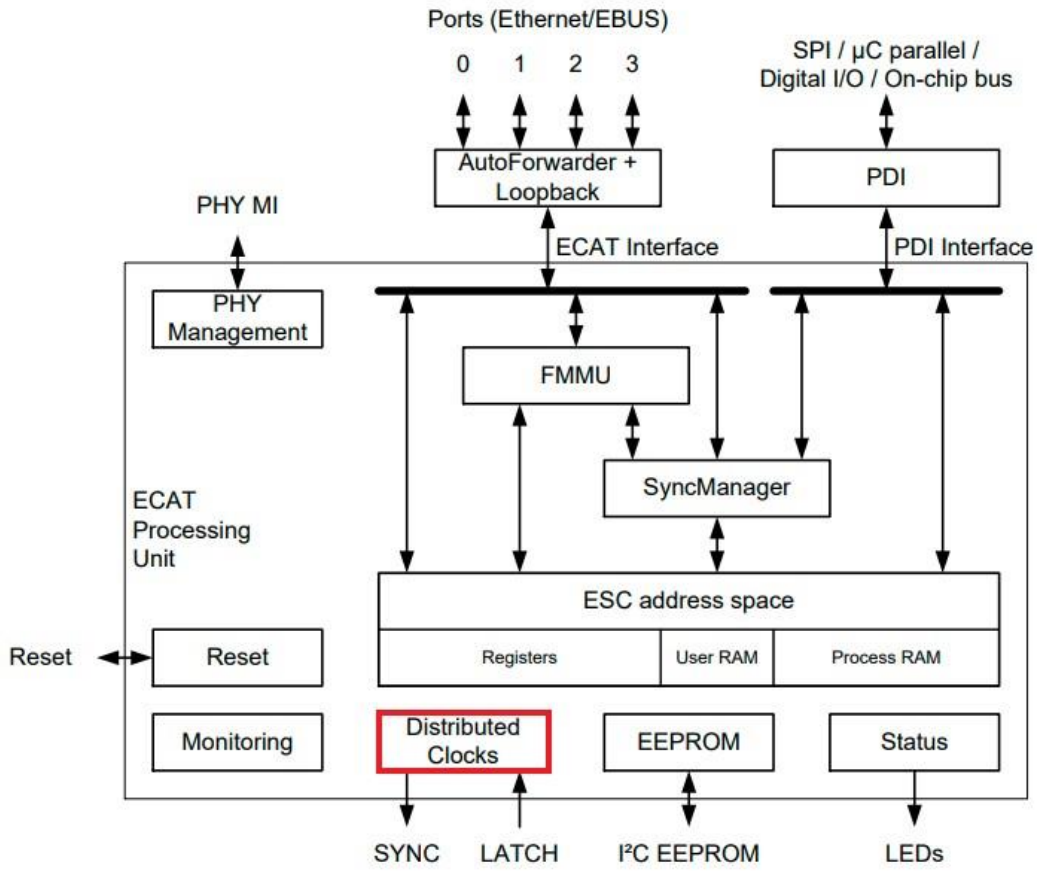
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

  Serial.println(slave.isSupportDC());
}

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

}
```

Distributed Clock (DC) Functions

The **Distributed Clocks (DC)** is an essential functional unit within the EtherCAT Slave Controller (ESC). It is responsible for implementing a time synchronization mechanism across the EtherCAT network, ensuring that all slave devices synchronize their clocks according to a unified time reference, thus ensuring consistency of time across the entire system.

For system synchronization all slaves are synchronized to one Reference Clock. Typically, the first ESC with Distributed Clocks capability after the master within one segment holds the reference time (System Time). This System Time is used as the reference clock to synchronize the DC slave clocks of other devices and of the master. The propagation delays, local clock sources drift, and local clock offsets are taken into account for the clock synchronization.

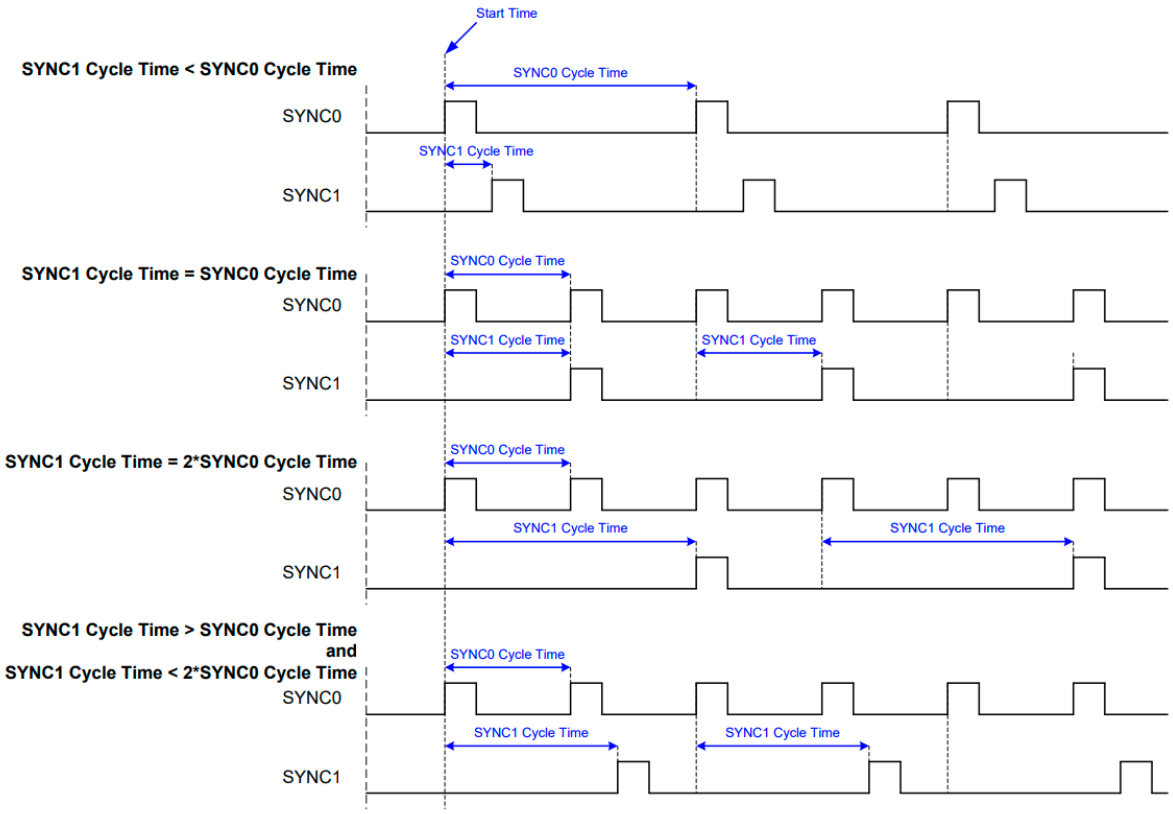
The ESCs can generate SyncSignals for local applications to be synchronized to the EtherCAT System Time. SyncSignals can be used directly (e.g., as interrupts) or for Digital Output updating/Digital Input sampling. Additionally, LatchSignals can be time stamped with respect to the EtherCAT System Time.



The DC unit supports the generation of a base SyncSignal **SYNC0** and a dependent SyncSignal **SYNC1**. The second SyncSignal (SYNC1) depends on SYNC0, it can be enerated with a predefined delay after SYNC0 pulses.

If the SYNC1 Cycle Time is larger than the SYNC0 Cycle Time, it will be generated as follows: when the Start Time Cyclic Operation is reached, a SYNC0 pulse is generated. The SYNC1 pulse is generated after the SYNC0 pulse with a delay of SYNC1 Cycle Time. The next SYNC1 pulse is generated when the next SYNC0 pulse was generated, plus the SYNC1 Cycle Time.

Some example configurations are shown in the following figure:



For more detailed information, please refer to ESC Hardware Data Sheet Section I.

Functions:

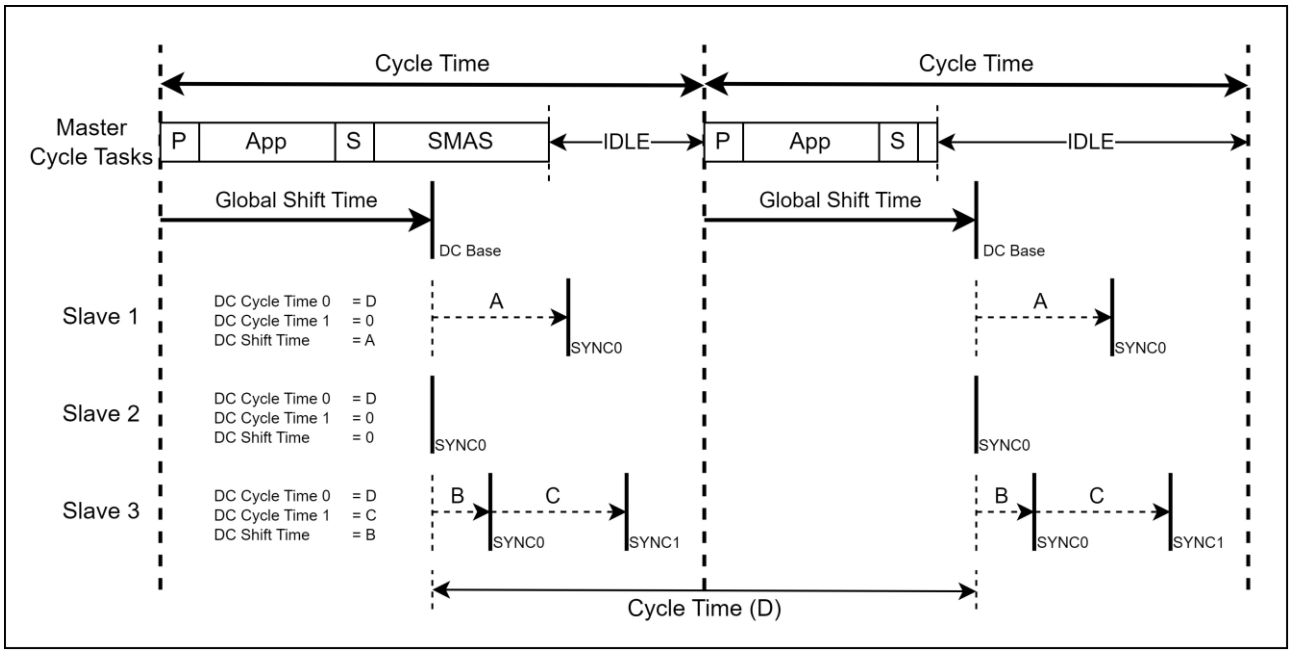
- `setDc()`

`setDc()`

Description

Configure DC parameters of the EtherCAT slave device. This function has three DC parameters to configure:

- ***DC Cycle Time 0*** is used to set the cycle time for the SYNC0 signal, typically aligned with the EtherCAT communication cycle time.
- ***DC Cycle Time 1*** is used to set the cycle time for the SYNC1 signal, which refers to the delay defined after the SYNC0 pulse. This parameter is optional.
- ***DC Shift Time*** is used to set the offset of the SYNC0 signal relative to the DC Base.



Syntax

```
int setDc(uint32_t cycletime0_ns, int32_t shifttime_ns = 0, uint32_t
cycletime1_ns = 0);
```

Parameters

- `[in] uint32_t cycletime0_ns`
  DC SYNC0 cycle time in nanoseconds.
- `[in] int32_t shifttime_ns`
  DC SYNC0 shift time in nanoseconds.
- `[in] uint32_t cycletime1_ns`
  DC SYNC1 cycle time in nanoseconds.

Return Value

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

Comment

The function must be called after EthercatMaster::begin() and before EthercatMaster::start(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.setDc(1000000);
  master.start(1000000);
}

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

}
```

Process Data Objects (PDO) Functions

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

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

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

Functions:

- pdoBitWrite()
- pdoBitRead()
- pdoGetOutputBuffer()
- pdoGetInputBuffer()
- pdoWrite()
- pdoWrite8()
- pdoWrite16()
- pdoWrite32()
- pdoWrite64()
- pdoRead()
- pdoRead8()
- pdoRead16()
- pdoRead32()
- pdoRead64()

pdoBitWrite()

Description

Write the specified bit value to the output process data of such slave device.

Syntax

```
int pdoBitWrite(uint32_t bit_offset, uint8_t value);
```

Parameters

- `[in] uint32_t bit_offset`
  The bit offset value of output process data for such EtherCAT slave device.
- `[in] unit_8_t value`
  The bit value to be written to output process data for such EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoBitWrite(0, 1);
  delay(500);
  slave.pdoBitWrite(0, 0);
  delay(1000);
}
```

pdoBitRead()

Description

Read the specified bit value from the input process data of such slave device.

Syntax

```
int pdoBitRead(uint32_t bit_offset);
```

Parameters

- `[in] uint32_t bit_offset`
  The bit offset value of input process data for such EtherCAT slave device.

Return Value

The specified bit value from the input process data of such slave device. If the returned value is less than zero, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}
void loop() {
  Serial.print("Bit-0 Value: ");
  Serial.println(slave.pdoBitRead(0));
  Serial.print("Bit-7 Value: ");
  Serial.println(slave.pdoBitRead(7));
  delay(1000);
}
```

pdoGetOutputBuffer()

Description

Get the memory pointer of the output process data for such EtherCAT slave device.

Syntax

`uint8_t *pdoGetOutputBuffer();`

Parameters

None.

Return Value

The memory pointer of the output process data for such EtherCAT slave device. If the returned value is `NULL`, it indicates an error code or that such slave device has no output process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t *pointer;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
  pointer = slave.pdoGetOutputBuffer();
}

void loop() {
  pointer[0] = 0x55;
  delay(500);
  pointer[0] = 0xaa;
  delay(1000);
}
```

pdoGetInputBuffer()

Description

Get the memory pointer of the input process data for such EtherCAT slave device.

Syntax

```
uint8_t *pdoGetInputBuffer();
```

Parameters

None.

Return Value

The memory pointer of the input process data for such EtherCAT slave device. If the returned value is NULL, it indicates an error code or that such slave device has no input process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t *pointer;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);

  pointer = slave.pdoGetInputBuffer();
}

void loop() {
  Serial.print("Byte-0 Value: %02X\n");
  Serial.println(pointer[0]);
  delay(1000);
}
```

pdoWrite()

Description

Write the output process data of a certain size which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoWrite(uint32_t offset, void *data, uint32_t size);
```

Parameters

- `[in] uint32_t offset`

  The offset value of output process data for such EtherCAT slave device.

- `[in] void *data`

  The data buffer for writing output process data.

- `[in] uint32_t size`

  The size of the data buffer for writing output process data.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t buffer[4] = {0x00, 0x55, 0xAA, 0xFF};

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  buffer[0] = ~buffer[0];
  buffer[1] = ~buffer[1];
  buffer[2] = ~buffer[2];
```

```
  buffer[3] = ~buffer[3];
  slave.pdoWrite(0, buffer, 4);
  delay(1000);
}
```

pdoWrite8()

Description

Write 8-bit output process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoWrite8(uint32_t offset, uint8_t value);
```

Parameters

- `[in] uint32_t offset`

  The offset value of output process data for such EtherCAT slave device.

- `[in] uint8_t value`

  The 8-bit value to be written to output process data for such EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoWrite8(0, 0x55);
  delay(500);
  slave.pdoWrite8(0, 0xAA);
  delay(1000);
}
```

pdoWrite16()

Description

Write 16-bit output process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoWrite16(uint32_t offset, uint16_t value);
```

Parameters

- `[in] uint32_t offset`
  The offset value of output process data for such EtherCAT slave device.
- `[in] uint16_t value`
  The 16-bit value to be written to output process data for such EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoWrite16(0, 0x5555);
  delay(500);
  slave.pdoWrite16(0, 0xAAAA);
  delay(1000);
}
```

pdoWrite32()

Description

Write 32-bit output process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoWrite32(uint32_t offset, uint32_t value);
```

Parameters

- `[in] uint32_t offset`
  The offset value of output process data for such EtherCAT slave device.
- `[in] uint32_t value`
  The 32-bit value to be written to output process data for such EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoWrite32(0, 0x55555555);
  delay(500);
  slave.pdoWrite32(0, 0xAAAAAAAA);
  delay(1000);
}
```

pdoWrite64()

Description

Write 64-bit output process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoWrite64(uint32_t offset, uint64_t value);
```

Parameters

- `[in] uint32_t offset`
  The offset value of output process data for such EtherCAT slave device.
- `[in] uint64_t value`
  The 64-bit value to be written to output process data for such EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoWrite64(0, 0x5555555555555555ULL);
  delay(500);
  slave.pdoWrite64(0, 0xAAAAAAAAAAAAAAAAULL);
  delay(1000);
}
```

pdoRead()

Description

Read the input process data of a certain size which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int pdoRead(uint32_t offset, void *data, uint32_t size);
```

Parameters

- `[in] uint32_t offset`
  The offset value of input process data for such EtherCAT slave device.
- `[out] void *data`
  The data buffer for reading input process data.
- `[in] uint32_t size`
  The size of the data buffer for reading input process data.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t buffer[4];

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.pdoRead(0, buffer, 4);
  Serial.print("Buffer: ");
```

```
  Serial.print(buffer[0]);
  Serial.print(buffer[1]);
  Serial.print(buffer[2]);
  Serial.println(buffer[3]);
  delay(1000);
}
```

pdoRead8()

Description

Read 8-bit input process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
uint8_t pdoRead8(uint32_t offset);
```

Parameters

- `[in] uint32_t offset`
  The offset value of input process data for such EtherCAT slave device.

Return Value

Return the 8-bit input process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup(){
    Serial.begin(115200);
    master.begin();
    slave.attach(0, master);
    master.start();
}
void loop() {
    slave.pdoWrite8(0, 0xFF); /* Write value of byte offset 0 of
slave's output process data. */
    Serial.println(slave.pdoRead8(0), HEX); /* Read value of byte
offset 0 of slave's input process data. */
    delay(100);
}
```

pdoRead16()

Description

Read 16-bit input process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
uint16_t pdoRead16(uint32_t offset);
```

Parameters

- `[in] uint32_t offset`
  The offset value of input process data for such EtherCAT slave device.

Return Value

Return the 16-bit input process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.pdoRead16(0));
  delay(1000);
}
```

pdoRead32()

Description

Read 32-bit input process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
uint32_t pdoRead32(uint32_t offset);
```

Parameters

- `[in] uint32_t offset`
  The offset value of input process data for such EtherCAT slave device.

Return Value

Return the 32-bit input process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.pdoRead32(0));
  delay(1000);
}
```

pdoRead64()

Description

Read 64-bit input process data which starting from the specified offset for such EtherCAT slave device.

Syntax

```
uint64_t pdoRead64(uint32_t offset);
```

Parameters

- `[in] uint32_t offset`
  The offset value of input process data for such EtherCAT slave device.

Return Value

Return the 64-bit input process data.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

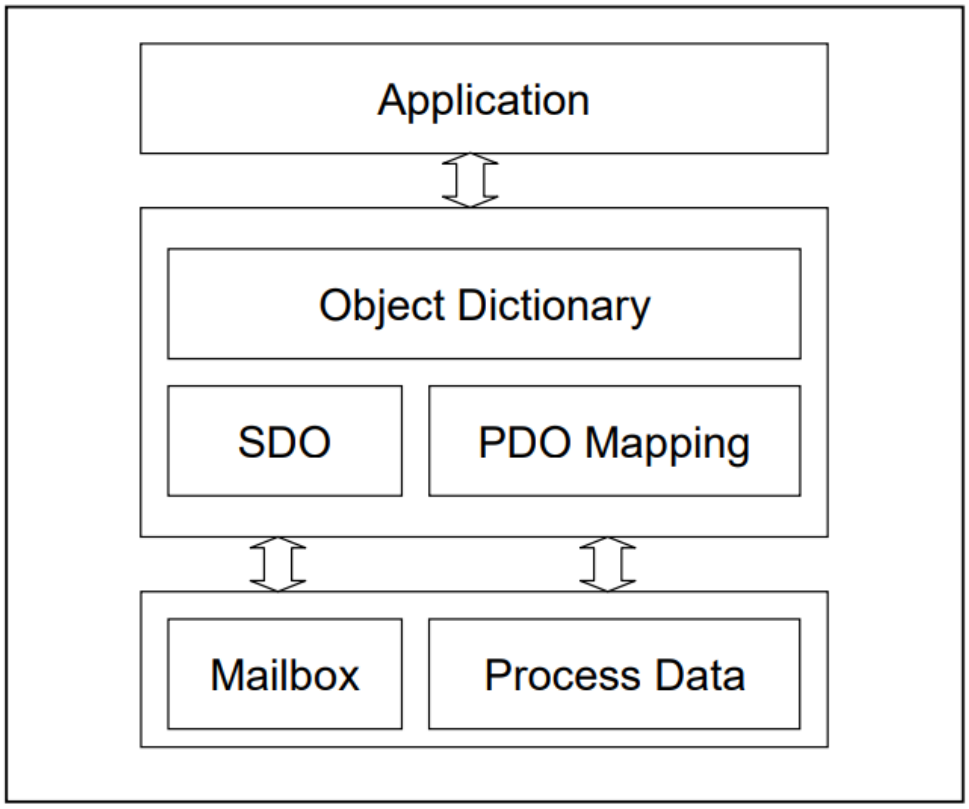
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  uint64_t value = slave.pdoRead64(0);
  char buffer[21];
  sprintf(buffer, "%llu", value);

  Serial.println(buffer);
  delay(1000);
}
```

CANopen over EtherCAT (CoE) Functions

CANopen is a high-level communication protocol based on the Controller Area Network (CAN) bus, commonly used for communication between control systems and devices in industrial applications. It defines a set of communication objects, data types, and network management functions to facilitate data exchange, configuration, and control between devices.

The CANopen protocol includes the following aspects:

- **Object Dictionary**
  Defines all data objects and parameters exchanged between devices. The object dictionary encompasses various types of objects such as variables, parameters, events, and functions.
- **PDO (Process Data Object)**
  Used for real-time data transmission. PDOs allow devices to transmit data between each other in a fixed or event-triggered manner, enabling real-time control and data exchange.
- **SDO (Service Data Object)**
  Used for configuring and managing device parameters. SDOs provide functionalities for reading, writing, and parameter configuration, allowing devices to dynamically exchange configuration information.



CoE (CAN application over EtherCAT) is a CANopen protocol based on the EtherCAT network. It enables communication using the CANopen protocol over EtherCAT networks. The Object Dictionary contains parameters, application data and the mapping information between process data interface and application date (PDO mapping). Its entries can be accessed via Service Data Objects (SDO).
The SDO services primarily consist of two types of commands. The SDO command is utilized for accessing objects stored in the Object Dictionary, while the SDO information command is employed to retrieve details about these objects.

Functions:

- SDO commands
  - sdoDownload()
  - sdoDownload8()
  - sdoDownload16()
  - sdoDownload32()
  - sdoDownload64()
  - sdoUpload()
  - sdoUpload8()
  - sdoUpload16()
  - sdoUpload32()
  - sdoUpload64()
- SDO Information commands
  - getODlist()
  - getObjectDescription()
  - getEntryDescription()

sdoDownload()

Description

Performs a CoE SDO download to the specified object for such EtherCAT slave device.

Syntax

```
int sdoDownload(
    uint16_t    od_index,
    uint8_t     od_subindex,
    void        *  data,
    uint32_t    size,
    uint32_t    *  abortcode = NULL,
    bool        complete_access = false,
    uint32_t    timeout_us = 100000
);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object. 0 or 1 if Complete Access.
- `[in] void data`
  The data buffer for writing.
- `[in] uint32_t size`
  The size of the data buffer for writing.
- `[out] uint32_t abortcode`
  The pointer of the variable used to store the SDO Abort Code.
- `[in] bool complete_access`
  Use Complete Access or not.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
```

```
EthercatDevice_Generic slave;

uint16_t value = 0x000F;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.sdoDownload(0x6040, 0x00, &value, sizeof(value));
  delay(1000);
}
```

sdoDownload8()

Description

Write 8-bit value to the specified object for such EtherCAT slave device.

Syntax

```
int sdoDownload8(uint16_t od_index, uint8_t od_subindex, uint8_t value,
uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint8_t value`
  The 8-bit value to be written to object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}
void loop() {
  slave.sdoDownload8(0x6040, 0x00, 0x0F);
  delay(1000);
}
```

sdoDownload16()

Description

Write 16-bit value to the specified object for such EtherCAT slave device.

Syntax

```
int sdoDownload16(uint16_t od_index, uint8_t od_subindex, uint16_t value, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint16_t value`
  The 16-bit value to be written to object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.sdoDownload16(0x6040, 0x00, 0x000F);
  delay(1000);
}
```

sdoDownload32()

Description

Write 32-bit value to the specified object for such EtherCAT slave device.

Syntax

```
int sdoDownload32(uint16_t od_index, uint8_t od_subindex, uint32_t value, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint32_t value`
  The 32-bit value to be written to object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.sdoDownload32(0x607A, 0x00, 1000);
  delay(1000);
  slave.sdoDownload32(0x607A, 0x00, 0);
  delay(1000);
```

```
}
```

sdoDownload64()

Description

Write 64-bit value to the specified object for such EtherCAT slave device.

Syntax

```
int sdoDownload64(uint16_t od_index, uint8_t od_subindex, uint64_t value, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint64_t value`
  The 64-bit value to be written to object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.sdoDownload64(0x5000, 0x00, 100000);
  delay(1000);
  slave.sdoDownload64(0x5000, 0x00, 0);
  delay(1000);
```

```
}
```

sdoUpload()

Description

Performs a CoE SDO upload to the specified object for such EtherCAT slave device.

Syntax

```
int sdoUpload(
    uint16_t    od_index,
    uint8_t     od_subindex,
    void     *  data,
    uint32_t    size,
    uint32_t *  abortcode = NULL,
    bool        complete_access = false,
    uint32_t    timeout_us = 100000
);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object. 0 or 1 if Complete Access.
- `[out] void *data`
  The data buffer for reading.
- `[in] uint32_t size`
  The size of the data buffer for reading.
- `[out] uint32_t *abortcode`
  The pointer of the variable used to store the SDO Abort Code.
- `[in] bool complete_access`
  Use Complete Access or not. The default is false.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
```

```
EthercatDevice_Generic slave;

uint16_t value;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  slave.sdoUpload(0x6040, 0x00, &value, sizeof(value));
  Serial.print("Value: ");
  Serial.println(value);
  delay(1000);
}
```

sdoUpload8()

Description

Read 8-bit value from the specified object for such EtherCAT slave device.

Syntax

```
uint8_t sdoUpload8(uint16_t od_index, uint8_t od_subindex, uint32_t
timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

Return the 8-bit value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.sdoUpload8(0x6040, 0x00));
  delay(1000);
}
```

sdoUpload16()

Description

Read 16-bit value from the specified object for such EtherCAT slave device.

Syntax

```
uint16_t sdoUpload16(uint16_t od_index, uint8_t od_subindex, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

Return the 16-bit value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.sdoUpload16(0x6040, 0x00));
  delay(1000);
}
```

sdoUpload32()

Description

Read 32-bit value from the specified object for such EtherCAT slave device.

Syntax

```
uint32_t sdoUpload32(uint16_t od_index, uint8_t od_subindex, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

Return the 32-bit value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.sdoUpload32(0x607A, 0x00));
  delay(1000);
}
```

sdoUpload64()

Description

Read 64-bit value from the specified object for such EtherCAT slave device.

Syntax

```
uint64_t sdoUpload64(uint16_t od_index, uint8_t od_subindex, uint32_t timeout_us = 100000);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

Return the 64-bit value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  master.begin();
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  uint64_t value = slave.sdoUpload64(0x5000, 0x00);

  uint32_t highPart = (uint32_t)(value >> 32);
  uint32_t lowPart = (uint32_t)(value & 0xFFFFFFFF);
```

```
  Serial.print("Value: ");
  Serial.print(highPart, HEX);
  Serial.print(lowPart, HEX);
  Serial.println();

  delay(1000);
}
```

getODlist()

Description

Gets a list of objects existing in the object dictionary for such EtherCAT slave device.

Syntax

```
int getODlist(
    uint16_t *  list,
    uint32_t    list_size,
    uint32_t *  abortcode = NULL,
    uint32_t    timeout_us = 100000
);
```

Parameters

- `[out] uint16_t *list`
  The data buffer used to read the list of objects.
- `[in] uint32_t list_size`
  The number of objects can be stored in the data buffer.
- `[out] uint32_t *abortcode`
  The pointer of the variable used to store the SDO Abort Code.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

Return the number of objects in the object list. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint16_t ODlist[1024];
int rc;

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

  master.begin();
```

```
  slave.attach(0, master);
  master.start(1000000);
}

void loop() {
  rc = sizeof(ODlist) / sizeof(ODlist[0]);
  rc = slave.getODlist(ODlist, rc);
  for (int i = 0; i < rc; i++) {
    Serial.print("Index ");
    Serial.println(ODlist[i]);
  }
}
```

getObjectDescription()

Description

Reads the object description of the specified object addressed by index for such EtherCAT slave device.

Syntax

```
int getObjectDescription(
    uint16_t    od_index,
    uint16_t *  datatype,
    uint8_t  *  max_od_subindex,
    uint8_t  *  objcode,
    char     *  objname,
    size_t      objname_size,
    uint32_t *  abortcode = NULL,
    uint32_t    timeout_us = 100000
);

int getObjectDescription(
    uint16_t    od_index,
    uint16_t &  datatype,
    uint8_t  &  max_od_subindex,
    uint8_t  &  objcode,
    char     *  objname,
    size_t      objname_size,
    uint32_t *  abortcode = NULL,
    uint32_t    timeout_us = 100000
);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[out] uint16_t *datatype`
  The variable used to store the data type. Please refer to Data Type.
- `[out] uint8_t *max_od_subindex`
  Maximum number of subindexes of the object.
- `[out] uint8_t *objcode`
  Object code.
  7: Variable
  8: Array
  9: Record

- `[out] char objname`
  Name of the object. The buffer used to store the object name.
- `[in] size_t objname_size`
  The size of the buffer for the object name.
- `[out] uint32_t *abortcode`
  The pointer of the variable used to store the SDO Abort Code.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint16_t DataType;
uint8_t MaxSubindex, ObjectCode;
char ObjectName[64];

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

  master.begin();
  slave.attach(0, master);
  slave.getObjectDescription(0x1C12, DataType, MaxSubindex, ObjectCode,
ObjectName, sizeof(ObjectName));
  Serial.print("Data Type: ");
  Serial.println(DataType);
  Serial.print("Object Code: ");
  Serial.println(ObjectCode);
  Serial.print("Max Subindex: ");
  Serial.println(MaxSubindex);
  Serial.print("Object Name: ");
  Serial.println(ObjectName);
}
```

```
void loop() {

}
```

getEntryDescription()

Description

Reads the entry description of the specified object addressed by index and subindex for such EtherCAT slave device.

Syntax

```
int getEntryDescription(
    uint16_t    od_index,
    uint8_t     od_subindex,
    uint8_t  *  valueinfo,
    uint16_t *  datatype,
    uint16_t *  bitlength,
    uint16_t *  objaccess,
    char     *  entryname,
    size_t      entryname_size,
    uint32_t *  abortcode = NULL,
    uint32_t    timeout_us = 100000
);

int getEntryDescription(
    uint16_t    od_index,
    uint8_t     od_subindex,
    uint8_t  &  valueinfo,
    uint16_t &  datatype,
    uint16_t &  bitlength,
    uint16_t &  objaccess,
    char     *  entryname,
    size_t      entryname_size,
    uint32_t *  abortcode = NULL,
    uint32_t    timeout_us = 100000
);
```

Parameters

- `[in] uint16_t od_index`
  Index of the object.
- `[in] uint8_t od_subindex`
  Subindex of the object.
- `[in] uint8_t valueinfo`
  NOTE: The parameter is currently invalid in the current version. The value info includes which elements shall be returned:

Bit 0: reserved
Bit 1: reserved
Bit 2: reserved
Bit 3: unit type
Bit 4: default value
Bit 5: minimum value
Bit 6: maximum value

- `[out] uint16_t datatype`
  The variable used to store the data type. Please refer to Data Type.
- `[out] uint16_t bitlength`
  Bit length of the object.
- `[out] uint16_t objaccess`
  The attribute of access.
  Bit 0: read access in Pre-Operational state
  Bit 1: read access in Safe-Operational state
  Bit 2: read access in Operational state
  Bit 3: write access in Pre-Operational state
  Bit 4: write access in Safe-Operational state
  Bit 5: write access in Operational state
  Bit 6: object is mappable in a RxPDO
  Bit 7: object is mappable in a TxPDO
  Bit 8: object can be used for backup
  Bit 9: object can be used for settings
  Bit 10-15: reserved
- `[out] char entryname`
  Name of the object entry. The buffer used to store the entry name.
- `[in] size_t entryname_size`
  The size of the buffer for the entry name.
- `[out] uint32_t abortcode`
  The pointer of the variable used to store the SDO Abort Code.
- `[in] uint32_t timeout_us`
  Timeout in microseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

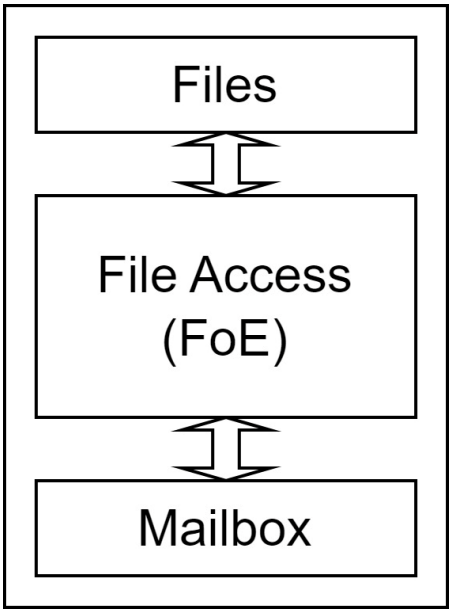
Example

```
#include "Ethercat.h"
```

```
EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  uint16_t DataType, BitLength, ObjectAccess;
  uint8_t ValueInfo = 0;
  char EntryName[64];

  Serial.begin(115200);

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

  slave.getEntryDescription(0x1C12, 0x01, ValueInfo, DataType,
BitLength, ObjectAccess, EntryName, sizeof(EntryName));
  Serial.print("Data Type      : ");
  Serial.print(DataType, HEX);
  Serial.println("h");
  Serial.print("Bit Length     : ");
  Serial.print(BitLength, HEX);
  Serial.println("h");
  Serial.print("Object Access   : ");
  Serial.print(ObjectAccess, HEX);
  Serial.println("h");
  Serial.print("Entry Name     : ");
  Serial.println(EntryName);
}

void loop() {
  // Do nothing here
}
```

File over EtherCAT (FoE) Functions

File access over EtherCAT (FoE) is a protocol extension of EtherCAT that enables file transfer capabilities over the EtherCAT network. It specifies a standard way to download a firmware or any other files to the EtherCAT slave device or to upload a firmware or any other files from the EtherCAT slave device.



Functions:

- readFoE()
- writeFoE()

readFoE()

Description

Read a file from the EtherCAT slave device using FoE.

Syntax

```
int readFoE(char *filename, uint32_t password, void *data, uint32_t size, uint32_t timeout_ms = 2000);
```

Parameters

- `[in] char *filename`
  Name of the file to be read.
- `[in] uint32_t password`
  32-bit password value. If the password is equal to zero, it indicates that the password is unused.
- `[in] void *data`
  The file data buffer to be read.
- `[in] uint32_t size`
  The size of the file data buffer to be read.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

Return the size of the file to be read. If the return value is less than 0, it indicates an error code.

Comment

The function must be called after `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

#define MAX_FILE_DATA_SIZE  (2 * 1024 * 1024)

EthercatMaster master;
EthercatDevice_Generic slave;

char destination[] = {"destination.bin"};
char filename[] = {"firmware.bin"};
```

```
uint32_t password = 0;
uint8_t *filedata;
int filesize;
FILE *file;

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

  filedata = (uint8_t *)malloc(MAX_FILE_DATA_SIZE);
  if (filedata != NULL) {
    filesize = slave.readFoE(filename, password, filedata,
MAX_FILE_DATA_SIZE);
    file = fopen(destination, "wb");
    if (file != NULL) {
      fwrite(filedata, sizeof(uint8_t), filesize, file);
      fclose(file);
    }
    free(filedata);
  }

}

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

}
```

writeFoE()

Description

Write a file to the EtherCAT slave device using FoE.

Syntax

```
int writeFoE(char *filename, uint32_t password, void *data, uint32_t size, uint32_t timeout_ms = 2000);
```

Parameters

- `[in] char *filename`
  Name of the file to be written.
- `[in] uint32_t password`
  32-bit password value. If the password is equal to zero, it indicates that the password is unused.
- `[in] void *data`
  The file data buffer to be written.
- `[in] uint32_t size`
  The size of the file data buffer to be written.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

char source[] = {"source.bin"};
char filename[] = {"firmware.bin"};
uint32_t password = 0;
uint8_t *filedata;
int filesize;
FILE *file;

void setup() {
```

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

  file = fopen(source, "rb");
  if (file != NULL) {
    fseek(file, 0, SEEK_END);
    filesize = ftell(file);
    fseek(file, 0, SEEK_SET);
    filedata = (uint8_t *)malloc(filesize);
    if (filedata != NULL) {
      fread(filedata, sizeof(uint8_t), filesize, file);
      slave.writeFoE(filename, password, filedata, filesize);
      free(filedata);
    }
    fclose(file);
  }
}

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

}
```

SII EEPROM Functions

The following APIs are still under development and are not recommended for use. EtherCAT slave controllers use a mandatory NVRAM, typically a serial EEPROM with I²C interface, to store EtherCAT Slave Information (ESI).

This information includes Vendor ID, Product Code, Mailbox Configuration, FMMU, PDO, and so on. EEPROM sizes from 1 Kbit up to 4 Mbit are supported, depending on the ESC.

The ESC Configuration Area (EEPROM word addresses 0 to 7) is automatically read by the ESC after power-on or reset. It contains the PDI configuration, DC settings, and the Configured Station Alias. The consistency of the ESC Configuration data is secured with a checksum. For more detailed information, please refer to ESC Hardware Data Sheet Section I.

Functions:

- `readSII()`
- `readSII8()`
- `readSII16()`
- `readSII32()`
- `writeSII()`
- `writeSII8()`
- `writeSII16()`
- `writeSII32()`

readSII()

Description

Read the data of SII EEPROM of a certain size which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int readSII(uint32_t offset, void *data, size_t len, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] void *data`
  The data buffer for reading SII EEPROM.
- `[in] size_t len`
  The size of the data buffer for reading SII EEPROM.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t buffer[4];

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

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

```
}

void loop() {
  slave.readSII(0, buffer, 4);  // Read SII data
  Serial.print("Buffer: ");
  Serial.print(buffer[0], HEX);
  Serial.print(", ");
  Serial.print(buffer[1], HEX);
  Serial.print(", ");
  Serial.print(buffer[2], HEX);
  Serial.print(", ");
  Serial.println(buffer[3], HEX);
  delay(1000);
}
```

readSII8()

Description

Read 8-bit data of SII EEPROM starting from the specified offset for the slave device.

Syntax

```
uint8_t readSII8(uint32_t offset, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

Return the 8-bit data of SII EEPROM.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
    int readSII8;
    Serial.begin(115200);
    while (!Serial);
    master.begin();
    slave.attach(0, master);
    master.start();
    readSII8 = slave.readSII8(0);
    Serial.println(readSII8);
}
void loop() {

}
```

readSII16()

Description

Read 16-bit data of SII EEPROM starting from the specified offset for the slave device.

Syntax

```
uint16_t readSII16(uint32_t offset, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

Return the 16-bit data of SII EEPROM.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.readSII16(0), HEX);
  delay(1000);
}
```

readSII32()

Description

Read 32-bit data of SII EEPROM starting from the specified offset for the slave device.

Syntax

```
uint32_t readSII32(uint32_t offset, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

Return the 32-bit data of SII EEPROM.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

void loop() {
  Serial.print("Value: ");
  Serial.println(slave.readSII32(0), HEX);
  delay(1000);
}
```

writeSII()

Description

Write the data of SII EEPROM of a certain size which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int writeSII(uint32_t offset, void *data, size_t len, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] void *data`
  The data buffer for writing SII EEPROM.
- `[in] size_t len`
  The size of the data buffer for writing SII EEPROM.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

uint8_t buffer[4] = {0x00, 0x55, 0xAA, 0xFF};

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.writeSII(64, buffer, 4);
}
```

```
void loop() {

}
```

writeSII8()

Description

Write 8-bit data of SII EEPROM which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int writeSII8(uint32_t offset, uint8_t value, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint8_t value`
  The 8-bit value to be written to SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.writeSII8(64, 0x55);
}

void loop() {

}
```

writeSII16()

Description

Write 16-bit data of SII EEPROM which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int writeSII16(uint32_t offset, uint16_t value, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint16_t value`
  The 16-bit value to be written to SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"
EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.writeSII16(64, 0x5555);
}

void loop() {

}
```

writeSII32()

Description

Write 32-bit data of SII EEPROM which starting from the specified offset for such EtherCAT slave device.

Syntax

```
int writeSII16(uint32_t offset, uint32_t value, uint32_t timeout_ms = 500);
```

Parameters

- `[in] uint32_t offset`
  The offset value of SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t value`
  The 32-bit value to be written to SII EEPROM for such EtherCAT slave device.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

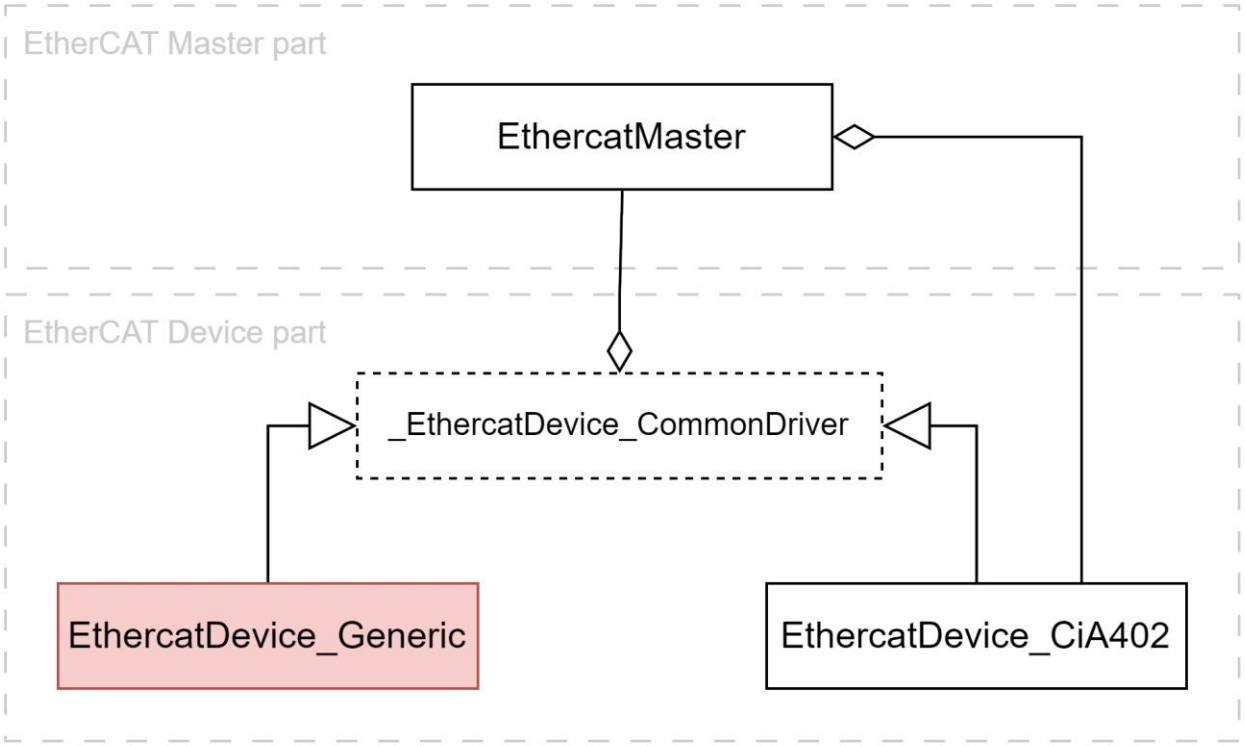
Example

```
#include "Ethercat.h"
EthercatMaster master;
EthercatDevice_Generic slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.writeSII32(64, 0x55555555);
}

void loop() {

}
```

2.2.2 EthercatDevice_Generic

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

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



Relationship

- *EthercatDevice_Generic* inherits from *_EthercatDevice_CommonDriver*.

Base Class

- _EthercatDevice_CommonDriver

Function Groups:

- Initialization

Initialization Functions

Initialization-related functions for the EthercatDevice_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of *EthercatMaster* class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of *EthercatMaster* class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

```
    master.start();
}

void loop() {
  //...

}
```

detach()

Description

Uninstall the slave object.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatDevice_Generic::attach().

WARNING: Prohibited from being called within the callback functions.

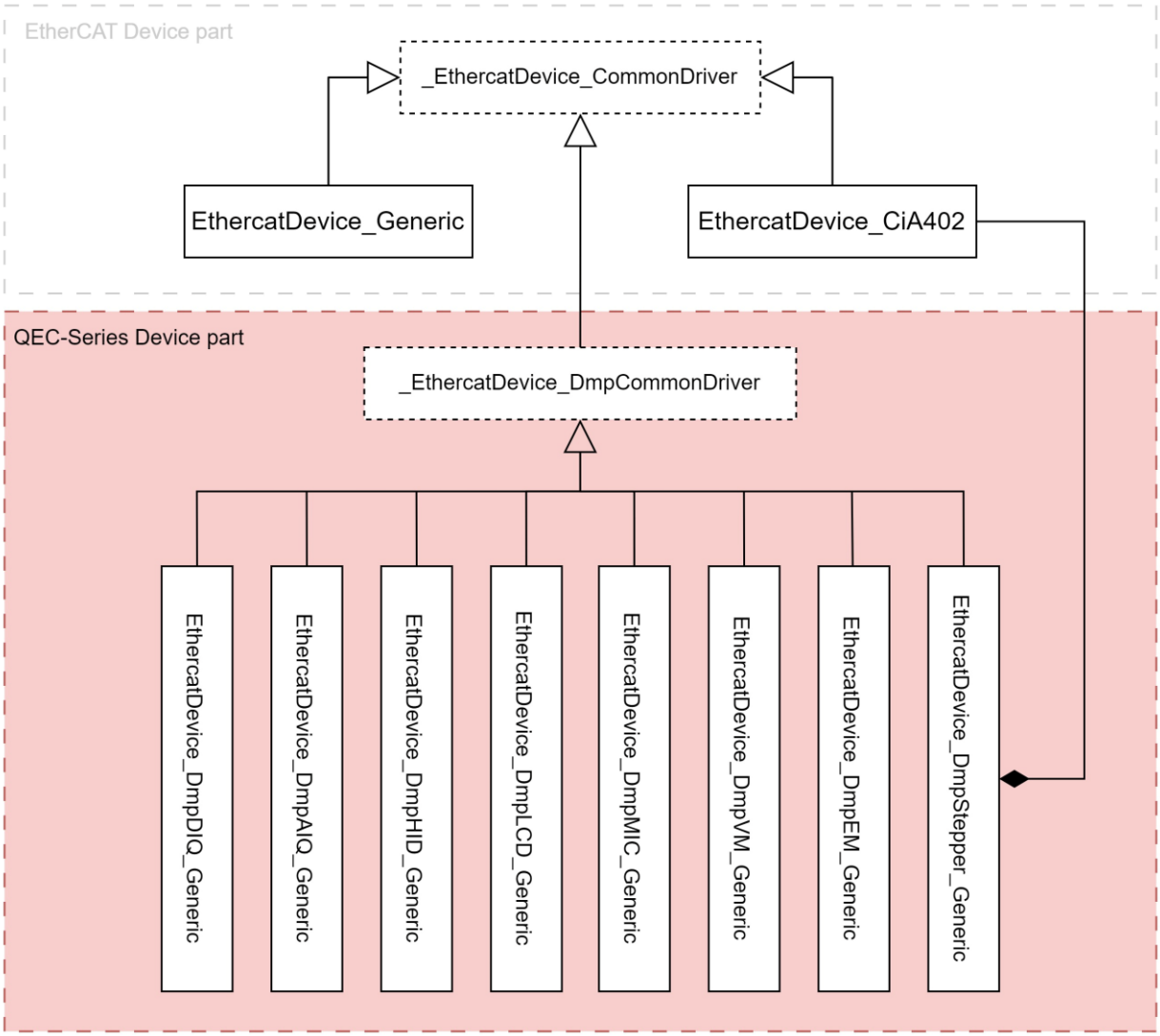
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

  delay(3000);
  slave.detach();
  master.end();
}

void loop() {
  // do something here.
}
```

2.3 QEC-Series Device

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

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



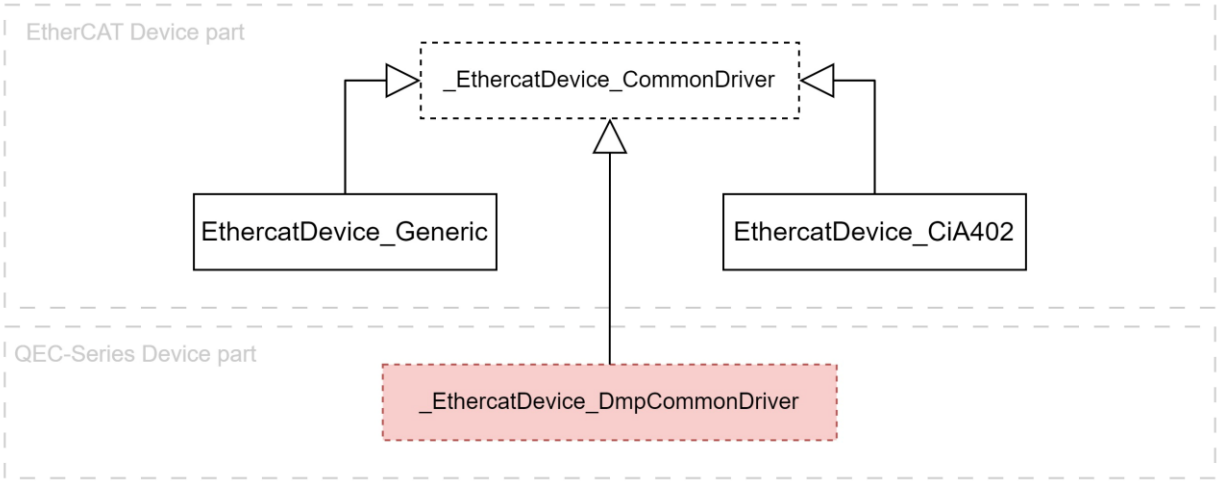
Classes:

- _EthercatDevice_DmpCommonDriver
- EthercatDevice_DmpDIQ_Generic
- EthercatDevice_DmpAIQ_Generic
- EthercatDevice_DmpHID_Generic
- EthercatDevice_DmpLCD_Generic
- EthercatDevice_DmpEM_Generic
- EthercatDevice_DmpStepper_Generic

2.3.1 _EthercatDevice_DmpCommonDriver

_EthercatDevice_DmpCommonDriver is an abstract class that provides dedicated access functions for EtherCAT slave-specific features developed by ICOP. These functions include system monitoring (temperature, voltage, current), order information, MTBF, etc.

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



- *_EthercatDevice_DmpCommonDriver* inherits from *_EthercatDevice_CommonDriver*.

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

Base Class:

- _EthercatDevice_CommonDriver

Function Groups:

- System Monitoring
- MTBF
- Order Information

System Monitoring Functions

The QEC series EtherCAT slave devices with MCU all provide CoE objects to obtain system monitoring information, including system temperature, system voltage, system current, peripheral voltage, and peripheral current. Therefore, this library provides functions to get system monitoring information, enabling users to promptly monitor the system and evaluate it for any signs of failure.

Functions:

- getSystemTemperature()
- getSystemPowerVoltage()
- getSystemPowerCurrent()
- getPeripheralPowerVoltage()
- getPeripheralPowerCurrent()
- tryToGetSystemTemperature()
- tryToGetSystemPowerVoltage()
- tryToGetSystemPowerCurrent()
- tryToGetPeripheralPowerVoltage()
- tryToGetPeripheralPowerCurrent()

getSystemTemperature()

Description

Get the system temperature.

Syntax

```
double getSystemTemperature();
```

Parameters

None.

Return Value

Return the system temperature. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetSystemTemperature()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("System Temperature: ");
  Serial.println(slave.getSystemTemperature());
}
void loop() {
  // ...
}
```

getSystemPowerVoltage()

Description

Get the system voltage.

Syntax

```
double getSystemPowerVoltage();
```

Parameters

None.

Return Value

Return the system voltage. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetSystemPowerVoltage()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("System Voltage: ");
  Serial.println(slave.getSystemPowerVoltage());
}
void loop() {
  // ...
}
```

getSystemPowerCurrent()

Description

Get the system current.

Syntax

```
double getSystemPowerCurrent();
```

Parameters

None.

Return Value

Return the system current. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetSystemPowerCurrent()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("System Current: ");
  Serial.println(slave.getSystemPowerCurrent());
}
void loop() {
  // ...
}
```

getPeripheralPowerVoltage()

Description

Get the peripheral voltage.

Syntax

```
double getPeripheralPowerVoltage();
```

Parameters

None.

Return Value

Return the peripheral voltage. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetPeripheralPowerVoltage()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("Peripheral Voltage: ");
  Serial.println(slave.getPeripheralPowerVoltage());
}
void loop() {
  // ...
}
```

getPeripheralPowerCurrent()

Description

Get the peripheral current.

Syntax

```
double getPeripheralPowerCurrent();
```

Parameters

None.

Return Value

Return the peripheral current. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetPeripheralPowerCurrent()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("Peripheral Current: ");
  Serial.println(slave.getPeripheralPowerCurrent());
}
void loop() {
  // ...
}
```

tryToGetSystemTemperature()

Description

Try to get the system temperature in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getSystemTemperature() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetSystemTemperature();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: Completed.
- false: In progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
double SystemTemperature;

void CyclicCallback() {
  if (slave.tryToGetSystemTemperature()) {
    SystemTemperature = slave.getSystemTemperature();
  }
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

}
```

```
void loop() {
  Serial.print("System Temperature: ");
  Serial.println(SystemTemperature);
  // ...
}
```

tryToGetSystemPowerVoltage()

Description

Try to get the system voltage in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getSystemPowerVoltage() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetSystemPowerVoltage();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: Completed.
- false: In progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
double SystemVoltage;

void CyclicCallback() {
  if (slave.tryToGetSystemPowerVoltage()) {
    SystemVoltage = slave.getSystemPowerVoltage();
  }
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

}
```

```
void loop() {
  Serial.print("System Voltage: ");
  Serial.println(SystemVoltage);
  // ...
}
```

tryToGetSystemPowerCurrent()

Description

Try to get the system current in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getSystemPowerCurrent() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetSystemPowerCurrent();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: Completed.
- false: In progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
double SystemCurrent;

void CyclicCallback() {
  if (slave.tryToGetSystemPowerCurrent())
    SystemCurrent = slave.getSystemPowerCurrent();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

}
void loop() {
```

```
  Serial.print("System Current: ");
  Serial.println(SystemCurrent);
  // ...
}
```

tryToGetPeripheralPowerVoltage()

Description

Try to get the peripheral voltage in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getPeripheralPowerVoltage() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetPeripheralPowerVoltage();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: Completed.
- false: In progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
double PeripheralVoltage;

void CyclicCallback() {
  if (slave.tryToGetPeripheralPowerVoltage())
    PeripheralVoltage = slave.getPeripheralPowerVoltage();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

}
void loop() {
```

```
  Serial.print("Peripheral Voltage: ");
  Serial.println(PeripheralVoltage);
  // ...
}
```

tryToGetPeripheralPowerCurrent()

Description

Try to get the peripheral current in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call `getPeripheralPowerCurrent()` to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetPeripheralPowerCurrent();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: Completed.
- false: In progress.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
double PeripheralCurrent;

void CyclicCallback() {
  if (slave.tryToGetPeripheralPowerCurrent())
    PeripheralCurrent = slave.getPeripheralPowerCurrent();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

}
void loop() {
```

```
  Serial.print("Peripheral Current: ");
  Serial.println(PeripheralCurrent);
  // ...
}
```

MTBF Functions

MTBF stands for Mean Time Between Failures. It is a reliability metric that measures the average time between failures of a system or component. It is calculated by dividing the total time of operation by the number of failures that occur during that time. The result is an average value that can be used to estimate the expected service life of the system or component.

MTBF is a useful metric for tracking the reliability of systems and components, and for identifying potential design flaws or manufacturing defects. It can also be used to make decisions about preventive maintenance schedules.

The QEC series EtherCAT slave devices with MCU all provide CoE objects to obtain MTBF-related information. Therefore, this library provides functions to get these MTBF-related information, allowing users or users to provide it to the manufacturer to assess and judge the life and failure of the device.

Functions:

- getWorkingHours()
- getBootTimes()
- tryToGetWorkingHours()
- tryToGetBootTimes()

getWorkingHours()

Description

Get the current working hours.

Syntax

```
int getWorkingHours();
```

Parameters

None.

Return Value

Return the current working hours. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetWorkingHours()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("Working Hours: ");
  Serial.println(slave.getWorkingHours());
}

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

getBootTimes()

Description

Get the current boot times.

Syntax

```
int getBootTimes();
```

Parameters

None.

Return Value

Return the current boot times. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function can be either blocking or non-blocking. If it is blocking, it should not be called within callback functions. If it is non-blocking, it must be used in conjunction with `tryToGetBootTimes()`, and it can be called within callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

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

  Serial.print("Boot Times: ");
  Serial.println(slave.getBootTimes());
}

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

tryToGetWorkingHours()

Description

Try to get the working hours in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getWorkingHours() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetWorkingHours();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: completed.
- false: in progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
int WorkingHours;

void CyclicCallback() {
  if (slave.tryToGetWorkingHours())
    WorkingHours = slave.getWorkingHours();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
```

```
  Serial.print("Working Hours: ");
  Serial.println(WorkingHours);
  // ...
}
```

tryToGetBootTimes()

Description

Try to get the boot times in a non-blocking manner. If it returns true, the reading process is complete, and you must immediately call getBootTimes() to get the value. Otherwise, it indicates that the process is not yet complete.

Syntax

```
bool tryToGetBootTimes();
```

Parameters

None.

Return Value

Return a boolean value indicating whether the non-blocking read process has completed.

- true: completed.
- false: in progress.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
int BootTimes;

void CyclicCallback() {
  if (slave.tryToGetBootTimes())
    BootTimes = slave.getBootTimes();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
```

```
  Serial.print("Boot Times: ");
  Serial.println(BootTimes);
  // ...
}
```

Order Information Functions

The QEC series EtherCAT slave devices with MCU all provide CoE objects to obtain customer order-related information, which is pre-burned into the devices before shipment.

Therefore, this library provides functions to get these customer order information, facilitating inquiries when necessary.

Functions:

- `getCustomer()`
- `getOrderNumber()`
- `getInvoiceNumber()`
- `getDeliveryDate()`

getCustomer()

Description

Get customer information.

Syntax

```
char *getCustomer(char *buffer = NULL);
```

Parameters

- `[out] char *buffer`
  String data buffer, please ensure that the string array size is greater than or equal to 7.
  If this parameter is not provided, the internal data buffer will be used.

Return Value

Return a pointer to the customer information string.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
char Customer[7];

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

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

  Serial.print("Customer: ");
  Serial.println(slave.getCustomer());
  Serial.print("Customer: ");
  Serial.println(slave.getCustomer(Customer));
  Serial.print("Customer: ");
  Serial.println(Customer);
}

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

getOrderNumber()

Description

Get the order number.

Syntax

```
char *getOrderNumber(char *buffer = NULL);
```

Parameters

- `[out] char *buffer`
  String data buffer, please ensure that the string array size is greater than or equal to 9.
  If this parameter is not provided, the internal data buffer will be used.

Return Value

Return a pointer to the order number string.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
char OrderNumber[9];

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

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

  Serial.print("Order Number: ");
  Serial.println(slave.getOrderNumber());
  Serial.print("Order Number: ");
  Serial.println(slave.getOrderNumber(OrderNumber));
  Serial.print("Order Number: ");
  Serial.println(OrderNumber);
}

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

getInvoiceNumber()

Description

Get the invoice number.

Syntax

```
char *getInvoiceNumber(char *buffer = NULL);
```

Parameters

- `[out] char *buffer`
  String data buffer, please ensure that the string array size is greater than or equal to 12. If this parameter is not provided, the internal data buffer will be used.

Return Value

Return a pointer to the invoice number string.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
char InvoiceNumber[12];

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

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

  Serial.print("Invoice Number: ");
  Serial.println(slave.getInvoiceNumber());
  Serial.print("Invoice Number: ");
  Serial.println(slave.getInvoiceNumber(InvoiceNumber));
  Serial.print("Invoice Number: ");
  Serial.println(InvoiceNumber);
}

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

getDeliveryDate()

Description

Get the delivery date.

Syntax

```
char *getDeliveryDate(char *buffer = NULL);
```

Parameters

- `[out] char *buffer`
  String data buffer, please ensure that the string array size is greater than or equal to 5. If this parameter is not provided, the internal data buffer will be used.

Return Value

Return a pointer to the delivery date string.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the callback functions.

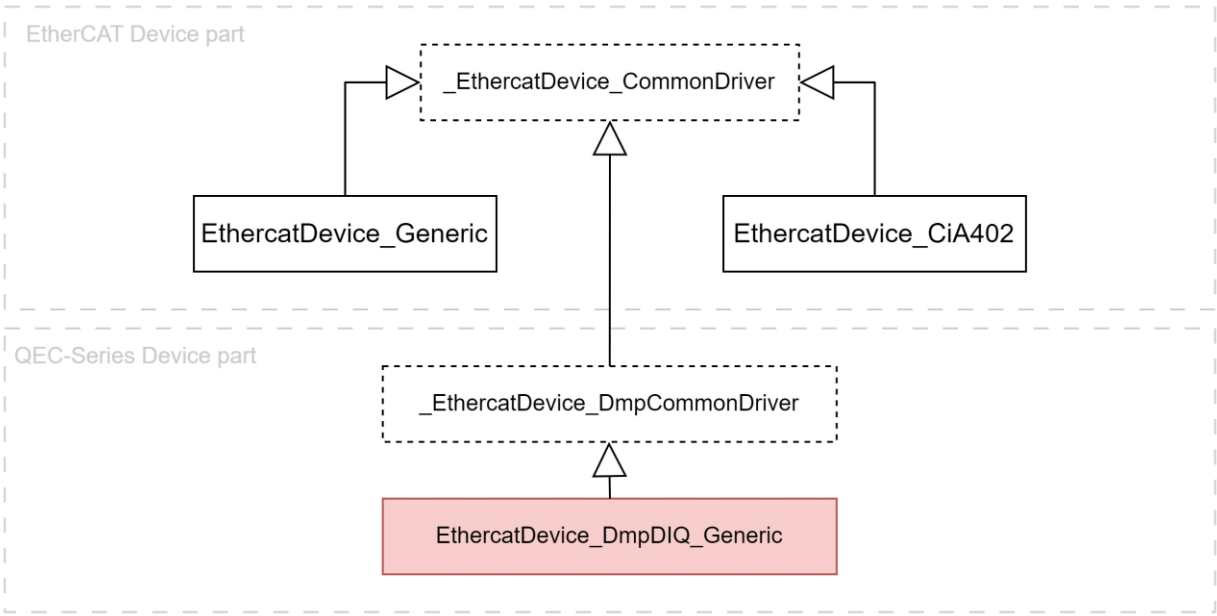
Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;
char DeliveryDate[5];

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

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

  Serial.print("Delivery Date: ");
  Serial.println(slave.getDeliveryDate());
  Serial.print("Delivery Date: ");
  Serial.println(slave.getDeliveryDate(DeliveryDate));
  Serial.print("Delivery Date: ");
  Serial.println(DeliveryDate);
}

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

2.3.2 EthercatDevice_DmpDIQ_Generic

EthercatDevice_DmpDIQ_Generic is an EtherCAT slave class specifically developed by ICOP for Digital I/O EtherCAT slave modules. It provides APIs for digital input, digital output, and other functionalities.

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



- *EthercatDevice_DmpDIQ_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.

Base Class:

- _EthercatDevice_CommonDriver

Function Groups:

- Initialization
- Digital I/O
- Broken Wire Detection

Derived Class:

Class Name	VID	PID	Inputs	Outputs	MCU	DC	Wire Detection
EthercatDevice_QECR00D0FS	0x00000bc3	0x0086d303	0	16	O		
EthercatDevice_QECR00D0FH	0x00000bc3	0x0086d30A	0	16	O	O	
EthercatDevice_QECR00D0TL	0x00000bc3	0x0086d327	0	32			
EthercatDevice_QECR00D0TH	0x00000bc3	0x0086d801	0	32	O	O	
EthercatDevice_QECR00DF0S	0x00000bc3	0x0086d30D	16	0	O		
EthercatDevice_QECR00DF0D	0x00000bc3	0x0086d300	16	0	O		O
EthercatDevice_QECR00DF0H	0x00000bc3	0x0086d30B	16	0	O	O	
EthercatDevice_QECR00DT0L	0x00000bc3	0x0086d323	32	0			
EthercatDevice_QECR00DT0H	0x00000bc3	0x0086d701	32	0	O	O	
EthercatDevice_QECR00D88S	0x00000bc3	0x0086d309	8	8	O		
EthercatDevice_QECR00D88D	0x00000bc3	0x0086d301	8	8	O		O
EthercatDevice_QECR00D88H	0x00000bc3	0x0086d30F	8	8	O	O	
EthercatDevice_QECR00DC4D	0x00000bc3	0x0086d304	12	4	O		O
EthercatDevice_QECR00D4CD	0x00000bc3	0x0086d302	4	12	O		O
EthercatDevice_QECR11D0FS	0x00000bc3	0x0086d0d4	0	16	O		
EthercatDevice_QECR11D0FH	0x00000bc3	0x0086d305	0	16	O	O	
EthercatDevice_QECR11D0TL	0x00000bc3	0x0086d324	0	32			
EthercatDevice_QECR11D0TH	0x00000bc3	0x0086d800	0	32	O	O	
EthercatDevice_QECR11DF0S	0x00000bc3	0x0086d30E	16	0	O		
EthercatDevice_QECR11DF0D	0x00000bc3	0x0086d0d2	16	0	O		O
EthercatDevice_QECR11DF0H	0x00000bc3	0x0086d306	16	0	O	O	
EthercatDevice_QECR11DT0L	0x00000bc3	0x0086d320	32	0			
EthercatDevice_QECR11DT0H	0x00000bc3	0x0086d700	32	0	O	O	
EthercatDevice_QECR11D88S	0x00000bc3	0x0086d0d5	8	8	O		
EthercatDevice_QECR11D88D	0x00000bc3	0x0086d307	8	8	O		O
EthercatDevice_QECR11D88H	0x00000bc3	0x0086d308	8	8	O	O	

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpDIQ_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of EthercatMaster class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of EthercatMaster class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00D0FH slave;

void setup() {
   master.begin();
```

```
    slave.attach(0, master);
    master.start();
}
void loop() {

}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of *EthercatMaster* class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11DT0H slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

}
```

Digital I/O Functions

Digital input and digital output functions for the EthercatDevice_DmpDIQ_Generic class.

Functions:

- digitalWrite()
- digitalWriteAll()
- digitalRead()
- digitalReadAll()

digitalWrite()

Description

Write a HIGH or a LOW value to a digital output pin on the EtherCAT Slave device.

Syntax

```
int digitalWrite(int pin, uint8_t value);
```

Parameters

- `[in] int pin`
  The digital output pin number of the EtherCAT slave device.
- `[in] uint8_t value`
  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.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00D0FH slave;

void setup() {
    master.begin();
    slave.attach(0, master);
    master.start();
}
void loop() {
    slave.digitalWrite(0, HIGH);
    delay(4000);
    slave.digitalWrite(0, LOW);
    delay(1000);
}
```

digitalWriteAll()

Description

Write the digital output state of all pins of the EtherCAT slave device simultaneously.

Syntax

```
int digitalWriteAll(uint32_t value);
```

Parameters

- `[in] uint32_t value`
  This parameter is a 32-bit unsigned integer that specifies the value to be written to all digital output pins. Each bit in the value corresponds to a digital output pin of the EtherCAT slave device, with the following mapping:
  1. Bit 0 indicates digital output pin 0.
  2. Bit 1 indicates digital output pin 1.
  3. And so on, up to bit 31 which indicates digital output pin 31.

  A value of 1 typically sets the corresponding pin to HIGH, while a value of 0 sets it to LOW.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00D88D slave;

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

void loop() {
  slave.digitalWriteAll(0x55555555);
  delay(1000);
```

```
  slave.digitalWriteAll(0xAAAAAAAA);
  delay(1000);
}
```

digitalRead()

Description

Read the value from a specified digital input pin of the EtherCAT slave device.

Syntax

```
int digitalRead(int pin);
```

Parameters

- `[in] int pin`

  The digital input pin number of the EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00D88D slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start();
}

void loop() {
  slave.digitalWrite(0, HIGH);
  delay(1000);

  Serial.print("DI0: ");
  Serial.println(slave.digitalRead(0));

  slave.digitalWrite(0, LOW);
  delay(1000);
```

```
  Serial.print("DI0: ");
  Serial.println(slave.digitalRead(0));
}
```

digitalReadAll()

Description

Read the digital input state of all pins of the EtherCAT slave device simultaneously.

Syntax

```
uint32_t digitalReadAll();
```

Parameters

None.

Return Value

Return a 32-bit unsigned integer that represents the combined state of all digital input pins. Each bit in the returned value corresponds to a digital input pin of the EtherCAT slave device, with the following mapping:

1. Bit 0 indicates digital input pin 0.
2. Bit 1 indicates digital input pin 1.
3. And so on, up to bit 31 which indicates digital input pin 31.

A value of 1 indicates that the corresponding pin is currently HIGH, while a value of 0 indicates that it is currently LOW.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00D88D slave;

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

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

void loop() {
  slave.digitalWriteAll(0x55555555);
  delay(1000);
```

```
  Serial.print("DI: ");
  Serial.println(slave.digitalReadAll());

  slave.digitalWriteAll(0xAAAAAAAA);
  delay(1000);
  Serial.print("DI: ");
  Serial.println(slave.digitalReadAll());
}
```

Broken Wire Detection Functions

Broken Wire Detection Functions for the EthercatDevice_DmpDIQ_Generic class.

Functions:

- startBrokenWireTest()

`startBrokenWireTest()`

Description

Initiates a broken wire detection test on the specified input pins of the EtherCAT slave device.

Syntax

```
uint16_t startBrokenWireTest(uint16_t bitMask, uint32_t timeout_ms = 1000);
```

Parameters

- `[in] uint16_t bitmask`
  This parameter is an unsigned 16-bit integer that specifies which input pins to include in the broken wire detection test. Each bit in the bitMask corresponds to an input pin of the EtherCAT slave device, with the following mapping:
  1. Bit 0 indicates digital input pin 0.
  2. Bit 1 indicates digital input pin 1.
  3. And so on, up to bit 15 which indicates digital input pin 15

  A value of 1 indicates that the corresponding pin should be included in the test, while a value of 0 indicates that it should be excluded.
  WARNING: It is crucial to ensure that this parameter does not contain bits set to 1 for input channels that do not exist on the EtherCAT slave device, otherwise this test will not be executed.
- `[in] uint32_t timeout_ms`
  Timeout in milliseconds.

Return Value

Return a 16-bit unsigned integer that represents the results of the broken wire detection test. Each bit in the returned value corresponds to an input pin included in the test. The interpretation of each bit is as follows:

- 0: The corresponding pin was either not included in the test or was detected as broken.
- 1: The corresponding pin was included in the test and was detected as connected and not broken.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

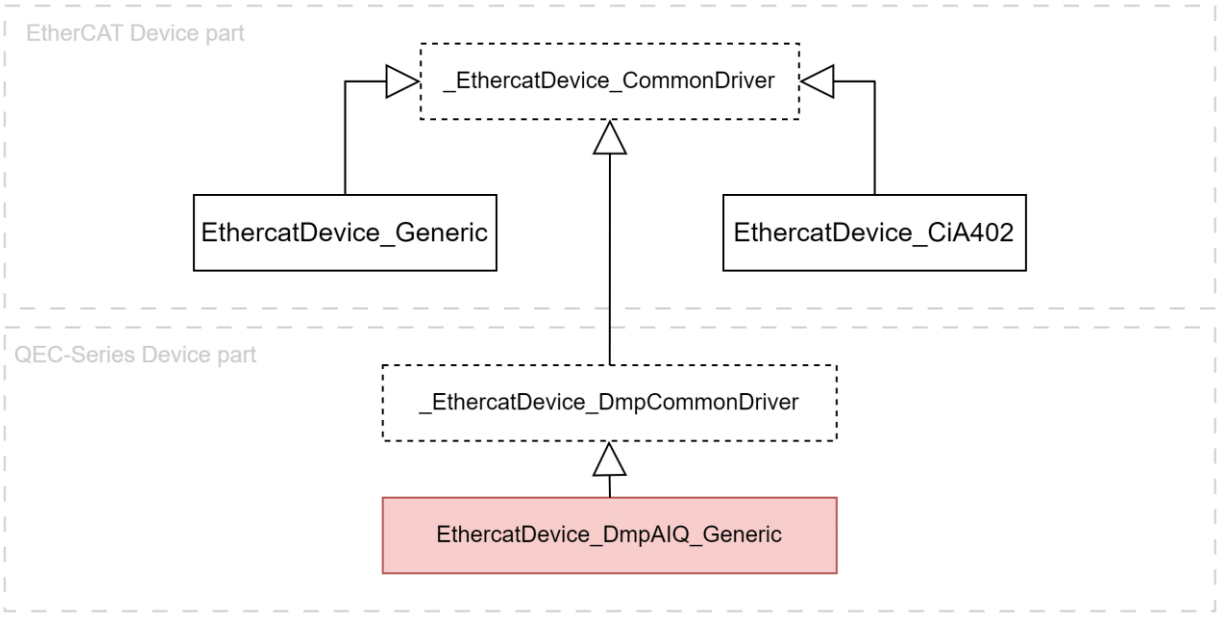
Example

```
#include "Ethercat.h"
```

```
EthercatMaster master;
EthercatDevice_QECR00D88D slave;

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

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

  Serial.print(slave.startBrokenWireTest(0xFF));
}

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

2.3.3 EthercatDevice_DmpAIQ_Generic

EthercatDevice_DmpAIQ_Generic is an EtherCAT slave class specifically developed by ICOP for Analog I/O EtherCAT slave modules. It provides APIs for analog input, analog output, and other functionalities.
The class relationships of *EthercatDevice_DmpAIQ_Generic* are illustrated in the following diagram:



- *EthercatDevice_DmpAIQ_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.

Base Class:

- _EthercatDevice_CommonDriver

Derived Class:

Class Name	Vendor ID	Product Code	Input Channels	Output Channels
EthercatDevice_QECR11A44S	0x00000bc3	0x0086d880	4	4

Function Groups:

- Initialization
- Analog Output
- Analog Input

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpAIQ_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of EthercatMaster class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of EthercatMaster class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

void setup(void) {
   master.begin();
```

```
    slave.attach(0, master);
    master.start();
}
void loop() {

}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of *EthercatMaster* class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

}
```

Analog Output Functions

Analog output functions for the EthercatDevice_DmpAIQ_Generic class.

Functions:

- analogWrite()
- voltageWrite()
- currentWrite()

`analogWrite()`

Description

Write a value to the specified analog output channel on the EtherCAT slave device.

Syntax

```
int analogWrite(int ch, int32_t value);
```

Parameters

- `[in] int ch`
  The specified analog output channel on the EtherCAT slave device.
- `[in] int32_t value`
  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, according to the mode configuration of the specified channel:
  - *Voltage Mode*: Maps a value of 0 to voltage 0V, 2^{31} to 64V, and -2^{31} to -64V. (i.e., 2^{25} would get mapped to 1V.)
  - *Current Mode*: Maps a value of 0 to current 0A, 2^{31} to 16A, and -2^{31} to -16A. (i.e., 2^{27} would get mapped to 1A.)

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

int voltage = 5;

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

void loop() {
```

```
  slave.analogWrite(0, voltage << 25);
  delay(1000);
  slave.analogWrite(0, 0);
  delay(1000);
  slave.analogWrite(0, -1 * (voltage << 25));
  delay(1000);
  slave.analogWrite(0, 0);
  delay(1000);
}
```

voltageWrite()

Description

Write a voltage value to the specified analog output channel on the EtherCAT slave device. This function is used to specify that the channel is configured for *Voltage Mode*.

Syntax

```
int voltageWrite(int ch, double voltage);
```

Parameters

- `[in] int ch`
  The specified analog output channel on the EtherCAT slave device.
- `[in] double voltage`
  The desired output voltage, in volts (*V*).

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

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

void loop() {
  slave.voltageWrite(0, 10.0);
  delay(1000);
  slave.voltageWrite(0, 0);
  delay(1000);
  slave.voltageWrite(0, -10.0);
  delay(1000);
  slave.voltageWrite(0, 0);
```

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

currentWrite()

Description

Write a current value to the specified analog output channel on the EtherCAT slave device. This function is used to specify that the channel is configured for *Current Mode*.

Syntax

```
int currentWrite(int ch, double current);
```

Parameters

- `[in] int ch`
  The specified analog output channel on the EtherCAT slave device.
- `[in] double current`
  The desired output current, in amperes (A).

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

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

void loop() {
  slave.currentWrite(0, 0.02);
  delay(1000);
  slave.currentWrite(0, 0);
  delay(1000);
  slave.currentWrite(0, 0.01);
  delay(1000);
  slave.currentWrite(0, 0);
```

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

Analog Input Functions

Analog input functions for the EthercatDevice_DmpAIQ_Generic class.

Functions:

analogRead()

Description

Read a value from an analog input pin on the EtherCAT Slave device.

Syntax

```
int analogRead(int ch);
```

Parameters

- `[in] int ch`

  The specified analog input channel on the EtherCAT slave device.

Return Value

Return a 32-bit signed integer within the range of -2^{31} to 2^{31} as the analog input value. This value is linearly mapped to a physical input voltage in volts, where 0 maps to 0V, 2^{31} maps to 64V, and -2^{31} maps to -64V.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

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

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

void loop() {
  Serial.print("AIO: ");
  Serial.println(slave.analogRead(0));
  delay(1000);
```

```
}
```

voltageRead()

Description

Read the voltage value of the specified analog input channel on the EtherCAT slave device.

Syntax

```
double voltageRead(int ch);
```

Parameters

- `[in] int ch`

  The specified analog input channel on the EtherCAT slave device.

Return Value

Return the voltage value expressed in volts (V) as a double.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11A44S slave;

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

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

void loop() {
  Serial.print("AIO: ");
  Serial.println(slave.voltageRead(0));
  delay(1000);
}
```

2.3.4 EthercatDevice_DmpHID_Generic

EthercatDevice_DmpHID_Generic is an EtherCAT slave class specifically developed by ICOP for QEC EtherCAT slave HID modules. It encompasses RS232/RS485, 4x4 Keypad, 16x2 LCM, Manual Pulse Generator (MPG), and Buzzer functionalities.

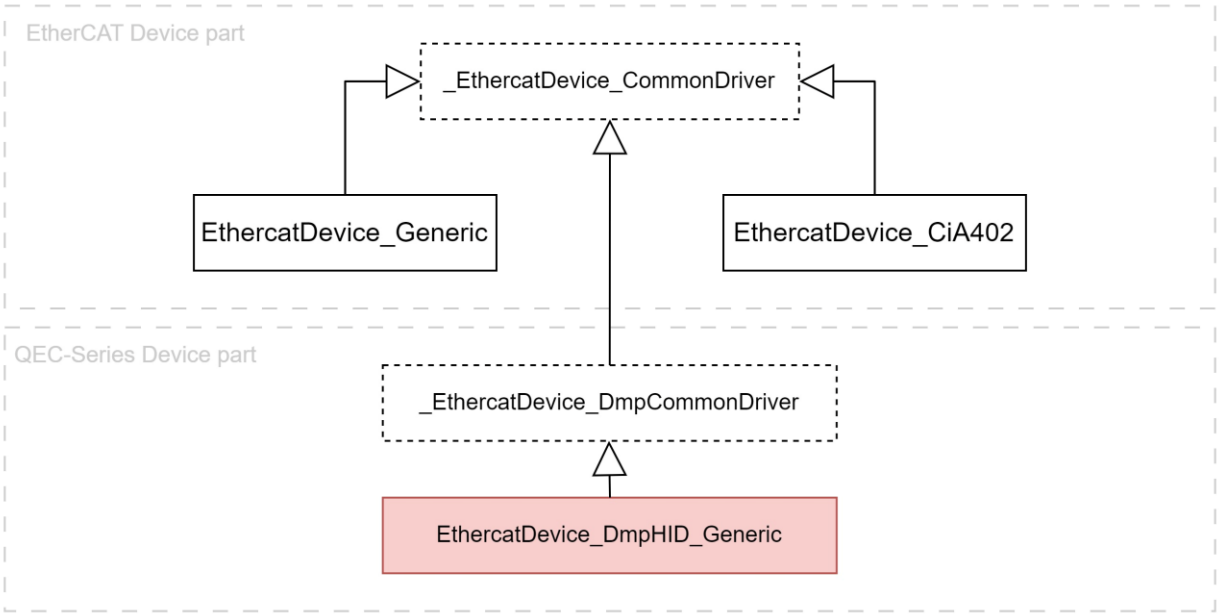
RS232 and **RS485** are electrical specifications that define the voltage levels, signal timing, and connector pinouts for implementing UART communication over physical cables. **UART (Universal Asynchronous Receiver Transmitter)** is a hardware interface standard for asynchronous serial data communication. It defines the format of data bits, start and stop bits, parity bits, and baud rate for serial communication. UART is commonly used for connecting microcontrollers, computers, and other devices for data exchange. This device features two UART ports that can be freely switched between RS232 or RS485 modes to accommodate user application requirements.

This device features a **4x4 keypad** that provides a user interface for inputting numbers, symbols, and characters. The keypad is arranged in a matrix format with four rows and four columns, with each key representing a specific character or function.

This device features a **2x16 LCD module** that provides a visual display for presenting information to the user. The LCM is arranged in a matrix format with two rows and 16 columns, allowing it to display up to 32 characters per line.

This device features a **Manual Pulse Generator (MPG)** interface that enables the connection of specific MPG devices, allowing precise control of machine or system movement. The interface that counts MPG pulses is referred to as an Encoder in this context. In addition to Encoder counter, the interface also offers Ratio knob, Axis knob, Emergency Stop switch, and Enable switch.

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



- *EthercatDevice_DmpHID_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.

Base Class:

- _EthercatDevice_CommonDriver

Derived Class:

Class Name	Vendor ID	Product Code	UART	Keypad	LCM	MPG
EthercatDevice_QECR11HU1S	0x00000bc3	0x0086d404	O			O
EthercatDevice_QECR11HU5S	0x00000bc3	0x0086d403	O			
EthercatDevice_QECR00HU5S	0x00000bc3	0x0086d400	O			
EthercatDevice_QECR11HU9S	0x00000bc3	0x0086d402	O	O	O	O
EthercatDevice_QECR00HU9S	0x00000bc3	0x0086d401	O	O	O	O

Function Groups:

- Initialization
- Control
- UART
- LCM
- Keypad
- MPG
- Buzzer

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of EthercatMaster class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of *EthercatMaster* class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00HU9S slave;

void setup(void) {
   master.begin();
```

```
    slave.attach(0, master);
    master.start();
}
void loop() {

}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of *EthercatMaster* class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

}
```

Control Functions

Control functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- update()

update()

Description

Update state machines and internal variables for each function on the EtherCAT slave device.

Syntax

```
int update();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

UART Functions

UART functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- uartIs485()
- uartSetBaud()
- uartSetFormat()
- uartSetFlowControl()
- uartGetRTS()
- uartGetCTS()
- uartGetDTR()
- uartGetDSR()
- uartSetRTS()
- uartSetDTR()
- uartClearFIFO()
- uartClearTxQueue()
- uartClearRxQueue()
- uartQueryTxQueue()
- uartQueryRxQueue()
- uartTxQueueEmpty()
- uartRxQueueEmpty()
- uartTxQueueFull()
- uartRxQueueFull()
- uartSend()
- uartWrite()
- uartReceive()
- uartRead()

uartIs485()

Description

Check if the specified UART port of the EtherCAT slave device is in RS485 mode.

Syntax

```
int uartIs485(int dev);
```

Parameters

- `[in] int dev`

  The specified UART port of the EtherCAT slave device.

  0 for COM1.

  1 for COM2.

Return Value

Return whether the specified UART port of the EtherCAT slave device is in RS485 mode.

- 1 means RS485.
- 0 means non-RS485.

if the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.

This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("RS485 mode: ");
  Serial.println(slave.uartIs485(0));
}

void loop() {
```

```
  // ...
}
```

uartSetBaud()

Description

Configure the baud rate for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSetBaud(int dev, uint32_t baud);
```

Parameters

- `[in] int dev`

  The specified UART port of the EtherCAT slave device.

  0 for COM1.

  1 for COM2.

- `[in] uint32_t baud`

  The baud rate to be configured.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.uartSetBaud(0, 115200);
}

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

uartSetFormat()

Description

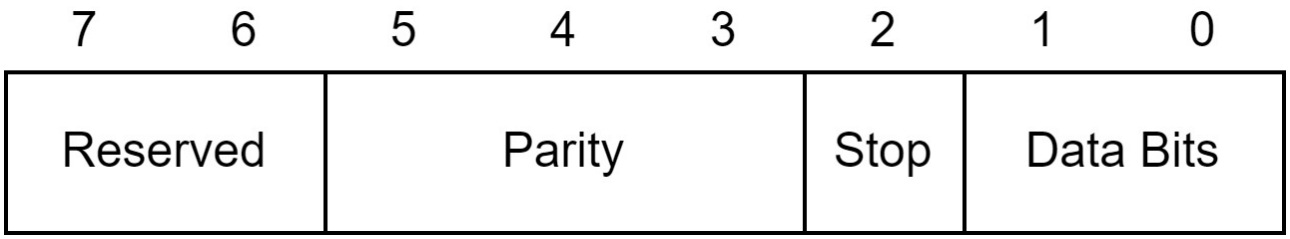
Configure the UART frame format for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSetFormat(int dev, uint8_t format);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] uint8_t format`
  The UART frame format to be configured. The bit definition of this parameter is as follows:



Data Bits: These bits define the word length of the data being transmitted and received.

Definition	Value	Description
ECAT_UART_BYTESIZE5	0x00	5 data bits.
ECAT_UART_BYTESIZE6	0x01	6 data bits.
ECAT_UART_BYTESIZE7	0x02	7 data bits.
ECAT_UART_BYTESIZE8	0x03	8 data bits.

Stop: This bit selects the number of stop bits to be transmitted.

Definition	Value	Description
ECAT_UART_STOPBIT1	0x00	One stop bit.
ECAT_UART_STOPBIT2	0x04	Two stop bits (1.5 with 5-bit data).

Parity: These bits select the way in which parity control is performed.

Definition	Value	Description
ECAT_UART_NOPARITY	0x00	No parity bit.
ECAT_UART_ODDPARITY	0x08	Odd parity.

ECAT_UART_EVENPARITY	0x18	Even parity.
ECAT_UART_MARKPARITY	0x28	The parity bit exists and is always 1.
ECAT_UART_SPACEPARITY	0x38	The parity bit exists and is always 0.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.uartSetFormat(0, ECAT_UART_BYTESIZE8 + ECAT_UART_NOPARITY +
ECAT_UART_STOPBIT1);
}

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

uartSetFlowControl()

Description

Configure the flow control mode for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSetFlowControl(int dev, int control);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] int control`
  The flow control mode to be configured.
  a. `ECAT_UART_NO_CONTROL` : Disable flow control.
  b. `ECAT_UART_RTS_CTS` : RTS/CTS flow control is a hardware flow control scheme that is commonly used in RS232.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

uartGetRTS()

Description

Get the current state of the RTS control signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartGetRTS(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return the current state of the RTS control signal. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
}

void loop() {
  slave.uartSetRTS(0, 1);
  Serial.print("RTS: ");
  Serial.println(slave.uartGetRTS(0));
  delay(1000);

  slave.uartSetRTS(0, 0);
  Serial.print("RTS: ");
  Serial.println(slave.uartGetRTS(0));
  delay(500);
  // ...
}
```

uartGetCTS()

Description

Get the current state of the CTS signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartGetCTS(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return the current state of the CTS signal. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
}

void loop() {
  Serial.print("CTS: ");
  Serial.println(slave.uartGetCTS(0));
  delay(1000);
  // ...
}
```

uartGetDTR()

Description

Get the current state of the DTR control signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartGetDTR(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return the current state of the DTR control signal. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
}

void loop() {
  slave.uartSetDTR(0, 1);
  Serial.print("DTR: ");
  Serial.println(slave.uartGetDTR(0));
  delay(1000);

  slave.uartSetDTR(0, 0);
  Serial.print("DTR: ");
  Serial.println(slave.uartGetDTR(0));
  delay(500);
  // ...
}
```

uartGetDSR()

Description

Get the current state of the DSR signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartGetDSR(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return the current state of the DSR signal. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
}

void loop() {
  Serial.print("DSR: ");
  Serial.println(slave.uartGetDSR(0));
  delay(1000);
  // ...
}
```

uartSetRTS()

Description

Control the RTS signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSetRTS(int dev, uint8_t value);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] unit8_t value`
  The RTS signal value to be set.
  0: Indicates an inactive RTS signal.
  1 to 255: Indicates an active RTS signal

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

```
  master.start();
}

void loop() {
  slave.uartSetRTS(0, 1);
  delay(1000);
  slave.uartSetRTS(0, 0);
  delay(500);
  // ...
}
```

uartSetDTR()

Description

Control the DTR signal for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSetDTR(int dev, uint8_t value);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] unit8_t value`
  The DTR signal value to be set.
  0: Indicates an inactive DTR signal.
  1 to 255: Indicates an active DTR signal.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

```
  master.start();
}

void loop() {
  slave.uartSetDTR(0, 1);
  delay(1000);
  slave.uartSetDTR(0, 0);
  delay(500);
  // ...
}
```

uartClearFIFO()

Description

Clear the TX and RX FIFOs for the specified UART port of the EtherCAT slave device.

Syntax

```
int uartClearFIFO(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

uartClearTxQueue()

Description

Clear the software TX FIFO for the specified UART port of the EtherCAT slave device in this library.

Syntax

```
int uartClearTxQueue(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

uartClearRxQueue()

Description

Clear the software RX FIFO for the specified UART port of the EtherCAT slave device in this library.

Syntax

```
int uartClearRxQueue(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

uartQueryTxQueue()

Description

Get the current number of bytes in the software TX FIFO for the specified UART port of the specified EtherCAT slave device in this library.

Syntax

```
int uartQueryTxQueue(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return the current number of bytes in the software TX FIFO for the specified UART port. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("TX FIFO: ");
  Serial.println(slave.uartQueryTxQueue(0));
}

void loop() {}
```

uartQueryRxQueue()

Description

Get the current number of bytes in the software RX FIFO for the specified UART port of the specified EtherCAT slave device in this library.

Syntax

```
int uartQueryRxQueue(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

The current number of bytes in the software RX FIFO for the specified UART port. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("RX FIFO: ");
  Serial.println(slave.uartQueryRxQueue(0));
}

void loop() {}
```

uartTxQueueEmpty()

Description

Check if the software TX FIFO for the specified UART port of the specified EtherCAT slave device in this library is empty.

Syntax

```
int uartTxQueueEmpty(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return whether the software TX FIFO in this library is empty. If the FIFO on the COM port is empty, the returned value is 1; if it is not, it is 0.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("TX FIFO is Empty: ");
  Serial.println(slave.uartTxQueueEmpty(0));
}
void loop() {}
```

uartRxQueueEmpty()

Description

Check if the software RX FIFO for the specified UART port on the EtherCAT slave device in this library is empty.

Syntax

```
int uartRxQueueEmpty(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return whether the software RX FIFO in this library is empty. If the FIFO on the COM port is empty, the returned value is 1; if it is not, it is 0.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("RX FIFO is Empty: ");
  Serial.println(slave.uartRxQueueEmpty(0));
}
void loop() {}
```

uartTxQueueFull()

Description

Check if the software TX FIFO for the specified UART port of the specified EtherCAT slave device in this library is full.

Syntax

```
int uartTxQueueFull(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return whether the software TX FIFO in this library is full. If the FIFO on the COM port is full, the returned value is 1; if it is not, it is 0.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("TX FIFO is Full: ");
  Serial.println(slave.uartTxQueueFull(0));
}
void loop() {}
```

uartRxQueueFull()

Description

Check if the software RX FIFO for the specified UART port of the specified EtherCAT slave device in this library is full.

Syntax

```
int uartRxQueueFull(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return whether the software RX FIFO in this library is full. If the FIFO on the COM port is full, the returned value is 1; if it is not, it is 0.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("RX FIFO is Full: ");
  Serial.println(slave.uartRxQueueFull(0));
}
void loop() {}
```

uartSend()

Description

Transmit multiple bytes data from the specified UART port of the EtherCAT slave device.

Syntax

```
int uartSend(int dev, void *buf, size_t size);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] void *buf`
  The data buffer to be transmitted.
- `[in] size_t size`
  The size of the data buffer.

Return Value

Return the number of bytes transmitted. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  char buffer[] = {"Hello world!"};

  Serial.begin(115200);
```

```
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.uartSend(0, buffer, strlen(buffer));

  Serial.print("Sent: ");
  Serial.println(buffer);
}

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

uartWrite()

Description

Transmit one byte data from the specified UART port of the EtherCAT slave device.

Syntax

```
int uartWrite(int dev, uint8_t value);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] unit8_t value`
  The one byte data to be transmitted.

Return Value

Return the number of bytes transmitted. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
  slave.uartWrite(0, 'H');
  slave.uartWrite(0, 'e');
  slave.uartWrite(0, 'l');
  slave.uartWrite(0, 'l');
  slave.uartWrite(0, 'o');
}

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

uartReceive()

Description

Receive multiple bytes data from the specified UART port of the EtherCAT slave device.

Syntax

```
int uartReceive(int dev, void *buf, size_t size);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.
- `[in] void *buf`
  The data buffer to be received.
- `[in] size_t size`
  The size of the data buffer.

Return Value

Return the number of bytes received. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

char buffer[256];
int rc;

void CyclicCallback() {
  slave.update();
}

void setup() {
```

```
  Serial.begin(115200);

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  rc = slave.uartReceive(0, buffer, 256);
  for (int i = 0; i < rc; i++) {
    Serial.print(buffer[i]);
  }

  // ...
}
```

uartRead()

Description

Read one byte data from the specified UART port of the EtherCAT slave device.

Syntax

```
int uartRead(int dev);
```

Parameters

- `[in] int dev`
  The specified UART port of the EtherCAT slave device.
  0 for COM1.
  1 for COM2.

Return Value

Return one byte data. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}
```

```
void loop() {
  int ch = slave.uartRead(0);
  if (ch >= 0)
    Serial.print((char)ch);

  // ...
}
```

LCM Functions

LCM functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- lcmHeight()
- lcmWidth()
- lcmWordWrap()
- lcmGotoXY()
- lcmClear()
- lcmPrint()
- lcmWrite()

lcmHeight()

Description

Get the height of the LCM on the EtherCAT slave device.

Syntax

```
int lcmHeight();
```

Parameters

None.

Return Value

Return the height of the LCM. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("LCM Height: ");
  Serial.println(slave.lcmHeight());
}

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

lcmWidth()

Description

Get the width of the LCM on the EtherCAT slave device.

Syntax

```
int lcmWidth();
```

Parameters

None.

Return Value

Return the width of the LCM. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

  master.begin();
  slave.attach(0, master);
  Serial.print("LCM Width: ");
  Serial.println(slave.lcmWidth());
}

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

lcmWordWrap()

Description

Enable or disable the word wrap feature on the LCM of the EtherCAT slave device.

Syntax

```
int lcmWordWrap(bool wrap);
```

Parameters

- `[in] bool wrap`
  A boolean value that specifies whether to enable or disable the word wrap feature on the LCM.
  1. true: The word wrap feature will be enabled.
  2. false: The word wrap feature will be disabled.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

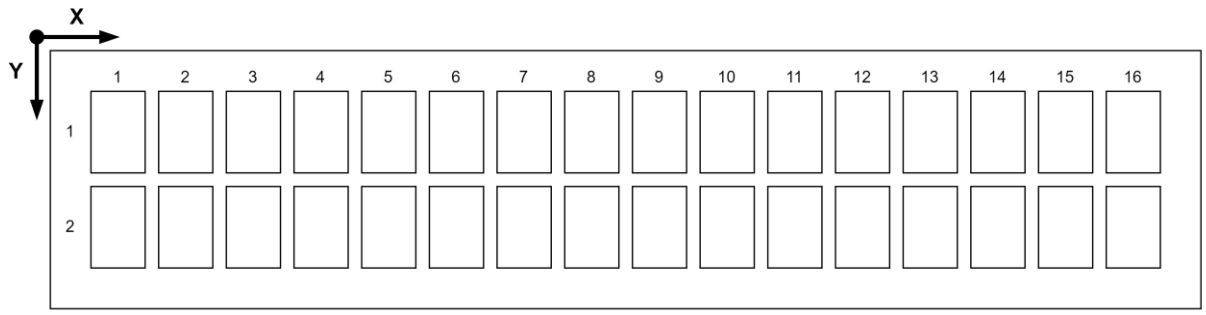
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcmWordWrap(true);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcmPrint("Hi, this is QEC HID slave.");
}
```

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

lcmGotoXY()

Description

Move the current cursor on the LCM of the EtherCAT slave device to the specified coordinates.



Syntax

```
int lcmGotoXY(int x, int y);
```

Parameters

- `[in] int x`

  X-axis position of the LCM.

- `[in] int y`

  Y-axis position of the LCM.

Return Value

Return the current cursor position of the LCM. The formula for calculating the cursor position P_{cursor} is as follows, where W is the width of the LCM:

$$Pcursor = W \times (Y - 1) + (X - 1)$$

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
```

```
}

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

  slave.lcmGotoXY(1, 1);
  slave.lcmPrint("Hello");
  slave.lcmGotoXY(8, 1);
  slave.lcmPrint("World!");
  slave.lcmGotoXY(2, 2);
  slave.lcmPrint("QEC HID slave");
}

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

lcmClear()

Description

Clear the screen of the LCM on the EtherCAT slave device.

Syntax

```
int lcmClear();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.lcmClear();
}

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

`lcmPrint()`

Description

Print the specified string to the LCM screen of the EtherCAT slave device.

Syntax

```
int lcmPrint(const char *fmt, ...);
```

Parameters

- `[in] const char *fmt`
  The string to be printed on the LCM screen. This is a pointer to a null-terminated string containing the format specification for the output. The format string follows the same format as the printf function in C, allowing for insertion of variables and formatting options.
- `[in] ...`
  This is a variable number of arguments that will be inserted into the formatted string according to the format specifiers in the fmt string.

Return Value

Return the number of characters printed to the LCM screen. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
  slave.lcmPrint("Hello world!");
}

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

lcmWrite()

Description

Print one character to the LCM screen of the EtherCAT slave device.

Syntax

```
int lcmWrite(char c);
```

Parameters

- `[in] char c`
  The character to be printed on the LCM screen.

Return Value

Return the number of characters printed to the LCM screen. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.lcmWrite('H');
  slave.lcmWrite('e');
  slave.lcmWrite('l');
```

```
  slave.lcmWrite('l');
  slave.lcmWrite('o');
  slave.lcmWrite('!');
}

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

Keypad Functions

Keypad functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- keypadSetTimeout()
- keypadClear()
- keypadRead()

keypadSetTimeout()

Description

Set the keypad input data buffer timeout for the EtherCAT slave device. If no keypad input data is read for a specified period, the input data buffer will be automatically cleared. The default timeout is 1000 milliseconds.

Syntax

```
int keypadSetTimeout(uint32_t timeout_ms);
```

Parameters

- `[in] uint32_t timeout_ms`
  Timeout in milliseconds. If this parameter is 0, it indicates that the keypad input data buffer will not be automatically cleared.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.keypadSetTimeout(3000);
}

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

keypadClear()

Description

Clear the input data buffer of the keypad of the EtherCAT slave device.

Syntax

```
int keypadClear();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

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

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

keypadRead()

Description

Read the input character from the keypad of the EtherCAT slave device.

Syntax

```
int keypadRead();
```

Parameters

None.

Return Value

Read a character from the keypad. Return '\0' if no data available. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  int ch;
```

```
  if ((ch = slave.keypadRead()) != '\0') {
    Serial.print((char)ch);
  }

  // ...
}
```

MPG Functions

MPG hand wheels functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- mpgSetCallback()
- mpgSetNoiseFilter()
- mpgInvertEncoderDirection()
- mpgWriteEncoderRaw()
- mpgWriteEncoder()
- mpgReadEncoderRaw()
- mpgReadEncoder()
- mpgReadEmergencyStop()
- mpgReadEnableSwitch()
- mpgReadAxis()
- mpgReadRatio()

mpgSetCallback()

Description

Register the MPG event callback function for the EtherCAT slave device. Since this callback function is called within update(), it is prohibited to call blocking functions and system call-related functions within this callback function if update() is called in an interrupt callback function (such as *Cyclic Callback*, *Error Callback*, or *Event Callback*).

Syntax

```
int mpgSetCallback(void (*callback)(int));
```

Parameters

- `[in] void (*callback)(int)`
  The MPG event callback function to be registered has an integer-type parameter that indicates the event type. The supported MPG event types are as follows:

Definition	Code	Description
ECAT_MPG_AXIS_CHANGE	1	The status of the Axis knob has changed.
ECAT_MPG_RATIO_CHANGE	2	The status of the Ratio knob has changed.
ECAT_MPG_EMERGENCY_STOP_CHANGE	3	The status of the Emergency Stop switch has changed.
ECAT_MPG_ENABLE_SWITCH_CHANGE	4	The status of the Enable switch has changed.
ECAT_MPG_ENCODER_CHANGE	5	The logical counter of the Encoder has changed.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void MpgCallback(int event) {
```

```
  switch (event) {
    case ECAT_MPG_AXIS_CHANGE:
      Serial.print("MPG Axis changed. ");
      Serial.println(slave.mpgReadAxis());
      break;
    case ECAT_MPG_RATIO_CHANGE:
      Serial.print("MPG Ratio changed. ");
      Serial.println(slave.mpgReadRatio());
      break;
    case ECAT_MPG_EMERGENCY_STOP_CHANGE:
      Serial.print("MPG Emergency Stop changed ");
      Serial.println(slave.mpgReadEmergencyStop());
      break;
    case ECAT_MPG_ENABLE_SWITCH_CHANGE:
      Serial.print("MPG Enable Switch changed. ");
      Serial.println(slave.mpgReadEnableSwitch());
      break;
    case ECAT_MPG_ENCODER_CHANGE:
      Serial.print("MPG Encoder changed. ");
      Serial.println(slave.mpgReadEncoder());
      break;
  }
}
void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  slave.mpgSetCallback(MpgCallback);
  master.start();
}
void loop() {
  slave.update();
}
```

mpgSetNoiseFilter()

Description

Configure the MPG IO noise filter for the EtherCAT slave device, including the noise filtering for the *Emergency Stop* switch, *Axis* knob, and *Ratio* knob.

Syntax

```
int mpgSetNoiseFilter(uint32_t time_us);
```

Parameters

- `[in] uint32_t time_us`

  The desired configuration time for the noise filter.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.mpgSetNoiseFilter(10000);
  master.start();
}

void loop() {
  slave.update();
}
```

mpgInvertEncoderDirection()

Description

Invert the counting direction of the Encoder of the MPG on the EtherCAT slave device.

Syntax

```
int mpgInvertEncoderDirection(bool invert);
```

Parameters

- `[in] bool invert`

  A boolean value that specifies whether to invert the counting direction of the Encoder.

  a. true: The counting direction will be inverted.

  b. false: The counting direction will not be changed.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.mpgInvertEncoderDirection(true);
  master.start();
}

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

mpgWriteEncoderRaw()

Description

Write the raw data to the *Encoder* of the MPG on the EtherCAT slave device.

Syntax

```
int mpgWriteEncoderRaw(int32_t value);
```

Parameters

- `[in] int32_t value`
  The raw data value to be written to the Encoder.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.mpgWriteEncoderRaw(100);
  master.start();
}

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

mpgWriteEncoder()

Description

Write the logical counter to the *Encoder* of the MPG on the EtherCAT slave device.

Syntax

```
int mpgWriteEncoder(int32_t value);
```

Parameters

- `[in] int32_t value`
  The logical counter to be written to the Encoder.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.mpgWriteEncoder(100);
  master.start();
}

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

mpgReadEncoderRaw()

Description

Read the raw data from the *Encoder* of the MPG on the EtherCAT slave device.

Syntax

```
int32_t mpgReadEncoderRaw();
```

Parameters

None.

Return Value

Return the raw data from the *Encoder*.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

int32_t raw = 0;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}
```

```
void loop() {
  int32_t newRaw = slave.mpgReadEncoderRaw();
  if (raw != newRaw) {
    raw = newRaw;
    Serial.print("Raw: ");
    Serial.println(raw);
  }
  // ...
}
```

mpgReadEncoder()

Description

Read the logical counter from the *Encoder* of the MPG on the EtherCAT slave device.

Syntax

```
int32_t mpgReadEncoder();
```

Parameters

None.

Return Value

Return the logical counter with 32-bit signed integer.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

int32_t encoder = 0;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}
```

```
void loop() {
  int32_t newEncoder = slave.mpgReadEncoder();
  if (encoder != newEncoder) {
    encoder = newEncoder;
    Serial.print("Encoder: ");
    Serial.println(encoder);
  }

  // ...
}
```

mpgReadEmergencyStop()

Description

Read the *Emergency Stop* value of the MPG for the EtherCAT slave device.

Syntax

```
int mpgReadEmergencyStop();
```

Parameters

None.

Return Value

Return the Emergency Stop value, 1 means the emergency stop signal pulls HIGH, and 0 means LOW. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
```

```
  Serial.print("Emergency Stop: ");
  Serial.println(slave.mpgReadEmergencyStop());
  delay(1000);
  // ...
}
```

mpgReadEnableSwitch()

Description

Read the *Enable* switch value of the MPG for the EtherCAT slave device.

Syntax

```
int mpgReadEnableSwitch();
```

Parameters

None.

Return Value

Return the Enable switch value, 1 means the enable switch signal pulls HIGH, and 0 means LOW. If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
```

```
  Serial.print("Enable: ");
  Serial.println(slave.mpgReadEnableSwitch());
  delay(1000);
  // ...
}
```

mpgReadAxis()

Description

Read the *Axis* value of the MPG for the EtherCAT slave device.

Syntax

```
int mpgReadAxis();
```

Parameters

None.

Return Value

Return the Axis value.

- 0: off.
- 1: X-axis.
- 2: Y-axis.
- 3: Z-axis.
- 4: 4-axis.
- 5: 5-axis.
- 6: 6-axis.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

```
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  Serial.print("Axis: ");
  Serial.println(slave.mpgReadAxis());
  delay(1000);
  // ...
}
```

mpgReadRatio()

Description

Read the *Ratio* value of the MPG for the EtherCAT slave device.

Syntax

```
int mpgReadRatio();
```

Parameters

None.

Returns

Return the Ratio value.

- 0: 1x.
- 1: 10x.
- 2: 100x.

If the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
}

void loop() {
  Serial.print("Ratio: ");
  Serial.println(slave.mpgReadRatio());
  delay(1000);
  // ...
}
```

Buzzer Functions

Buzzer functions for the EthercatDevice_DmpHID_Generic class.

Functions:

- buzzer()

buzzer()

Description

Emit a sound of the specified frequency from the buzzer on the EtherCAT slave device.

Syntax

```
int buzzer(uint32_t hz, uint32_t duration_ms = 0);
```

Parameters

- `[in] uint32_t hz`
  The frequency of the sound in Hz. If this parameter is set to 0, it indicates that the sound should be stopped.If the value of the frequency is greater than 100,000 (100K), this library would replace the input value with 100K.
- `[in] uint32_t duration_ms`
  The duration of the sound in milliseconds. If this parameter is omitted or set to 0, the sound will continue indefinitely.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

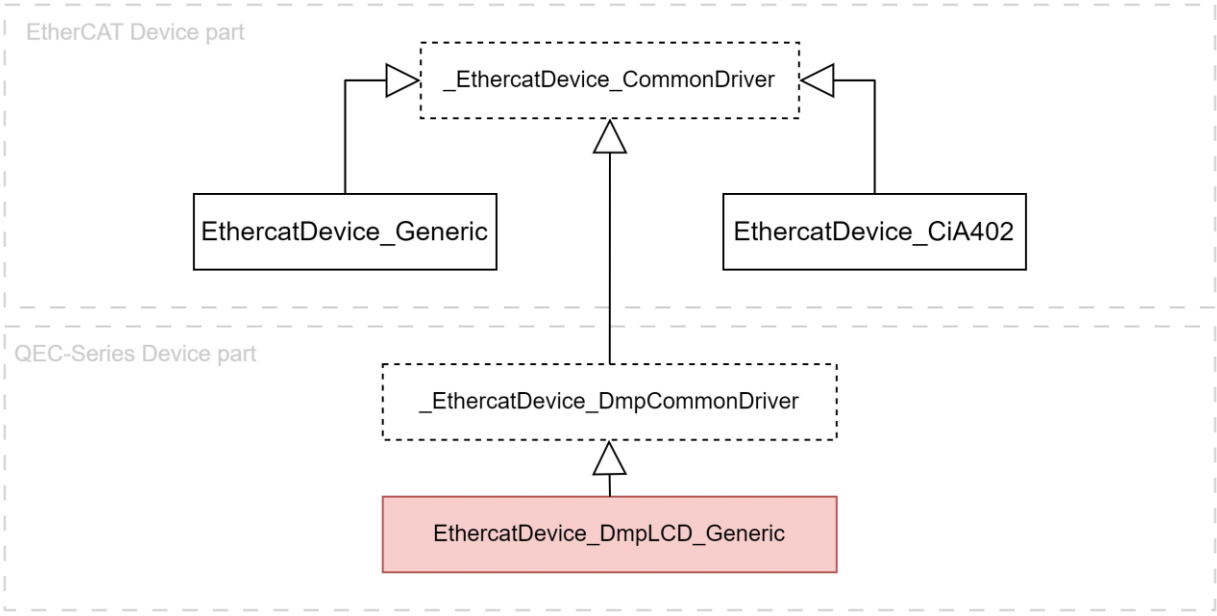
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11HU9S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
  slave.buzzer(3000);
  delay(1000);
  slave.buzzer(500);
  delay(1000);
  slave.buzzer(0);
  delay(1000);
}

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

2.3.5 EthercatDevice_DmpLCD_Generic

EthercatDevice_DmpLCD_Generic is an EtherCAT slave class specifically developed by ICOP for LCD EtherCAT slave modules. It provides a variety of drawing APIs.
The class relationships of *EthercatDevice_DmpLCD_Generic* are illustrated in the following diagram:



- *EthercatDevice_DmpLCD_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.

Base Class:

- _EthercatDevice_CommonDriver

Derived Class:

Class Name	Vendor ID	Product Code
EthercatDevice_QECR11UN01	0x00000bc3	0x0086d103
EthercatDevice_QECR00UN01	0x00000bc3	0x0086d100

Function Groups:

- Initialization
- Control
- LCD
- Touch

LCD Module Table:

ID	LCM Driver IC	Resolution	Xp	Yp	Xm	Ym
0	ILI9341	240 X 320	D9	A2	A3	D8
1	ILI9341	240 X 320	D6	A1	A2	D7
2	ILI9488	320 X 480	D8	A3	A2	D9
3	ILI9486	320 X 480	X	X	X	X
4	HX8347-I(T)	240 X 320	D9	A2	A3	D8
5	HX8347-D	240 X 320	D9	A2	A3	D8

LCD Module List:

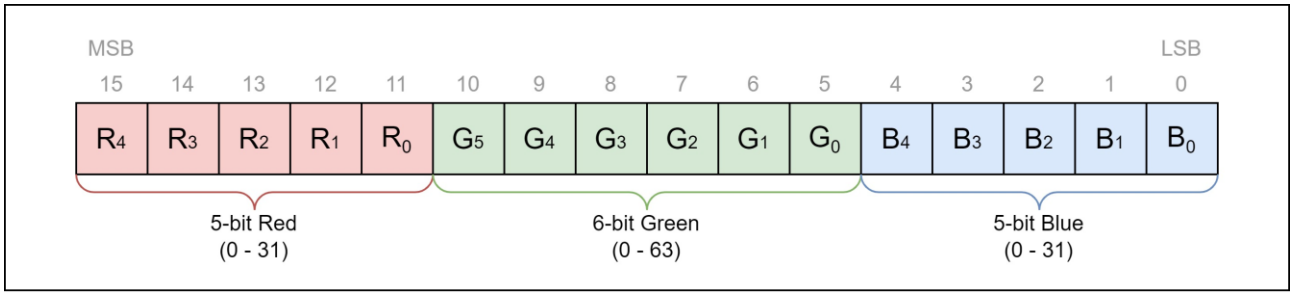
- ECAT_LCD_UNKNOWN_ID (0xFFFF)
- ECAT_LCD_ILI9341_1 (0)
- ECAT_LCD_ILI9341_2 (1)
- ECAT_LCD_ILI9488_1 (2)
- ECAT_LCD_ILI9486_1 (3)
- ECAT_LCD_HX8347I_1 (4)
- ECAT_LCD_HX8347D_1 (5)

About RGB565

RGB565 is a color format used to represent color information for pixels in an image. It utilizes 16 bits (2 bytes) to encode the color information for a single pixel. The name "RGB565" indicates the distribution of bits among the three primary colors: red, green, and blue.

- **Red (R)**: 5 bits
- **Green (G)**: 6 bits
- **Blue (B)**: 5 bits

This prioritizes green over red and blue because the human eye is more sensitive to variations in green compared to red and blue.



Due to the fewer bits allocated to each color component, RGB565 offers a smaller color palette compared to 24-bit RGB or 16-bit RGB. It can represent 2^5 = 32 possible values for red and blue, and 2^6 = 64 possible values for green, resulting in a total of 32 x 64 x 32 = 65,536 possible colors.

Here are some RGB565 color codes for quick reference and easy testing:

Color	Code	Sample
Black	0x0000	
Blue	0x001F	
Red	0xF800	
Green	0x07E0	
Cyan	0x07FF	
Magenta	0xF81F	
Yellow	0xFFE0	
White	0xFFFF	

Converting RGB888 to RGB565

Converting from 24-bit RGB to RGB565 involves a process called color quantization. Similar to converting between 24-bit and 16-bit RGB, this process reduces the number of colors and assigns them the closest available color within the RGB565 palette. This can introduce some color loss, but for certain applications, the trade-off in color accuracy for efficiency might be acceptable. The following is an example of converting RGB888 to RGB565 in C code.

```
uint16_t rgb888_to_rgb565(uint8_t r, uint8_t g, uint8_t b)
{
    return ((r >> 3) << 11) | ((g >> 2) << 5) | (b >> 3);
}
```

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpLCD_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of *EthercatMaster* class based on the ID of the slave device on the network. If the lcd_id parameter is provided, the lcdInit() function will be called to initialize the LCD module.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster *master, uint16_t lcd_id, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, uint16_t lcd_id, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of EthercatMaster class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.
- `[in] uint16_t lcd_id`
  The ID of the LCD module to be initialized. For a list of supported LCD modules, please refer to lcdInit().

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11UN01 slave;

void setup(void) {
    master.begin();
    slave.attach(0, master);
}
void loop() {
  // ...
}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of EthercatMaster class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

Control Functions

Control functions for the EthercatDevice_DmpLCD_Generic class.

Functions:

- update()

update()

Description

Update state machines and internal variables for each function on the EtherCAT slave device.

Syntax

```
int update();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

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

LCD Functions

LCD functions for the EthercatDevice_DmpLCD_Generic class.

Functions:

- lcdInit()
- lcdFlush()
- lcdWidth()
- lcdHeight()
- lcdGetRotation()
- lcdSetRotation()
- lcdDrawPixel()
- lcdDrawFastHLine()
- lcdDrawFastVLine()
- lcdDrawLine()
- lcdFillRect()
- lcdDrawRect()
- lcdFillCircle()
- lcdDrawCircle()
- lcdFillTriangle()
- lcdDrawTriangle()
- lcdFillRoundRect()
- lcdDrawRoundRect()
- lcdFillScreen()
- lcdSetAddrWindow()
- lcdPushColors()
- lcdSetTextCursor()
- lcdSetTextColor()
- lcdSetTextSize()
- lcdSetTextWrap()
- lcdPrint()

`lcdInit()`

Description

Initialize the LCD module on the EtherCAT slave device.

Syntax

```
int lcdInit(uint16_t lcd_id = ECAT_LCD_UNKNOWN_ID);
```

Parameters

- `[in] uint16_t lcd_id`
  The ID of the LCD module to be initialized. The list of supported LCD modules is as follows, including the configuration of the touch pins (X_p, Y_p, X_m, Y_m):

Definition	ID	IC	Resolution	Xp	Yp	Xm	Ym
`ECAT_LCD_ILI9341_1`	0	ILI9341	240 X 320	D9	A2	A3	D8
`ECAT_LCD_ILI9341_2`	1	ILI9341	240 X 320	D6	A1	A2	D7
`ECAT_LCD_ILI9488_1`	2	ILI9488	320 X 480	D8	A3	A2	D9
`ECAT_LCD_ILI9486_1`	3	ILI9486	320 X 480	-	-	-	-
`ECAT_LCD_HX8347I_1`	4	HX8347-I(T)	240 X 320	D9	A2	A3	D8
`ECAT_LCD_HX8347D_1`	5	HX8347-D	240 X 320	D9	A2	A3	D8
`ECAT_LCD_UNKNOWN_ID`	0xFFFF	-	-	-	-	-	-

  If this parameter is `ECAT_LCD_UNKNOWN_ID`, the ID of the LCD module and calibration parameters are loaded from the storage on the EtherCAT slave device.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  // ...
}
```

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

lcdFlush()

Description

Wait for all graphics commands for the LCD on the EtherCAT slave device to complete execution.

Syntax

```
int lcdFlush();
```

Parameters

None.

Return Value

Return 1 to indicate that all graphics commands have been completed. Return 0 to indicate that they are still pending. If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  // Graphics commands.
  // ...
  slave.lcdFlush();
}
```

lcdWidth()

Description

Get the display width of the LCD on the EtherCAT slave device. This width will rotate according to the configuration of lcdSetRotation().

Syntax

```
int16_t lcdWidth();
```

Parameters

None.

Return Value

Return the display width of the LCD.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);

  Serial.print("LCD Width: ");
  Serial.println(slave.lcdWidth());
}

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

lcdHeight()

Description

Get the display height of the LCD on the EtherCAT slave device. This height will rotate according to the configuration of lcdSetRotation().

Syntax

```
int16_t lcdHeight();
```

Parameters

None.

Return Value

Return the display height of the LCD.

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);

  Serial.print("LCD Height: ");
  Serial.println(slave.lcdHeight());
}

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

lcdGetRotation()

Description

Get the current rotation for display of the LCD on the EtherCAT slave device.

Syntax

```
uint8_t lcdGetRotation();
```

Parameters

None.

Return Value

Return the current rotation for display of the LCD. For detailed rotation modes, please refer to lcdSetRotation().

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is non-blocking and can be called within the Cyclic Callback.

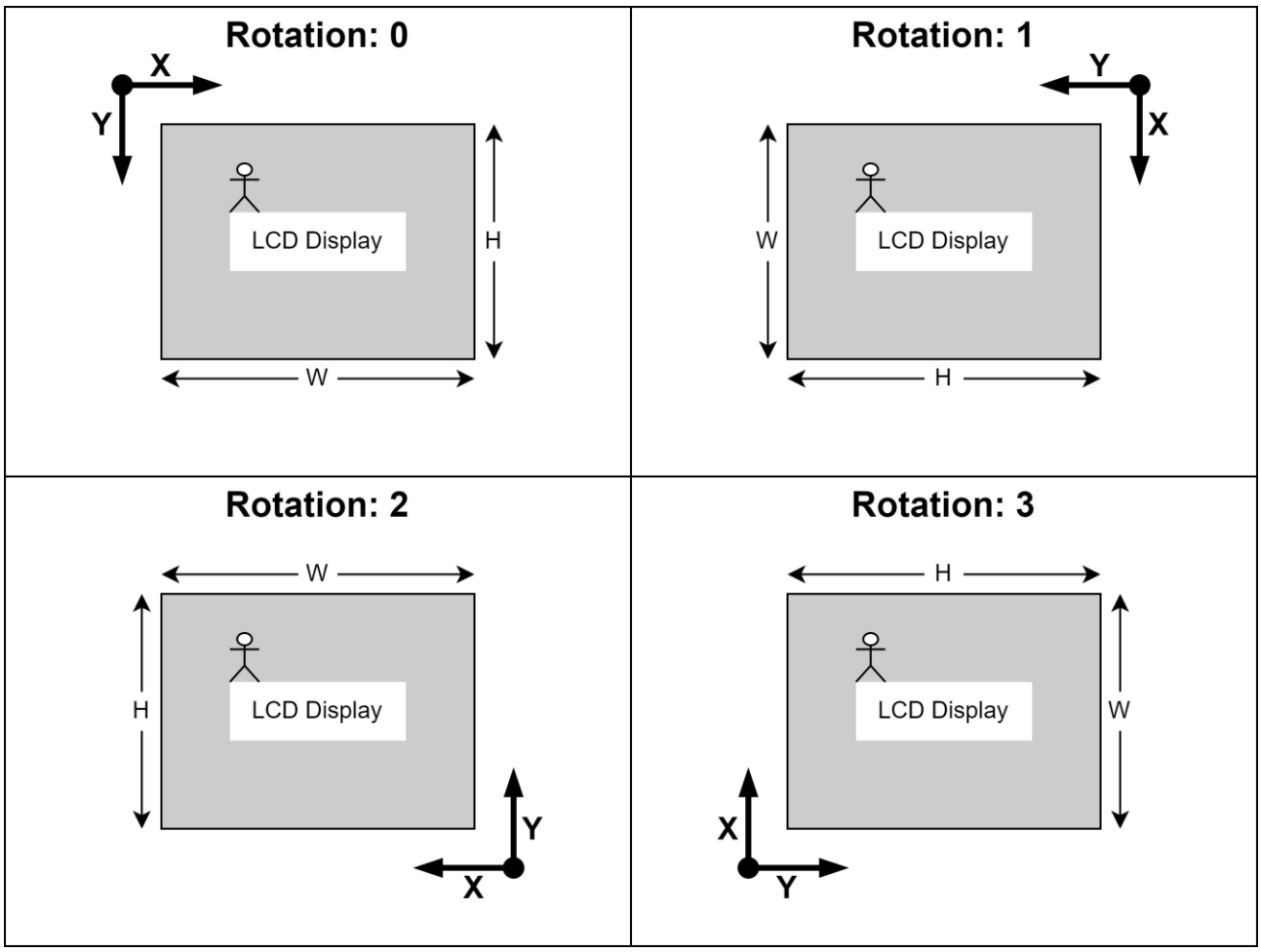
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);

  Serial.print("LCD Rotation: ");
  Serial.println(slave.lcdGetRotation());
}

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

`lcdSetRotation()`

Description

Set the current rotation for display of the LCD on the EtherCAT slave device. For details on rotation modes, please refer to the following figure.



Syntax

```
int lcdSetRotation(uint8_t x);
```

Parameters

- `[in] uint8_t x`

  The rotation mode to be configured. Since only four rotation modes are supported, this function will perform bitwise operations on the input parameter x, retaining only its lowest 2 bits and clearing the remaining bits to zero.

Return Value

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

Comment

This function must be called after a successful execution of [EthercatMaster::begin()](#).
This function is non-blocking and can be called within the Cyclic Callback.

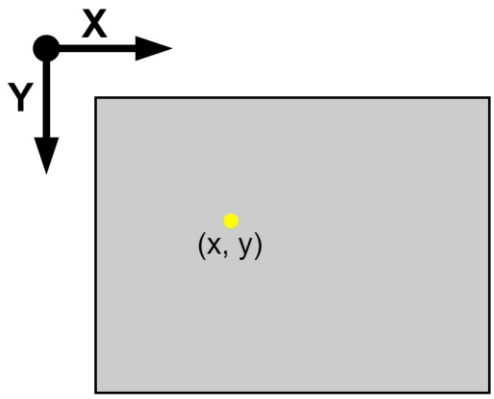
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  slave.lcdSetRotation(1);
}

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

`lcdDrawPixel()`

Description

Draw a single pixel on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawPixel(int16_t x, int16_t y, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the pixel to be drawn.
- `[in] int16_t y`
  The Y-axis position of the pixel to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

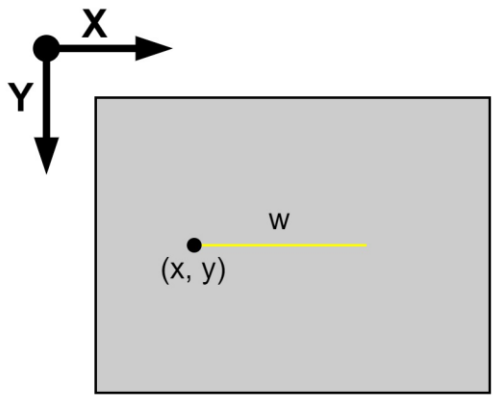
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
```

```
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawPixel(100, 100, 0xFFE0);
}

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

`lcdDrawFastHLine()`

Description

Draw a horizontal line on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a horizontal line to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a horizontal line to be drawn.
- `[in] int16_t w`
  The length of a horizontal line to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

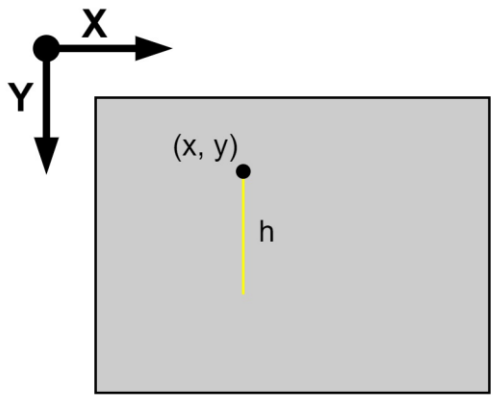
```
#include "Ethercat.h"

EthercatMaster master;
```

```
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawFastHLine(100, 100, 100, 0xFFE0);
}

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

`lcdDrawFastVLine()`

Description

Draw a vertical line on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a vertical line to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a vertical line to be drawn.
- `[in] int16_t h`
  The length of a vertical line to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

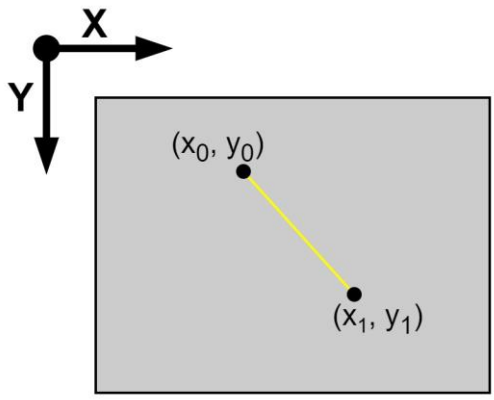
```
#include "Ethercat.h"

EthercatMaster master;
```

```
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawFastVLine(100, 100, 100, 0xFFE0);
}

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

`lcdDrawLine()`

Description

Draw a line on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color);
```

Parameters

- `[in] int16_t x0`
  The X-axis position of the starting point of a line to be drawn.
- `[in] int16_t y0`
  The Y-axis position of the starting point of a line to be drawn.
- `[in] int16_t x1`
  The X-axis position of the ending point of a line to be drawn.
- `[in] int16_t y1`
  The Y-axis position of the ending point of a line to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

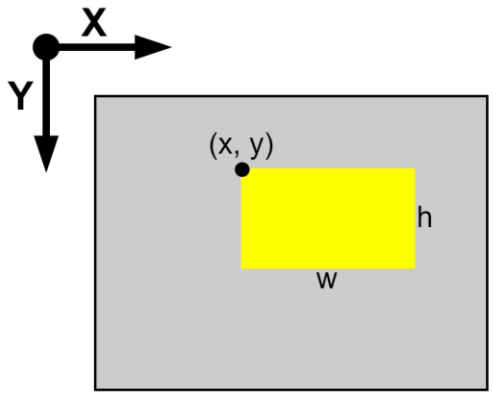
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawLine(100, 100, 200, 200, 0xFFE0);
}

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

`lcdFillRect()`

Description

Draw a rectangle outline on the LCD display of the EtherCAT slave device with a given color and fill it with the same color.



Syntax

```
int lcdFillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a rectangle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a rectangle to be drawn.
- `[in] int16_t w`
  The width of a rectangle to be drawn.
- `[in] int16_t h`
  The height of a rectangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

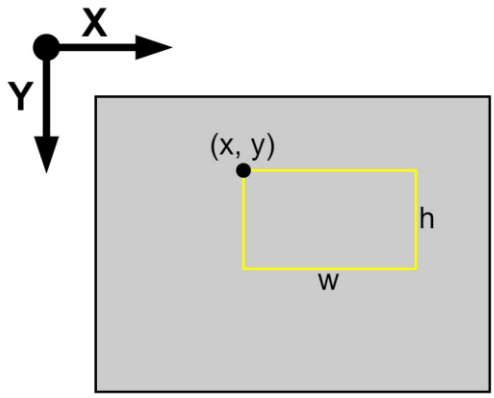
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdFillRect(100, 100, 100, 100, 0xFFE0);
}

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

`lcdDrawRect()`

Description

Draw a rectangle outline on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a rectangle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a rectangle to be drawn.
- `[in] int16_t w`
  The width of a rectangle to be drawn.
- `[in] int16_t h`
  The height of a rectangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

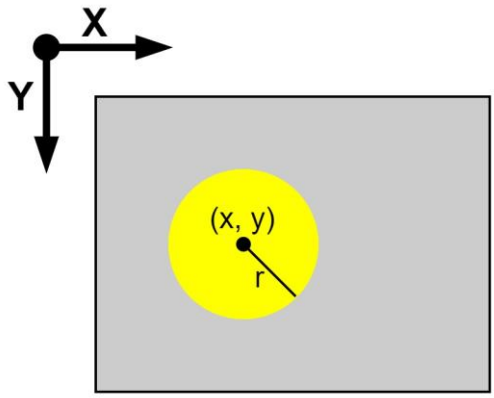
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawRect(100, 100, 100, 100, 0xFFE0);
}

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

`lcdFillCircle()`

Description

Draw a circle outline on the LCD display of the EtherCAT slave device with a given color and fill it with the same color.



Syntax

```
int lcdFillCircle(int16_t x, int16_t y, int16_t r, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the center point of a circle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the center point of a circle to be drawn.
- `[in] int16_t r`
  The radius of a circle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

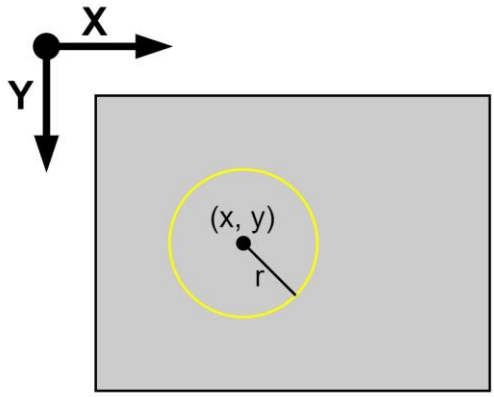
```
#include "Ethercat.h"
```

```
EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdFillCircle(100, 100, 50, 0xFFE0);
}

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

`lcdDrawCircle()`

Description

Draw a circle outline on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawCircle(int16_t x, int16_t y, int16_t r, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the center point of a circle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the center point of a circle to be drawn.
- `[in] int16_t r`
  The radius of a circle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

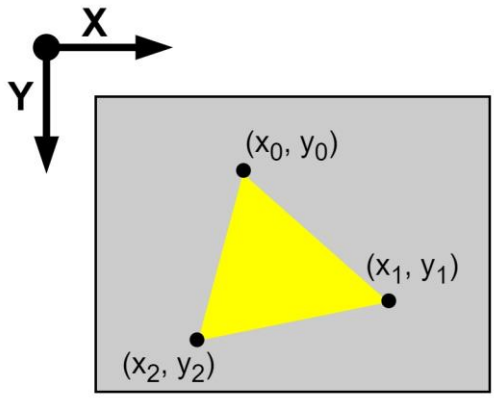
```
#include "Ethercat.h"

EthercatMaster master;
```

```
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawCircle(100, 100, 50, 0xFFE0);
}

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

`lcdFillTriangle()`

Description

Draw a triangle outline on the LCD display of the EtherCAT slave device with a given color and fill it with the same color.



Syntax

```
int lcdFillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
int16_t x2, int16_t y2, uint16_t color);
```

Parameters

- `[in] int16_t x0`
  The X-axis position of the 1st point of a triangle to be drawn.
- `[in] int16_t y0`
  The Y-axis position of the 1st point of a triangle to be drawn.
- `[in] int16_t x1`
  The X-axis position of the 2nd point of a triangle to be drawn.
- `[in] int16_t y1`
  The Y-axis position of the 2nd point of a triangle to be drawn.
- `[in] int16_t x2`
  The X-axis position of the 3rd point of a triangle to be drawn.
- `[in] int16_t y2`
  The Y-axis position of the 3rd point of a triangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

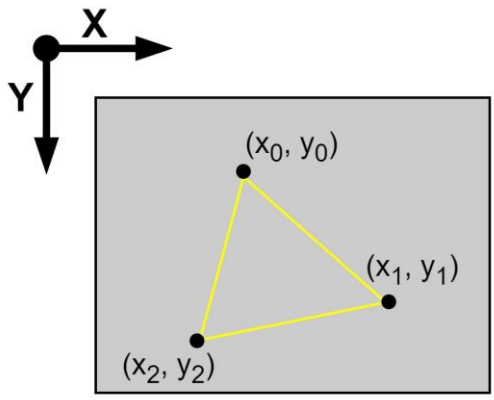
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdFillTriangle(100, 100, 200, 200, 80, 220, 0xFFE0);
}

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

lcdDrawTriangle()

Description

Draw a triangle outline on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
int16_t x2, int16_t y2, uint16_t color);
```

Parameters

- `[in] int16_t x0`
  The X-axis position of the 1st point of a triangle to be drawn.
- `[in] int16_t y0`
  The Y-axis position of the 1st point of a triangle to be drawn.
- `[in] int16_t x1`
  The X-axis position of the 2nd point of a triangle to be drawn.
- `[in] int16_t y1`
  The Y-axis position of the 2nd point of a triangle to be drawn.
- `[in] int16_t x2`
  The X-axis position of the 3rd point of a triangle to be drawn.
- `[in] int16_t y2`
  The Y-axis position of the 3rd point of a triangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin().
This function is non-blocking and can be called within the Cyclic Callback.

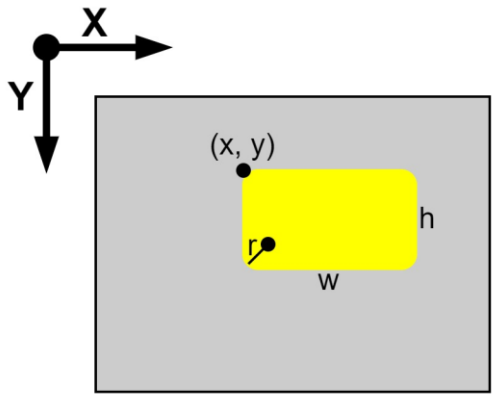
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawTriangle(100, 100, 200, 200, 80, 220, 0xFFE0);
}

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

`lcdFillRoundRect()`

Description

Draw a rounded rectangle outline on the LCD display of the EtherCAT slave device with a given color and fill it with the same color.



Syntax

```
int lcdFillRoundRect(int16_t x, int16_t y, int16_t w, int16_t h, int16_t r, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a rounded rectangle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a rounded rectangle to be drawn.
- `[in] int16_t w`
  The width of a rounded rectangle to be drawn.
- `[in] int16_t h`
  The height of a rounded rectangle to be drawn.
- `[in] int16_t r`
  The corner radius of a rounded rectangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin().
This function is non-blocking and can be called within the Cyclic Callback.

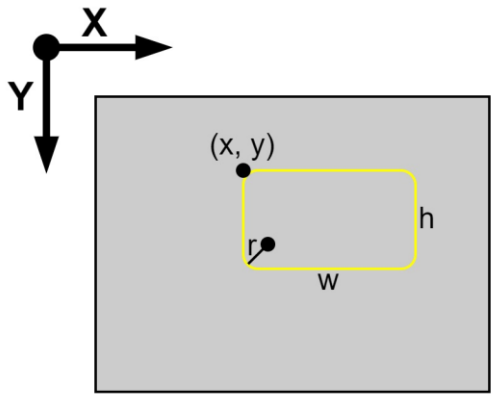
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdFillRoundRect(100, 100, 100, 100, 20, 0xFFE0);
}

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

`lcdDrawRoundRect()`

Description

Draw a rounded rectangle outline on the LCD display of the EtherCAT slave device with a given color.



Syntax

```
int lcdDrawRoundRect(int16_t x, int16_t y, int16_t w, int16_t h, int16_t r, uint16_t color);
```

Parameters

- `[in] int16_t x`
  The X-axis position of the starting point of a rounded rectangle to be drawn.
- `[in] int16_t y`
  The Y-axis position of the starting point of a rounded rectangle to be drawn.
- `[in] int16_t w`
  The width of a rounded rectangle to be drawn.
- `[in] int16_t h`
  The height of a rounded rectangle to be drawn.
- `[in] int16_t r`
  The corner radius of a rounded rectangle to be drawn.
- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdDrawRoundRect(100, 100, 100, 100, 20, 0xFFE0);
}

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

lcdFillScreen()

Description

Fill the entire LCD display with a given color on the EtherCAT slave device.

Syntax

```
int lcdFillScreen(uint16_t color);
```

Parameters

- `[in] uint16_t color`
  The color of the pixel to be drawn. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

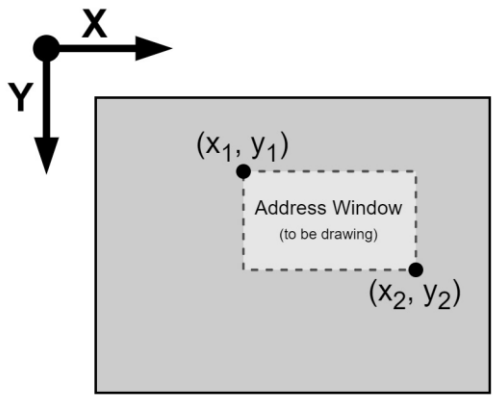
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdFillScreen(0xFFE0);
}

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

```
}
```

`lcdSetAddrWindow()`

Description

Set the address window on the LCD display of the EtherCAT slave device. Subsequent pixel-drawing operations will be performed within this area. Used in conjunction with `lcdPushColors()` function.



Syntax

```
int lcdSetAddrWindow(int x1, int y1, int x2, int y2);
```

Parameters

- `[in] int x1`
  The X-axis position of the top-left corner of the window.
- `[in] int y1`
  The Y-axis position of the top-left corner of the window.
- `[in] int x2`
  The X-axis position of the bottom-right corner of the window. x2 must be greater than or equal to x1 for a valid window.
- `[in] int y2`
  The Y-axis position of the bottom-right corner of the window. y2 must be greater than or equal to y1 for a valid window.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

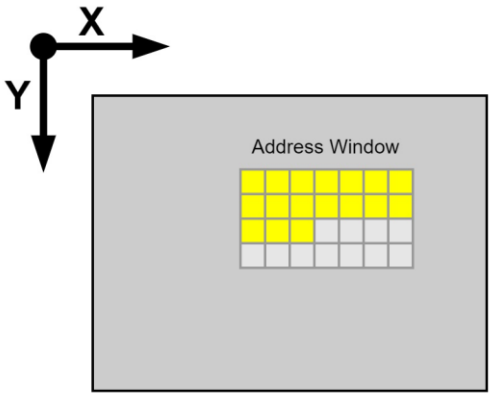
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

uint16_t buffer[256];

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  for (int i = 0; i < 128; i++)
    buffer[i] = 0xFFE0;
  for (int i = 0; i < 128; i++)
    buffer[i + 128] = 0xF800;

  slave.lcdSetAddrWindow(100, 100, 115, 115);
  slave.lcdPushColors(&buffer[0], 128, true);
  slave.lcdPushColors(&buffer[128], 128, false);
}

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

lcdPushColors()

Description

Send an array of 16-bit color values to the LCD display on the EtherCAT slave device for pixel-drawing. This function assumes that the lcdSetAddrWindow() function has been previously called to define the address window for the drawing operation.



Syntax

```
int lcdPushColors(uint16_t *data, uint8_t len, bool first);
```

Parameters

- `[in] uint16_t *data`
  Pointer to an array of 16-bit color values. Each element in the array represents the color of a pixel to be displayed.
- `[in] uint8_t len`
  The length of 16-bit color values in the data array. This indicates the total number of pixels to be drawn. If the length of the data array exceeds the number of pixels in the LCD address window, the excess portion will be drawn starting from the top-left corner of the LCD address window.
- `[in] bool first`
  Setting this parameter to true moves the current drawing position back to the top-left corner of the LCD address window. Setting it to false keeps the current drawing position unchanged.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin().
This function is non-blocking and can be called within the Cyclic Callback.

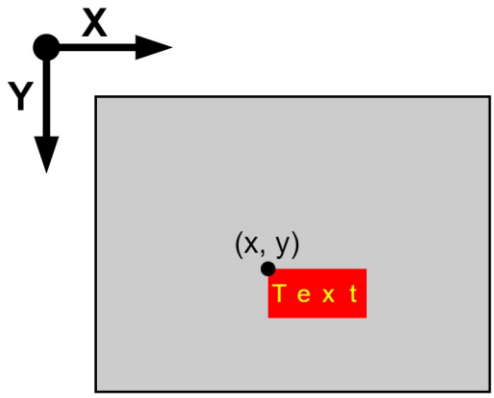
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

uint16_t buffer[256];

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  for (int i = 0; i < 128; i++)
    buffer[i] = 0xFFE0;
  for (int i = 0; i < 128; i++)
    buffer[i + 128] = 0xF800;

  slave.lcdSetAddrWindow(100, 100, 115, 115);
  slave.lcdPushColors(&buffer[0], 128, true);
  slave.lcdPushColors(&buffer[128], 128, false);
}

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

lcdSetTextCursor()

Description

Move the text cursor on the LCD display of the EtherCAT slave device to the specified pixel position. Subsequent text will be printed starting from that position.



Syntax

```
int lcdSetTextCursor(int16_t x, int16_t y);
```

Parameters

- `[in] int16_t x`

  The X-axis position of the desired text cursor.

- `[in] int16_t y`

  The Y-axis position of the desired text cursor.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

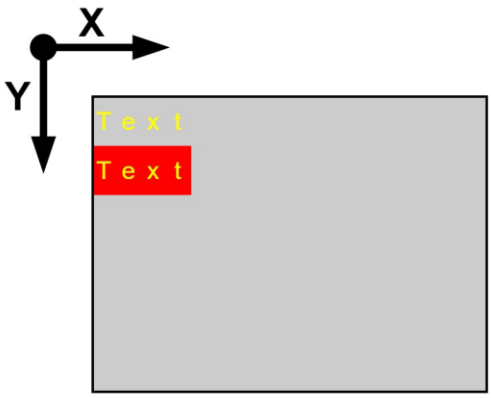
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}
```

```
void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdPrint("Hello World!\n");
  slave.lcdSetTextCursor(100, 100);
  slave.lcdPrint("Hello World!\n");
}

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

lcdSetTextColor()

Description

Set the font color for the text to be printed on the LCD display of the EtherCAT slave device. Optionally, set the background color for the text as well.



Syntax

```
int lcdSetTextColor(uint16_t color);
int lcdSetTextColor(uint16_t color, uint16_t background);
```

Parameters

- `[in] uint16_t color`
  The font color for the text to be printed. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.
- `[in] uint16_t background`
  The background color for the text to be printed. It is a 16-bit unsigned integer that encodes the color information using the RGB565 format.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;
```

```
void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdSetTextColor(0xFFE0);
  slave.lcdPrint("Hello World!\n");
  slave.lcdSetTextColor(0xFFE0, 0xF800);
  slave.lcdPrint("Hello World!\n");
}

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

lcdSetTextSize()

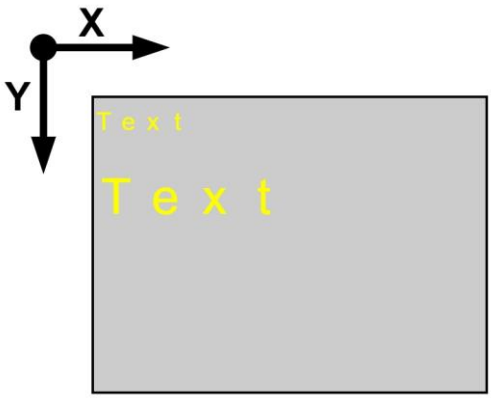
Description

Set the font size multiplier for the text to be printed on the LCD display of the EtherCAT slave device. The default font size multiplier is 1, which corresponds to a 6 x 8 font. This means the text will be 6 pixels wide and 8 pixels tall. You can increase the font size multiplier by setting the value to 2, 3, or higher. Each increment in the font size multiplier results in a larger font. For example:

- s = 2: 12 x 16 pixels.
- s = 3: 18 x 24 pixels.
- s = 4: 24 x 32 pixels.
- And so on.

Beyond uniform scaling, fonts can also be scaled individually in width and height. This means that the font can be stretched horizontally or vertically without affecting the other dimension. For example:

- w = 1, h = 2: 6 x 16 pixels.
- w = 2, h = 1: 12 x 8 pixels.
- w = 2, h = 4: 12 x 32 pixels.



Syntax

```
int lcdSetTextSize(uint8_t s);
int lcdSetTextSize(uint8_t w, uint8_t h);
```

Parameters

- `[in] uint8_t s`

  The font size multiplier for the text to be printed.

- `[in] uint8_t w`

  The font width multiplier for the text to be printed.

- `[in] uint8_t h`

The font height multiplier for the text to be printed.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

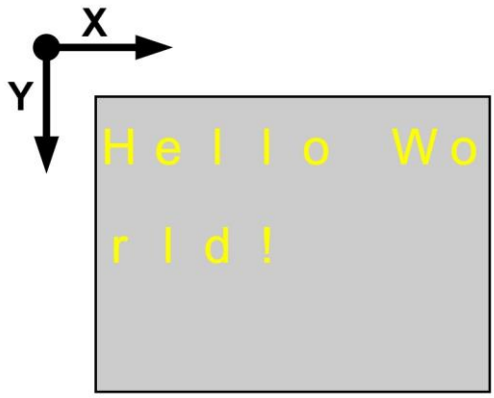
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdSetTextSize(2);
  slave.lcdPrint("Hello World!\n");
  slave.lcdSetTextSize(2, 4);
  slave.lcdPrint("Hello World!\n");
}

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

lcdSetTextWrap()

Description

Set whether text that is too long for the width of the LCD display should automatically wrap around to the next line or clip right.



Syntax

```
int lcdSetTextWrap(bool wrap);
```

Parameters

- `[in] bool wrap`
  A Boolean value used to control whether text wrapping is enabled or disabled.
  true: Enable text wrapping.
  false: Disable text wrapping.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

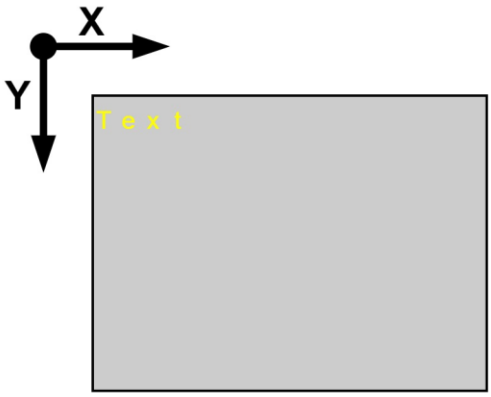
```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

void CyclicCallback() {
  slave.update();
}
```

```
void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdSetTextWrap(true);
  slave.lcdPrint("Hello World!\n");
}

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

`lcdPrint()`

Description

Print the specified string to the LCD display on the EtherCAT slave device.



Syntax

```
int lcdPrint(const char *fmt, ...);
```

Parameters

- `[in] const char *fmt`
  The string to be printed on the LCD display. This is a pointer to a null-terminated string containing the format specification for the output. The format string follows the same format as the printf function in C, allowing for insertion of variables and formatting options.
- `[in] ...`
  This is a variable number of arguments that will be inserted into the formatted string according to the format specifiers in the fmt string.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;
```

```
void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.lcdPrint("Hello World!\n");
}

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

Touch Screen Functions

Touch Screen functions for the EthercatDevice_DmpLCD_Generic class.

Functions:

- touchCalibration()
- touchX()
- touchY()
- isTouched()

touchCalibration()

Description

Touchscreen calibration routine for the LCD module on the EtherCAT slave device. This function is a non-blocking function that contains the calibration routine state machine. It must be called continuously until it returns a non-zero value, indicating that the calibration routine is complete.

Syntax

```
int touchCalibration(int flag = 0);
```

Parameters

- `[in] int flag`

  If the value of this parameter is -1, the current touchscreen calibration routine will be canceled. In this case, the function will return 1 to indicate cancellation. Other values for this parameter have no effect on the calibration routine.

Return Value

Return the current status of the touchscreen calibration routine.

- 0: Calibration in progress.
- 1: Calibration successful.
- -1: Calibration failed.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

int rc;

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

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

```
  slave.lcdInit(ECAT_LCD_ILI9341_1);

  while ((rc = slave.touchCalibration()) == 0);
  if (rc > 0)
    Serial.println("Touch Screen calibration successful.");
  else
    Serial.println("Touch Screen calibration failed.");
}

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

touchX()

Description

Read the X-axis position of the touch point on the touchscreen of the LCD module on the EtherCAT slave device. The coordinate will be rotated according to the configuration of `lcdSetRotation()`.

Syntax

```
int touchX(size_t point = 0);
```

Parameters

- `[in] size_t point`
  Touch point sequence number. If the device supports multi-touch, this parameter can be used to read the X-axis position of the specified touch point.
  0: 1st touch point.
  1: 2nd touch point.
  And so on.

Return Value

Return the X-axis position of the touch point. if the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

bool Touched = false;
int TouchX;
int TouchY;

void CyclicCallback() {
  if (!Touched && slave.isTouched() > 0) {
    Touched = true;
    TouchX = slave.touchX();
    TouchY = slave.touchY();
  }
}
```

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  if (Touched) {
    Serial.print("Touched. X: ");
    Serial.print(TouchX);
    Serial.print(", Y: ");
    Serial.println(TouchY);
    delay(500);
    Touched = false;
  }
  // ...
}
```

touchY()

Description

Read the Y-axis position of the touch point on the touchscreen of the LCD module on the EtherCAT slave device. The coordinate will be rotated according to the configuration of `lcdSetRotation()`.

Syntax

```
int touchY(size_t point = 0);
```

Parameters

- `[in] size_t point`
  Touch point sequence number. If the device supports multi-touch, this parameter can be used to read the Y-axis position of the specified touch point.
  0: 1st touch point.
  1: 2nd touch point.
  And so on.

Return Value

Return the Y-axis position of the touch point. if the return value is smaller than 0, it means an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

bool Touched = false;
int TouchX;
int TouchY;

void CyclicCallback() {
  if (!Touched && slave.isTouched() > 0) {
    Touched = true;
    TouchX = slave.touchX();
    TouchY = slave.touchY();
  }
}
```

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  if (Touched) {
    Serial.print("Touched. X: ");
    Serial.print(TouchX);
    Serial.print(", Y: ");
    Serial.println(TouchY);
    delay(500);
    Touched = false;
  }
  // ...
}
```

isTouched()

Description

Get the number of touch points on the touchscreen of the EtherCAT slave device. For devices that only support single-touch, this function can be used to check if the screen is being touched.

Syntax

```
int isTouched();
```

Parameters

None.

Return Value

Return the number of touch points on the touchscreen. If the return value is less than 0, it indicates an error code.

Comment

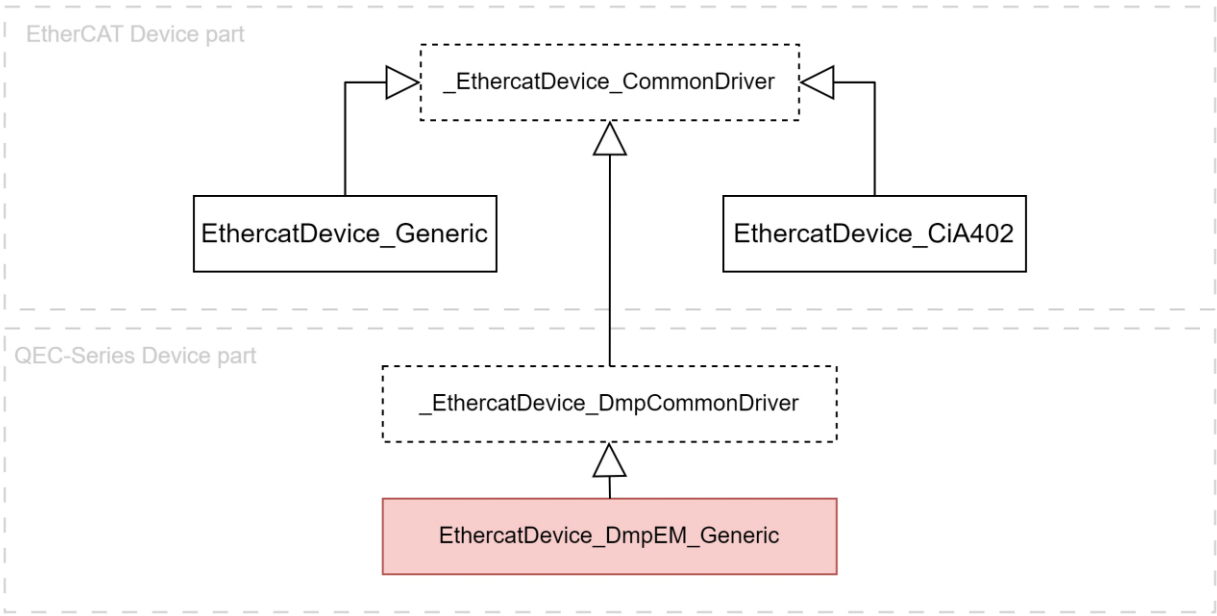
This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00UN01 slave;

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

  master.begin();
  slave.attach(0, master);
  slave.lcdInit(ECAT_LCD_ILI9341_1);
  master.start();
}

void loop() {
  if (slave.isTouched() > 0) {
    Serial.println("Touched");
    delay(500);
  }
}
```

2.3.6 EthercatDevice_DmpEM_Generic

EthercatDevice_DmpEM_Generic is an EtherCAT slave class developed for the DM&P Group's environmental monitoring EtherCAT slave module. This module features environmental monitoring (temperature, humidity, pressure, PM2.5, CO2, etc.), battery status monitoring, speaker, and digital input/output functions.
The class relationships of *EthercatDevice_DmpEM_Generic* are illustrated in the following diagram:



- *EthercatDevice_DmpEM_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.

Base Class:

- _EthercatDevice_CommonDriver

Derived Class:

Class Name	Vendor ID	Product Code	Input Channels	Output Channels	Speaker
EthercatDevice_QECRxxEMxS	0x00000bc3	0x0086d7ff	1	2	O

Function Groups:

- Initialization
- Control
- Environment
- Battery
- Digital I/O

- Speaker

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of EthercatMaster class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of EthercatMaster class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECRxxEMxS slave;

void setup(void) {
  master.begin();
```

```
  slave.attach(0, master);
  master.start();
}
void loop() {
  // ...
}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of *EthercatMaster* class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECRxxEMxS slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

}
```

Control Functions

Control functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- update()

update()

Description

Update state machines and internal variables for each function on the EtherCAT slave device.

Syntax

```
int update();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Slave1.update();
}
```

Environment Functions

Environment functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- environmentReadTemperature()
- environmentReadHumidity()
- environmentReadPressure()
- environmentReadPM25()
- environmentReadCO2()
- environmentReadFlammableGas()
- environmentReadCO()
- environmentReadO2()
- environmentReadIllumination()
- environmentReadPIR()

environmentReadTemperature()

Description

Read Temperature value.

Syntax

```
double environmentReadTemperature();
```

Parameters

None.

Return Value

Temperature value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadTemperature());
  delay(1000);
}
```

environmentReadHumidity()

Description

Read Humidity value.

Syntax

```
double environmentReadHumidity();
```

Parameters

None.

Return Value

Humidity value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadHumidity());
  delay(1000);
}
```

environmentReadPressure()

Description

Read Pressure value.

Syntax

```
int environmentReadPressure();
```

Parameters

None.

Return Value

Pressure value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadPressure());
  delay(1000);
}
```

environmentReadPM25()

Description

Read PM2.5 value.

Syntax

```
int environmentReadPM25();
```

Parameters

None.

Return Value

PM2.5 value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadPM25());
  delay(1000);
}
```

environmentReadCO2()

Description

Read CO^2 value.

Syntax

```
int environmentReadCO2();
```

Parameters

None.

Return Value

CO^2 value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadCO2());
  delay(1000);
}
```

environmentReadFlammableGas()

Description

Read Flammable Gas value.

Syntax

```
int environmentReadFlammableGas();
```

Parameters

None.

Return Value

Flammable Gas value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadFlammableGas());
  delay(1000);
}
```

environmentReadCO()

Description

Read CO value.

Syntax

```
int environmentReadCO();
```

Parameters

None.

Return Value

CO value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadCO());
  delay(1000);
}
```

environmentReadO2()

Description

Read O^2 value.

Syntax

```
int environmentReadO2();
```

Parameters

None.

Return Value

O^2 value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadO2());
  delay(1000);
}
```

environmentReadIllumination()

Description

Read Illumination value.

Syntax

```
int environmentReadIllumination();
```

Parameters

None.

Return Value

Illumination value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadIllumination());
  delay(1000);
}
```

environmentReadPIR()

Description

Read PIR value.

Syntax

```
int environmentReadPIR();
```

Parameters

None.

Return Value

PIR value.

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.environmentReadPIR());
  delay(1000);
}
```

Battery Functions

Battery functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- batteryReadStatus()

batteryReadStatus()

Description

Read Battery Status.

Syntax

```
int batteryReadStatus();
```

Return Value

Battery Status.

- 0: Full
- 1: Charging
- 2: No Battery
- 3: Discharging

Comment

This function must be called after a successful execution of EthercatMaster::start().
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.batteryReadStatus());
  delay(1000);
}
```

Digital I/O Functions

Digital I/O functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- digitalWrite()
- digitalRead()
- emergencyLightWrite()

digitalWrite()

Description

Write a HIGH or a LOW value to a digital output pin on the EtherCAT Slave device.

Syntax

```
int digitalWrite(int pin, uint8_t value);
```

Parameters

- `[in] int pin`
  Digital Input/Output pin number.
- `[in] uint8_t value`
  HIGH or LOW.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

void setup() {
  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Slave1.digitalWrite(0, HIGH);
  delay(1000);
  Slave1.digitalWrite(0, LOW);
  delay(1000);
}
```

digitalRead()

Description

Read a HIGH or LOW value from a digital input pin on the EtherCAT Slave device.

Syntax

```
int digitalRead(uint8_t pin);
```

Parameters

- `[in] uint8_t pin`

  Digital Input/Output pin number.

Return Value

Int: the value of the digital input pin on the EtherCAT Slave.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`.
This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

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

  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}

void loop() {
  Serial.println(Slave1.digitalRead(0));
  delay(1000);
}
```

emergencyLightWrite()

Description

Write emergency Light.

Syntax

```
int emergencyLightWrite(uint8_t value);
```

Parameters

- `[in] uint8_t value`

  Emergency Light Value.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster EcatMaster;
EthercatDevice_QECRxxEMxS Slave1;

void setup() {
  EcatMaster.begin();
  Slave1.attach(0, EcatMaster);
  EcatMaster.start(1000000, ECAT_FREERUN);
}
void loop() {
  Slave1.emergencyLightWrite(1);
  delay(1000);
  Slave1.emergencyLightWrite(0);
  delay(1000);
}
```

Speaker Functions

Speaker functions for the EthercatDevice_DmpEM_Generic class.

Functions:

- speakerSetVolume()
- speakerWrite()

speakerSetVolume()

Description

Set Speaker Volume.

Syntax

```
int speakerSetVolume(uint16_t volume);
```

Parameters

- `[in] uint16_t volume`

  Speaker Volume, from 0 to 256.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

speakerWrite()

Description

Set Speaker Audio Raw path.

Syntax

```
int speakerWrite(EthercatAudioRawType raw);
```

Parameters

- `[in] EthercatAudioRawType raw`
  Speaker Audio Raw path.

Return Value

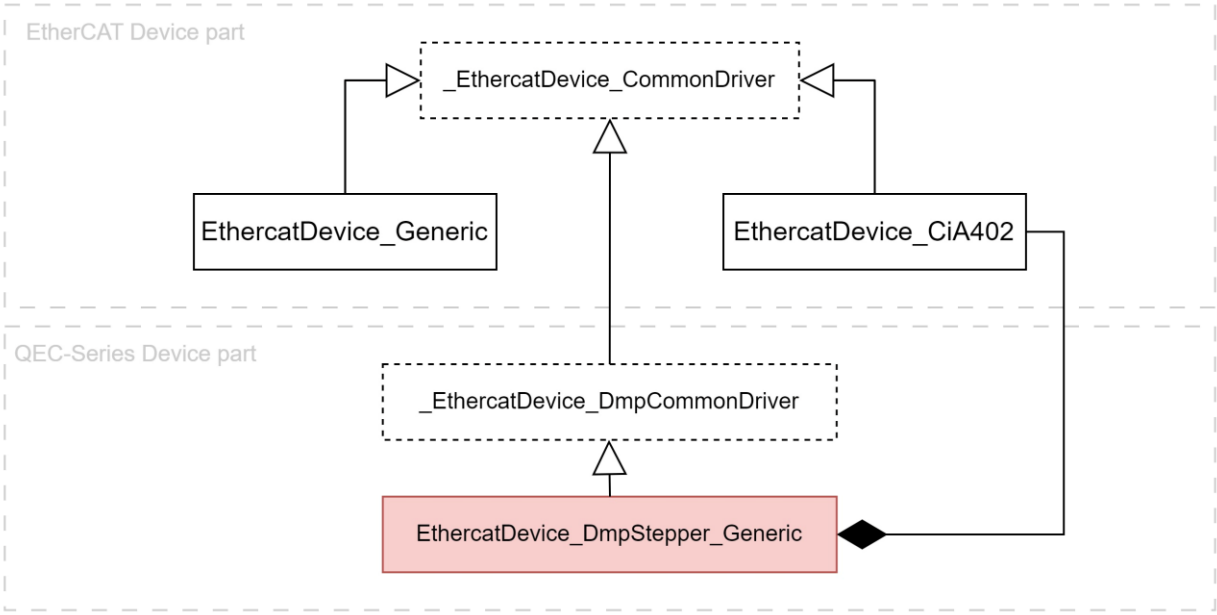
Return an error code. If the returned value is zero, it indicates a successful execution of this function.

Comment

This function must be called after a successful execution of `EthercatMaster::start()`. This function is non-blocking and can be called within the Cyclic Callback.

2.3.7 EthercatDevice_DmpStepper_Generic

EthercatDevice_DmpStepper_Generic is an EtherCAT slave class developed for the DM&P Group's 3-axis stepper motor controller EtherCAT slave module. This module features motor drivers, encoder inputs, and other CNC-related functions. In the motor control section, it not only supports the CiA 402 mode but also the 3-axis synchronous G-code mode.
The class relationships of *EthercatDevice_DmpStepper_Generic* are illustrated in the following diagram:



- *EthercatDevice_DmpStepper_Generic* inherits from *_EthercatDevice_DmpCommonDriver*.
- *EthercatDevice_DmpStepper_Generic* is composed of *EthercatDevice_CiA402*.

Base Class:

- _EthercatDevice_CommonDriver

Derived Class:

Class Name	Vendor ID	Product Code
EthercatDevice_QECR11MP3S	0x00000bc3	0x0086d0d6
EthercatDevice_QECR00MP3S	0x00000bc3	0x0086d0d9

Function Groups:

- Initialization
- Control

- CiA 402
- Machine
- Encoder
- Configuration

Machine Axis

- `ECAT_MACHINE_X_AXIS (0)`
- `ECAT_MACHINE_Y_AXIS (1)`
- `ECAT_MACHINE_Z_AXIS (2)`

Encoder List

- `ECAT_ENCODER_1 (0x01)`
- `ECAT_ENCODER_2 (0x02)`
- `ECAT_ENCODER_3 (0x03)`
- `ECAT_ENCODER_X (0x11)`
- `ECAT_ENCODER_Y (0x12)`
- `ECAT_ENCODER_Z (0x13)`

Index List

- ECAT_EMERGENCY_STOPPED (1)
- ECAT_MACHINE_X_AXIS_LIMIT_TOUCHED (2)
- ECAT_MACHINE_Y_AXIS_LIMIT_TOUCHED (3)
- ECAT_MACHINE_Z_AXIS_LIMIT_TOUCHED (4)
- IS_ECAT_ENCODER_1_INDEX_RESET(event) (((event) & 0x00000300) == 0x00000100)
- IS_ECAT_ENCODER_2_INDEX_RESET(event) (((event) & 0x00000C00) == 0x00000400)
- IS_ECAT_ENCODER_3_INDEX_RESET(event) (((event) & 0x00003000) == 0x00001000)
- IS_ECAT_ENCODER_X_INDEX_RESET(event) (((event) & 0x0000C000) == 0x00004000)
- IS_ECAT_ENCODER_Y_INDEX_RESET(event) (((event) & 0x00030000) == 0x00010000)
- IS_ECAT_ENCODER_Z_INDEX_RESET(event) (((event) & 0x000C0000) == 0x00040000)
- IS_ECAT_ENCODER_1_OVERFLOW(event) (((event) & 0x00000300) == 0x00000200)
- IS_ECAT_ENCODER_2_OVERFLOW(event) (((event) & 0x00000C00) == 0x00000800)
- IS_ECAT_ENCODER_3_OVERFLOW(event) (((event) & 0x00003000) == 0x00002000)
- IS_ECAT_ENCODER_X_OVERFLOW(event) (((event) & 0x0000C000) == 0x00008000)
- IS_ECAT_ENCODER_Y_OVERFLOW(event) (((event) & 0x00030000) == 0x00020000)
- IS_ECAT_ENCODER_Z_OVERFLOW(event) (((event) & 0x000C0000) == 0x00080000)
- IS_ECAT_ENCODER_1_UNDERFLOW(event) (((event) & 0x00000300) == 0x00000300)
- IS_ECAT_ENCODER_2_UNDERFLOW(event) (((event) & 0x00000C00) == 0x00000C00)
- IS_ECAT_ENCODER_3_UNDERFLOW(event) (((event) & 0x00003000) == 0x00003000)
- IS_ECAT_ENCODER_X_UNDERFLOW(event) (((event) & 0x0000C000) == 0x0000C000)
- IS_ECAT_ENCODER_Y_UNDERFLOW(event) (((event) & 0x00030000) == 0x00030000)
- IS_ECAT_ENCODER_Z_UNDERFLOW(event) (((event) & 0x000C0000) == 0x000C0000)

Initialization Functions

Initialization-related functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- attach()
- detach()

attach()

Description

Initialize the object of this EtherCAT slave device class and attach it to the object of *EthercatMaster* class based on the ID of the slave device on the network.

Syntax

```
int attach(uint16_t slave_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] uint16_t slave_id`
  The ID of the slave device on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] EthercatMaster *master`
  The object of *EthercatMaster* class to which it should be attached.
- `[in] EthercatAttachMode mode`
  The definition of slave_id:
  a. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT slave device on the network, 0 indicates the first slave device, 1 indicates the second slave device, and so on.
  b. `ECAT_ALIAS_ADDRESS`
     The alias address of the slave device on the network, which is defined at byte offset 8 in the SII EEPROM of the slave device.

Return Value

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

Comment

The function must be called after `EthercatMaster::begin()` and before `EthercatMaster::start()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
   master.begin();
```

```
    slave.attach(0, master);
    master.start();
}
void loop() {
  // ...
}
```

detach()

Description

Deinitialize the object of this EtherCAT slave device class and detach it from the object of *EthercatMaster* class.

Syntax

```
int detach();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `attach()`.

WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  delay(3000);

  slave.detach();
  master.end();
}

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

}
```

Control Functions

Control functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- update()
- attachInterrupt()

update()

Description

Update state machines and internal variables for each function on the EtherCAT slave device.

Syntax

```
int update();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is non-blocking and can be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

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

attachInterrupt()

Description

Register the event callback function for the EtherCAT slave device. Since this callback function is called within update(), it is prohibited to call blocking functions and system call-related functions within this callback function if update() is called in an interrupt callback function (such as *Cyclic Callback*, *Error Callback*, or *Event Callback*).

Syntax

```
int attachInterrupt(void (*callback)(int));
```

Parameters

- `[in] void (*callback)(int)`

  The event callback function to be registered has an integer-type parameter that indicates the event type. The supported event types are as follows:

Definition	Code	Description
ECAT_EMERGENCY_STOPPED	1	Emergency stop occurred.
ECAT_MACHINE_X_AXIS_LIMIT_TOUCHED	2	The X-axis limit switch has been touched.
ECAT_MACHINE_Y_AXIS_LIMIT_TOUCHED	3	The Y-axis limit switch has been touched.
ECAT_MACHINE_Z_AXIS_LIMIT_TOUCHED	4	The Z-axis limit switch has been touched.

  The remaining event types are determined using the following macros. Since the status of the following events are stored in the process data and this data is not latched, it is recommended to call update() in the cyclic callback to handle these events and prevent event loss.

Definition	Description
IS_ECAT_ENCODER_1_INDEX_RESET(event)	The Index Reset event of Encoder 1 has been triggered.
IS_ECAT_ENCODER_2_INDEX_RESET(event)	The Index Reset event of Encoder 2 has been triggered.
IS_ECAT_ENCODER_3_INDEX_RESET(event)	The Index Reset event of Encoder 3 has been triggered.
IS_ECAT_ENCODER_X_INDEX_RESET(event)	The Index Reset event of Encoder X has been triggered.
IS_ECAT_ENCODER_Y_INDEX_RESET(event)	The Index Reset event of Encoder Y has been triggered.

IS_ECAT_ENCODER_Z_INDEX_RESET(event)	The Index Reset event of Encoder Z has been triggered.
IS_ECAT_ENCODER_1_OVERFLOW(event)	The Overflow event of Encoder 1 has been triggered.
IS_ECAT_ENCODER_2_OVERFLOW(event)	The Overflow event of Encoder 2 has been triggered.
IS_ECAT_ENCODER_3_OVERFLOW(event)	The Overflow event of Encoder 3 has been triggered.
IS_ECAT_ENCODER_X_OVERFLOW(event)	The Overflow event of Encoder X has been triggered.
IS_ECAT_ENCODER_Y_OVERFLOW(event)	The Overflow event of Encoder Y has been triggered.
IS_ECAT_ENCODER_Z_OVERFLOW(event)	The Overflow event of Encoder Z has been triggered.
IS_ECAT_ENCODER_1_UNDERFLOW(event)	The Underflow event of Encoder 1 has been triggered.
IS_ECAT_ENCODER_2_UNDERFLOW(event)	The Underflow event of Encoder 2 has been triggered.
IS_ECAT_ENCODER_3_UNDERFLOW(event)	The Underflow event of Encoder 3 has been triggered.
IS_ECAT_ENCODER_X_UNDERFLOW(event)	The Underflow event of Encoder X has been triggered.
IS_ECAT_ENCODER_Y_UNDERFLOW(event)	The Underflow event of Encoder Y has been triggered.
IS_ECAT_ENCODER_Z_UNDERFLOW(event)	The Underflow event of Encoder Z has been triggered.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

int emergency_stopped;
int x_limit_touched;
```

```
int y_limit_touched;
int z_limit_touched;
int encoder_index_reset[3];
int encoder_overflow[3];
int encoder_underflow[3];
int encoder_xyz_index_reset[3];
int encoder_xyz_overflow[3];
int encoder_xyz_underflow[3];

void Callback(int event) {
  switch (event) {
    case ECAT_EMERGENCY_STOPPED:
      emergency_stopped = 1;
      return;
    case ECAT_MACHINE_X_AXIS_LIMIT_TOUCHED:
      x_limit_touched = 1;
      return;
    case ECAT_MACHINE_Y_AXIS_LIMIT_TOUCHED:
      y_limit_touched = 1;
      return;
    case ECAT_MACHINE_Z_AXIS_LIMIT_TOUCHED:
      z_limit_touched = 1;
      return;
    default:
      break;
  }

  if (IS_ECAT_ENCODER_1_INDEX_RESET(event))
    encoder_index_reset[0] = 1;
  else if (IS_ECAT_ENCODER_2_INDEX_RESET(event))
    encoder_index_reset[1] = 1;
  else if (IS_ECAT_ENCODER_3_INDEX_RESET(event))
    encoder_index_reset[2] = 1;
  else if (IS_ECAT_ENCODER_1_OVERFLOW(event))
    encoder_overflow[0] = 1;
  else if (IS_ECAT_ENCODER_2_OVERFLOW(event))
    encoder_overflow[1] = 1;
  else if (IS_ECAT_ENCODER_3_OVERFLOW(event))
    encoder_overflow[2] = 1;
  else if (IS_ECAT_ENCODER_1_UNDERFLOW(event))
```

```
    encoder_underflow[0] = 1;
  else if (IS_ECAT_ENCODER_2_UNDERFLOW(event))
    encoder_underflow[1] = 1;
  else if (IS_ECAT_ENCODER_3_UNDERFLOW(event))
    encoder_underflow[2] = 1;

  if (IS_ECAT_ENCODER_X_INDEX_RESET(event))
    encoder_xyz_index_reset[0] = 1;
  else if (IS_ECAT_ENCODER_Y_INDEX_RESET(event))
    encoder_xyz_index_reset[1] = 1;
  else if (IS_ECAT_ENCODER_Z_INDEX_RESET(event))
    encoder_xyz_index_reset[2] = 1;
  else if (IS_ECAT_ENCODER_X_OVERFLOW(event))
    encoder_xyz_overflow[0] = 1;
  else if (IS_ECAT_ENCODER_Y_OVERFLOW(event))
    encoder_xyz_overflow[1] = 1;
  else if (IS_ECAT_ENCODER_Z_OVERFLOW(event))
    encoder_xyz_overflow[2] = 1;
  else if (IS_ECAT_ENCODER_X_UNDERFLOW(event))
    encoder_xyz_underflow[0] = 1;
  else if (IS_ECAT_ENCODER_Y_UNDERFLOW(event))
    encoder_xyz_underflow[1] = 1;
  else if (IS_ECAT_ENCODER_Z_UNDERFLOW(event))
    encoder_xyz_underflow[2] = 1;
}

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  slave.attachInterrupt(Callback);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}
```

```
void loop() {
  if (emergency_stopped) {
    emergency_stopped = 0;
    Serial.println("ECAT_EMERGENCY_STOPPED");
  }
  if (x_limit_touched) {
    x_limit_touched = 0;
    Serial.println("ECAT_MACHINE_X_AXIS_LIMIT_TOUCHED");
  }
  if (y_limit_touched) {
    y_limit_touched = 0;
    Serial.println("ECAT_MACHINE_Y_AXIS_LIMIT_TOUCHED");
  }
  if (z_limit_touched) {
    z_limit_touched = 0;
    Serial.println("ECAT_MACHINE_Z_AXIS_LIMIT_TOUCHED");
  }
  if (encoder_index_reset[0]) {
    encoder_index_reset[0] = 0;
    Serial.println("IS_ECAT_ENCODER_1_INDEX_RESET");
  }
  if (encoder_index_reset[1]) {
    encoder_index_reset[1] = 0;
    Serial.println("IS_ECAT_ENCODER_2_INDEX_RESET");
  }
  if (encoder_index_reset[2]) {
    encoder_index_reset[2] = 0;
    Serial.println("IS_ECAT_ENCODER_3_INDEX_RESET");
  }
  if (encoder_overflow[0]) {
    encoder_overflow[0] = 0;
    Serial.println("IS_ECAT_ENCODER_1_OVERFLOW");
  }
  if (encoder_overflow[1]) {
    encoder_overflow[1] = 0;
    Serial.println("IS_ECAT_ENCODER_2_OVERFLOW");
  }
  if (encoder_overflow[2]) {
    encoder_overflow[2] = 0;
    Serial.println("IS_ECAT_ENCODER_3_OVERFLOW");
```

```
  }
  if (encoder_underflow[0]) {
    encoder_underflow[0] = 0;
    Serial.println("IS_ECAT_ENCODER_1_UNDERFLOW");
  }
  if (encoder_underflow[1]) {
    encoder_underflow[1] = 0;
    Serial.println("IS_ECAT_ENCODER_2_UNDERFLOW");
  }
  if (encoder_underflow[2]) {
    encoder_underflow[2] = 0;
    Serial.println("IS_ECAT_ENCODER_3_UNDERFLOW");
  }
  if (encoder_xyz_index_reset[0]) {
    encoder_xyz_index_reset[0] = 0;
    Serial.println("IS_ECAT_ENCODER_X_INDEX_RESET");
  }
  if (encoder_xyz_index_reset[1]) {
    encoder_xyz_index_reset[1] = 0;
    Serial.println("IS_ECAT_ENCODER_Y_INDEX_RESET");
  }
  if (encoder_xyz_index_reset[2]) {
    encoder_xyz_index_reset[2] = 0;
    Serial.println("IS_ECAT_ENCODER_Z_INDEX_RESET");
  }
  if (encoder_xyz_overflow[0]) {
    encoder_xyz_overflow[0] = 0;
    Serial.println("IS_ECAT_ENCODER_X_OVERFLOW");
  }
  if (encoder_xyz_overflow[1]) {
    encoder_xyz_overflow[1] = 0;
    Serial.println("IS_ECAT_ENCODER_Y_OVERFLOW");
  }
  if (encoder_xyz_overflow[2]) {
    encoder_xyz_overflow[2] = 0;
    Serial.println("IS_ECAT_ENCODER_Z_OVERFLOW");
  }
  if (encoder_xyz_underflow[0]) {
    encoder_xyz_underflow[0] = 0;
    Serial.println("IS_ECAT_ENCODER_X_UNDERFLOW");
```

```
  }
  if (encoder_xyz_underflow[1]) {
    encoder_xyz_underflow[1] = 0;
    Serial.println("IS_ECAT_ENCODER_Y_UNDERFLOW");
  }
  if (encoder_xyz_underflow[2]) {
    encoder_xyz_underflow[2] = 0;
    Serial.println("IS_ECAT_ENCODER_Z_UNDERFLOW");
  }
  // ...
}
```

CiA 402 Functions

CiA 402 functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- cia402GetServo()

cia402GetServo()

Description

Initialize the *EthercatDevice_CiA402* object for the specified motor with the given identifier on the EtherCAT slave device and return a pointer to the initialized *EthercatDevice_CiA402* object.

Syntax

```
EthercatDevice_CiA402 *cia402GetServo(int motor, EthercatDevice_CiA402 *servo = NULL);
```

Parameters

- `[in] int motor`
  The specified motor number on the EtherCAT slave device:
  1: Motor 1.
  2: Motor 2.
  3: Motor 3.
- `[in] EthercatDevice_CiA402 *servo`
  The pointer to an object of the EthercatDevice_CiA402 class declared by the user. If this parameter is NULL, initialize it with an object of the EthercatDevice_CiA402 class from the library and return a pointer to that object. Otherwise, initialize it with a pointer to an object of the EthercatDevice_CiA402 class provided by the user and return that pointer.

Return Value

Return an object pointer of the *EthercatDevice_CiA402* class.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function only works in CiA 402 on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.
WARNING: Prohibited from being called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;
EthercatDevice_CiA402 motor1;
EthercatDevice_CiA402 motor2;

EthercatDevice_CiA402 *pMotor[3];
```

```
void setup() {
  master.begin();
  slave.attach(0, master);
  pMotor[0] = slave.cia402GetServo(1, &motor1);
  pMotor[1] = slave.cia402GetServo(2, &motor2);
  pMotor[2] = slave.cia402GetServo(3);
}

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

Machine Functions

Machine functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- machineEnableSoftLimit()
- machineDisableSoftLimit()
- machineIsEmergencyStopped()
- machineSetEmergencyStop()
- machineClearEmergencyStop()
- machineIsLimitTouched()
- machineIsHomingAttained()
- machineIsServoOn()
- machineIsMoving()
- machineServoOn()
- machineServoOff()
- machineSetHomingSpeed()
- machineStartHoming()
- machineGcode()
- machineActualPosition()

machineEnableSoftLimit()

Description

Enable the software limit function for the machine on the EtherCAT slave device. For more information about software limits, please refer to configMachineSoftLimit().

Syntax

```
int machineEnableSoftLimit();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode(). This function does not work in the following case:

- The machine is servo-off.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  slave.configMachineSoftLimit(ECAT_MACHINE_X_AXIS, -50, 50);
  slave.configMachineSoftLimit(ECAT_MACHINE_Y_AXIS, -50, 50);
  slave.configMachineSoftLimit(ECAT_MACHINE_Z_AXIS, -50, 50);
  master.attachCyclicCallback(CyclicCallback);
```

```
  master.start();

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineEnableSoftLimit();

  slave.machineGcode("G1 X-100 Y-100 Z-100");
  slave.machineGcode("G1 X100 Y100 Z100");
  delay(10);

  while (slave.machineIsMoving()) {
    Serial.print("Acutal Position => ");
    Serial.print("X: ");
    Serial.print(slave.machineActualPosition(ECAT_MACHINE_X_AXIS), 2);
    Serial.print(", Y: ");
    Serial.print(slave.machineActualPosition(ECAT_MACHINE_Y_AXIS), 2);
    Serial.print(", Z: ");
    Serial.println(slave.machineActualPosition(ECAT_MACHINE_Z_AXIS),
2);
    delay(10);
    // ...
  }

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineDisableSoftLimit()

Description

Disable the software limit function for the machine on the EtherCAT slave device. For more information about software limits, please refer to configMachineSoftLimit().

Syntax

```
int machineDisableSoftLimit();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode(). This function does not work in the following case:

- The machine is servo-off.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  master.begin();
  slave.attach(0, master);
  slave.configMachineSoftLimit(ECAT_MACHINE_X_AXIS, -50, 50);
  slave.configMachineSoftLimit(ECAT_MACHINE_Y_AXIS, -50, 50);
  slave.configMachineSoftLimit(ECAT_MACHINE_Z_AXIS, -50, 50);
  master.attachCyclicCallback(CyclicCallback);
```

```
  master.start();

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineDisableSoftLimit();

  slave.machineGcode("G1 X-100 Y-100 Z-100");
  slave.machineGcode("G1 X100 Y100 Z100");
  delay(10);

  while (slave.machineIsMoving()) {
    Serial.print("Acutal Position => ");
    Serial.print("X: ");
    Serial.print(slave.machineActualPosition(ECAT_MACHINE_X_AXIS), 2);
    Serial.print(", Y: ");
    Serial.print(slave.machineActualPosition(ECAT_MACHINE_Y_AXIS), 2);
    Serial.print(", Z: ");
    Serial.println(slave.machineActualPosition(ECAT_MACHINE_Z_AXIS),
2);
    delay(10);
    // ...
  }

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineIsEmergencyStopped()

Description

Check if the machine on the EtherCAT slave device is emergency stopped. There are two primary conditions that can trigger the emergency stop:

- Hardware emergency stop
  This occurs when the hardware emergency stop switch is physically activated. This switch is typically a physical button located on the machine. It is designed to halt the machine's operation in case of an immediate safety hazard.
- User-initiated emergency stop
  This occurs when the user calls the machineSetEmergencyStop() function. This function is typically used to manually activate the emergency stop state, often through a software interface or control panel.

Syntax

```
int machineIsEmergencyStopped();
```

Parameters

None.

Return Value

Return whether the machine is emergency stopped.

- 1 means the Machine is in the Emergency Stop.
- 0 means none.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

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

```
  master.start();
}

void loop() {
  Serial.print("Emergency Stopped: ");
  Serial.println(slave.machineIsEmergencyStopped());
  delay(1000);
  //...
}
```

machineSetEmergencyStop()

Description

Initiate the emergency stop for the machine on the EtherCAT slave device. This function is typically used to manually activate the emergency stop state, often through a software interface or control panel.

Syntax

```
int machineSetEmergencyStop();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start();

  slave.machineSetEmergencyStop();
  delay(10);
  Serial.print("Emergency Stopped: ");
  Serial.println(slave.machineIsEmergencyStopped());
}

void loop() {}
```

machineClearEmergencyStop()

Description

Clear the emergency stop for the machine on the EtherCAT slave device. To clear the emergency stop state and resume normal machine operation, ensure the physical hardware emergency stop switch is deactivated or reset to its normal position before calling this function.

Since this function is a non-blocking function and the update() function needs to be called continuously to execute the state machine, it may take some time to complete, so the related status may take some time to respond, such as machineIsEmergencyStopped().

Syntax

```
int machineClearEmergencyStop();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.machineClearEmergencyStop();
}
void loop() {
  slave.update();
}
```

machineIsLimitTouched()

Description

Check if the limit switch of the specified machine axis on the EtherCAT slave device is touched.

Syntax

```
int machineIsLimitTouched(int machine_axis);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

Return Value

Return whether the limit switch of the specified machine axis is touched.

- 1 means the Machine Limit Switch is touched.
- 0 means none.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
```

```
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
}

void loop() {
  Serial.print("Limit Switch => ");
  Serial.print("X: ");
  Serial.print(slave.machineIsLimitTouched(ECAT_MACHINE_X_AXIS));
  Serial.print(", Y: ");
  Serial.print(slave.machineIsLimitTouched(ECAT_MACHINE_Y_AXIS));
  Serial.print(", Z: ");
  Serial.println(slave.machineIsLimitTouched(ECAT_MACHINE_Z_AXIS));
  delay(1000);
  // ...
}
```

machineIsHomingAttained()

Description

Check if the machine on the EtherCAT slave device is homing attained.

Syntax

```
int machineIsHomingAttained();
```

Parameters

None.

Return Value

Return whether the machine is homing attained.

- 1 means the Machine is homing attained.
- 0 means none.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineStartHoming();
```

```
  delay(10);
  while (!slave.machineIsHomingAttained());
  // ..

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineIsServoOn()

Description

Check if the machine on the EtherCAT slave device is servo-on.

Syntax

```
int machineIsServoOn();
```

Parameters

None.

Return Value

Return whether the machine is servo-on.

- 1 means the Machine is Servo-on.
- 0 means none.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);
  delay(1000);
  // ..
```

```
  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineIsMoving()

Description

Check if the machine on the EtherCAT slave device is moving.

Syntax

```
int machineIsMoving();
```

Parameters

None.

Return Value

Return whether the machine is moving.

- 1 means the Machine is moving.
- 0 means none.

If the return value is less than 0, it indicates an error code.

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineGcode("G1 X0 Y0 Z0");
```

```
  slave.machineGcode("G1 X100 Y50 Z25");
  delay(10);
  while (slave.machineIsMoving());
  // ...

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineServoOn()

Description

Turn on all motors of the machine on the EtherCAT slave device.
Since this function is a non-blocking function and the update() function needs to be called continuously to execute the state machine, it may take some time to complete, so the related status may take some time to respond, such as machineIsServoOn().

Syntax

```
int machineServoOn();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.
This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);
  delay(1000);
```

```
  // ..
  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineServoOff()

Description

Turn off all motors of the machine on the EtherCAT slave device.

Since this function is a non-blocking function and the update() function needs to be called continuously to execute the state machine, it may take some time to complete, so the related status may take some time to respond, such as machineIsServoOn().

Syntax

```
int machineServoOff();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

This function does not work in the following cases:

- The machine is homing.
- The machine is moving.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);
  delay(1000);
  // ..
  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineSetHomingSpeed()

Description

Set the homing speed of the specified machine axis on the EtherCAT slave device.

Syntax

```
int machineSetHomingSpeed(int machine_axis, double mm_per_min);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] double mm_per_min`
  The desired homing speed in millimeters per minute.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::start()` and before `EthercatMaster::stop()`. This function is blocking and cannot be called within the callback functions.

This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to `configDeviceMode()`.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
```

```
  slave.attach(0, master);
  slave.machineSetHomingSpeed(ECAT_MACHINE_X_AXIS, 3000);
  slave.machineSetHomingSpeed(ECAT_MACHINE_Y_AXIS, 3000);
  slave.machineSetHomingSpeed(ECAT_MACHINE_Z_AXIS, 3000);
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineStartHoming();
  delay(10);
  while (!slave.machineIsHomingAttained());
  // ..

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineStartHoming()

Description

Initiate the homing procedure for the machine on the EtherCAT slave device.
Since this function is a non-blocking function and the update() function needs to be called continuously to execute the state machine, it may take some time to complete, so the related status may take some time to respond, such as machineIsHomingAttained().

Syntax

```
int machineStartHoming();
```

Parameters

None.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.
This function only works in G-code Controller mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().
This function does not work in the following cases:

- The machine is emergency stopped.
- The machine is servo-off.
- The machine is homing.
- The machine is moving.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

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

```
  master.attachCyclicCallback(CyclicCallback);
  master.start();

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  slave.machineStartHoming();
  delay(10);
  while (!slave.machineIsHomingAttained());
  // ..

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);

}

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

machineGcode()

Description

Send a G-code command to the EtherCAT slave device.

Since this function is a non-blocking function and the update() function needs to be called continuously to execute the state machine, it may take some time to complete, so the related status may take some time to respond, such as machineIsMoving().

Syntax

```
int machineGcode(const char *fmt, ...);
```

Parameters

- `[in] const char *fmt`
  The string of the G-code command to be transmitted to the EtherCAT slave device. This is a pointer to a null-terminated string containing the format specification for the output. The format string follows the same format as the printf function in C, allowing for insertion of variables and formatting options.
- `[in] ...`
  This is a variable number of arguments that will be inserted into the formatted string according to the format specifiers in the fmt string.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in *G-code Controller* mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

This function does not work in the following cases:

- The machine is emergency stopped.
- The machine is servo-off.
- The machine is homing.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
```

```
}

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

  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);

  while (1) {
    slave.machineGcode("G1 X100 Y50 Z25");
    delay(1000);
    slave.machineGcode("G1 X0 Y0 Z0");
    delay(1000);
    // ...
  }

  slave.machineServoOff();
  while (slave.machineIsServoOn() == 1);
}

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

machineActualPosition()

Description

Get the current position of the specified machine axis on the EtherCAT slave device.

Syntax

```
double machineActualPosition(int machine_axis);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

Return Value

Return the current position of the specified machine axis in millimeters.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

This function only works in *G-code Controller* mode on the EtherCAT slave device. For more details, please refer to configDeviceMode().

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void CyclicCallback() {
  slave.update();
}

void setup() {
  master.begin();
  slave.attach(0, master);
  master.attachCyclicCallback(CyclicCallback);
  master.start();
```

```
  slave.machineServoOn();
  while (slave.machineIsServoOn() == 0);
}

void loop() {
  if (slave.machineIsMoving() == 0) {
    slave.machineGcode("G1 X0 Y0 Z0");
    slave.machineGcode("G1 X100 Y50 Z25");
    delay(10);
  }
  printf("Acutal Position => ");
  printf("X:%.2f, ", slave.machineActualPosition(ECAT_MACHINE_X_AXIS));
  printf("Y:%.2f, ", slave.machineActualPosition(ECAT_MACHINE_Y_AXIS));
  printf("Z:%.2f\n", slave.machineActualPosition(ECAT_MACHINE_Z_AXIS));
  delay(10);
  // ...
}
```

Encoder Functions

Encoder functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- encoderWrite()
- encoderDirectionRead()
- encoderRead()

encoderWrite()

Description

Write the counter of the specified encoder on the EtherCAT slave device.

Syntax

```
int encoderWrite(int encoder, int32_t value);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

- `[in] int32_t value`
  The value to be written.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  master.begin();
  slave.attach(0, master);
  slave.encoderWrite(ECAT_ENCODER_1, 100);
  slave.encoderWrite(ECAT_ENCODER_2, 100);
  slave.encoderWrite(ECAT_ENCODER_3, 100);
```

```
  master.start();
}

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

encoderDirectionRead()

Description

Get the current direction of the specified encoder on the EtherCAT slave device.

Syntax

```
int encoderDirectionRead(int encoder);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

Return Value

Return the current direction of the specified encoder. A return value of 0 indicates forward rotation, while a return value of 1 indicates reverse rotation. If the returned value is less than zero, it indicates an error code.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start();
}
```

```
void loop() {
  Serial.print("Encoder 1 Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_1));
  Serial.print("Encoder 2 Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_2));
  Serial.print("Encoder 3 Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_3));
  Serial.print("Encoder X Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_X));
  Serial.print("Encoder Y Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_Y));
  Serial.print("Encoder Z Direction: ");
  Serial.println(slave.encoderDirectionRead(ECAT_ENCODER_Z));
  delay(1000);
  // ...
}
```

encoderRead()

Description

Get the counter of the specified encoder on the EtherCAT slave device.

Syntax

```
int32_t encoderRead(int encoder);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

Return Value

Return the counter of the specified encoder with 32-bit signed integer.

Comment

This function must be called after a successful execution of EthercatMaster::start() and before EthercatMaster::stop(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  master.start();
}

void loop() {
```

```
  Serial.print("Encoder 1: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_1));
  Serial.print("Encoder 2: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_2));
  Serial.print("Encoder 3: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_3));
  Serial.print("Encoder X: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_X));
  Serial.print("Encoder Y: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_Y));
  Serial.print("Encoder Z: ");
  Serial.println(slave.encoderRead(ECAT_ENCODER_Z));
  delay(1000);
  // ...
}
```

Configuration Functions

Configuration functions for the EthercatDevice_DmpStepper_Generic class.

Functions:

- getDeviceMode()
- configDeviceMode()
- configCiA402MotorResolution()
- configMachineAxisMapping()
- configMachineDefaultFeedrate()
- configMachineDefaultHomingSpeed()
- configMachineHomingDirection()
- configMachineHomingPriority()
- configMachineMaxVelocity()
- configMachineMaxAcceleration()
- configMachineSoftLimit()
- configMachinePPU()
- configMachineAxisDirection()
- configEncoderMode()
- configEncoderDigitalFilter()
- configEncoderRange()
- configEncoderInputPolarity()
- configEncoderIndexReset()

getDeviceMode()

Description

Get the device mode for the EtherCAT slave device.

Syntax

```
int getDeviceMode();
```

Parameters

None.

Return Value

Return the device mode. If the return value is less than 0, it indicates an error code. For details about device modes, please refer to `configDeviceMode()`.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);

  Serial.print("Device Mode: ");
  Serial.println(slave.getDeviceMode());
}

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

configDeviceMode()

Description

Configure the device mode for the EtherCAT slave device. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program. After calling this function, the device mode of the EtherCAT slave device will not change immediately. The EtherCAT slave device must be repowered for the change to take effect.

This EtherCAT slave device supports the following two device modes:

- *CiA 402 Servo*
  This EtherCAT slave device is equipped with 3-axis CiA 402 stepper motors. Users can individually control each CiA 402 stepper motor.
- *G-code Controller*
  This EtherCAT slave device functions as a G-code controller, responsible for parsing and executing G-code instructions.

Syntax

```
int configDeviceMode(uint8_t mode);
```

Parameters

- `[in] uint8_t mode`

  The device modes supported by this EtherCAT slave device are as follows:

Definition	Value	Description
`ECAT_CIA402_SERVO_MODE`	0	*CiA 402 Servo* mode.
`ECAT_GCODE_CONTROLLER_MODE`	1	*G-code Controller* mode.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`.
This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

```
  slave.configDeviceMode(ECAT_GCODE_CONTROLLER_MODE);
}

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

configCiA402MotorResolution()

Description

Configure the motor resolution for each motor on the EtherCAT slave device in *CiA 402 Servo* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configCiA402MotorResolution(int motor, uint32_t steps_per_rev);
```

Parameters

- `[in] int motor`
  The specified motor number on the EtherCAT slave device:
  1: Motor 1.
  2: Motor 2.
  3: Motor 3.
- `[in] uint32_t steps_per_rev`
  The motor resolution to be configured, specified in terms of steps per revolution.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configCiA402MotorResolution(1, 3200);
  slave.configCiA402MotorResolution(2, 3200);
  slave.configCiA402MotorResolution(3, 3200);
  // ...
}
```

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

configMachineAxisMapping()

Description

Configure the mapping between mechanical axes and motors for the EtherCAT slave device in *G-code Controller* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program. After calling this function, the mapping will not change immediately. The EtherCAT slave device must be repowered for the change to take effect.

Syntax

```
int configMachineAxisMapping(uint8_t mapping);
```

Parameters

- `[in] uint8_t mapping`
  The mapping of the mechanical axis to be configured. The EtherCAT slave device offers the following mapping options:

Code	X-axis	Y-axis	Z-axis
0	Motor 1	Motor 2	Motor 3
1	Motor 1	Motor 3	Motor 2
2	Motor 2	Motor 1	Motor 3
3	Motor 2	Motor 3	Motor 1
4	Motor 3	Motor 1	Motor 2
5	Motor 3	Motor 2	Motor 1

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachineAxisMapping(0);
```

```
}

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

configMachineDefaultFeedrate()

Description

Configure the default feed rate for the EtherCAT slave device in *G-code Controller* mode. If the user has not yet specified a feed rate in a G-code movement command sent through machineGcode(), this default feed rate will be used. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineDefaultFeedrate(double mm_per_min);
```

Parameters

- `[in] double mm_per_min`
  The default feed rate to be configured is in millimeters per minute.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachineDefaultFeedrate(3000);
  // ...
}

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

configMachineDefaultHomingSpeed()

Description

Configure the default homing speed of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. This default homing speed will be used for homing operations if the user has not yet specified a homing speed for each axis using the machineSetHomingSpeed() function. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineDefaultHomingSpeed(int machine_axis, double mm_per_min);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] double mm_per_min`
  The default homing speed to be configured is in millimeters per minute.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

```
  slave.configMachineDefaultHomingSpeed(ECAT_MACHINE_X_AXIS, 1000);
  slave.configMachineDefaultHomingSpeed(ECAT_MACHINE_Y_AXIS, 1000);
  slave.configMachineDefaultHomingSpeed(ECAT_MACHINE_Z_AXIS, 1000);
  // ...
}

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

configMachineHomingDirection()

Description

Configure the homing direction of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineHomingDirection(int machine_axis, bool positive);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] bool positive`
  A Boolean value used to configure the homing direction.
  - true: Search for the home switch in the ***positive*** direction.
  - false: Search for the home switch in the ***negative*** direction.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

```
  slave.configMachineHomingDirection(ECAT_MACHINE_X_AXIS, false);
  slave.configMachineHomingDirection(ECAT_MACHINE_Y_AXIS, false);
  slave.configMachineHomingDirection(ECAT_MACHINE_Z_AXIS, false);
  // ...
}

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

configMachineHomingPriority()

Description

Configure the homing priority of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineHomingPriority(int machine_axis, uint8_t priority);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] uint8_t priority`
  The homing priority to be configured. The value range is 0 to 255, and the lower the value, the higher the priority. If the values are equal, the priority order is not guaranteed.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

```
  slave.configMachineHomingPriority(ECAT_MACHINE_X_AXIS, 1);
  slave.configMachineHomingPriority(ECAT_MACHINE_Y_AXIS, 2);
  slave.configMachineHomingPriority(ECAT_MACHINE_Z_AXIS, 3);
  // ...
}

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

configMachineMaxVelocity()

Description

Configure the maximum velocity of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. The purpose of this parameter is primarily to limit maximum speed of the motor to prevent exceeding the motor's specifications. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineMaxVelocity(int machine_axis, double mm_per_sec);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] double mm_per_sec`
  The maximum velocity to be configured is in millimeters per second.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

```
  slave.configMachineMaxVelocity(ECAT_MACHINE_X_AXIS, 300);
  slave.configMachineMaxVelocity(ECAT_MACHINE_Y_AXIS, 300);
  slave.configMachineMaxVelocity(ECAT_MACHINE_Z_AXIS, 300);
  // ...
}

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

configMachineMaxAcceleration()

Description

Configure the maximum acceleration for the EtherCAT slave device in *G-code Controller* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineMaxAcceleration(double mm_per_sec2);
```

Parameters

- `[in] double mm_per_sec2`
  The maximum acceleration to be configured is in millimeters per second squared.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the Cyclic Callback.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachineMaxAcceleration(100000);
  // ...
}

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

configMachineSoftLimit()

Description

Configure the software limit of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. These parameters are written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure these parameters each time before running the program.

Upon receiving a movement command from the user calling machineGcode(), the EtherCAT slave device internally calculates the target position for each axis.

- If the target position *P* for an axis is greater than the maximum position value P_{max} for that axis, then *P* will be adjusted to P_{max}.
- If the target position *P* for an axis is less than the minimum position value P_{min} for that axis, then *P* will be adjusted to P_{min}.

Syntax

```
int configMachineSoftLimit(int machine_axis, double min, double max);
```

Parameters

- `[in] int machine_axis`

  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] double min`

  The minimum position value P_{min} of the software limit to be configured, in millimeters.

- `[in] double max`

  The maximum position value P_{max} of the software limit to be configured, in millimeters.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"
```

```
EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachineSoftLimit(ECAT_MACHINE_X_AXIS, 0, 100);
  slave.configMachineSoftLimit(ECAT_MACHINE_Y_AXIS, 0, 100);
  slave.configMachineSoftLimit(ECAT_MACHINE_Z_AXIS, 0, 100);
}

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

configMachinePPU()

Description

Configure the PPU (Pulse Per Unit) of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. This parameter defines the relationship between the movement of the machine and the motors. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachinePPU(int machine_axis, double pulses_per_mm);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] double pulses_per_mm`
  The PPU to be configured is in pulses per millimeter.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachinePPU(ECAT_MACHINE_X_AXIS, 100);
```

```
  slave.configMachinePPU(ECAT_MACHINE_Y_AXIS, 100);
  slave.configMachinePPU(ECAT_MACHINE_Z_AXIS, 100);
  // ...
}

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

configMachineAxisDirection()

Description

Configure the motor direction of the specified machine axis on the EtherCAT slave device in *G-code Controller* mode. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configMachineAxisDirection(int machine_axis, int invert);
```

Parameters

- `[in] int machine_axis`
  The specified machine axis number on the EtherCAT slave device:

Definition	Value	Description
ECAT_MACHINE_X_AXIS	0	X-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Y_AXIS	1	Y-axis, which mapping is determined by the configMachineAxisMapping().
ECAT_MACHINE_Z_AXIS	2	Z-axis, which mapping is determined by the configMachineAxisMapping().

- `[in] int invert`
  The motor direction to be configured. A value of 0 indicates normal direction, while any other value indicates inverted direction.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configMachineAxisDirection(ECAT_MACHINE_X_AXIS, 0);
```

```
  slave.configMachineAxisDirection(ECAT_MACHINE_Y_AXIS, 0);
  slave.configMachineAxisDirection(ECAT_MACHINE_Z_AXIS, 0);
  // ...
}

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

configEncoderMode()

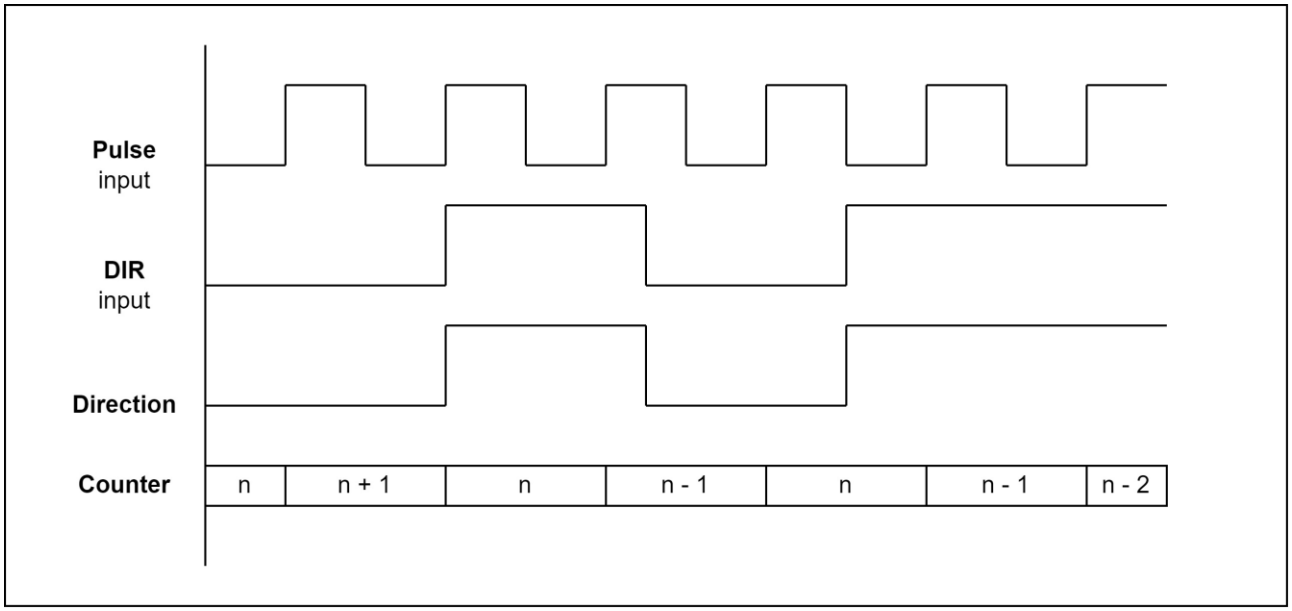
Description

Configure the encoder mode of the specified encoder on the EtherCAT slave device. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

This EtherCAT slave device supports a total of six encoder modes, as follows:

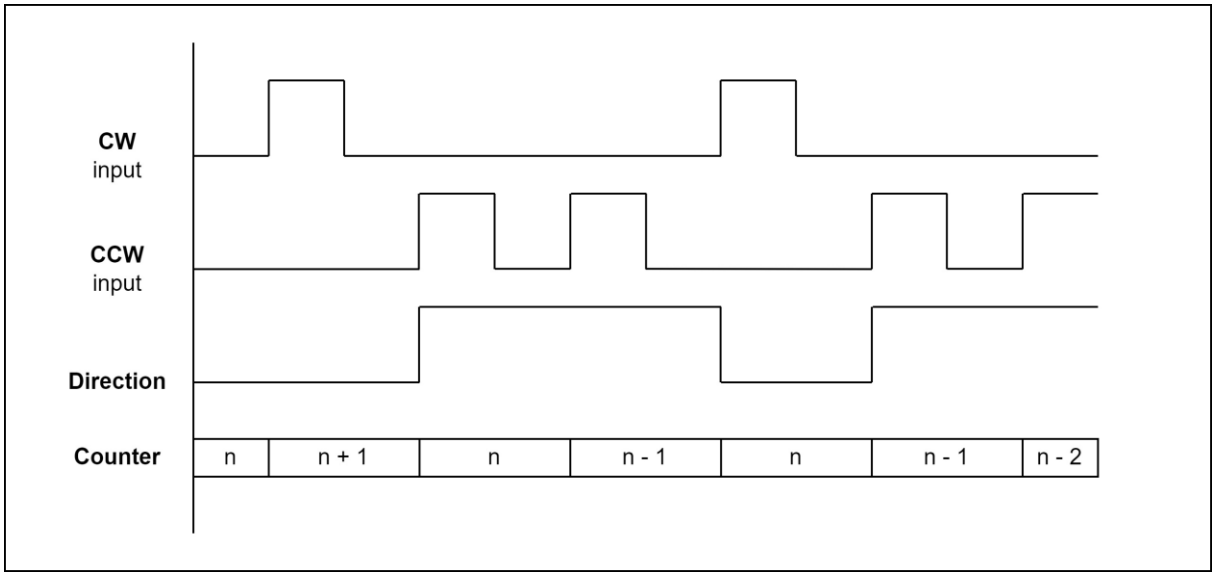
- Pulse/DIR (*1-Pulse* mode)

This mode is also known as *1-Pulse* mode and has two input pins for accepting signals from the encoder: the *Pulse* input and the *DIR* input. The *Pulse* input is used to count the number of pulses from the encoder, and the *DIR* input is used to indicate the current counting direction of the encoder counter. In this mode, the counter only counts on the rising edge of the *Pulse* input.



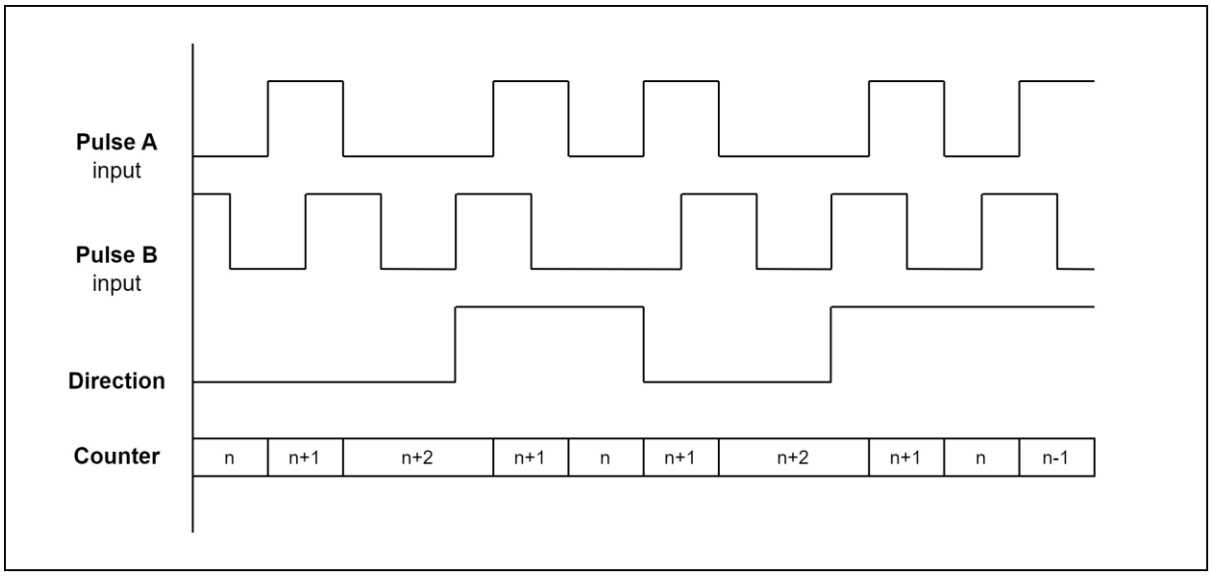
- CW/CCW (*2-Pulse* mode)

This mode is also known as *2-Pulse* mode. It has two input pins for accepting signals from the encoder: the *CW* input and the *CCW* input. The pulses from the *CW* input represent the number of pulses counted in the forward direction by the encoder counter, while the pulses from the *CCW* input represent the number of pulses counted in the reverse direction by the encoder counter. In this mode, the counter only counts on the rising edge of the *CW* and *CCW* inputs.



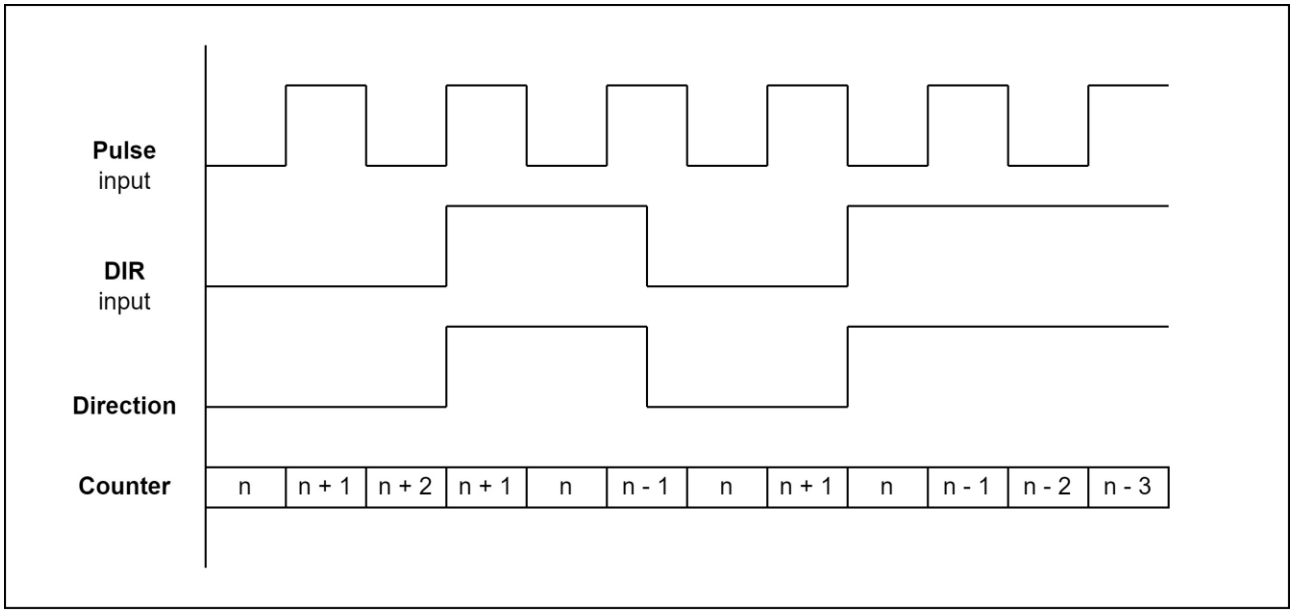
- Pulse A/B (*QEI* mode)

This mode is also known as *QEI (Quadrature Encoder Interface)* mode. It has two input pins for accepting signals from the encoder: the *Pulse A* input and the *Pulse B* input. The *Pulse A* and *Pulse B* inputs have a unique relationship. If *Pulse A* leads *Pulse B*, the counting direction is deemed forward. If *Pulse A* lags *Pulse B*, the counting direction is deemed reverse. In this mode, the counter only counts on the rising and falling edges of the *Pulse A* input.



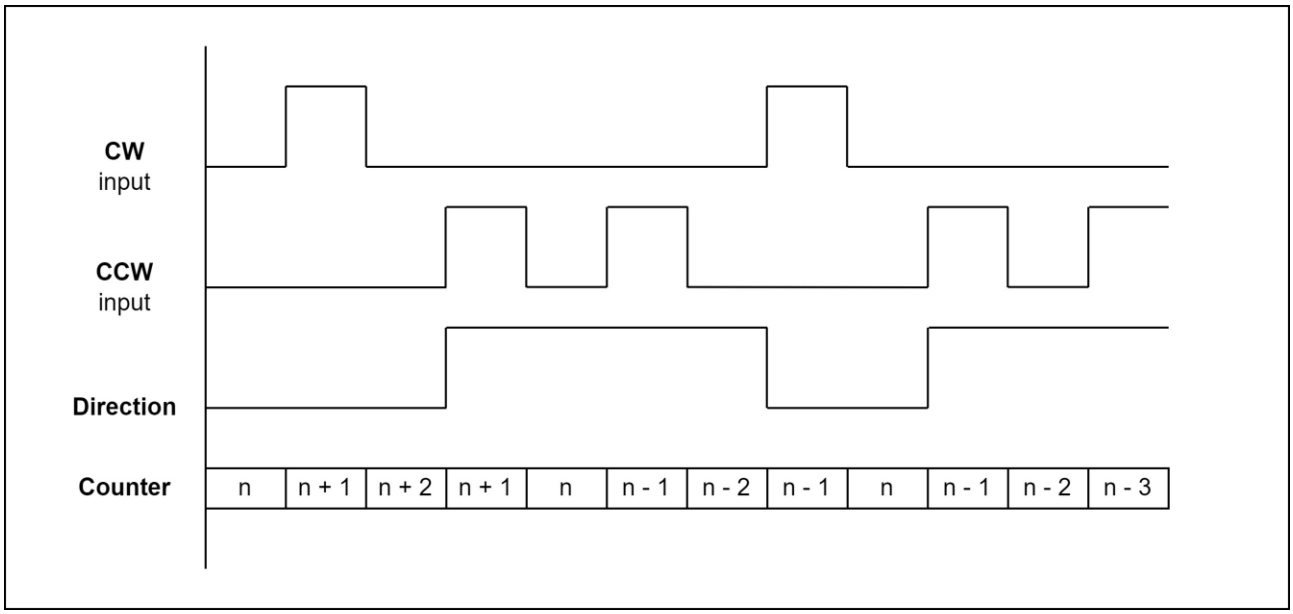
- Pulse/DIR x2

This mode is similar to **Pulse/DIR** mode, with the only difference being that the counter in this mode counts on both the rising and falling edges of the *Pulse* input.



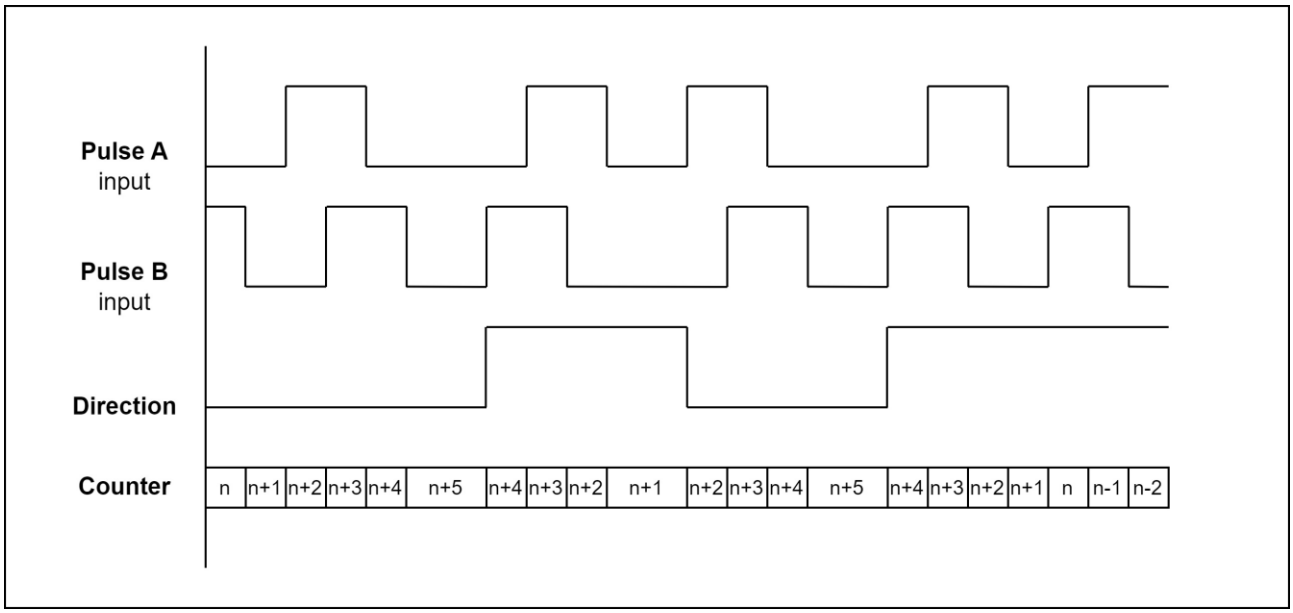
- CW/CCW x2

This mode is similar to **CW/CCW** mode, with the only difference being that the counter in this mode counts on both the rising and falling edges of the *CW* and *CCW* inputs.



- Pulse A/B x2

This mode is similar to **Pulse A/B** mode, with the only difference being that the counter in this mode counts on both the rising and falling edges of the *Pulse A* and *Pulse B* inputs.



Syntax

```
int configEncoderMode(int encoder, uint8_t mode);
```

Parameters

- `[in] int encoder`

  The specified encoder number:

Definition	Value	Description
`ECAT_ENCODER_1`	0x01	Encoder 1 on the EtherCAT slave device.
`ECAT_ENCODER_2`	0x02	Encoder 2 on the EtherCAT slave device.
`ECAT_ENCODER_3`	0x03	Encoder 3 on the EtherCAT slave device.
`ECAT_ENCODER_X`	0x11	Encoder X, which mapping is determined by the `configMachineAxisMapping()`.
`ECAT_ENCODER_Y`	0x12	Encoder Y, which mapping is determined by the `configMachineAxisMapping()`.
`ECAT_ENCODER_Z`	0x13	Encoder Z, which mapping is determined by the `configMachineAxisMapping()`.

- `[in] uint8_t mode`

  The encoder mode to be configured. Users can select from six encoder modes, as detailed in the table below. For input values not listed in the table, the default configuration will be `ECAT_ENCODER_MODE_AB_PHASE_x2`.

Definition	Value	Description
`ECAT_ENCODER_MODE_STEP_DIR`	0	Please see Pulse/DIR mode.
`ECAT_ENCODER_MODE_CWCCW`	1	Please see CW/CCW mode.
`ECAT_ENCODER_MODE_AB_PHASE`	2	Please see Pulse A/B mode.
`ECAT_ENCODER_MODE_STEP_DIR_x2`	5	Please see Pulse/DIR x2 mode.

ECAT_ENCODER_MODE_CWCCW_x2	6	Please see CW/CCW x2 mode.
ECAT_ENCODER_MODE_AB_PHASE_x2	7	Please see Pulse A/B x2 mode.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the callback functions.

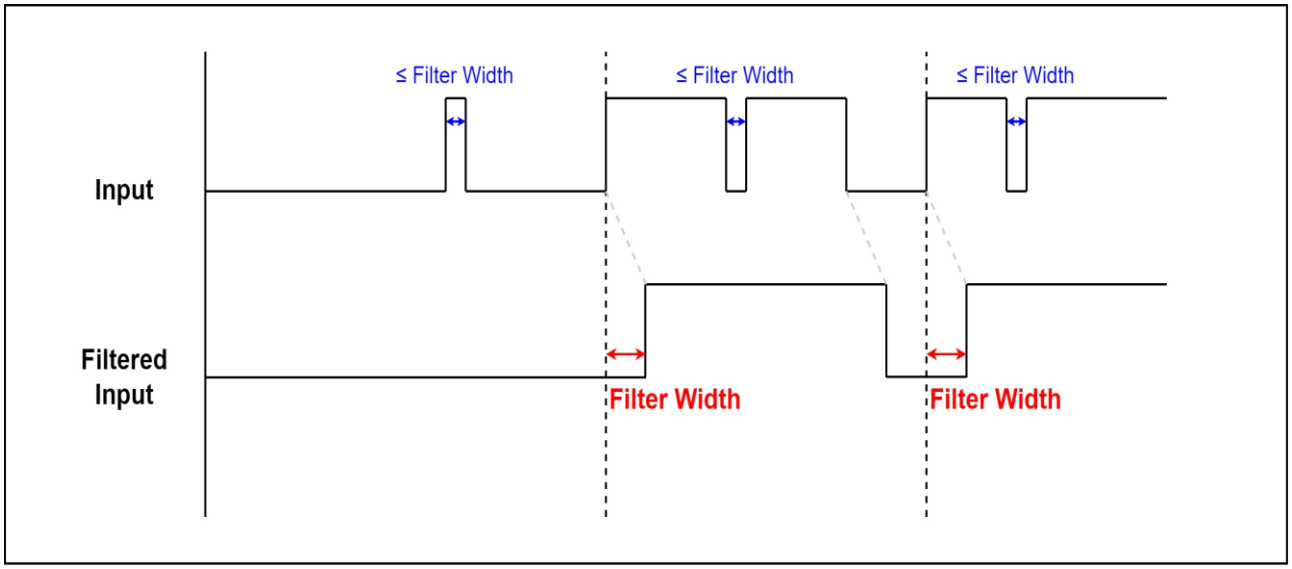
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configEncoderMode(ECAT_ENCODER_1,
ECAT_ENCODER_MODE_AB_PHASE_x2);
  slave.configEncoderMode(ECAT_ENCODER_2,
ECAT_ENCODER_MODE_AB_PHASE_x2);
  slave.configEncoderMode(ECAT_ENCODER_3,
ECAT_ENCODER_MODE_AB_PHASE_x2);
  // ...
}

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

configEncoderDigitalFilter()

Description

Configure the digital filter of the specified encoder on the EtherCAT slave device. As shown in the figure, pulses with a width smaller than the filter bandwidth are filtered out, and pulses that are not filtered out are delayed by the time of the filter bandwidth.
This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.



Syntax

```
int configEncoderDigitalFilter(int encoder, uint32_t width);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

- `[in] uint32_t width`
  The digital filter of the encoder to be configured is in units of 10 nanoseconds. If this value is 100, it means the filter width is 1000 nanoseconds.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the callback functions.

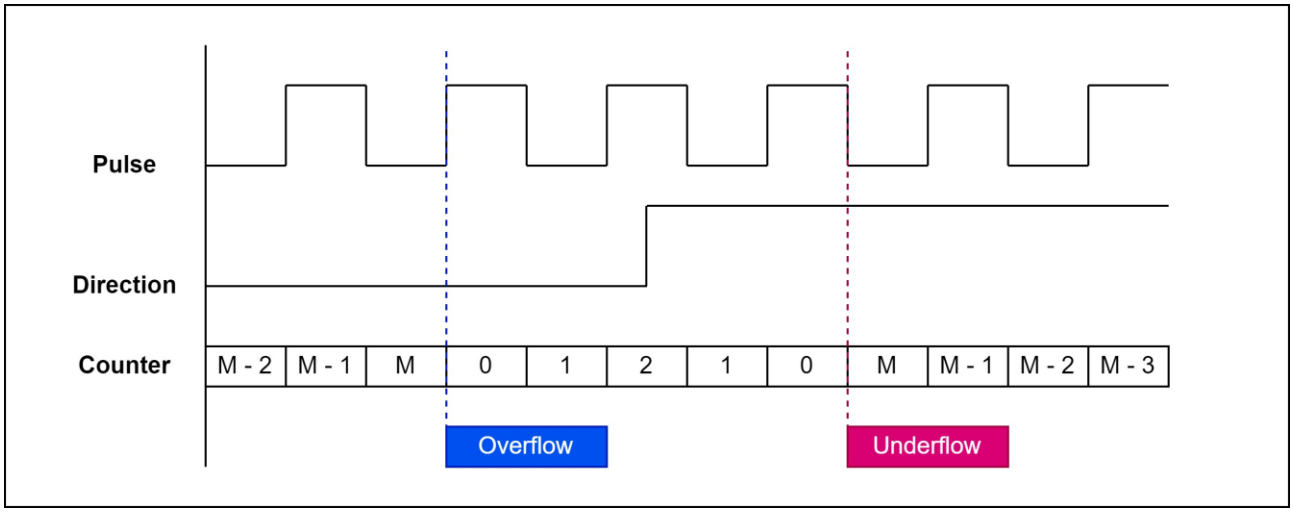
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configEncoderDigitalFilter(ECAT_ENCODER_1, 100);
  slave.configEncoderDigitalFilter(ECAT_ENCODER_2, 100);
  slave.configEncoderDigitalFilter(ECAT_ENCODER_3, 100);
  // ...
}

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

configEncoderRange()

Description

Configure the maximum value of the counter for the specified encoder on the EtherCAT slave device. As shown in the figure, upon reaching its maximum value (*M*), the counter overflows, resets to zero, and triggers an *Overflow* event. Conversely, when the counter decrements to zero, it underflows, resets to the maximum value, and triggers an *Underflow* event. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.



Syntax

```
int configEncoderRange(int encoder, uint32_t value);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

- `[in] uint32_t value`
  The maximum value of the encoder counter to be configured.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configEncoderRange(ECAT_ENCODER_1, 8000);
  slave.configEncoderRange(ECAT_ENCODER_2, 8000);
  slave.configEncoderRange(ECAT_ENCODER_3, 8000);
  // ...
}

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

configEncoderInputPolarity()

Description

Configure the polarity of the input pins for the specified encoder on the EtherCAT slave device. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.

Syntax

```
int configEncoderInputPolarity(int encoder, bool pin_a, bool pin_b, bool pin_z);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

- `[in] bool pin_a`
  The polarity of input pin A for the specified encoder to be configured.
  - true: High voltage represents logic 1, while low voltage represents logic 0.
  - false: High voltage represents logic 0, while low voltage represents logic 1.
- `[in] bool pin_b`
  The polarity of input pin B for the specified encoder to be configured. The explanation for the input value is the same as pin_a.
- `[in] bool pin_z`
  The polarity of input pin Z for the specified encoder to be configured. The explanation for the input value is the same as pin_a.

Return Value

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

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is blocking and cannot be called within the callback functions.

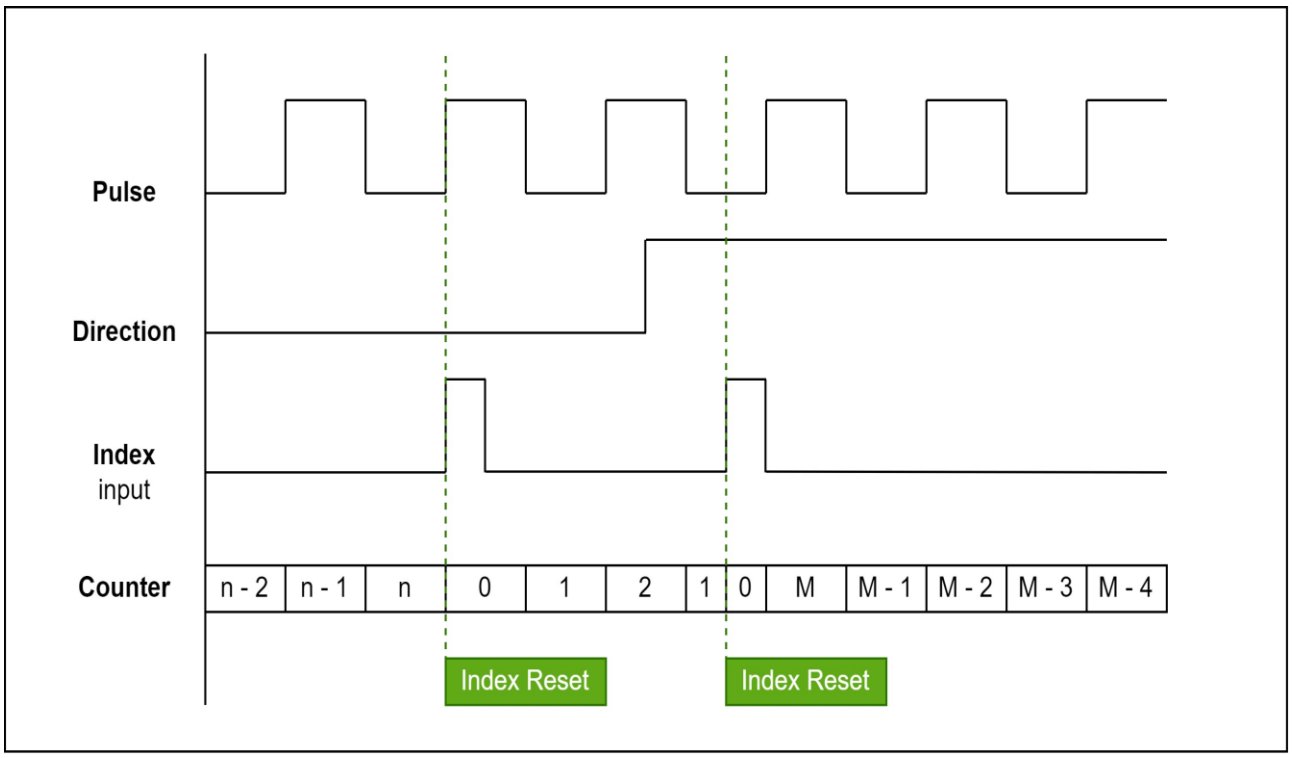
Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configEncoderInputPolarity(ECAT_ENCODER_1, true, true, true);
  slave.configEncoderInputPolarity(ECAT_ENCODER_2, true, true, true);
  slave.configEncoderInputPolarity(ECAT_ENCODER_3, true, true, true);
  // ...
}

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

configEncoderIndexReset()

Description

Enable or disable the index signal reset counter function for the specified encoder on the EtherCAT slave device. As shown in the figure, upon enabling this function, the counter will be reset to zero and the *Index Reset* event will be triggered when the index signal (signal Z) is detected. This parameter is written to the EEPROM of the EtherCAT slave device and loaded during startup, so users do not need to configure this parameter each time before running the program.



Syntax

```
int configEncoderIndexReset(int encoder, bool enable);
```

Parameters

- `[in] int encoder`
  The specified encoder number:

Definition	Value	Description
ECAT_ENCODER_1	0x01	Encoder 1 on the EtherCAT slave device.
ECAT_ENCODER_2	0x02	Encoder 2 on the EtherCAT slave device.
ECAT_ENCODER_3	0x03	Encoder 3 on the EtherCAT slave device.
ECAT_ENCODER_X	0x11	Encoder X, which mapping is determined by the configMachineAxisMapping().
ECAT_ENCODER_Y	0x12	Encoder Y, which mapping is determined by the configMachineAxisMapping().

ECAT_ENCODER_Z	0x13	Encoder Z, which mapping is determined by the configMachineAxisMapping().

- [in] bool enable
  A boolean value that specifies whether to enable or disable the index signal reset function for the specified encoder.
  - true: The index signal reset function will be enabled.
  - false: The index signal reset function will be disabled.

Return Value

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

Comment

This function must be called after a successful execution of EthercatMaster::begin(). This function is blocking and cannot be called within the callback functions.

Example

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR11MP3S slave;

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

  slave.configEncoderIndexReset(ECAT_ENCODER_1, false);
  slave.configEncoderIndexReset(ECAT_ENCODER_2, false);
  slave.configEncoderIndexReset(ECAT_ENCODER_3, false);
  // ...
}

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

Ch. 3

Examples

3.1 Slave Information

```
#include "Ethercat.h"

EthercatMaster master;

void setup(void) {
    uint16_t slavecount, i;

    Serial.begin(115200);
    while (!Serial);

    master.begin();

    slavecount = master.getSlaveCount();
    for (i = 0; i < slavecount; i++) {
        Serial.print("Slave");
        Serial.print(i);
        Serial.print(" => Vendor ID: 0x");
        Serial.print(master.getVendorID(i), HEX);
        Serial.print(", Product Code: 0x");
        Serial.println(master.getProductCode(i), HEX);
    }
}

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

3.2 PDO Read/Write

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

void loop()
{
    /* Write value of byte offset 0 of slave's output process data. */
    slave.pdoWrite8(0, 0xFF);

    /* Read value of byte offset 0 of slave's input process data. */
    Serial.println(slave.pdoRead8(0), HEX);

    delay(100);
}
```

3.3 SDO Upload/Download

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

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

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

    /* Write 0x08 to the object with index 0x6060. */
    slave.sdoDownload8(0x6060, 0x00, 0x08);
    /* Read value from the object with index 0x6061. */
    Serial.prinln(slave.sdoUpload8(0x6061, 0x00));

    master.start();
}

void loop() {

}
```

3.4 Object Description Information

```
#include <Ethercat.h>

EthercatMaster master;
EthercatDevice_Generic slave;

uint16_t DataType;
uint8_t MaxSubindex, ObjectCode;
char ObjectName[64];

void setup() {
  Serial.begin(115200);
  master.begin();
  slave.attach(0, master);
  slave.getObjectDescription(0x1C12, DataType, MaxSubindex, ObjectCode,
ObjectName, sizeof(ObjectName));

  Serial.print("Data Type      : ");
  Serial.print(DataType, HEX);
  Serial.println("h");

  Serial.print("Object Code    : ");
  Serial.print(ObjectCode, HEX);
  Serial.println("h");

  Serial.print("Max Subindex   : ");
  Serial.println(MaxSubindex);

  Serial.print("Object Name    : ");
  Serial.println(ObjectName);
}

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

3.5 Cyclic Callback

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_Generic slave;

void CyclicCallback() {
    // put your cyclic Callback function here.
}
void setup() {
    master.begin();
    slave.attach(0, master);
    slave.setDc(1000000); // 1000000 ns = 1 ms
    master.attachCyclicCallback(CyclicCallback);
    master.start(1000000, ECAT_SYNC);  // 1000000 ns = 1 ms
}
void loop() {

}
```

3.6 QEC-DIO Series

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_QECR00DF0H slave;

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

  master.begin();
  slave.attach(0, master);
  master.start();
}
void loop() {
  Serial.println(slave.digitalReadAll());
  delay(4000);
}
```

3.7 QEC-HID Series

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster EcatMaster; // Create an EtherCAT Master Object
EthercatDevice_QECR11HU9S Slave1; // Create an EtherCAT Slave Object
for QEC R11HU9S

int incomingByte = 0; // Variable for incoming serial data
char read_ch; // Variable for read serial data (char)

void setup() {
    Serial.begin(115200); // Initialize serial communication at 115200
baud rate

    // Initialize the EtherCAT Master. If successful, all slaves enter
PRE OPERATIONAL state
    EcatMaster.begin();

    // Attach QECR11HU9S slave device to the EtherCAT Master at
position 0
    Slave1.attach(0, EcatMaster);

    // Start the EtherCAT Master. If successful, all slaves enter
OPERATIONAL state
    // FreenRun Mode, and the parameter 1000000 sets the cycle time in
nanoseconds
    EcatMaster.start(1000000, ECAT_FREERUN);

    // Configure UART settings for two COM ports of the slave device
    Slave1.uartSetBaud(COM1, 115200); // Set baud rate for COM1
    Slave1.uartSetFormat(COM1, SERIAL_8N1); // Set data format for COM1
    Slave1.uartSetBaud(COM2, 115200); // Set baud rate for COM2
    Slave1.uartSetFormat(COM2, SERIAL_8N1); // Set data format for COM2
}

void loop() {
    // send data only when you receive data:
```

```
    if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        // Send the byte via UART COM1 of the slave device
        Slave1.uartWrite(COM1, incomingByte);

        while (Slave1.uartQueryRxQueue(COM2) <1) Slave1.update();//
Because the function is non-blocking, so we need to call update(); by
ourselves

        // Read the received character from slave's UART COM2
        if((read_ch = (char)Slave1.uartRead(COM2)) > 0) {
            // Print the received data to the serial monitor
            Serial.print("COM2 receive: ");
            Serial.println(read_ch);
        }
    }
}
```

3.8 QEC-Stepper Series

```
#include "Ethercat.h" // Include the EtherCAT Library

EthercatMaster EcatMaster; // Create an EtherCAT Master Object
EthercatDevice_QECR11MP3S Slave1; // Create an EtherCAT Slave Objects
for QECR11MP3S

// Callback function for cyclic updates
void myCallback() {
  Slave1.update(); // Update the Ethercat slave
}

void setup() {
  Serial.begin(115200);
  // Initialize the EtherCAT Master. If successful, all slaves enter
PRE OPERATIONAL state
  EcatMaster.begin();

  // Attach the QECR11MP3S to the EtherCAT Master at position 0
  Slave1.attach(0, EcatMaster);

  // Set a cyclic callback for the Ethercat Master
  EcatMaster.attachCyclicCallback(myCallback);

  // Start the EtherCAT Master. If successful, all slaves enter
OPERATIONAL state
  // Sync Mode, and the parameter 1000000 sets the cycle time in
nanoseconds
  EcatMaster.start(1000000, ECAT_SYNC);

  // Enable motor (G-code) to prepare the device for movement commands
  Slave1.machineServoOn();
}

void loop() {
  // Move to position X=100 at speed F=1000. This command moves the
machine axis to position 100
```

```
  // at a feed rate of 1000. The operation will take some time to
complete.
  Slave1.machineGcode("G1 X100 F1000");
  delay(6000); // Wait for the command to be processed

  // Move back to position X=0 at speed F=10000. This command returns
the machine axis to the starting position
  // at a faster feed rate of 10000. This operation is quicker due to
the higher feed rate.
  Slave1.machineGcode("G1 X0 F10000");
  delay(1000); // Wait for the command to be processed
}
```

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_SII_READ_FAIL	-2100
ECAT_ERR_DEVICE_SII_WRITE_FAIL	-2101

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_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

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 master 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 slaves 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 master 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 slaves 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 master 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 slaves are functioning properly.

-210: ECAT_ERR_MASTER_NO_DEVICE

Description
EtherCAT network has no slaves.
Corrective Actions
Please connect at least one EtherCAT slave.

-300: ECAT_ERR_MASTER_ACYCLIC_INIT_FAIL

Description
Ayclic 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 slave 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 master 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 master command interface failed.
Corrective Actions
Please contact the manufacturer.

-1007: ECAT_ERR_MASTER_CIA402_INIT_FAIL

Description
Initialization of EtherCAT master CiA 402 framework failed.
Corrective Actions
Please contact the manufacturer.

-1008: ECAT_ERR_MASTER_SETUP_PDO_FAIL

Description
Configuration of PDO mapping for the slave using SDO Download failed.
Corrective Actions
Please ensure that all slaves are functioning properly. If the issue is persist, 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 slaves are powered on, or verify the presence of the network controller.

-1010: ECAT_ERR_MASTER_START_MASTER_FAIL

Description
Failed to start EtherCAT master.
Corrective Actions
Please ensure that the configuration of PDO mapping for all slaves 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 slaves are functioning correctly, and verify that the configuration of PDO mapping is correct.

-1013: ECAT_ERR_MASTER_CONFIG_DEVICE_FAIL

Description
EtherCAT slave's 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 slaves.
Corrective Actions
Please contact the manufacturer.

-1016: ECAT_ERR_MASTER_WAIT_MASTER_SYNC_TIMEOUT

Description
Synchronization timeout between master and slaves.
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 slave information in the ENI file does not match the scanned EtherCAT network slaves.
Corrective Actions
Please adjust the order of the slaves on the EtherCAT network or configure the slave identifications.

-1100: ECAT_ERR_MASTER_STOPPED

Description
EtherCAT master has not been started.
Corrective Actions
Please call `EthercatMaster::start()` or avoid calling this function.

-1101: ECAT_ERR_MASTER_STARTED

Description
EtherCAT master has already been started.
Corrective Actions
Please call `EthercatMaster::stop()` or avoid calling this function.

-1102: ECAT_ERR_MASTER_NOT_IN_PREOP

Description
EtherCAT master is not in PRE-OP state.
Corrective Actions
Please ensure that all slaves are in PRE-OP state before calling this function.

-1103: ECAT_ERR_MASTER_NOT_IN_SAFEOP

Description
EtherCAT master is not in SAFE-OP state.
Corrective Actions
Please ensure that all slaves are in SAFE-OP state before calling this function.

-1104: ECAT_ERR_MASTER_NOT_IN_OP

Description
EtherCAT master is not in OP state.
Corrective Actions
Please ensure that all slaves are in OP state before calling this function.

-1200: ECAT_ERR_MASTER_II_TRANSITION_FAIL

Description
Switching all slaves to INIT state during EtherCAT master initialization failed.
Corrective Actions
Please power cycle all slaves and then run the EtherCAT master 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 slaves 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 slaves. 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 slave does not exist.
Corrective Actions
Please do not access the specified slave.

-2001: ECAT_ERR_DEVICE_NOT_ATTACH

Description
The object of such slave has not been initialized.
Corrective Actions
Please call attach() to initialize the slave object.

-2002: ECAT_ERR_DEVICE_NO_MAILBOX

Description
The slave does not support Mailbox.
Corrective Actions
Please do not call Mailbox-related functions for the slave.

-2003: ECAT_ERR_DEVICE_NO_DC

Description
The slave does not support DC synchronization.
Corrective Actions
Please do not call DC-related functions for the slave.

-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 slave object.
Corrective Actions
Please contact the manufacturer.

-2006: ECAT_ERR_DEVICE_VENDOR_ID_MISMATCH

Description
The Vendor ID of the slave does not match that of the slave object.
Corrective Actions
Please use the correct slave object.

-2007: ECAT_ERR_DEVICE_PRODUCT_CODE_MISMATCH

Description
The Product Code of the slave does not match that of the slave object.
Corrective Actions
Please use the correct slave object.

-2008: ECAT_ERR_DEVICE_NO_SUCH_FUNCTION

Description
The slave 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 slave has not been initialized.
Corrective Actions
Please initialize that function.

-2010: ECAT_ERR_DEVICE_BUSY

Description
The slave is busy.
Corrective Actions
Please try again later.

-2011: ECAT_ERR_DEVICE_TIMEOUT

Description
The slave has encountered a timeout.
Corrective Actions
Please try again later or confirm that the slave is functioning properly.

-2012: ECAT_ERR_DEVICE_NO_DATA

Description
The slave has no available data.
Corrective Actions
Please try again later.

-2100: ECAT_ERR_DEVICE_SII_READ_FAIL

Description
Failed to read the SII EEPROM of the slave.
Corrective Actions
Please ensure that this function is called when the EtherCAT state of the slave is INIT or PRE-OP.

-2101: ECAT_ERR_DEVICE_SII_WRITE_FAIL

Description
Failed to write the SII EEPROM of the slave.
Corrective Actions
Please ensure that this function is called when the EtherCAT state of the slave is INIT or PRE-OP.

-2200: ECAT_ERR_DEVICE_PDO_NOT_EXIST

Description
The slave 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 slave.
Corrective Actions
Please access the correct range of process data for the slave.

-2300: ECAT_ERR_DEVICE_FOE_NOT_SUPPORT

Description
The slave does not support FoE.
Corrective Actions
Please do not call FoE-related functions for the slave.

-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 slave supports FoE and is functioning properly.

-2312: ECAT_ERR_DEVICE_FOE_ERROR

Description
FoE communication encountered an error.
Corrective Actions
Please verify that the slave 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 slave 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 slave supports writing files via FoE communication.

-2400: ECAT_ERR_DEVICE_COE_SDO_NOT_SUPPORT

Description
The slave 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 slave 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 slave 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 slave does not have objects related to CiA 402.
Corrective Actions
Please do not call functions related to CiA 402 for the slave.

-2501: ECAT_ERR_DEVICE_CIA402_ADD_FAIL

Description
Failed to insert the CiA 402 slave object to the CiA 402 framework of EtherCAT master.
Corrective Actions
Please confirm whether the CiA 402 slave object has been successfully inserted into the CiA 402 framework of the EtherCAT master.

-2502: ECAT_ERR_DEVICE_CIA402_TYPE_MISMATCH

Description
The content of the object of Device Type (Index 1000h) for the slave is not CiA 402 type.
Corrective Actions
Please do not call functions related to CiA 402 for the slave.

-2503: ECAT_ERR_DEVICE_CIA402_NO_MODE_SUPPORT

Description
This CiA 402 slave does not support any operation modes defined by CiA 402.
Corrective Actions
Please do not call functions related to CiA 402 for the slave. Also, ensure that the CiA 402 slave 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 slave.
Corrective Actions
Please change the CiA 402 operation mode of this slave to the correct mode before calling this function.

-2505: ECAT_ERR_DEVICE_CIA402_MODE_NOT_SUPPORT

Description
The slave does not support the specified CiA 402 operation mode.
Corrective Actions
Please do not change the CiA 402 operation mode of this slave 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 slave does not support EoE.
Corrective Actions
Please do not call EoE-related functions for the slave.

-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 slave 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 Error Callback Code

Error Callback Code list:

ECAT_ERR_WKC_SINGLE_FAULT	Working Counter Fault.	2000001
ECAT_ERR_WKC_MULTIPLE_FAULTS	Working Counter Multiple Faults.	2000002
ECAT_ERR_SINGLE_LOST_FRAME	Single Lost Frame.	2000003
ECAT_ERR_MULTIPLE_LOST_FRAMES	Multiple Lost Frames.	2000004
ECAT_ERR_LOST_SLAVE	Lost Slave.	2000005
ECAT_ERR_STATE_MISMATCH	State Mismatch.	2000006
ECAT_ERR_CABLE_BROKEN	Cable Broken.	2000007
ECAT_ERR_WAIT_ACK_TIMEOUT	Wait ACK Timeout.	2001000

A.4 Event Callback Code

Event Callback Code list:

ECAT_EVT_STATE_CHANGED	State Changed.	1000001
ECAT_EVT_CABLE_RECONNECTED	Cable Reconnected.	1000002

A.5 SDO Abort Code

The CoE SDO Abort Codes defined in ETG.1000.6:

Value	Meaning
0x05030000	Toggle bit not changed.
0x05040000	SDO protocol timeout.
0x05040001	Client/Server command specifier not valid or unknown.
0x05040005	Out of memory.
0x06010000	Unsupported access to an object.
0x06010001	Attempt to read to a write only object.
0x06010002	Attempt to write to a read only object.
0x06010003	Subindex cannot be written, SI0 must be 0 for write access.
0x06010004	SDO Complete access not supported for objects of variable length such as ENUM object types.
0x06010005	Object length exceeds mailbox size.
0x06010006	Object mapped to RxPDO, SDO Download blocked.
0x06020000	The object does not exist in the object directory.
0x06040041	The object can not be mapped into the PDO.
0x06040042	The number and length of the objects to be mapped would exceed the PDO length.
0x06040043	General parameter incompatibility reason.
0x06040047	General internal incompatibility in the device.
0x06060000	Access failed due to a hardware error.
0x06070010	Data type does not match, length of service parameter does not match.
0x06070012	Data type does not match, length of service parameter too high.
0x06070013	Data type does not match, length of service parameter too low.
0x06090011	Subindex does not exist.
0x06090030	Value range of parameter exceeded (only for write access).
0x06090031	Value of parameter written too high.
0x06090032	Value of parameter written too low.
0x06090036	Maximum value is less than minimum value.
0x08000000	General error.
0x08000020	Data cannot be transferred or stored to the application. NOTE: This is the general Abort Code in case no further detail on the reason can determined. It is recommended to use one of the more detailed Abort Codes. (0x08000021, 0x08000022)

0x08000021	Data cannot be transferred or stored to the application because of local control. NOTE: “local control” means an application specific reason. It does not mean the ESM-specific control.
0x08000022	Data cannot be transferred or stored to the application because of the present device state. NOTE: “device state” means the ESM state.
0x08000023	Object dictionary dynamic generation fails or no object dictionary is present.

The extended CoE SDO Abort Codes defined in ETG.1020:

Value	**Meaning**
0x06010003	Subindex cannot be written, SI0 must be 0 for write access.
0x06010004	SDO Complete access not supported for objects of variable length such as ENUM object types.
0x06010005	Object length exceeds mailbox size.
0x06010006	Object mapped to RxPDO, SDO Download blocked. This optional Abort Code is used only in states SafeOp and Op.
0x06090033	configured module list does not match detected module list. It shall be used if Object 0xF03x is written but does not fit to object 0xF05x.

A.6 Data Type

The Basic Data Types defined in ETG.1000.6:

Index (hex)	Object Type	Name
0001	DEFTYPE	BOOLEAN
0002	DEFTYPE	INTEGER8
0003	DEFTYPE	INTEGER16
0004	DEFTYPE	INTEGER32
0005	DEFTYPE	UNSIGNED8
0006	DEFTYPE	UNSIGNED16
0007	DEFTYPE	UNSIGNED32
0008	DEFTYPE	REAL32
0009	DEFTYPE	VISIBLE_STRING
000A	DEFTYPE	OCTET_STRING
000B	DEFTYPE	UNICODE_STRING
000C	DEFTYPE	TIME_OF_DAY
000D	DEFTYPE	TIME_DIFFERENCE
000F	DEFTYPE	DOMAIN
0010	DEFTYPE	INTEGER24
0011	DEFTYPE	REAL64
0012	DEFTYPE	INTEGER40
0013	DEFTYPE	INTEGER48
0014	DEFTYPE	INTEGER56
0015	DEFTYPE	INTEGER64
0016	DEFTYPE	UNSIGNED24
0018	DEFTYPE	UNSIGNED40
0019	DEFTYPE	UNSIGNED48
001A	DEFTYPE	UNSIGNED56
001B	DEFTYPE	UNSIGNED64
001D	DEFTYPE	GUID
001E	DEFTYPE	BYTE
002D	DEFTYPE	BITARR8
002E	DEFTYPE	BITARR16
002F	DEFTYPE	BITARR32

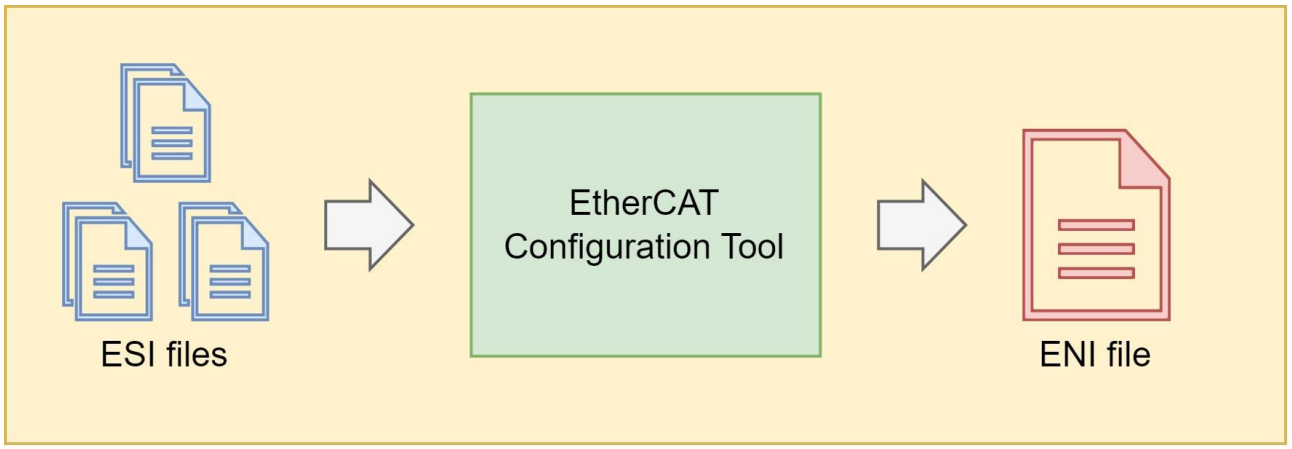
The Extended Data Types defined in ETG.1000.6:

Index (hex)	Object Type	Name
0021	DEFSTRUCT	PDO_MAPPING
0023	DEFSTRUCT	IDENTITY
0025	DEFSTRUCT	COMMAND_PAR
0029	DEFSTRUCT	SYNC_PAR
0030	DEFTYPE	BIT1
0031	DEFTYPE	BIT2
0032	DEFTYPE	BIT3
0033	DEFTYPE	BIT4
0034	DEFTYPE	BIT5
0035	DEFTYPE	BIT6
0036	DEFTYPE	BIT7
0037	DEFTYPE	BIT8
0040-005F	DEFSTRUCT	Manufacturer Specific Complex Data Types
0060-007F	DEFTYPE	Device Profile 0 Specific Standard Data Types
0080-009F	DEFSTRUCT	Device Profile 0 Specific Complex Data Types
00A0-00BF	DEFTYPE	Device Profile 1 Specific Standard Data Types
00C0-00DF	DEFSTRUCT	Device Profile 1 Specific Complex Data Types
00E0-00FF	DEFTYPE	Device Profile 2 Specific Standard Data Types
0100-011F	DEFSTRUCT	Device Profile 2 Specific Complex Data Types
0120-013F	DEFTYPE	Device Profile 3 Specific Standard Data Types
0140-015F	DEFSTRUCT	Device Profile 3 Specific Complex Data Types
0160-017F	DEFTYPE	Device Profile 4 Specific Standard Data Types
0180-019F	DEFSTRUCT	Device Profile 4 Specific Complex Data Types
01A0-01BF	DEFTYPE	Device Profile 5 Specific Standard Data Types
01C0-01DF	DEFSTRUCT	Device Profile 5 Specific Complex Data Types
01E0-01FF	DEFTYPE	Device Profile 6 Specific Standard Data Types
0200-021F	DEFSTRUCT	Device Profile 6 Specific Complex Data Types
0220-023F	DEFTYPE	Device Profile 7 Specific Standard Data Types
0240-025F	DEFSTRUCT	Device Profile 7 Specific Complex Data Types

A.7 EtherCAT Network Information

The EtherCAT Network Information (ENI) contains the necessary settings to configure an EtherCAT network. The XML-based file contains general information about the master and the configurations of every slave device connected to the master.

The EtherCAT Configuration Tool reads the ESI files or online scans the network for all slaves, 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 Master Software must support at least one of the following in the Network Configuration section: Online Scanning or Reading ENI. This library, however, supports both. In the case of Reading ENI, this library currently extracts only partial information from the ENI file for network configuration.

The extracted information includes the following:

`EtherCATConfig : Config : Slave : Info`

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

`EtherCATConfig : Config : Slave : Mailbox`

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

`EtherCATConfig : Config : Slave : Mailbox : CoE`

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

`EtherCATConfig : Config : Slave : ProcessData`

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

`EtherCATConfig : Config : Slave : ProcessData : Sm`

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

`EtherCATConfig : Config : Slave : DC`

- **Elements**
  - CycleTime0
  - CycleTime1
  - ShiftTime
- **Purpose**
  Used to configure the DC parameters of an EtherCAT slave.

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.