User Manual

EtherCAT CiA402 API

86Duino Coding IDE 501+

EtherCAT Library - EthercatDevice_CiA402 Class

(Version1.2)

REVISION

DATE	VERSION	DESCRIPTION
2024/12/20	**Version1.0**	New Release.
2025/1/3	**Version1.1**	Replace 'Master' by 'MainDevice' or 'MDevice', and 'Slave' by 'SubDevice', and will show the terms MainDevice and SubordinateDevice in the list of abbreviations.
2025/3/3	**Version1.2**	Add Homing Methods' instructions.

COPYRIGHT

Ver.1.2 March, 2025

TRADEMARKS ACKNOWLEDGMENT

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

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

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

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

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

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

This Manual is for the QEC series.

SAFETY INFORMATION

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

WARNING!

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

Content

Ch. 1

Introduction

1.1 About QEC MainDevice (MDevice)

QEC MDevice is an EtherCAT MDevice compatible with 86Duino Coding IDE 501+. It offers real-time EtherCAT communication between EtherCAT MDevice and EtherCAT SubDevice. Except for the EtherCAT Library of 86Duino IDE, QEC MDevice also provides Modbus, Ethernet TCP/IP, CAN bus, etc. industrial communication protocols and uses a rich high-level C/C++ programming language for rapid application development.

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 MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from https://www.qec.tw/software/.

QEC MDevice'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 SubDevice scanning
- Import ENI file
- Setting EtherCAT MDevice
- Configure EtherCAT SubDevice

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

1.1.2 QEC EtherCAT MDevice Architecture

The EtherCAT MDevice software is primarily divided into two parts, each running on the respective systems of the Vortex86EX2 CPU.

They are responsible for the following tasks:

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

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

1.1.3 Hardware Platform

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

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

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

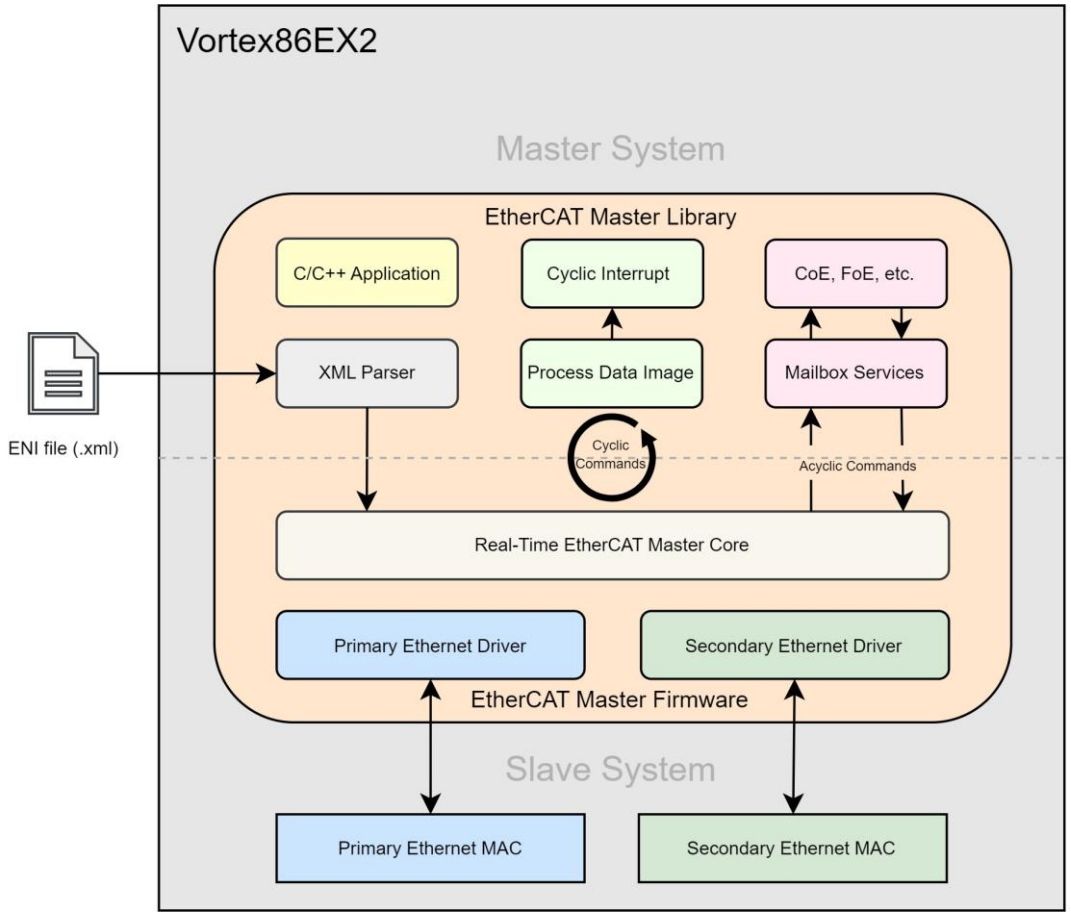


Figure 1: Vortex86EX2 Dual-core hardware platform diagram

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 MDevice Library blocks run on the Master System, while the Real-Time EtherCAT MDevice Core runs on the Slave System.

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

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

When the EtherCAT MDevice 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.

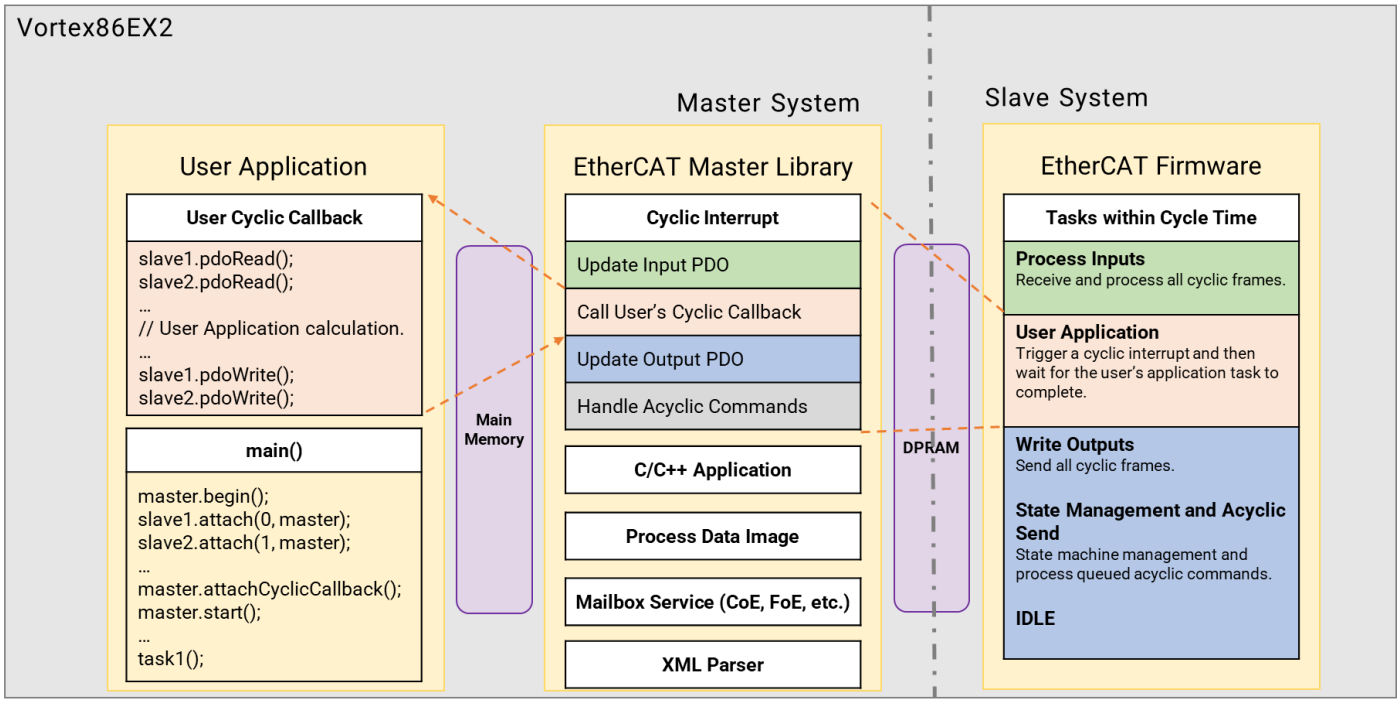


Figure 2: Vortex86EX2 Dual-core Synchronization

1.1.5 Callback Functions

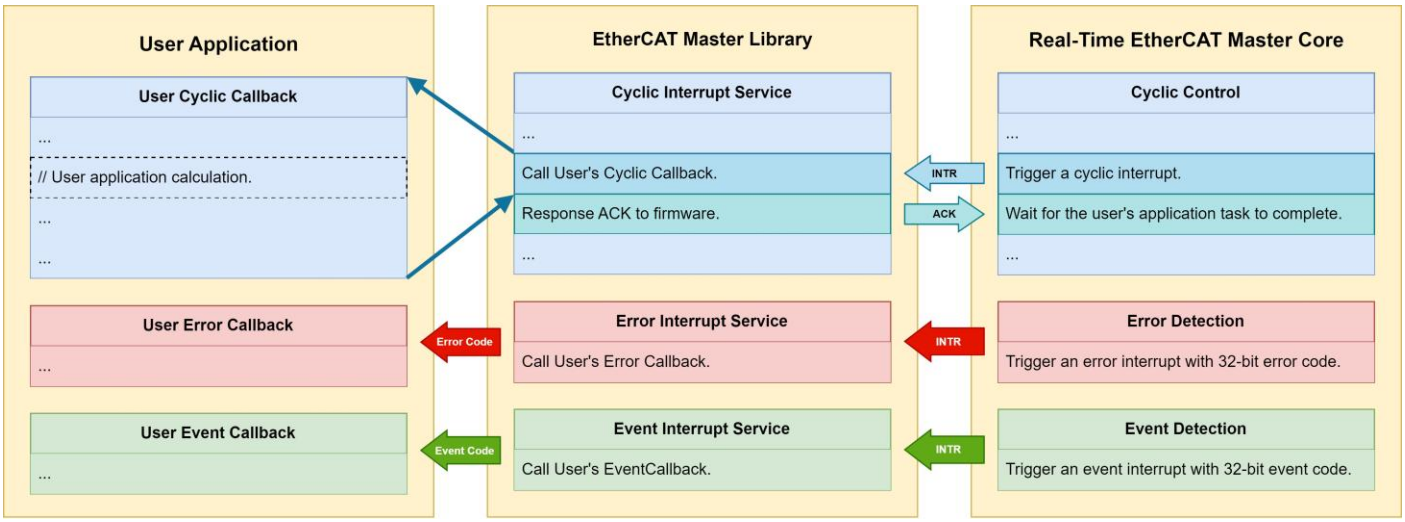


Figure 3: Vortex86EX2 Dual-core Callback Functions diagram

This library provides three types of callbacks as follows:

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

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

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

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

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

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

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

1.2 Features

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

- Class A: Standard EtherCAT MDevice
- Class B: Minimum EtherCAT MDevice

1.2.1 Feature Table

Word Usage:

- **O** equals supported.

Feature Name	Short Description	QEC MDevice
Basic Features		
Service Commands	Support of all commands	O
SubDevice with Device Emulation	Support SubDevice with and without application controller	O
EtherCAT State Machine	Support of ESM special behavior	O
Error Handling	Checking of network or SubDevice 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 MDevice	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 Synchronization

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

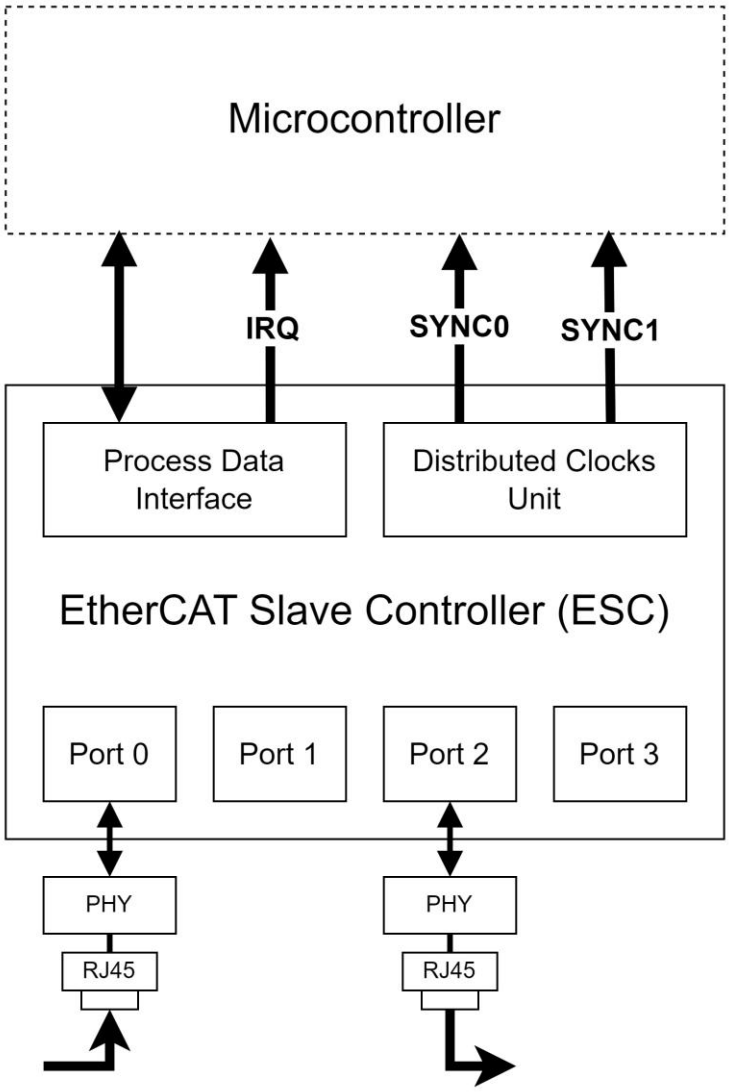


Figure 4: EtherCAT SubDevice time synchronization

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

There are three synchronization modes in EtherCAT:

- Free Run
- SM-Synchronous
- DC-Synchronous

1.3.1 Free Run

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

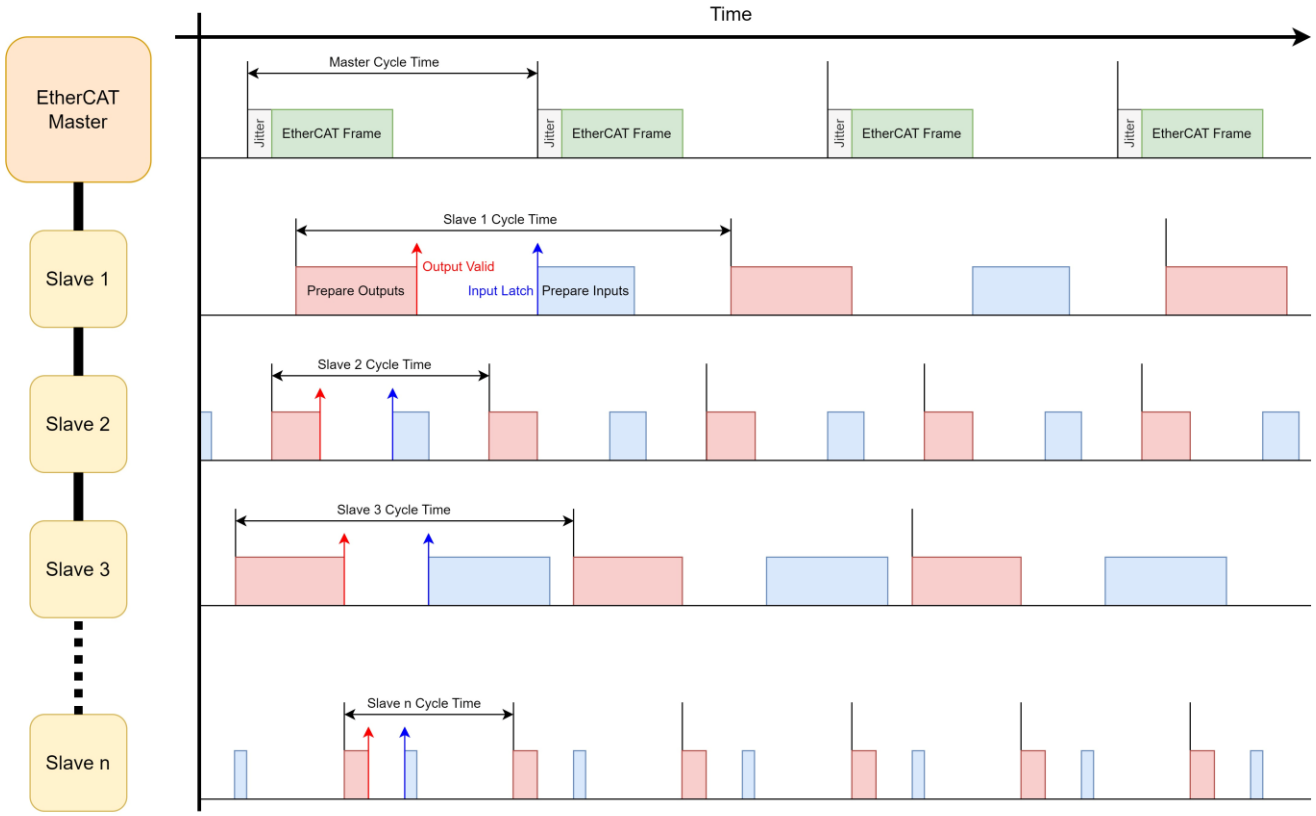


Figure 5: EtherCAT Free-run mode

1.3.2 SM-Synchronous

The IRQ pin generates a SM event when the cyclic frame is received by ESC, this event will trigger the execution of the local application in µC. As shown in the diagram below, cyclic frames are received by SubDevice with the same jitter of MDevice in sending them. Even assuming zero jitter, due to finite hardware ***Propagation Delay*** the last SubDevice 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 SubDevice, resulting in an accuracy of SM-Synchronous at the ***microsecond*** level.

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

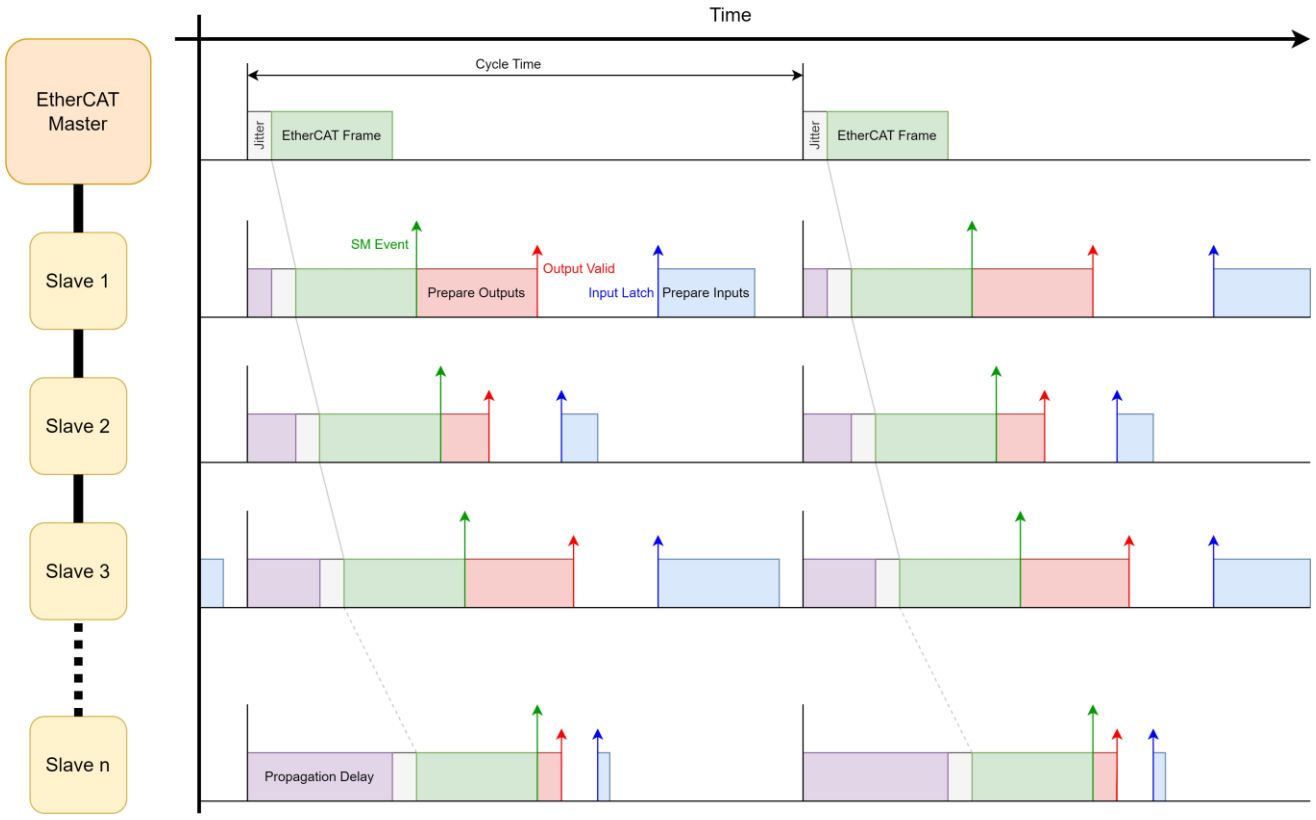


Figure 6: EtherCAT SM-Synchronous mode

1.3.3 DC-Synchronous

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

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

If the SubDevice 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 SubDevice 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 SubDevice.
- 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*.

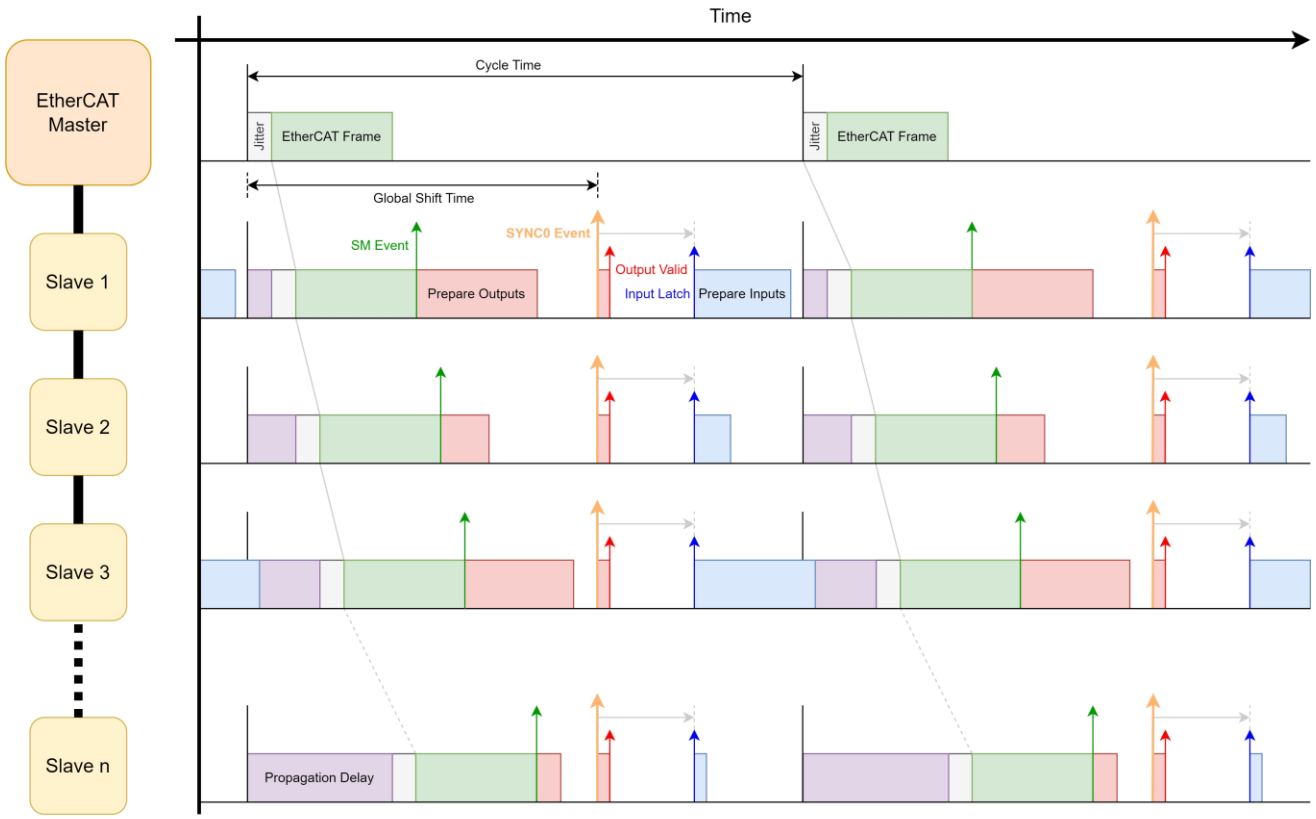


Figure 7: EtherCAT DC-Synchronous mode

1.4 About EthercatDevice_CiA402

EthercatDevice_CiA402 is a generic CiA 402 EtherCAT SubDevice class designed to control any EtherCAT servo drive that supports the CiA 402 standard.

It provides access functions for commonly used CiA 402 objects and operation functions for several CiA 402 operation modes and function groups, including:

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

Implementation Directive for CiA402 Drive Profile.
Directive for using IEC 61800-7-201 within EtherCAT-based servo drives.

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

- CiA Draft Standard 402: CiA® 402-CANopen Drives and Motion Control Profile.
- ETG.6010 Implementation Directive for CiA402 Drive Profile
- User manual for the currently used CiA 402 drive device

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

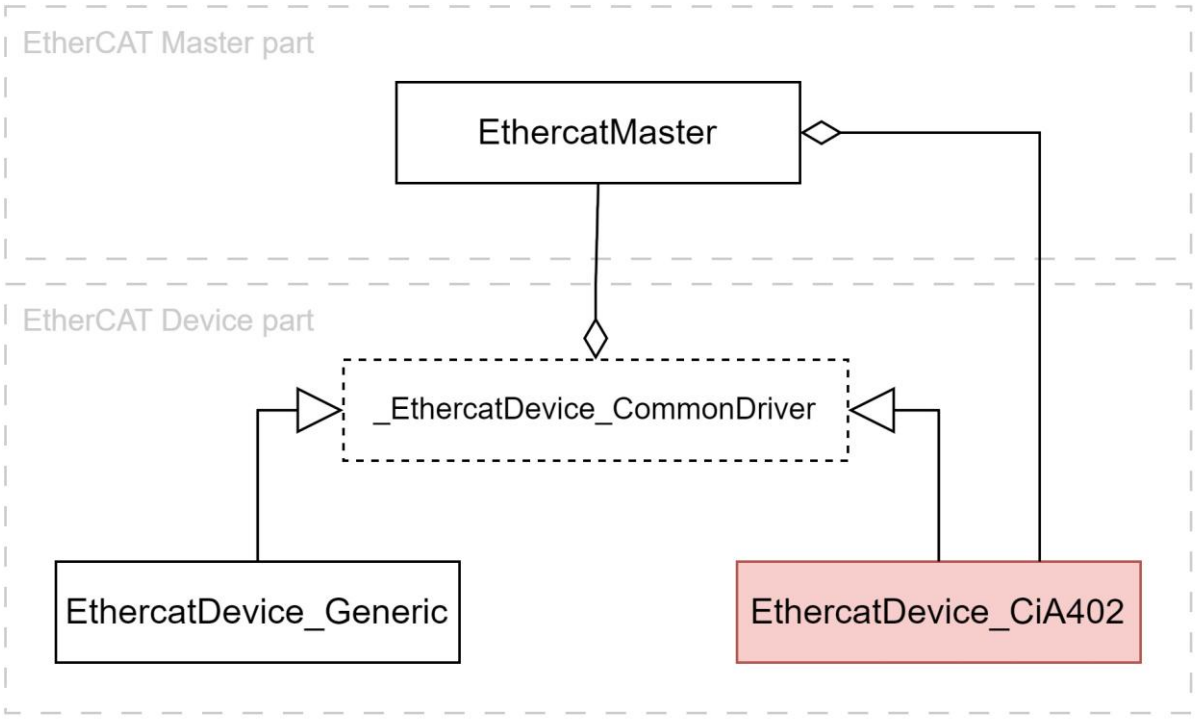


Figure 8: Relationships of EthercatDevice_CiA402 class

- EthercatDevice_CiA402 inherits from _EthercatDevice_CommonDriver.

For more detailed information about EtherCAT Device Class, please refer to [EtherCAT Library API User Manual - QEC].

1.4.1 Functions Table

This section will briefly introduce the EthercatDevice_CiA402 API usage list.

Function Name	Description	Callback Available
Initialization-related functions		
attach()	Initialize the object of this EtherCAT SubDevice class.	
detach()	Deinitialize the object of this EtherCAT SubDevice class.	
isStepper()	Check if the EtherCAT SubDevice is a stepper motor drive.	O
Control-related functions		
getCiA402Mode()	Get the current mode of operation. (6061_h)	O[1]
setCiA402Mode()	Switch the mode of operation. (6060_h, 6061_h, 6502_h)	O[1, 2]
getCiA402State()	Get the current CiA 402 state. (6041_h)	O
setCiA402State()	Switch the CiA 402 state. (6040h, 6041_h)	O[2]
enable()	Enable the drive function and power on the motor. (6040_h, 6041_h)	O[2]
disable()	Disable the drive function and power off the motor. (6040_h, 6041_h)	O[2]
Operation-related functions		
setTargetPosition()	Set the target position. ($607A_h$)	O[1]
setTargetVelocity()	Set the target velocity. ($60FF_h$)	O[1]
setTargetTorque()	Set the target torque. (6071_h)	O[1]
setProfileAcceleration()	Set the profile acceleration. (6083_h)	O[1]
setProfileDeceleration()	Set the profile deceleration. (6084_h)	O[1]
setMaxAcceleration()	Set the max acceleration. ($60C5_h$)	O[1]
setMaxDeceleration()	Set the max deceleration. ($60C6_h$)	O[1]
setMaxProfileVelocity()	Set the max profile velocity. ($607F_h$)	O[1]
setMotionProfileType()	Set the motion profile type. (6086_h)	O[1]
setPositionWindow()	Set the position window. (6067_h)	O[1]
setPositionWindowTime()	Set the position window time. (6068_h)	O[1]
setPositionOffset()	Set the position offset. ($60B0_h$)	O[1]
setSoftwarePositionLimit()	Set the software position limit. ($607D_h$)	O[1]
setFollowingErrorWindow()	Set the following error window. (6065_h)	O[1]
setPositionPolarity()	Set the position polarity. ($607E_h$)	O[1]
setVelocityWindow()	Set the velocity window. ($606D_h$)	O[1]
setVelocityWindowTime()	Set the velocity window time. ($606E_h$)	O[1]
setVelocityThreshold()	Set the velocity threshold. ($606F_h$)	O[1]
setVelocityOffset()	Set the velocity offset. ($60B1_h$)	O[1]
setMaxMotorSpeed()	Set the max motor speed. (6080_h)	O[1]
setVelocityPolarity()	Set the velocity polarity. ($607E_h$)	O[1]
setTorqueOffset()	Set the torque offset. ($60B2_h$)	O[1]

setMaxTorque()	Set the max torque. (6072_h)	O[1]
setPositiveTorqueLimit()	Set the positive torque limit. ($60E0_h$)	O[1]
setNegativeTorqueLimit()	Set the negative torque limit. ($60E1_h$)	O[1]
setQuickStopDeceleration()	Set the quick stop deceleration. (6085_h)	O[1]
setQuickStopOptionCode()	Set the quick stop option code. ($605A_h$)	
setShutdownOptionCode()	Set the shutdown option code. ($605B_h$)	
setDisableOperationOptionCode()	Set the disable operation option code. ($605C_h$)	
setHaltOptionCode()	Set the halt option code. ($605D_h$)	
setFaultReactionOptionCode()	Set the fault reaction option code. ($605E_h$)	
getErrorCode()	Get the error code. ($603F_h$)	O[1]
getSupportedDriveModes()	Get the supported drive modes. (6502_h)	O[1]
getMotorResolution()	Get the motor resolution. ($60EF_h$)	
getPositionActualValue()	Get the position actual value. (6064_h)	O[1]
getVelocityActualValue()	Get the velocity actual value. ($606C_h$)	O[1]
getTorqueActualValue()	Get the torque actual value. (6077_h)	O[1]
getCurrentActualValue()	Get the current actual value. (6078_h)	O[1]
getPositionDemandValue()	Get the position demand value. (6062_h)	O[1]
getPositionDemandInternalValue()	Get the position demand internal value. ($60FC_h$)	O[1]
getPositionActualInternalValue()	Get the position actual internal value. (6063_h)	O[1]
getAdditionalPositionActualValue()	Get the additional position actual value. ($60E4_h$)	O[1]
getFollowingErrorActualValue()	Get the following error actual value. ($60F4_h$)	O[1]
getVelocityDemandValue()	Get the velocity demand value. ($606B_h$)	O[1]
getTorqueDemandValue()	Get the torque demand value. (6074_h)	O[1]
getDigitalInputs()	Get the digital inputs. ($60FD_h$)	O[1]
Profile Position mode (pp) related functions		
pp_SetVelocity()	Set the profile velocity. (6081_h)	
pp_SetAcceleration()	Set the profile acceleration. (6083_h)	
pp_SetDeceleration()	Set the profile deceleration. (6084_h)	
pp_SetMotionProfileType()	Set the motion profile type. (6086_h)	
pp_Run()	Move to the target position. (6040_h, 6041_h, $607A_h$)	
pp_IsTargetReached()	Check if the target position has been reached. (6041_h)	O
pp_CheckFollowingError()	Check if the following error occurs. (6041_h)	O
pp_Halt()	Pause the current operation. (6040h, 6041_h)	
pp_Resume()	Resume the paused operation. (6040_h, 6041_h)	
Profile Velocity mode (pv) related functions		
pv_SetAcceleration()	Set the profile acceleration. (6083_h)	
pv_SetDeceleration()	Set the profile deceleration. (6084_h)	
pv_SetMotionProfileType()	Set the motion profile type. (6086_h)	
pv_Run()	Move at a target velocity continuously. (6041_h, $60FF_h$)	
pv_IsTargetReached()	Check if the target velocity has been reached. (6041_h)	O
pv_CheckZeroSpeed()	Check if the speed is zero. (6041_h)	O

pv_CheckMaxSlippageError()	Check if the maximum slippage error occurs. (6041_h)	O
pv_Halt()	Pause the current operation. (6040_h, 6041_h)	
pv_Resume()	Resume the paused operation. (6040_h, 6041_h)	
Profile Torque mode (tq) related functions		
tq_SetTorqueSlope()	Set the torque slope. (6087_h)	
tq_SetTorqueProfileType()	Set the torque profile type. (6088_h)	
tq_SetMotorRatedCurrent()	Set the motor rated current. (6075_h)	
tq_SetMotorRatedTorque()	Set the motor rated torque. (6076_h)	
tq_Run()	Drive continuously at the target torque. (6041_h, 6071_h)	
tq_IsTargetReached()	Check if the target torque has been reached. (6041_h)	O
tq_Halt()	Pause the current operation. (6040_h, 6041_h)	
tq_Resume()	Resume the paused operation. (6040_h, 6041_h)	
Homing mode (hm) related functions		
hm_SetHomeOffset()	Set the home offset. ($607C_h$)	
hm_SetHomingMethod()	Set the homing method. (6098_h)	
hm_SetHomingSpeeds()	Set the homing speeds. (6099_h)	
hm_SetHomingAcceleration()	Set the homing acceleration. ($609A_h$)	
hm_Run()	Initiate a homing operation. (6040_h, 6041_h)	
hm_IsAttained()	Check the status of the homing operation. (6041_h)	O
hm_Stop()	Stop the homing operation. (6040_h, 6041_h)	
Function Group "Touch Probe" related functions		
enableTouchProbe1()	Enable the touch probe 1. ($60B8_h$, $60B9_h$, $60D0_h$)	
enableTouchProbe2()	Enable the touch probe 2. ($60B8_h$, $60B9_h$, $60D0_h$)	
disableTouchProbe1()	Disable the touch probe 1. ($60B8_h$, $60B9_h$)	
disableTouchProbe2()	Disable the touch probe 2. ($60B8_h$, $60B9_h$)	
isTouchProbe1ValueReady()	Check if a positive or negative edge has occurred on the touch probe 1 signal. ($60B9_h$)	O[1]
isTouchProbe2ValueReady()	Check if a positive or negative edge has occurred on the touch probe 2 signal. ($60B9_h$)	O[1]
readTouchProbe1Value()	Read the touch probe position 1. ($60BA_h$, $60BB_h$)	O[1]
readTouchProbe2Value()	Read the touch probe position 2. ($60BC_h$, $60BD_h$)	O[1]
Low-level functions for mode-specific flow control		
setHaltBit()	Set the halt bit in the controlword. (6040_h)	O
isTargetReached()	Check if the target has reached. (6041_h)	O
setModeSpecificBit4()	Set the bit 4 in the controlword. (6040_h)	O
setModeSpecificBit5()	Set the bit 5 in the controlword. (6040_h)	O
setModeSpecificBit6()	Set the bit 6 in the controlword. (6040_h)	O
checkModeSpecificBit12()	Check the value of bit 12 in the statusword. (6041_h)	O
checkModeSpecificBit13()	Check the value of bit 13 in the statusword. (6041_h)	O

- **Note 1:** This function can be used in callback functions if the related object is mapped to PDO.
- **Note 2:** This function will ignore the timeout parameter and will not wait for the actual value to match the set value when used in a callback.

1.4.2 Drives and Motion Control

The device control function block controls all functions of the drive (drive function and power section). It is divided into device control of the state machine and operation mode function.

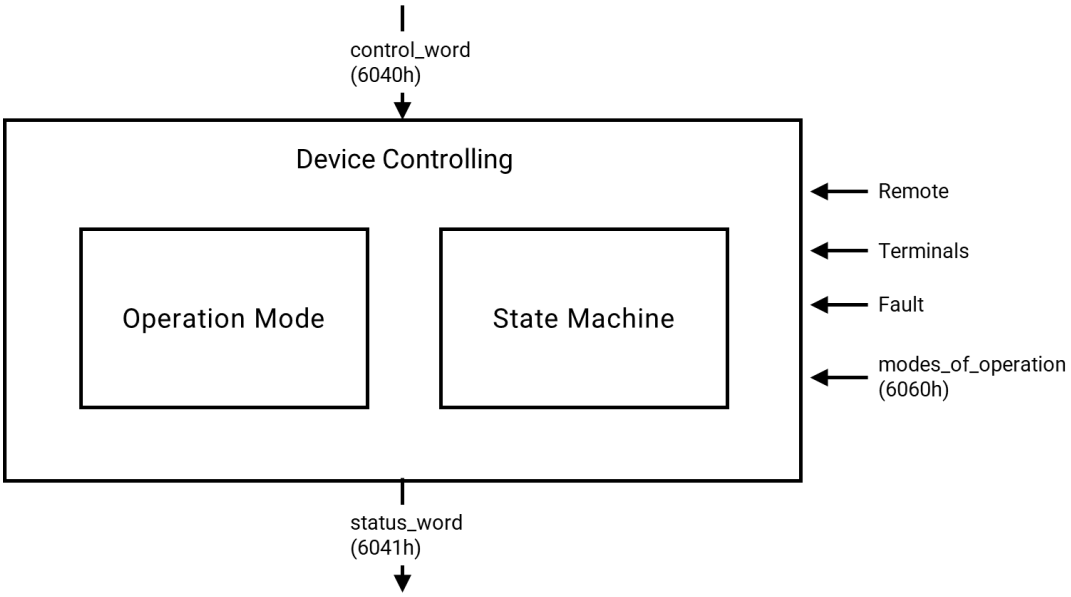


Figure 9: Device controlling

The state of the drive can be controlled by the controlword.
The state of the drive is shown in the statusword.
In remote mode the device is controlled directly from the CANopen network by PDO and SDO.
The state machine is controlled externally by the controlword and external signals. The write access to the controlword is controlled by the optional hardware signal 'Remote'. The state machine is also controlled by internal signals like faults and modes of operation.

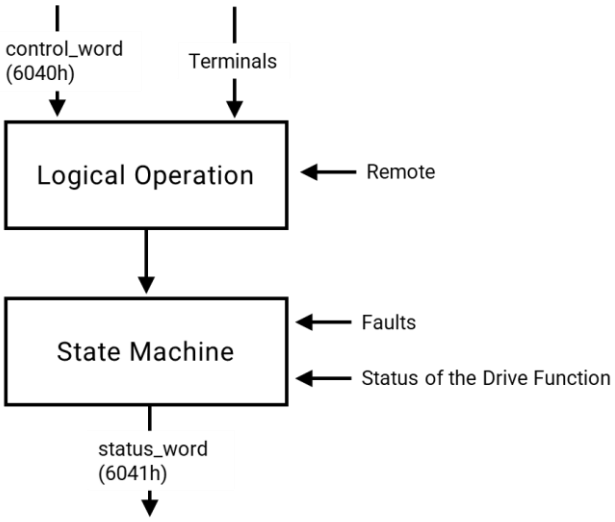


Figure 10: Remote mode

1.4.2.1 State machine

The state machine describes the device status and the possible control sequence of the drive. A single state represents a special internal or external behavior. The state of the drive also determines which commands are accepted. E.g. it is only possible to start a point-to-point move when the drive is in state OPERATION ENABLE.
States may be changed using the controlword and/or according to internal events. The current state can be read using the statusword.

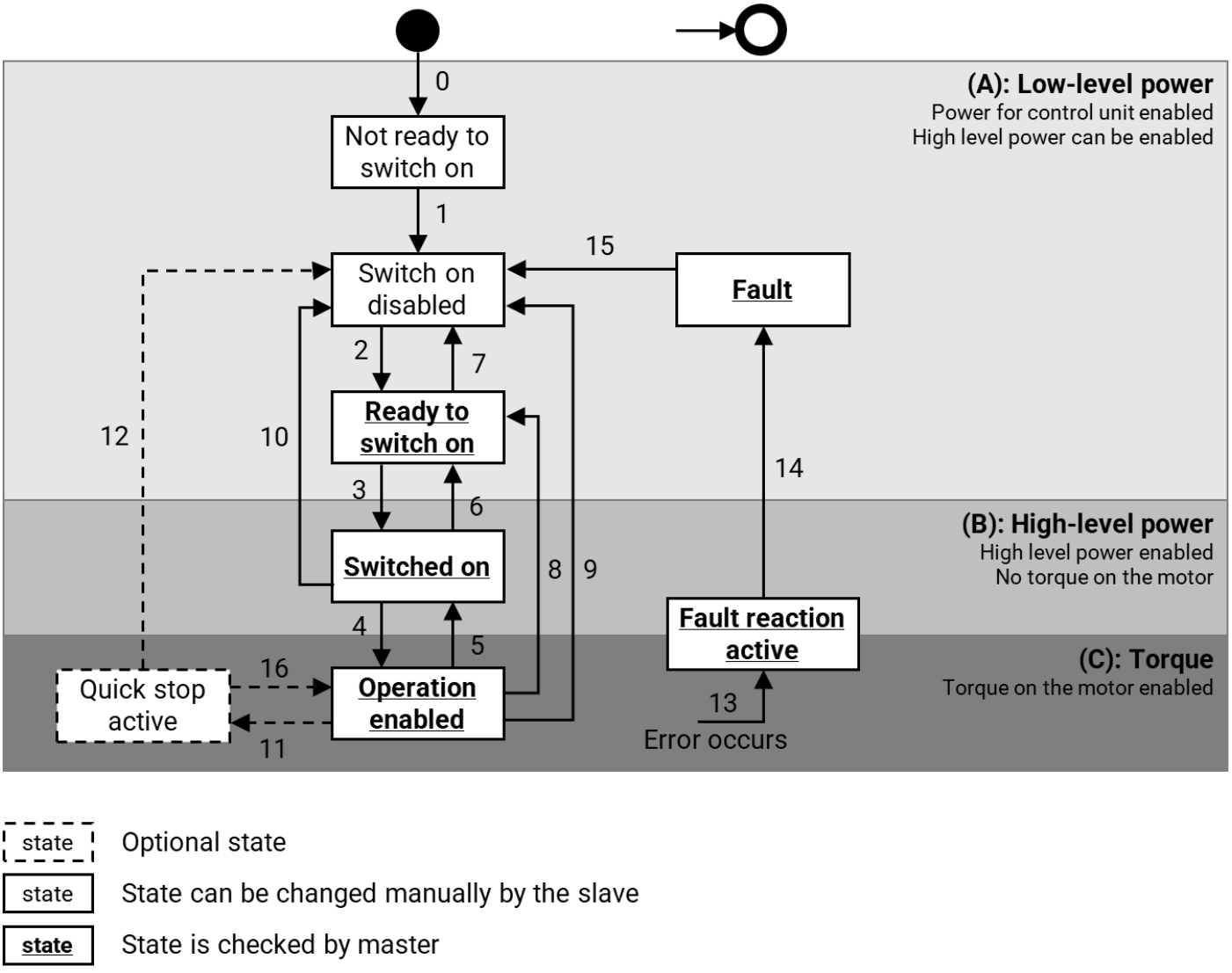


Figure 11: State Machine

State Description

State	Description
Not Ready to Switch On	Drive is initializing or running a self-test. Brake (if present) is applied. Function disabled.
Switch On Disabled	Initialization complete, parameters set. High voltage not applied for safety. Function disabled.
Ready to Switch On	High voltage may be applied. Drive parameters can be modified. Function disabled.
Switched On	High voltage applied. Power amplifier is ready. Drive parameters can be modified. Function disabled.
Operation Enable	No faults detected. Drive function enabled, power applied to motor. Normal operation state.
Quick Stop Active	Quick stop executed. Power removed, function disabled. To resume, send Enable Operation.
Fault Reaction Active	Fault detected. Quick stop is being executed. Power removed, function disabled.
Fault	Fault detected. Drive function disabled. High voltage switch-off depends on application.

Notes:

- If a command is received which causes a change of state, this command must be processed completely and the new state attained before the next command can be processed.

- 'Drive function is disabled' implies no energy is supplied to the motor. This may be achieved by different manufacturers in different ways. Reference values are not processed.

- 'Drive function is enabled' implies that energy can be supplied to the motor. The reference values (torque, velocity, position) are processed.

- 'Fault occurred' implies that a fault in the drive has occurred. In this case there is a transition to the state FAULT REACTION ACTIVE. In this state the device will execute a special fault reaction. After the execution of this fault reaction the device will switch to the state FAULT. This state can only be left by the command 'Fault Reset', but only if the fault is not active any more.

Ch. 2

Functions

86Duino Coding IDE 501+.

EtherCAT Library - EthercatDevice_CiA402 Class

2.1 Initialization-related Functions

Initialization-related functions for the EthercatDevice_CiA402 class.

Functions:

- attach()
- detach()
- isStepper()

attach()

Description

Initialize the object of this EtherCAT SubDevice class and attach it to the object of EthercatMaster class based on the ID of the SubDevice 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);
int attach(uint16_t slave_id, uint8_t drive_id, EthercatMaster *master, EthercatAttachMode mode = ECAT_SLAVE_NO);
int attach(uint16_t slave_id, uint8_t drive_id, EthercatMaster &master, EthercatAttachMode mode = ECAT_SLAVE_NO);
```

Parameters

- `[in] slave_id`
  The ID of the SubDevice on the EtherCAT bus. The definition of this ID is determined based on the mode parameter.
- `[in] drive_id`
  The ID of CiA 402 drive in the EtherCAT SubDevice. Typically used when a single EtherCAT SubDevice has multiple CiA 402 drive axes.
- `[in] master`
  The object of EthercatMaster class to which it should be attached.
- `[in] mode`
  The definition of slave_id :
  I. `ECAT_SLAVE_NO`
     The sequence number of the EtherCAT SubDevice on the network, 0 indicates the first SubDevice, 1 indicates the second SubDevice, and so on.
  II. `ECAT_ALIAS_ADDRESS`
     The alias address of the SubDevice on the network, which is defined at byte offset 8 in the SII EEPROM of the SubDevice.

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()`.

- WARNING: Prohibited from being called in the callback functions.

Example Code

- Single-axis CiA 402 EtherCAT Servo Drive

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

}
```

- Multi-axis CiA 402 EtherCAT Servo Drive

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor1;
EthercatDevice_CiA402 motor2;
EthercatDevice_CiA402 motor3;

void setup() {
  master.begin();
  motor1.attach(0, 0, master);
  motor2.attach(0, 1, master);
  motor3.attach(0, 2, master);
}

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

}
```

detach()

Description

Deinitialize the object of this EtherCAT SubDevice 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 in the callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  delay(1000);
  motor.detach();
}

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

}
```

isStepper()

Description

Check if the EtherCAT SubDevice is a stepper motor drive.

Syntax

```
int isStepper();
```

Parameters

None.

Return Value

Return whether the EtherCAT SubDevice is a stepper motor drive. 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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  Serial.print("Stepper Motor Drive: ");
  Serial.println(motor.isStepper());
}

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

}
```

2.2 Control-related Functions

Control-related functions for the EthercatDevice_CiA402 class.

Functions:

- getCiA402Mode()
- setCiA402Mode()
- getCiA402State()
- setCiA402State()
- enable()
- disable()

getCiA402Mode()

Description

Get the current mode of operation.

Relevant Objects

- Object 6061_h: Modes of operation display

Syntax

```
int getCiA402Mode();
```

Parameters

None.

Return Value

Return the current mode of operation:

Definition	Code	Description
CIA402_PP_MODE	1	Profile Position mode (pp)
CIA402_PV_MODE	3	Profile Velocity mode (pv)
CIA402_TQ_MODE	4	Profile Torque mode (tq)
CIA402_HOMING_MODE	6	Homing mode (hm)
CIA402_CSP_MODE	8	Cyclic Synchronous Position mode (csp)
CIA402_CSV_MODE	9	Cyclic Synchronous Velocity mode (csv)
CIA402_CST_MODE	10	Cyclic Synchronous Torque mode (cst)

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  Serial.print("Operation Mode: ");
  Serial.println(motor.getCiA402Mode());
}
```

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

}
```

setCiA402Mode()

Description

Switch the mode of operation.

Relevant Objects

- Object 6060_h: Modes of operation
- Object 6061_h: Modes of operation display
- Object 6502_h: Supported drive modes

Syntax

```
int setCiA402Mode(int mode, uint32_t timeout_ms = 1000);
```

Parameters

- `[in] mode`
  The mode of operation to be switched:

Definition	Code	Description
CIA402_PP_MODE	1	Profile Position mode (pp)
CIA402_PV_MODE	3	Profile Velocity mode (pv)
CIA402_TQ_MODE	4	Profile Torque mode (tq)
CIA402_HOMING_MODE	6	Homing mode (hm)
CIA402_CSP_MODE	8	Cyclic Synchronous Position mode (csp)
CIA402_CSV_MODE	9	Cyclic Synchronous Velocity mode (csv)
CIA402_CST_MODE	10	Cyclic Synchronous Torque mode (cst)

- `[in] 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 non-blocking and can be used in callback functions if the related object is mapped to a PDO. However, when used in a callback, this function will ignore the timeout parameter and will not wait for the actual value to match the set value.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
```

```
}

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

}
```

getCiA402State()

Description

Get the current CiA 402 state.

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int getCiA402State();
```

Parameters

None.

Return Value

Return the current CiA 402 state:

Definition	Code	State
CIA402_NOT_READY_TO_SWITCH_ON	0	Not ready to switch on.
CIA402_SWITCH_ON_DISABLED	1	Switch on disabled.
CIA402_READY_TO_SWITCH_ON	2	Ready to switch on.
CIA402_SWITCHED_ON	3	Switched on.
CIA402_OPERATION_ENABLED	4	Operation enabled.
CIA402_FAULT	5	Fault.
CIA402_FAULT_REACTION_ACTIVE	6	Fault reaction active.
CIA402_QUICK_STOP_ACTIVE	7	Quick stop active.

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()`. This function is non-blocking and can be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
```

```
  Serial.print("CiA402 State: ");
  Serial.println(motor.getCiA402State());
  delay(1000);
}
```

setCiA402State()

Description

Switch the CiA 402 state.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int setCiA402State(int state, uint32_t timeout_ms = 1000);
```

Parameters

- `[in] mode`
  The CiA 402 state to be switched:

Definition	Code	State
CIA402_SWITCH_ON_DISABLED	1	Switch on disabled.
CIA402_READY_TO_SWITCH_ON	2	Ready to switch on.
CIA402_SWITCHED_ON	3	Switched on.
CIA402_OPERATION_ENABLED	4	Operation enabled.
CIA402_QUICK_STOP_ACTIVE	7	Quick stop active.

- `[in] 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::start()`. This function is blocking and will wait for the actual value to match the set value until a timeout occurs. When used in a callback function, it becomes non-blocking and will ignore the timeout parameter, not waiting for the actual value to match the set value.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  motor.setCiA402State(CIA402_OPERATION_ENABLED);
  delay(4000);
```

```
  motor.setCiA402State(CIA402_SWITCH_ON_DISABLED);
}

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

}
```

enable()

Description

Enable the drive function and power on the motor. The behavior of this function is identical to setCiA402State(CIA402_OPERATION_ENABLED).

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int enable(uint32_t timeout_ms = 1000);
```

Parameters

- `[in] 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::start()`. This function is blocking and will wait for the actual value to match the set value until a timeout occurs. When used in a callback function, it becomes non-blocking and will ignore the timeout parameter, not waiting for the actual value to match the set value.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  motor.enable();
}

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

}
```

disable()

Description

Disable the drive function and power off the motor. The behavior of this function is identical to `setCiA402State(CIA402_SWITCH_ON_DISABLED)`.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int disable(uint32_t timeout_ms = 1000);
```

Parameters

- `[in] 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::start()`. This function is blocking and will wait for the actual value to match the set value until a timeout occurs. When used in a callback function, it becomes non-blocking and will ignore the timeout parameter, not waiting for the actual value to match the set value.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  motor.enable();
  delay(4000);
  motor.disable();
}

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

2.3 Operation-related Functions

Operation-related functions for the EthercatDevice_CiA402 class.

Functions:

- setTargetPosition()
- setTargetVelocity()
- setTargetTorque()
- setProfileAcceleration()
- setProfileDeceleration()
- setMaxAcceleration()
- setMaxDeceleration()
- setMaxProfileVelocity()
- setMotionProfileType()
- setPositionWindow()
- setPositionWindowTime()
- setPositionOffset()
- setSoftwarePositionLimit()
- setFollowingErrorWindow()
- setPositionPolarity()
- setVelocityWindow()
- setVelocityWindowTime()
- setVelocityThreshold()
- setVelocityOffset()
- setMaxMotorSpeed()
- setVelocityPolarity()
- setTorqueOffset()
- setMaxTorque()
- setPositiveTorqueLimit()
- setNegativeTorqueLimit()
- setQuickStopDeceleration()
- setQuickStopOptionCode()
- setShutdownOptionCode()
- setDisableOperationOptionCode()
- setHaltOptionCode()
- setFaultReactionOptionCode()
- getErrorCode()
- getSupportedDriveModes()
- getMotorResolution()
- getPositionActualValue()

- getVelocityActualValue()
- getTorqueActualValue()
- getCurrentActualValue()
- getPositionDemandValue()
- getPositionDemandInternalValue()
- getPositionActualInternalValue()
- getAdditionalPositionActualValue()
- getFollowingErrorActualValue()
- getVelocityDemandValue()
- getTorqueDemandValue()
- getDigitalInputs()

setTargetPosition()

Description

Set the target position object.

Relevant Objects

- Object 607A$_h$: Target position

Syntax

```
int setTargetPosition(int32_t value);
```

Parameters

- `[in] value`
  The target position to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
  motor.setTargetPosition(10000);
}

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

}
```

setTargetVelocity()

Description

Set the target velocity object.

Relevant Objects

- Object 60FF$_h$: Target velocity

Syntax

```
int setTargetVelocity(int32_t value);
```

Parameters

- `[in] value`
  The target velocity to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
  motor.setTargetVelocity(1000);
}

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

}
```

setTargetTorque()

Description

Set the target torque object.

Relevant Objects

- Object 6071_h: Target torque

Syntax

```
int setTargetTorque(int16_t value);
```

Parameters

- `[in] value`
  The target torque to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
  motor.setTargetTorque(30);
}

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

}
```

setProfileAcceleration()

Description

Set the profile acceleration object.

Relevant Objects

- Object 6083_h: Profile acceleration

Syntax

```
int setProfileAcceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile acceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setProfileAcceleration(5000);
}

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

}
```

setProfileDeceleration()

Description

Set the profile deceleration object.

Relevant Objects

- Object 6084_h: Profile deceleration

Syntax

```
int setProfileDeceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile deceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setProfileDeceleration(5000);
}

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

}
```

setMaxAcceleration()

Description

Set the max acceleration object.

Relevant Objects

- Object 60C5$_h$: Max acceleration

Syntax

```
int setMaxAcceleration(uint32_t value);
```

Parameters

- `[in] value`
  The max acceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMaxAcceleration(200);
}

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

}
```

setMaxDeceleration()

Description

Set the max deceleration object.

Relevant Objects

- Object 60C6$_h$: Max deceleration

Syntax

```
int setMaxDeceleration(uint32_t value);
```

Parameters

- `[in] value`
  The max deceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMaxDeceleration(200);
}

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

}
```

setMaxProfileVelocity()

Description

Set the max profile velocity object.

Relevant Objects

- Object 607F$_h$: Max profile velocity

Syntax

```
int setMaxProfileVelocity(uint32_t value);
```

Parameters

- `[in] value`
  The max profile velocity to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMaxProfileVelocity(50000);
}

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

}
```

setMotionProfileType()

Description

Set the motion profile type object.

Relevant Objects

- Object 6086_h: Motion profile type

Syntax

```
int setMotionProfileType(int16_t value);
```

Parameters

- `[in] value`

  The motion profile type to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Linear ramp (trapezoidal profile)
1	$\sin^2$ ramp
2	Jerk-free ramp
3	Jerk-limited ramp
4 .. 32767	Reserved

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMotionProfileType(0);
}

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

setPositionWindow()

Description

Set the position window object.

Relevant Objects

- Object 6067_h: Position window

Syntax

```
int setPositionWindow(uint32_t value);
```

Parameters

- `[in] value`
  The position window to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setPositionWindow(100);
}

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

}
```

setPositionWindowTime()

Description

Set the position window time object.

Relevant Objects

- Object 6068_h: Position window time

Syntax

```
int setPositionWindowTime(uint16_t value);
```

Parameters

- `[in] value`
  The position window time to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setPositionWindowTime(10);
}

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

}
```

setPositionOffset()

Description

Set the position offset object.

Relevant Objects

- Object 60B0$_h$: Position offset

Syntax

```
int setPositionOffset(int32_t value);
```

Parameters

- `[in] value`
  The position offset to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setPositionOffset(10000);
}

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

}
```

setSoftwarePositionLimit()

Description

Set the software position limit object.

Relevant Objects

- Object 607D$_h$: Software position limit

Syntax

```
int setSoftwarePositionLimit(int32_t min_value, int32_t max_value);
```

Parameters

- `[in] min_value`
  The minimum software position limit to be set.
- `[in] max_value`
  The maximum software position limit to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setSoftwarePositionLimit(-2147483648, 2147483647);
}

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

}
```

setFollowingErrorWindow()

Description

Set the following error window object.

Relevant Objects

- Object 6065_h: Following error window

Syntax

```
int setFollowingErrorWindow(uint32_t value);
```

Parameters

- `[in] value`
  The following error window to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setFollowingErrorWindow(3840000);
}

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

}
```

setPositionPolarity()

Description

Set the position polarity object.

Relevant Objects

- Object 607E$_h$: Polarity

Syntax

```
int setPositionPolarity(int polarity);
```

Parameters

- `[in] polarity`
  The position polarity to be set:

Definition	Code	Description
CIA402_POLARITY_POSITIVE	0	multiply by 1
CIA402_POLARITY_NEGATIVE	1	multiply by -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 non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setPositionPolarity(CIA402_POLARITY_POSITIVE);
}

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

}
```

setVelocityWindow()

Description

Set the velocity window object.

Relevant Objects

- Object 606D$_h$: Velocity window

Syntax

```
int setVelocityWindow(uint16_t value);
```

Parameters

- `[in] value`
  The velocity window to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setVelocityWindow(100);
}

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

}
```

setVelocityWindowTime()

Description

Set the velocity window time object.

Relevant Objects

- Object 606E$_h$: Velocity window time

Syntax

```
int setVelocityWindowTime(uint16_t value);
```

Parameters

- `[in] value`
  The velocity window time to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setVelocityWindowTime(10);
}

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

}
```

setVelocityThreshold()

Description

Set the velocity threshold object.

Relevant Objects

- Object 606F$_h$: Velocity threshold

Syntax

```
int setVelocityThreshold(uint16_t value);
```

Parameters

- `[in] value`
  The velocity threshold to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setVelocityThreshold(100);
}

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

}
```

setVelocityOffset()

Description

Set the velocity offset object.

Relevant Objects

- Object 60B1$_h$: Velocity offset

Syntax

```
int setVelocityOffset(int32_t value);
```

Parameters

- `[in] value`
  The velocity offset to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setVelocityOffset(1000);
}

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

}
```

setMaxMotorSpeed()

Description

Set the max motor speed object.

Relevant Objects

- Object 6080_h: Max motor speed

Syntax

```
int setMaxMotorSpeed(uint32_t value);
```

Parameters

- `[in] value`
  The max motor speed to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMaxMotorSpeed(5000);
}

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

}
```

setVelocityPolarity()

Description

Set the velocity polarity object.

Relevant Objects

- Object 607E$_h$: Polarity

Syntax

```
int setVelocityPolarity(int polarity);
```

Parameters

- [in] polarity

  The velocity polarity to be set:

Definition	Code	Description
CIA402_POLARITY_POSITIVE	0	multiply by 1
CIA402_POLARITY_NEGATIVE	1	multiply by -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 non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setVelocityPolarity(CIA402_POLARITY_POSITIVE);
}

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

}
```

setTorqueOffset()

Description

Set the torque offset object.

Relevant Objects

- Object $60B2_h$: Torque offset

Syntax

```
int setTorqueOffset(int16_t value);
```

Parameters

- `[in] value`
  The torque offset to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setTorqueOffset(500);
}

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

}
```

setMaxTorque()

Description

Set the max torque object.

Relevant Objects

- Object 6072_h: Max torque

Syntax

```
int setMaxTorque(uint16_t value);
```

Parameters

- `[in] value`
  The max torque to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setMaxTorque(3000);
}

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

}
```

setPositiveTorqueLimit()

Description

Set the positive torque limit object.

Relevant Objects

- Object 60E0$_h$: Positive torque limit value

Syntax

```
int setPositiveTorqueLimit(uint16_t value);
```

Parameters

- `[in] value`
  The positive torque limit to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setPositiveTorqueLimit(3000);
}

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

}
```

setNegativeTorqueLimit()

Description

Set the negative torque limit object.

Relevant Objects

- Object 60E1$_h$: Negative torque limit value

Syntax

```
int setNegativeTorqueLimit(uint16_t value);
```

Parameters

- `[in] value`
  The negative torque limit to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setNegativeTorqueLimit(3000);
}

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

}
```

setQuickStopDeceleration()

Description

Set the quick stop deceleration object.

Relevant Objects

- Object 6085_h: Quick stop deceleration

Syntax

```
int setQuickStopDeceleration(uint32_t value);
```

Parameters

- `[in] value`
  The quick stop deceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setQuickStopDeceleration(200);
}

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

}
```

setQuickStopOptionCode()

Description

Set the quick stop option code object.

Relevant Objects

- Object 605A$_h$: Quick stop option code

Syntax

```
int setQuickStopOptionCode(int16_t value);
```

Parameters

- `[in] value`

  The quick stop option code to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Disable drive function
1	Slow down on slow down ramp
2	Slow down on quick stop ramp
3	Slow down on the current limit
4	Slow down on the voltage limit
5	Slow down on slow down ramp and stay in QUICK STOP
6	Slow down on quick stop ramp and stay in QUICK STOP
7	Slow down on the current limit and stay in QUICK STOP
8	Slow down on the voltage limit and stay in QUICK STOP
9 ... 32767	Reserved

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setQuickStopOptionCode(1);
}
```

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

}
```

setShutdownOptionCode()

Description

Set the shutdown option code object.

Relevant Objects

- Object 605B$_h$: Shutdown option code

Syntax

```
int setShutdownOptionCode(int16_t value);
```

Parameters

- `[in] value`

  The shutdown option code to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Disable drive function
1	Slow down with slow down ramp; disable of the drive function
2 ... 32767	Reserved

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setShutdownOptionCode(1);
}

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

}
```

setDisableOperationOptionCode()

Description

Set the disable operation option code object.

Relevant Objects

- Object 605C$_h$: Disable operation option code

Syntax

```
int setDisableOperationOptionCode(int16_t value);
```

Parameters

- [in] value

  The disable operation option code to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Disable drive function
1	Slow down with slow down ramp and then disabling of the drive function
2 ... 32767	Reserved

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setDisableOperationOptionCode(1);
}

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

}
```

setHaltOptionCode()

Description

Set the halt option code object.

Relevant Objects

- Object 605D$_h$: Halt option code

Syntax

```
int setHaltOptionCode(int16_t value);
```

Parameters

- `[in] value`

  The halt option code to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Disable drive, motor is free to rotaten
1	Slow down on slow down ramp
2	Slow down on quick stop ramp
3	Slow down on the current limit
4	Slow down on the voltage limit
5 ... 32767	Reserved

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setHaltOptionCode(1);
}

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

setFaultReactionOptionCode()

Description

Set the fault reaction option code object.

Relevant Objects

- Object 605E$_h$: Fault reaction option code

Syntax

```
int setFaultReactionOptionCode(int16_t value);
```

Parameters

- `[in] value`

  The fault reaction option code to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Disable drive, motor is free to rotaten
1	Slow down on slow down ramp
2	Slow down on quick stop ramp
3	Slow down on the current limit
4	Slow down on the voltage limit
5 ... 32767	Reserved

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setFaultReactionOptionCode(1);
}

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

getErrorCode()

Description

Get the error code object.

Relevant Objects

- Object 603F$_h$: Error code

Syntax

```
uint16_t getErrorCode();
```

Parameters

None.

Return Value

Return the value of the error code object.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Error Code: ");
  Serial.println(motor.getErrorCode());
  delay(1000);
  // ...
}
```

getSupportedDriveModes()

Description

Get the supported drive modes object.

Relevant Objects

- Object 6502_h: Supported drive modes

Syntax

```
uint32_t getSupportedDriveModes();
```

Parameters

None.

Return Value

Return the value of the supported drive modes object:

Bit	Description
0	Profile Position mode (pp)
1	Velocity mode (vl)
2	Profile Velocity mode (pv)
3	Profile Torque mode (tq)
4	Reserved
5	Homing mode (hm)
6	Interpolated Position mode (ip)
7	Cyclic synchronous position mode (csp)
8	Cyclic synchronous velocity mode (csv)
9	Cyclic synchronous torque mode (cst)
10	Cyclic synchronous torque mode with commutation angle (cstca)
11 ... 15	Reserved
16 ... 31	Manufacturer Specific

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
```

```
  motor.attach(0, master);
  Serial.print("Supported Drive Modes: ");
  Serial.println(motor.getSupportedDriveModes());
}

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

}
```

getMotorResolution()

Description

Get the motor resolution object.

Relevant Objects

- Object 60EF$_h$: Motor resolution

Syntax

```
uint32_t getMotorResolution();
```

Parameters

None.

Return Value

Return the value of the motor resolution object.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  Serial.print("Motor Resolution: ");
  Serial.println(motor.getMotorResolution());
}

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

}
```

getPositionActualValue()

Description

Get the position actual value object.

Relevant Objects

- Object 6064_h: Position actual value

Syntax

```
int32_t getPositionActualValue();
```

Parameters

None.

Return Value

Return the position actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
  Serial.print("Position: ");
  Serial.println(motor.getPositionActualValue());
  delay(1000);
  // ...
}
```

getVelocityActualValue()

Description

Get the velocity actual value object.

Relevant Objects

- Object 606C$_h$: Velocity actual value

Syntax

```
int32_t getVelocityActualValue();
```

Parameters

None.

Return Value

Return the velocity actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
  Serial.print("Velocity: ");
  Serial.println(motor.getVelocityActualValue());
  delay(1000);
  // ...
}
```

getTorqueActualValue()

Description

Get the torque actual value object.

Relevant Objects

- Object 6077_h: Torque actual value

Syntax

```
int16_t getTorqueActualValue();
```

Parameters

None.

Return Value

Return the torque actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Torque: ");
  Serial.println(motor.getTorqueActualValue());
  delay(1000);
  // ...
}
```

getCurrentActualValue()

Description

Get the current actual value object.

Relevant Objects

- Object 6078_h: Current actual value

Syntax

```
int16_t getCurrentActualValue();
```

Parameters

None.

Return Value

Return the current actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Current: ");
  Serial.println(motor.getCurrentActualValue());
  delay(1000);
  // ...
}
```

getPositionDemandValue()

Description

Get the position demand value object.

Relevant Objects

- Object 6062_h: Position demand value

Syntax

```
int32_t getPositionDemandValue();
```

Parameters

None.

Return Value

Return the position demand value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Position: ");
  Serial.println(motor.getPositionDemandValue());
  delay(1000);
  // ...
}
```

getPositionDemandInternalValue()

Description

Get the position demand internal value object.

Relevant Objects

- Object 60FC$_h$: Position demand internal value

Syntax

```
int32_t getPositionDemandInternalValue();
```

Parameters

None.

Return Value

Return the position demand internal value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Position: ");
  Serial.println(motor.getPositionDemandValue());
  delay(1000);
  // ...
}
```

getPositionActualInternalValue()

Description

Get the position actual internal value object.

Relevant Objects

- Object 6063_h: Position actual internal value

Syntax

```
int32_t getPositionActualInternalValue();
```

Parameters

None.

Return Value

Return the position actual internal value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Position: ");
  Serial.println(motor.getPositionActualInternalValue());
  delay(1000);
  // ...
}
```

getAdditionalPositionActualValue()

Description

Get the additional position actual value object.

Relevant Objects

- Object 60E4$_h$: Additional position actual value

Syntax

```
int32_t getAdditionalPositionActualValue(int ordinal = 1);
```

Parameters

None.

Return Value

Return the additional position actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
  Serial.print("Position: ");
  Serial.println(motor.getAdditionalPositionActualValue());
  delay(1000);
  // ...
}
```

getFollowingErrorActualValue()

Description

Get the following error actual value object.

Relevant Objects

- Object 60F4$_h$: Following error actual value

Syntax

```
int32_t getFollowingErrorActualValue();
```

Parameters

None.

Return Value

Return the following error actual value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
  Serial.print("Following Error: ");
  Serial.println(motor.getFollowingErrorActualValue());
  delay(1000);
  // ...
}
```

getVelocityDemandValue()

Description

Get the velocity demand value object.

Relevant Objects

- Object 606B$_h$: Velocity demand value

Syntax

```
int32_t getVelocityDemandValue();
```

Parameters

None.

Return Value

Return the velocity demand value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Velocity: ");
  Serial.println(motor.getVelocityDemandValue());
  delay(1000);
  // ...
}
```

getTorqueDemandValue()

Description

Get the torque demand value object.

Relevant Objects

- Object 6074_h: Torque demand value

Syntax

```
int16_t getTorqueDemandValue();
```

Parameters

None.

Return Value

Return the torque demand value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();

  motor.enable();
}

void loop() {
  Serial.print("Torque: ");
  Serial.println(motor.getTorqueDemandValue());
  delay(1000);
  // ...
}
```

getDigitalInputs()

Description

Get the digital inputs object.

Relevant Objects

- Object 60FD$_h$: Digital inputs

Syntax

```
uint32_t getDigitalInputs();
```

Parameters

None.

Return Value

Return the value of the digital inputs object:

Bit	Description
0	Negative limit switch
1	Positive limit switch
2	Home switch
3	Interlock
4 ... 15	Reserved
16 ... 31	Manufacturer Specific

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  master.start();
}

void loop() {
  Serial.print("Digital Inputs: ");
  Serial.println(motor.getDigitalInputs());
```

```
  delay(1000);
  // ...
}
```

2.4 Profile Position mode (pp) Related Functions

Profile Position mode (pp) related functions for the EthercatDevice_CiA402 class.

Functions:

- pp_SetVelocity()
- pp_SetAcceleration()
- pp_SetDeceleration()
- pp_SetMotionProfileType()
- pp_Run()
- pp_IsTargetReached()
- pp_CheckFollowingError()
- pp_Halt()
- pp_Resume()

pp_SetVelocity()

Description

Set the profile velocity object.

Relevant Objects

- Object 6081_h: Profile velocity

Syntax

int pp_SetVelocity(uint32_t value);

Parameters

- `[in] value`
  The profile velocity to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
```

```
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(30000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_SetAcceleration()

Description

Set the profile acceleration object.

Relevant Objects

- Object 6083_h: Profile acceleration

Syntax

```
int pp_SetAcceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile acceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
```

```
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(30000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_SetDeceleration()

Description

Set the profile deceleration object.

Relevant Objects

- Object 6084_h: Profile deceleration

Syntax

```
int pp_SetDeceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile deceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
```

```
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(30000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_SetMotionProfileType()

Description

Set the motion profile type object.

Relevant Objects

- Object 6086_h: Motion profile type

Syntax

```
int pp_SetMotionProfileType(int16_t value);
```

Parameters

- `[in] value`

  The motion profile type to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Linear ramp (trapezoidal profile)
1	$\sin^2$ ramp
2	Jerk-free ramp
3	Jerk-limited ramp
4 .. 32767	Reserved

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
```

```
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);

  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(30000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_Run()

Description

Move to the target position in Profile Position mode (pp).

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword
- Object $607A_h$: Target position

Syntax

```
int pp_Run(int32_t position, int relative = CIA402_PP_ABSOLUTE, bool
change_immediately = false);
```

Parameters

- `[in] position`
  The target position to be set.
- `[in] relative`
  A value indicating whether the position parameter is relative or absolute:

Definition	Code	Description
CIA402_PP_ABSOLUTE	0	position is an absolute value.
CIA402_PP_RELATIVE	1	position is a relative value.

- `[in] change_immediately`
  A Boolean to determine if the target position change should be executed instantly.
  - true: Interrupt the actual positioning and start the next positioning.
  - false: Finish the actual positioning and then start the next positioning.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();
```

```
  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);

  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(30000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_IsTargetReached()

Description

Check if the target position has been reached in Profile Position mode (pp).

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int pp_IsTargetReached();
```

Parameters

None.

Return Value

Return whether the target position has been reached.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_CheckFollowingError()

Description

Check if the following error occurs in Profile Position mode (pp).

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int pp_CheckFollowingError();
```

Parameters

None.

Return Value

Return whether a following error has occurred.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
  if (motor.pp_CheckFollowingError())
    Serial.println("Following Error.");

  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
```

```
  if (motor.pp_CheckFollowingError())
    Serial.println("Following Error.");
}
```

pp_Halt()

Description

Pause the current operation and wait for the drive to stop based on the configuration specified in `setHaltOptionCode()`.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int pp_Halt(uint32_t timeout_ms = 10000);
```

Parameters

- `[in] 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::start()`. This function is blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(1000000, CIA402_PP_RELATIVE, true);
  delay(3000);
```

```
  motor.pp_Halt();
  delay(3000);
  motor.pp_Resume();
  while (motor.pp_IsTargetReached() == 0);

  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-1000000, CIA402_PP_RELATIVE, true);
  delay(3000);
  motor.pp_Halt();
  delay(3000);
  motor.pp_Resume();
  while (motor.pp_IsTargetReached() == 0);
}
```

pp_Resume()

Description

Resume the paused operation that was called before pp_Halt().

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int pp_Resume();
```

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(1000000, CIA402_PP_RELATIVE, true);
  delay(3000);
  motor.pp_Halt();
  delay(3000);
```

```
  motor.pp_Resume();
  while (motor.pp_IsTargetReached() == 0);

  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
  motor.pp_Run(-1000000, CIA402_PP_RELATIVE, true);
  delay(3000);
  motor.pp_Halt();
  delay(3000);
  motor.pp_Resume();
  while (motor.pp_IsTargetReached() == 0);
}
```

2.5 Profile Velocity mode (pv) Related Functions

Profile Velocity mode (pv) related functions for the EthercatDevice_CiA402 class.

Functions:

- pv_SetAcceleration()
- pv_SetDeceleration()
- pv_SetMotionProfileType()
- pv_Run()
- pv_IsTargetReached()
- pv_CheckZeroSpeed()
- pv_CheckMaxSlippageError()
- pv_Halt()
- pv_Resume()

pv_SetAcceleration()

Description

Set the profile acceleration object.

Relevant Objects

- Object 6083_h: Profile acceleration

Syntax

```
int pv_SetAcceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile acceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

pv_SetDeceleration()

Description

Set the profile deceleration object.

Relevant Objects

- Object 6084_h: Profile deceleration

Syntax

```
int pv_SetDeceleration(uint32_t value);
```

Parameters

- `[in] value`
  The profile deceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

pv_SetMotionProfileType()

Description

Set the motion profile type object.

Relevant Objects

- Object 6086_h: Motion profile type

Syntax

```
int pv_SetMotionProfileType(int16_t value);
```

Parameters

- `[in] value`

  The motion profile type to be set:

Value	Description
-32768 ... -1	Manufacturer Specific
0	Linear ramp (trapezoidal profile)
1	$\sin^2$ ramp
2	Jerk-free ramp
3	Jerk-limited ramp
4 .. 32767	Reserved

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
```

```
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);

  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

pv_Run()

Description

Move at a target velocity continuously in Profile Velocity mode (pv).

Relevant Objects

- Object 6041_h: Statusword
- Object $60FF_h$: Target velocity

Syntax

```
int pv_Run(int32_t velocity);
```

Parameters

- `[in] velocity`
  The target velocity to be set.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

pv_IsTargetReached()

Description

Check if the target velocity has been reached in Profile Velocity mode (pv).

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int pv_IsTargetReached();
```

Parameters

None.

Return Value

Return whether the target velocity has been reached.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);

  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
```

```
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

pv_CheckZeroSpeed()

Description

Check if the speed is zero in Profile Velocity mode (pv).

Relevant Objects

- Object 6041h: Statusword

Syntax

```
int pv_CheckZeroSpeed();
```

Parameters

None.

Return Value

Return whether the speed is zero.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);

  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
```

```
  motor.pv_Run(0);
  while (motor.pv_CheckZeroSpeed() == 0);
  delay(3000);
}
```

pv_CheckMaxSlippageError()

Description

Check if the maximum slippage error occurs in Profile Velocity mode (pv).

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int pv_CheckMaxSlippageError();
```

Parameters

None.

Return Value

Return whether a maximum slippage error has occurred.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  if (motor.pv_CheckMaxSlippageError())
    Serial.println("Max skippage Error.");
```

```
  delay(3000);

  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  if (motor.pv_CheckMaxSlippageError())
    Serial.println("Max skippage Error.");
  delay(3000);
}
```

pv_Halt()

Description

Pause the current operation and wait for the drive to stop based on the configuration specified in `setHaltOptionCode()`.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int pv_Halt(uint32_t timeout_ms = 10000);
```

Parameters

- `[in] 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::start()`. This function is blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
}

void loop() {
  motor.pv_Halt();
  delay(1000);
```

```
  motor.pv_Resume();
  delay(1000);
}
```

pv_Resume()

Description

Resume the paused operation that was called before pv_Halt().

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int pv_Resume();
```

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();
  motor.enable();
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
  motor.pv_Run(1000);
}
void loop() {
  motor.pv_Halt();
  delay(1000);
  motor.pv_Resume();
  delay(1000);
}
```

2.6 Profile Torque mode (tq) Related Functions

Profile Torque mode (tq) related functions for the EthercatDevice_CiA402 class.

Functions:

- tq_SetTorqueSlope()
- tq_SetTorqueProfileType()
- tq_SetMotorRatedCurrent()
- tq_SetMotorRatedTorque()
- tq_Run()
- tq_IsTargetReached()
- tq_Halt()
- tq_Resume()

tq_SetTorqueSlope()

Description

Set the torque slope object.

Relevant Objects

- Object 6087_h: Torque slope

Syntax

```
int tq_SetTorqueSlope(uint32_t value);
```

Parameters

- `[in] value`
  The torque slope to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_SetTorqueProfileType()

Description

Set the torque profile type object.

Relevant Objects

- Object 6088_h: Torque profile type

Syntax

```
int tq_SetTorqueProfileType(int16_t value);
```

Parameters

- `[in] value`

  The torque profile type to be set:

Value	Description
0000_h	Linear ramp (trapezoidal profile)
0001_h	$\sin^2$ ramp
0002_h .. $7FFF_h$	Reserved
8000_h .. $FFFF_h$	Manufacturer Specific

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
```

```
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);

  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_SetMotorRatedCurrent()

Description

Set the motor rated current object.

Relevant Objects

- Object 6075_h: Motor rated current

Syntax

```
int tq_SetMotorRatedCurrent(uint32_t value);
```

Parameters

- `[in] value`
  The motor rated current to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_SetMotorRatedTorque()

Description

Set the motor rated torque object.

Relevant Objects

- Object 6076_h: Motor rated torque

Syntax

```
int tq_SetMotorRatedTorque(uint32_t value);
```

Parameters

- `[in] value`
  The motor rated torque to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_Run()

Description

Drive continuously at the target torque in Profile Torque mode (tq).

Relevant Objects

- Object 6041_h: Statusword
- Object 6071_h: Target torque

Syntax

```
int tq_Run(int16_t torque);
```

Parameters

- `[in] torque`
  The target torque to be set.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
```

```
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_IsTargetReached()

Description

Check if the target torque has been reached in Profile Torque mode (tq).

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int tq_IsTargetReached();
```

Parameters

None.

Return Value

Return whether the target torque has been reached.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);

  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
```

```
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

tq_Halt()

Description

Pause the current operation and wait for the drive to stop based on the configuration specified in `setHaltOptionCode()`.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int tq_Halt(uint32_t timeout_ms = 10000);
```

Parameters

- `[in] 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::start()`. This function is blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
}

void loop() {
  motor.tq_Halt();
```

```
  delay(1000);
  motor.tq_Resume();
  delay(1000);
}
```

tq_Resume()

Description

Resume the paused operation that was called before tq_Halt().

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int tq_Resume();
```

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
  motor.tq_Run(50);
}

void loop() {
  motor.tq_Halt();
  delay(1000);
  motor.tq_Resume();
```

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

2.7 Homing mode (hm) Related Functions

Homing mode is the process by which a drive seeks the home position, also referred to as the datum, reference point, or zero point. This ensures that the motor starts from a known and repeatable position.
Homing can be achieved through various methods, typically using limit switches at the ends of travel or a home switch (zero point switch) in mid-travel. Many homing methods also utilize the index (zero) pulse train from an incremental encoder for precise positioning.

Homing mode (hm) related functions for the EthercatDevice_CiA402 class.
Functions:

- hm_SetHomeOffset()
- hm_SetHomingMethod()
- hm_SetHomingSpeeds()
- hm_SetHomingAcceleration()
- hm_Run()
- hm_IsAttained()
- hm_Stop()

hm_SetHomeOffset()

Description

Set the home offset object.

Relevant Objects

- Object 607C$_h$: Home offset

Syntax

```
int hm_SetHomeOffset(int32_t value);
```

Parameters

- `[in] value`
  The home offset to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
}

void loop() {
```

```
  // put your main code here, to run repeatedly:

}
```

hm_SetHomingMethod()

Description

Set the homing method object.

Relevant Objects

- Object 6098_h: Homing method

Syntax

```
int hm_SetHomingMethod(int8_t value);
```

Parameters

- `[in] value`

  The homing method to be set:

Value	Description
-128 .. -1	Manufacturer Specific
0	No homing operation required
1..35	Methods 1 to 35. You can refer to A.7 Homing Methods.
36 .. 127	Reserved

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
```

```
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
}

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

}
```

hm_SetHomingSpeeds()

Description

Set the homing speeds object.

Relevant Objects

- Object 6099_h: Homing speeds

Syntax

```
int hm_SetHomingSpeeds(uint32_t speed_during_search_switch, uint32_t
speed_during_search_zero);
```

Parameters

- `[in] speed_during_search_switch`
  The speed during search for switch to be set.
- `[in] speed_during_search_zero`
  The speed during search for zero to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
```

```
}

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

}
```

hm_SetHomingAcceleration()

Description

Set the homing acceleration object.

Relevant Objects

- Object 609A$_h$: Homing acceleration

Syntax

```
int hm_SetHomingAcceleration(uint32_t value);
```

Parameters

- `[in] value`
  The homing acceleration to be set.

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 in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
}

void loop() {
```

```
  // put your main code here, to run repeatedly:

}
```

hm_Run()

Description

Initiate a homing operation in Homing mode (hm).

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int hm_Run();
```

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
}

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

```
}
```

hm_IsAttained()

Description

Check the status of the homing operation in Homing mode (hm). This status represents bits 12 and 13 of the controlword object.

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int hm_IsAttained();
```

Parameters

None.

Return Value

Return the status of the homing operation. When the status value equals CIA402_HM_RUNNING, it indicates that the homing process is still in progress. The bit definitions of the status are shown in the table below.

Bit	Name	Value	Description
0	CIA402_HM_ATTAINED	0	Homing mode not yet completed.
		1	Homing mode carried out successfully.
1	CIA402_HM_ERROR	0	No homing error.
		1	Homing error occurred.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  int status;

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

  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();
```

```
  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while ((status = motor.hm_IsAttained()) == CIA402_HM_RUNNING);

  if (status & CIA402_HM_ATTAINED)
    Serial.println("Homing mode carried out successfully.");
  if (status & CIA402_HM_ERROR)
    Serial.println("Homing error occurred.");
}

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

}
```

hm_Stop()

Description

Stop the homing operation and wait for the drive to stop based on the configuration specified in `setHaltOptionCode()`.

Relevant Objects

- Object 6040_h: Controlword
- Object 6041_h: Statusword

Syntax

```
int hm_Stop(uint32_t timeout_ms = 10000);
```

Parameters

- `[in] 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::start()`. This function is blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  delay(1000);
  motor.hm_Stop();
}
```

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

}
```

2.8 Function Group "Touch Probe" Related Functions

Touch Probe related functions for the EthercatDevice_CiA402 class.

Functions:

- enableTouchProbe1()
- enableTouchProbe2()
- disableTouchProbe1()
- disableTouchProbe2()
- isTouchProbe1ValueReady()
- isTouchProbe2ValueReady()
- readTouchProbe1Value()
- readTouchProbe2Value()

enableTouchProbe1()

Description

Enable the touch probe 1.

Relevant Objects

- Object $60B8_h$: Touch probe function
- Object $60B9_h$: Touch probe status
- Object $60D0_h$: Touch probe source

Syntax

```
int enableTouchProbe1(int trigger_mode, int trigger_src, int trigger_timing);
```

Parameters

- `[in] trigger_mode`
  Choose the trigger mode of touch probe 1:

Definition	Code	Description
CIA402_TPMODE_FIRST_EVENT	0	Trigger the first event.
CIA402_TPMODE_CONTINUOUS	1	Trigger events continuously.

- `[in] trigger_src`
  Choose the trigger source of touch probe 1:

Definition	Code	Description
CIA402_TPSRC_TP_INPUT	0x00010000	Trigger with touch probe input.
CIA402_TPSRC_ENCODER_Z	0x00020000	Trigger with zero impulse signal or position encoder.
CIA402_TPSRC_DIGITAL_INPUT1	1	Trigger with digital input 1.
CIA402_TPSRC_DIGITAL_INPUT2	2	Trigger with digital input 2.
CIA402_TPSRC_DIGITAL_INPUT3	3	Trigger with digital input 3.
CIA402_TPSRC_DIGITAL_INPUT4	4	Trigger with digital input 4.
CIA402_TPSRC_HW_ENCODER_Z	5	Trigger with digital input 5.
CIA402_TPSRC_SW_ENCODER_Z	6	Trigger with digital input 6.

- `[in] trigger_timing`
  Choose the trigger timing of touch probe 1:

Definition	Code	Description
CIA402_TPTIMING_NONE	0	Switch off all sampling of touch probe 1.
CIA402_TPTIMING_POS_EDGE	1	Enable sampling at positive edge of touch probe 1.
CIA402_TPTIMING_NEG_EDGE	2	Enable sampling at negative edge of touch probe 1
CIA402_TPTIMING_BOTH	3	Enable sampling at both positive and negative edges of touch probe 1 simultaneously.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe1(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe1ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 1 positive value: ");
  Serial.println(motor.readTouchProbe1Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe1();
}

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

}
```

enableTouchProbe2()

Description

Enable the touch probe 2.

Relevant Objects

- Object $60B8_h$: Touch probe function
- Object $60B9_h$: Touch probe status
- Object $60D0_h$: Touch probe source

Syntax

```
int enableTouchProbe2(int trigger_mode, int trigger_src, int trigger_timing);
```

Parameters

- `[in] trigger_mode`
  Choose the trigger mode of touch probe 2:

Definition	Code	Description
CIA402_TPMODE_FIRST_EVENT	0	Trigger the first event.
CIA402_TPMODE_CONTINUOUS	1	Trigger events continuously.

- `[in] trigger_src`
  Choose the trigger source of touch probe 2:

Definition	Code	Description
CIA402_TPSRC_TP_INPUT	0x00010000	Trigger with touch probe input.
CIA402_TPSRC_ENCODER_Z	0x00020000	Trigger with zero impulse signal or position encoder.
CIA402_TPSRC_DIGITAL_INPUT1	1	Trigger with digital input 1.
CIA402_TPSRC_DIGITAL_INPUT2	2	Trigger with digital input 2.
CIA402_TPSRC_DIGITAL_INPUT3	3	Trigger with digital input 3.
CIA402_TPSRC_DIGITAL_INPUT4	4	Trigger with digital input 4.
CIA402_TPSRC_HW_ENCODER_Z	5	Trigger with digital input 5.
CIA402_TPSRC_SW_ENCODER_Z	6	Trigger with digital input 6.

- [in] trigger_timing
  Choose the trigger timing of touch probe 2:

Definition	Code	Description
CIA402_TPTIMING_NONE	0	Switch off all sampling of touch probe 2.
CIA402_TPTIMING_POS_EDGE	1	Enable sampling at positive edge of touch probe 2.
CIA402_TPTIMING_NEG_EDGE	2	Enable sampling at negative edge of touch probe 2
CIA402_TPTIMING_BOTH	3	Enable sampling at both positive and negative edges of touch probe 2 simultaneously.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe2(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe2ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 2 positive value: ");
  Serial.println(motor.readTouchProbe2Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe2();
}

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

}
```

disableTouchProbe1()

Description

Disable the touch probe 1.

Relevant Objects

- Object $60B8_h$: Touch probe function
- Object $60B9_h$: Touch probe status

Syntax

```
int disableTouchProbe1();
```

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 blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe1(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe1ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 1 positive value: ");
  Serial.println(motor.readTouchProbe1Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe1();
}

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

```
}
```

disableTouchProbe2()

Description

Disable the touch probe 2.

Relevant Objects

- Object $60B8_h$: Touch probe function
- Object $60B9_h$: Touch probe status

Syntax

```
int disableTouchProbe2();
```

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 blocking and cannot be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe2(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe2ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 2 positive value: ");
  Serial.println(motor.readTouchProbe2Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe2();
}

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

```
}
```

isTouchProbe1ValueReady()

Description

Check if a positive or negative edge has occurred on the touch probe 1 signal source.

Relevant Objects

- Object $60B9_h$: Touch probe status

Syntax

```
bool isTouchProbe1ValueReady(int value_selector);
```

Parameters

- `[in] value_selector`
  Choose the event to check on touch probe 1:

Definition	Code	Description
CIA402_TPVALUE_POS_EDGE	0	Check if a positive edge has occurred on the touch probe 1 signal source.
CIA402_TPVALUE_NEG_EDGE	1	Check if a negative edge has occurred on the touch probe 1 signal source.

Return Value

Return whether a positive or negative edge has occurred on the touch probe 1 signal source.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe1(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe1ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 1 positive value: ");
  Serial.println(motor.readTouchProbe1Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe1();
```

```
}

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

}
```

isTouchProbe2ValueReady()

Description

Check if a positive or negative edge has occurred on the touch probe 2 signal source.

Relevant Objects

- Object $60B9_h$: Touch probe status

Syntax

```
bool isTouchProbe2ValueReady(int value_selector);
```

Parameters

- `[in] value_selector`
  Choose the event to check on touch probe 2:

Definition	Code	Description
CIA402_TPVALUE_POS_EDGE	0	Check if a positive edge has occurred on the touch probe 2 signal source.
CIA402_TPVALUE_NEG_EDGE	1	Check if a negative edge has occurred on the touch probe 2 signal source.

Return Value

Return whether a positive or negative edge has occurred on the touch probe 2 signal source.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe2(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe2ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 2 positive value: ");
  Serial.println(motor.readTouchProbe2Value(CIA402_TPVALUE_POS_EDGE));
  motor.disableTouchProbe2();
```

```
}

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

}
```

readTouchProbe1Value()

Description

Read the touch probe position 1 positive/negative value object.

Relevant Objects

- Object $60BA_h$: Touch probe position 1 positive value
- Object $60BB_h$: Touch probe position 1 negative value

Syntax

```
int32_t readTouchProbe1Value(int value_selector);
```

Parameters

- `[in] value_selector`
  Choose the data source for reading the position value of touch probe 1:

Definition	Code	Description
CIA402_TPVALUE_POS_EDGE	0	Read the touch probe position 1 positive value. ($60BA_h$)
CIA402_TPVALUE_NEG_EDGE	1	Read the touch probe position 1 negative value. ($60BB_h$)

Return Value

Return the touch probe position 1 value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe1(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe1ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 1 positive value: ");
  Serial.println(motor.readTouchProbe1Value(CIA402_TPVALUE_POS_EDGE));
```

```
  motor.disableTouchProbe1();
}

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

}
```

readTouchProbe2Value()

Description

Read the touch probe position 2 positive/negative value object.

Relevant Objects

- Object $60BC_h$: Touch probe position 2 positive value
- Object $60BD_h$: Touch probe position 2 negative value

Syntax

```
int32_t readTouchProbe2Value(int value_selector);
```

Parameters

- `[in] value_selector`

  Choose the data source for reading the position value of touch probe 2:

Definition	Code	Description
CIA402_TPVALUE_POS_EDGE	0	Read the touch probe position 2 positive value. ($60BC_h$)
CIA402_TPVALUE_NEG_EDGE	1	Read the touch probe position 2 negative value. ($60BD_h$)

Return Value

Return the touch probe position 2 value.

Comment

This function must be called after a successful execution of `EthercatMaster::begin()`. This function is non-blocking and can be called in callback functions if the related object is mapped to PDO.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

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

  motor.enableTouchProbe2(CIA402_TPMODE_FIRST_EVENT, CIA402_TPSRC_TP_INPUT,
CIA402_TPTIMING_POS_EDGE);
  while (motor.isTouchProbe2ValueReady(CIA402_TPVALUE_POS_EDGE) == false);
  Serial.print("Touch probe position 2 positive value: ");
  Serial.println(motor.readTouchProbe2Value(CIA402_TPVALUE_POS_EDGE));
```

```
  motor.disableTouchProbe2();
}

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

}
```

2.9 Low-level functions for mode-specific flow control

Low-level functions for mode-specific flow control related functions for the EthercatDevice_CiA402 class.

Functions:

- setHaltBit()
- isTargetReached()
- setModeSpecificBit4()
- setModeSpecificBit5()
- setModeSpecificBit6()
- checkModeSpecificBit12()
- checkModeSpecificBit13()

setHaltBit()

Description

Set the halt bit in the controlword object.

Relevant Objects

- Object 6040_h: Controlword

Syntax

```
int setHaltBit(bool halt);
```

Parameters

- `[in] halt`
  A Boolean value used to halt the servo drive.
  - true: Halt the servo drive.
  - false: Resume the servo drive.

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 in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();

  motor.setTargetVelocity(1000);
  delay(3000);
  motor.setHaltBit(true);
  delay(3000);
  motor.setHaltBit(false);
  delay(3000);
  motor.setTargetVelocity(0);
  delay(3000);
```

```
}

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

}
```

isTargetReached()

Description

Check if the target has been reached. It checks the target reached bit in the statusword object.

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int isTargetReached();
```

Parameters

None.

Return Value

Return whether the target has been reached.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);

  motor.setTargetPosition(-100000);
```

```
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
}
```

setModeSpecificBit4()

Description

Set the bit 4 in the controlword object.

Relevant Objects

- Object 6040_h: Controlword

Syntax

```
int setModeSpecificBit4(bool bit_value);
```

Parameters

- `[in] bit_value`
  A Boolean value used to set or clear the bit 4 in the controlword object.
  - true: Set the bit value to 1.
  - false: Clear the bit value to 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::start()`. This function is non-blocking and can be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
```

```
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);

  motor.setTargetPosition(-100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
}
```

setModeSpecificBit5()

Description

Set the bit 5 in the controlword object.

Relevant Objects

- Object 6040_h: Controlword

Syntax

```
int setModeSpecificBit5(bool bit_value);
```

Parameters

- `[in] bit_value`
  A Boolean value used to set or clear the bit 5 in the controlword object.
  - true: Set the bit value to 1.
  - false: Clear the bit value to 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::start()`. This function is non-blocking and can be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
```

```
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);

  motor.setTargetPosition(-100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
}
```

setModeSpecificBit6()

Description

Set the bit 6 in the controlword object.

Relevant Objects

- Object 6040_h: Controlword

Syntax

```
int setModeSpecificBit6(bool bit_value);
```

Parameters

- `[in] bit_value`
  A Boolean value used to set or clear the bit 6 in the controlword object.
  - true: Set the bit value to 1.
  - false: Clear the bit value to 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::start()`. This function is non-blocking and can be called in callback functions.

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
```

```
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);

  motor.setTargetPosition(-100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
}
```

checkModeSpecificBit12()

Description

Check the value of bit 12 in the statusword object.

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int checkModeSpecificBit12();
```

Parameters

None.

Return Value

Return the value of bit 12 in the statusword.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);

  motor.setTargetPosition(-100000);
```

```
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
}
```

checkModeSpecificBit13()

Description

Check the value of bit 13 in the statusword object.

Relevant Objects

- Object 6041_h: Statusword

Syntax

```
int checkModeSpecificBit13();
```

Parameters

None.

Return Value

Return the value of bit 13 in the statusword.

Comment

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

Example Code

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

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

  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
}

void loop() {
  motor.setTargetPosition(100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
```

```
  while (motor.isTargetReached() == 0);
  if (motor.checkModeSpecificBit13())
    Serial.println("Following Error.");

  motor.setTargetPosition(-100000);
  motor.setModeSpecificBit6(true);
  motor.setModeSpecificBit5(true);
  motor.setModeSpecificBit4(true);
  while (motor.checkModeSpecificBit12() == 0);
  motor.setModeSpecificBit4(false);
  while (motor.checkModeSpecificBit12() == 1);
  while (motor.isTargetReached() == 0);
  if (motor.checkModeSpecificBit13())
    Serial.println("Following Error.");

}
```

Ch. 3

Example

3.1 Profile Position (pp) control

Implement position control on a CiA 402 EtherCAT SubDevice supporting Profile Position mode (pp).

- Move a relative distance of 10,000 units in the positive direction.
- Move a relative distance of 10,000 units in the negative direction.

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);
  master.start();

  motor.enable();
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
}

void loop() {
  motor.pp_Run(100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
  motor.pp_Run(-100000, CIA402_PP_RELATIVE, true);
  while (motor.pp_IsTargetReached() == 0);
}
```

3.2 Profile Position (pp) control in cyclic callback

Implement position control on a CiA 402 EtherCAT SubDevice supporting Profile Position mode (pp) in cyclic callback.

- Move a relative distance of 10,000 units in the positive direction.
- Move a relative distance of 10,000 units in the negative direction.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- **Output PDO (RxPDO)**
  - Object 6040h: Controlword
  - Object 607Ah: Target position
- **Input PDO (TxPDO)**
  - Object 6041h: Statusword
  - Object 6064h: Position actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

#define STATE_SET_COMMAND          (0)
#define STATE_CHECK_ACK_SET        (1)
#define STATE_CHECK_ACK_CLEAR      (2)
#define STATE_WAIT_TARGET_REACHED   (3)
int state = STATE_SET_COMMAND;
int toggle = 0;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  switch (state) {
    case STATE_SET_COMMAND:
      toggle = !toggle;
      motor.setTargetPosition(100000 * toggle - 100000 * !toggle);
      motor.setModeSpecificBit6(true);
      motor.setModeSpecificBit5(false);
      motor.setModeSpecificBit4(true);
```

```
      state = STATE_CHECK_ACK_SET;
      break;
    case STATE_CHECK_ACK_SET:
      if (motor.checkModeSpecificBit12()) {
        motor.setModeSpecificBit4(false);
        state = STATE_CHECK_ACK_CLEAR;
      }
      break;
    case STATE_CHECK_ACK_CLEAR:
      if (motor.checkModeSpecificBit12() == 0)
        state = STATE_WAIT_TARGET_REACHED;
      break;
    case STATE_WAIT_TARGET_REACHED:
      if (motor.pp_IsTargetReached())
        state = STATE_SET_COMMAND;
      break;
  }
}

void setup() {
  master.begin();

  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PP_MODE);

  /* RxPDO mapping configuration. */
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x607A0020);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x60640020);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);
```

```
  master.attachCyclicCallback(MyCyclicCallback);
  master.start();

  motor.enable();
  motor.pp_SetMotionProfileType(0);
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
  motor.pp_SetDeceleration(5000);
}

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

3.3 Profile Velocity (pv) control

Implement velocity control on a CiA 402 EtherCAT SubDevice supporting Profile Velocity mode (pv).

- Move in the positive direction at a speed of 1,000 units per second for 3 seconds.
- Move in the negative direction at a speed of 1,000 units per second for 3 seconds.

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_PV_MODE);
  master.start();

  motor.enable();
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
}

void loop() {
  motor.pv_Run(1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);

  motor.pv_Run(-1000);
  while (motor.pv_IsTargetReached() == 0);
  delay(3000);
}
```

3.4 Profile Velocity (pv) control in cyclic callback

Implement velocity control on a CiA 402 EtherCAT SubDevice supporting Profile Velocity mode (pv).

- Move in the positive direction at a speed of 1,000 units per second for 3 seconds.
- Move in the negative direction at a speed of 1,000 units per second for 3 seconds.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- **Output PDO (RxPDO)**
  - Object 6040h: Controlword
  - Object 60FFh: Target velocity
- **Input PDO (TxPDO)**
  - Object 6041h: Statusword
  - Object 606Ch: Velocity actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

int toggle = 0;
int cycle_count = 3000;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  if (++cycle_count < 3000)
    return;

  cycle_count = 0;
  toggle = !toggle;
  motor.setTargetVelocity(1000 * toggle - 1000 * !toggle);
}

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

```
  motor.setCiA402Mode(CIA402_PV_MODE);

  /* RxPDO mapping configuration. */
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x60FF0020);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x606C0020);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);

  master.attachCyclicCallback(MyCyclicCallback);
  master.start(1000000);

  motor.enable();
  motor.pv_SetMotionProfileType(0);
  motor.pv_SetAcceleration(5000);
  motor.pv_SetDeceleration(5000);
}

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

3.5 Profile Torque (tq) control

Implement torque control on a CiA 402 EtherCAT SubDevice supporting Profile Torque mode (tq).

- Maintain a positive torque of 50 units for 3 seconds.
- Maintain a negative torque of 50 units for 3 seconds.

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
  master.start();

  motor.enable();
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
}

void loop() {
  motor.tq_Run(50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);

  motor.tq_Run(-50);
  while (motor.tq_IsTargetReached() == 0);
  delay(3000);
}
```

3.6 Profile Torque (tq) control in cyclic callback

Implement torque control on a CiA 402 EtherCAT SubDevice supporting Profile Torque mode (tq).

- Maintain a positive torque of 50 units for 3 seconds.
- Maintain a negative torque of 50 units for 3 seconds.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- **Output PDO (RxPDO)**
  - Object 6040h: Controlword
  - Object 6071h: Target torque
- **Input PDO (TxPDO)**
  - Object 6041h: Statusword
  - Object 6077h: Torque actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

int toggle = 0;
int cycle_count = 3000;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  if (++cycle_count < 3000)
    return;

  cycle_count = 0;
  toggle = !toggle;
  motor.setTargetTorque(50 * toggle - 50 * !toggle);
}

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_TQ_MODE);
```

```
  /* RxPDO mapping configuration. */
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x60710010);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x60770010);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);

  master.attachCyclicCallback(MyCyclicCallback);
  master.start(1000000);

  motor.enable();
  motor.tq_SetTorqueProfileType(0);
  motor.tq_SetTorqueSlope(200);
  motor.tq_SetMotorRatedTorque(0);
  motor.tq_SetMotorRatedCurrent(0);
}

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

3.7 Homing (hm) operation

Initiate the homing method 33 operation on a CiA 402 compliant EtherCAT SubDevice.

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

void setup() {
  master.begin();
  motor.attach(0, master);
  motor.setCiA402Mode(CIA402_HOMING_MODE);
  master.start();

  motor.enable();

  motor.hm_SetHomingMethod(CIA402_HM33);
  motor.hm_SetHomeOffset(0);
  motor.hm_SetHomingSpeeds(100, 20);
  motor.hm_SetHomingAcceleration(100);
  motor.hm_Run();
  while (motor.hm_IsAttained() == CIA402_HM_RUNNING);
}

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

Using Serial input operation to control CiA402 single axis driver. Type ‘1’ for pp operation; type ‘0’ for Initiate the homing method 33 operation.

```
#include "Ethercat.h"


EthercatMaster master;
EthercatDevice_CiA402 motor;

#define STATE_IDLE    (0)
#define STATE_HOMING  (1)
#define STATE_MOVING  (2)
int state = STATE_IDLE;

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

    master.begin();
    motor.attach(0, master);
    master.start();
    motor.enable();

    Serial.println("Device is IDLE. (\'1\' for Moving, \'0\' for
Homing.)");
}

void loop()
{
    if (Serial.available()) {
        int ch = Serial.read();
        if (state == STATE_IDLE) {
            switch (ch) {
            case '1':
                motor.setCiA402Mode(CIA402_PP_MODE);
                motor.pp_SetVelocity(100000);
                motor.pp_SetAcceleration(5000);
                motor.pp_SetDeceleration(5000);
                motor.pp_Run(1000000, CIA402_PP_RELATIVE);
                state = STATE_MOVING;
                Serial.println("Moving task is starting.");
```

```
                break;
            case '0':
                motor.setCiA402Mode(CIA402_HOMING_MODE);
                motor.hm_SetHomingMethod(CIA402_HM33);
                motor.hm_SetHomeOffset(0);
                motor.hm_SetHomingSpeeds(100, 20);
                motor.hm_SetHomingAcceleration(100);
                motor.hm_Run();
                state = STATE_HOMING;
                Serial.println("Homing task is starting.");
                break;
            }
        }
    }

    if (state == STATE_HOMING) {
        switch (motor.hm_IsAttained()) {
        case CIA402_HM_RUNNING:
            break;
        case CIA402_HM_ATTAINED:
            Serial.println("Homing task is completed.");
            state = STATE_IDLE;
            break;
        case CIA402_HM_ERROR:
            Serial.println("Homing task has an error.");
            state = STATE_IDLE;
            break;
        }
    }

    if (state == STATE_MOVING) {
        if (motor.pp_IsTargetReached()) {
            Serial.println("Moving task is completed.");
            state = STATE_IDLE;
        }
    }
}
```

3.8 Cyclic synchronous position (csp) control in cyclic callback

Implement Cyclic Synchronous Position (csp) control on a CiA 402 EtherCAT SubDevice.

- The target position is incremented by 1,000 units in each cycle.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- **Output PDO (RxPDO)**
  - Object 6040h: Controlword
  - Object 607Ah: Target position
- **Input PDO (TxPDO)**
  - Object 6041h: Statusword
  - Object 6064h: Position actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

int32_t position = 0;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  motor.setTargetPosition(position += 1000);
}

void setup() {
  master.begin();

  motor.attach(0, master);
  motor.setDc(1000000);
  motor.setCiA402Mode(CIA402_CSP_MODE);

  /* RxPDO mapping configuration. */
```

```
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x607A0020);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x60640020);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);

  master.attachCyclicCallback(MyCyclicCallback);
  master.start(1000000, ECAT_SYNC);

  motor.setTargetPosition(position = motor.getPositionActualValue());
  motor.enable();
}

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

3.9 Cyclic synchronous velocity (csv) control in cyclic callback

Implement Cyclic Synchronous Velocity (csv) control on a CiA 402 EtherCAT SubDevice.

- The target velocity is incremented by 1 unit each cycle until it reaches 3,000 units.
- The target velocity is decremented by 1 unit each cycle until it reaches -3,000 units.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- **Output PDO (RxPDO)**
  - Object 6040h: Controlword
  - Object 60FFh: Target velocity
- **Input PDO (TxPDO)**
  - Object 6041h: Statusword
  - Object 606Ch: Velocity actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

int32_t velocity = 0;
int toggle = 0;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  if (abs(velocity) >= 3000)
    toggle = !toggle;

  velocity = velocity +  toggle - !toggle;
  motor.setTargetVelocity(velocity);
}

void setup() {
  master.begin();
```

```
  motor.attach(0, master);
  motor.setDc(1000000);
  motor.setCiA402Mode(CIA402_CSV_MODE);

  /* RxPDO mapping configuration. */
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x60FF0020);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x606C0020);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);

  master.attachCyclicCallback(MyCyclicCallback);
  master.start(1000000, ECAT_SYNC);

  motor.setTargetVelocity(0);
  motor.enable();
}

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

3.10 Cyclic synchronous torque (cst) control in cyclic callback

Implement Cyclic Synchronous Torque (cst) control on a CiA 402 EtherCAT SubDevice.

- The target torque is incremented by 1 unit each cycle until it reaches 50 units.
- The target torque is decremented by 1 unit each cycle until it reaches -50 units.

To operate in cyclic callback, the relevant objects must be mapped to PDOs as follows:

- Output PDO (RxPDO)
  - Object 6040h: Controlword
  - Object 6071h: Target torque
- Input PDO (TxPDO)
  - Object 6041h: Statusword
  - Object 6077h: Torque actual value

Here is the example code.

```
#include "Ethercat.h"

EthercatMaster master;
EthercatDevice_CiA402 motor;

int16_t torque = 0;
int toggle = 0;

void MyCyclicCallback()
{
  if (motor.getCiA402State() != CIA402_OPERATION_ENABLED)
    return;

  if (abs(torque) >= 50)
    toggle = !toggle;

  torque = torque +  toggle - !toggle;
  motor.setTargetTorque(torque);
}

void setup() {
  master.begin();
```

```
  motor.attach(0, master);
  motor.setDc(1000000);
  motor.setCiA402Mode(CIA402_CST_MODE);

  /* RxPDO mapping configuration. */
  motor.sdoDownload8(0x1C12, 0x00, 0);
  motor.sdoDownload8(0x1601, 0x00, 0);
  motor.sdoDownload32(0x1601, 0x01, 0x60400010);
  motor.sdoDownload32(0x1601, 0x02, 0x60710010);
  motor.sdoDownload8(0x1601, 0x00, 2);
  motor.sdoDownload16(0x1C12, 0x01, 0x1601);
  motor.sdoDownload8(0x1C12, 0x00, 1);
  /* TxPDO mapping configuration. */
  motor.sdoDownload8(0x1C13, 0x00, 0);
  motor.sdoDownload8(0x1A01, 0x00, 0);
  motor.sdoDownload32(0x1A01, 0x01, 0x60410010);
  motor.sdoDownload32(0x1A01, 0x02, 0x60770010);
  motor.sdoDownload8(0x1A01, 0x00, 2);
  motor.sdoDownload16(0x1C13, 0x01, 0x1A01);
  motor.sdoDownload8(0x1C13, 0x00, 1);

  master.attachCyclicCallback(MyCyclicCallback);
  master.start(1000000, ECAT_SYNC);

  motor.setTargetTorque(0);
  motor.enable();
}

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

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 MDevice encountered an error.

Corrective Actions

Please contact the manufacturer.

-201: ECAT_ERR_MASTER_SET_DEVICE_SETTINGS_FAIL

Description

The command to set the configuration of the EtherCAT SubDevice encountered an error.

Corrective Actions

Please contact the manufacturer.

-202: ECAT_ERR_MASTER_GET_GROUP_INFO_FAIL

Description

The command to get the configuration of the EtherCAT group encountered an error.

Corrective Actions

Please contact the manufacturer.

-203: ECAT_ERR_MASTER_GET_MASTER_INFO_FAIL

Description

The command to get the configuration of the EtherCAT MDevice encountered an error.

Corrective Actions

Please contact the manufacturer.

-204: ECAT_ERR_MASTER_GET_DEVICE_INFO_FAIL

Description
The command to get the configuration of the EtherCAT SubDevice encountered an error.
Corrective Actions
Please contact the manufacturer.

-205: ECAT_ERR_MASTER_SET_GROUP_SETTINGS_FAIL

Description
The command to set the configuration of the EtherCAT group encountered an error.
Corrective Actions
Please contact the manufacturer.

-206: ECAT_ERR_MASTER_MAPPING_INIT_FAIL

Description
Failed to allocate memory for PDO mapping.
Corrective Actions
Please contact the manufacturer.

-207: ECAT_ERR_MASTER_INTERRUPT_INIT_FAIL

Description
Initialization of the interrupt function failed.
Corrective Actions
Please contact the manufacturer.

-208: ECAT_ERR_MASTER_ACTIVE_FAIL

Description
The command to activate the EtherCAT MDevice failed.
Corrective Actions
Please contact the manufacturer.

-209: ECAT_ERR_MASTER_ENI_INITCMDS_FAIL

Description
The execution of the SDO Download command in the ENI file failed.
Corrective Actions
Please verify the correctness of the ENI file content and ensure that all SubDevices are functioning properly.

-210: ECAT_ERR_MASTER_NO_DEVICE

Description
EtherCAT network has no SubDevices.
Corrective Actions
Please connect at least one EtherCAT SubDevice.

-300: ECAT_ERR_MASTER_ACYCLIC_INIT_FAIL

Description
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 SubDevice is functioning properly.

-304: ECAT_ERR_MASTER_ACYCLIC_ERROR

Description
Acyclic transfer encountered an error.
Corrective Actions
Please check the error code. Such as the CoE communication, you can check Abort Code.

-405: ECAT_ERR_MASTER_ACYCLIC_WRONG_STATUS

Description
Acyclic transfer obtained an invalid status.
Corrective Actions
Please contact the manufacturer.

-400: ECAT_ERR_MASTER_GENERIC_SEND_FAIL

Description
Failed to send data during generic transmission.
Corrective Actions
Please contact the manufacturer.

-401: ECAT_ERR_MASTER_GENERIC_RECV_FAIL

Description
Failed to receive data during generic transmission.
Corrective Actions
Please contact the manufacturer.

-1000: ECAT_ERR_MASTER_NOT_BEGIN

Description
The EtherCAT MDevice has not been initialized.
Corrective Actions
Please call EthercatMaster::begin().

-1001: ECAT_ERR_MASTER_WRONG_BUFFER_SIZE

Description
The size of the input buffer is incorrect.
Corrective Actions
Please input a buffer with correct size.

-1002: ECAT_ERR_MASTER_REDUNDANCY_NO_DC

Description
Cable Redundancy mode does not support DC (Distributed Clocks).
Corrective Actions
Do not call this function or avoid Cable Redundancy mode.

-1003: ECAT_ERR_MASTER_MEMORY_ALLOCATION_FAIL

Description
Failed to allocate memory.
Corrective Actions
Please contact the manufacturer.

-1004: ECAT_ERR_MASTER_OSLAYER_INIT_FAIL

Description
Operating system layer initialization failed.
Corrective Actions
Please contact the manufacturer.

-1005: ECAT_ERR_MASTER_NIC_INIT_FAIL

Description
Network controller initialization failed.
Corrective Actions
Please ensure the network controller is present or contact the manufacturer.

-1006: ECAT_ERR_MASTER_BASE_INIT_FAIL

Description
Initialization of EtherCAT MDevice command interface failed.
Corrective Actions
Please contact the manufacturer.

-1007: ECAT_ERR_MASTER_CIA402_INIT_FAIL

Description
Initialization of EtherCAT MDevice CiA 402 framework failed.
Corrective Actions
Please contact the manufacturer.

-1008: ECAT_ERR_MASTER_SETUP_PDO_FAIL

Description
Configuration of PDO mapping for the SubDevice using SDO Download failed.
Corrective Actions
Please ensure that all SubDevices are functioning properly. If the issue is 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 SubDevices are powered on, or verify the presence of the network controller.

-1010: ECAT_ERR_MASTER_START_MASTER_FAIL

Description
Failed to start EtherCAT MDevice.
Corrective Actions
Please ensure that the configuration of PDO mapping for all SubDevices is correct. If the issue is persisted, please contact the manufacturer.

-1011: ECAT_ERR_MASTER_CYCLETIME_TOO_SMALL

Description
The input cycle time is too small.
Corrective Actions
Please try increasing the cycle time.

-1012: ECAT_ERR_MASTER_DUMP_OUTPUT_PDO_FAIL

Description
Failed to update output process data using SDO Upload.
Corrective Actions
Please ensure that all SubDevices are functioning correctly, and verify that the configuration of PDO mapping is correct.

-1013: ECAT_ERR_MASTER_CONFIG_DEVICE_FAIL

Description
EtherCAT SubDevice initialization failed.
Corrective Actions
Please contact the manufacturer.

-1014: ECAT_ERR_MASTER_CONFIG_MAPPING_FAIL

Description
Failed to create PDO mapping.
Corrective Actions
Please contact the manufacturer.

-1015: ECAT_ERR_MASTER_WAIT_BUS_SYNC_TIMEOUT

Description
Synchronization timeout for all SubDevices.
Corrective Actions
Please contact the manufacturer.

-1016: ECAT_ERR_MASTER_WAIT_MASTER_SYNC_TIMEOUT

Description
Synchronization timeout between MDevice and SubDevice.
Corrective Actions
Please contact the manufacturer.

-1017: ECAT_ERR_MASTER_CYCLIC_START_FAIL

Description
Failed to start cyclic transmission.
Corrective Actions
Please contact the manufacturer.

-1018: ECAT_ERR_MASTER_WRONG_BUFFER_POINTER

Description
Incorrect data buffer pointer.
Corrective Actions
Please input the correct data buffer pointer.

-1050: ECAT_ERR_MASTER_ENI_INIT_FAIL

Description
ENI initialization failed.
Corrective Actions
Please confirm that the specified ENI file exists. If this issue persists, please contact the manufacturer.

-1051: ECAT_ERR_MASTER_ENI_MISMATCH

Description
The SubDevice information in the ENI file does not match the scanned EtherCAT network SubDevices.
Corrective Actions
Please adjust the order of the SubDevices on the EtherCAT network or configure the SubDevice identifications.

-1100: ECAT_ERR_MASTER_STOPPED

Description
EtherCAT MDevice has not been started.
Corrective Actions
Please call `EthercatMaster::start()` or avoid calling this function.

-1101: ECAT_ERR_MASTER_STARTED

Description
EtherCAT MDevice has already been started.
Corrective Actions
Please call `EthercatMaster::stop()` or avoid calling this function.

-1102: ECAT_ERR_MASTER_NOT_IN_PREOP

Description
EtherCAT MDevice is not in PRE-OP state.
Corrective Actions
Please ensure that all SubDevices are in PRE-OP state before calling this function.

-1103: ECAT_ERR_MASTER_NOT_IN_SAFEOP

Description
EtherCAT MDevice is not in SAFE-OP state.
Corrective Actions
Please ensure that all SubDevices are in SAFE-OP state before calling this function.

-1104: ECAT_ERR_MASTER_NOT_IN_OP

Description
EtherCAT MDevice is not in OP state.
Corrective Actions
Please ensure that all SubDevices are in OP state before calling this function.

-1200: ECAT_ERR_MASTER_II_TRANSITION_FAIL

Description
Switching all SubDevices to INIT state during EtherCAT MDevice initialization failed.
Corrective Actions
Please power cycle all SubDevices and then run the EtherCAT MDevice program again.

-1201: ECAT_ERR_MASTER_IP_TRANSITION_FAIL

Description
Failed to transition EtherCAT state from INIT to PRE-OP.
Corrective Actions
Please ensure that all SubDevices are functioning properly, or check the quality of the network connections.

-1202: ECAT_ERR_MASTER_PS_TRANSITION_FAIL

Description
Failed to transition EtherCAT state from PRE-OP to SAFE-OP.
Corrective Actions
Please check the correctness of the PDO mapping configuration for all SubDevices. If DC synchronization is enabled, try adjusting the related parameters for DC synchronization.

-1203: ECAT_ERR_MASTER_SO_TRANSITION_FAIL

Description
Failed to transition EtherCAT state from SAFE-OP to OP.
Corrective Actions
Please try adjusting the related parameters for DC synchronization and test again.

-2000: ECAT_ERR_DEVICE_NOT_EXIST

Description
The specified SubDevice does not exist.
Corrective Actions
Please do not access the specified SubDevice.

-2001: ECAT_ERR_DEVICE_NOT_ATTACH

Description
The object of such SubDevice has not been initialized.
Corrective Actions
Please call `attach()` to initialize the SubDevice object.

-2002: ECAT_ERR_DEVICE_NO_MAILBOX

Description
The SubDevice does not support Mailbox.
Corrective Actions
Please do not call Mailbox-related functions for the SubDevice.

-2003: ECAT_ERR_DEVICE_NO_DC

Description
The SubDevice does not support DC synchronization.
Corrective Actions
Please do not call DC-related functions for the SubDevice.

-2004: ECAT_ERR_DEVICE_WRONG_INPUT

Description
Incorrect input parameter.
Corrective Actions
Please input the correct parameter.

-2005: ECAT_ERR_DEVICE_MEMORY_ALLOCATION_FAIL

Description
Failed to allocate memory for the SubDevice object.
Corrective Actions
Please contact the manufacturer.

-2006: ECAT_ERR_DEVICE_VENDOR_ID_MISMATCH

Description
The Vendor ID of the SubDevice does not match that of the SubDevice object.
Corrective Actions
Please use the correct SubDevice object.

-2007: ECAT_ERR_DEVICE_PRODUCT_CODE_MISMATCH

Description
The Product Code of the SubDevice does not match that of the SubDevice object.
Corrective Actions
Please use the correct SubDevice object.

-2008: ECAT_ERR_DEVICE_NO_SUCH_FUNCTION

Description
The SubDevice does not support this feature.
Corrective Actions
Please do not call this function.

-2009: ECAT_ERR_DEVICE_FUNCTION_NOT_INIT

Description
The specific function of the SubDevice has not been initialized.
Corrective Actions
Please initialize that function.

-2010: ECAT_ERR_DEVICE_BUSY

Description
The SubDevice is busy.
Corrective Actions
Please try again later.

-2011: ECAT_ERR_DEVICE_TIMEOUT

Description
The SubDevice has encountered a timeout.
Corrective Actions
Please try again later or confirm that the SubDevice is functioning properly.

-2012: ECAT_ERR_DEVICE_NO_DATA

Description
The SubDevice has no available data.
Corrective Actions
Please try again later.

-2100: ECAT_ERR_DEVICE_SII_READ_FAIL

Description
Failed to read the SII EEPROM of the SubDevice.
Corrective Actions
Please ensure that this function is called when the EtherCAT state of the SubDevice is INIT or PRE-OP.

-2101: ECAT_ERR_DEVICE_SII_WRITE_FAIL

Description
Failed to write the SII EEPROM of the SubDevice.
Corrective Actions
Please ensure that this function is called when the EtherCAT state of the SubDevice is INIT or PRE-OP.

-2200: ECAT_ERR_DEVICE_PDO_NOT_EXIST

Description
The SubDevice has no output process data.
Corrective Actions
Please do not call this function.

-2201: ECAT_ERR_DEVICE_PDO_OUT_OF_RANGE

Description
Accessing beyond the process data range of the SubDevice.
Corrective Actions
Please access the correct range of process data for the SubDevice.

-2300: ECAT_ERR_DEVICE_FOE_NOT_SUPPORT

Description
The SubDevice does not support FoE.
Corrective Actions
Please do not call FoE-related functions for the SubDevice.

-2310: ECAT_ERR_DEVICE_FOE_REQUEST_FAIL

Description
Failed to request FoE communication.
Corrective Actions
Please try again later.

-2311: ECAT_ERR_DEVICE_FOE_TIMEOUT

Description
FoE communication timeout.
Corrective Actions
Please verify that the SubDevice supports FoE and is functioning properly.

-2312: ECAT_ERR_DEVICE_FOE_ERROR

Description
FoE communication encountered an error.
Corrective Actions
Please verify that the SubDevice supports FoE and is functioning properly.

-2313: ECAT_ERR_DEVICE_FOE_BUFFER_TOO_SMALL

Description
The size of the input buffer for FoE communication is too small.
Corrective Actions
Please input a buffer with suitable size.

-2314: ECAT_ERR_DEVICE_FOE_READ_FAIL

Description
Failed to read the file via FoE communication.
Corrective Actions
Please verify that the SubDevice supports reading files via FoE communication.

-2315: ECAT_ERR_DEVICE_FOE_WRITE_FAIL

Description
Failed to write the file via FoE communication.
Corrective Actions
Please verify that the SubDevice supports writing files via FoE communication.

-2400: ECAT_ERR_DEVICE_COE_SDO_NOT_SUPPORT

Description
The SubDevice does not support SDO commands of CoE communication.
Corrective Actions
Please do not call functions related to SDO commands of CoE communication.

-2401: ECAT_ERR_DEVICE_COE_SDO_INFO_NOT_SUPPORT

Description
The SubDevice does not support SDO Information commands of CoE communication.
Corrective Actions
Please do not call functions related to SDO Information commands of CoE communication.

-2410: ECAT_ERR_DEVICE_COE_BUSY

Description
CoE communication is busy.
Corrective Actions
Please try again later.

-2411: ECAT_ERR_DEVICE_COE_REQUEST_FAIL

Description
Failed to request CoE communication.
Corrective Actions
Please try again later.

-2412: ECAT_ERR_DEVICE_COE_TIMEOUT

Description
CoE communication timeout.
Corrective Actions
Please try again later or confirm that the SubDevice is functioning properly.

-2413: ECAT_ERR_DEVICE_COE_ERROR

Description
CoE communication encountered an error.
Corrective Actions
Please check the abort code for CoE communication.

-2500: ECAT_ERR_DEVICE_CIA402_NOT_EXIST

Description
This SubDevice does not have objects related to CiA 402.
Corrective Actions
Please do not call functions related to CiA 402 for the SubDevice.

-2501: ECAT_ERR_DEVICE_CIA402_ADD_FAIL

Description
Failed to insert the CiA 402 SubDevice object to the CiA 402 framework of EtherCAT MDevice.
Corrective Actions
Please confirm whether the CiA 402 SubDevice object has been successfully inserted into the CiA 402 framework of the EtherCAT MDevice.

-2502: ECAT_ERR_DEVICE_CIA402_TYPE_MISMATCH

Description

The content of the object of Device Type (Index 1000h) for the SubDevice is not CiA 402 type.

Corrective Actions

Please do not call functions related to CiA 402 for the SubDevice.

-2503: ECAT_ERR_DEVICE_CIA402_NO_MODE_SUPPORT

Description

This CiA 402 SubDevice does not support any operation modes defined by CiA 402.

Corrective Actions

Please do not call functions related to CiA 402 for the SubDevice. Also, ensure that the CiA 402 SubDevice supports CiA 402.

-2504: ECAT_ERR_DEVICE_CIA402_WRONG_MODE

Description

This function cannot be called in the current CiA 402 operation mode for the SubDevice.

Corrective Actions

Please change the CiA 402 operation mode of this SubDevice to the correct mode before calling this function.

-2505: ECAT_ERR_DEVICE_CIA402_MODE_NOT_SUPPORT

Description

The SubDevice does not support the specified CiA 402 operation mode.

Corrective Actions

Please do not change the CiA 402 operation mode of this SubDevice to an unsupported mode.

-2506: ECAT_ERR_DEVICE_CIA402_CHANGE_WRONG_STATE

Description

The current CiA 402 state does not allow switching to the specified CiA 402 state.

Corrective Actions

Please check the current CiA 402 state. If it is in FAULT state, switch to SWITCH_ON_DISABLED state first, and then switch to the target state.

-2507: ECAT_ERR_DEVICE_CIA402_WRITE_OBJECT_FAIL

Description

Accessing CiA 402 objects is not allowed using SDO Upload or SDO Download in Cyclic Callback.

Corrective Actions

Please avoid using SDO Upload/Download to access CiA 402 objects in Cyclic Callback. If needed, map the CiA 402 objects to process data.

-2580: ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE

Description

The input number for the Touch Probe functionality is incorrect.

Corrective Actions

Please input the correct Touch Probe number, ranging from 0 to 1.

-2581: ECAT_ERR_DEVICE_CIA402_NO_SUCH_TOUCH_PROBE_SOURCE

Description

The input signal source for the Touch Probe functionality is incorrect.

Corrective Actions

Please input the correct signal source, ranging from 0 to 2.

-2600: ECAT_ERR_DEVICE_EOE_NOT_SUPPORT

Description
The SubDevice does not support EoE.
Corrective Actions
Please do not call EoE-related functions for the SubDevice.

-2601: ECAT_ERR_DEVICE_EOE_NO_SUCH_PORT

Description
Incorrect EoE port number.
Corrective Actions
Please input the correct EoE port number.

-2602: ECAT_ERR_DEVICE_EOE_TOO_MUCH_CONTENT

Description
The input content is too much.
Corrective Actions
Please provide the correct content.

-2610: ECAT_ERR_DEVICE_EOE_BUSY

Description
EoE communication is busy.
Corrective Actions
Please try again later.

-2611: ECAT_ERR_DEVICE_EOE_REQUEST_FAIL

Description
Failed to request EoE communication.
Corrective Actions
Please try again later.

-2612: ECAT_ERR_DEVICE_EOE_TIMEOUT

Description

EoE communication timeout.

Corrective Actions

Please verify that the SubDevice supports EoE and is functioning properly.

-3000: ECAT_ERR_GROUP_WRONG_INPUT

Description

The input parameter is incorrect.

Corrective Actions

Please input the correct parameter.

-3001: ECAT_ERR_GROUP_NOT_ATTACH

Description

The object of such group has not been initialized.

Corrective Actions

Please initialize the group object.

A.3 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.4 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.5 About CiA DSP 402

The CiA DSP 402 represents the standardized CANopen device profile for digital controlled motion products like servo controllers, frequency converters or stepper motors.

All the devices mentioned above use communication techniques which conform to those described in the CiA Draft Standard DS 301 (CANopen Application Layer and Communication Profile). This document should be consulted in parallel to CiA® 402-CANopen Drives and Motion Control Profile.

REFERENCES

1. ISO 7498, 1984, Information Processing Systems - Open Systems Interconnection – Basic Reference Model
2. ISO 11898-1, 1999, Road Vehicles, Interchange of Digital Information - Controller Area Network (CAN) for high-speed Communication
3. CiA DS 301, CANopen Application Layer and Communication Profile, Version 4.02, February 2002
4. CiA DS 401, CANopen Device Profile I/O Modules, Version 2.1, May 2002
5. DRIVECOM Profil Antriebstechnik/Profil 21
6. DRIVECOM Profil Antriebstechnik/Servo 22, Jan. 1994

DEFINITIONS AND ABBREVIATION

Abbr.	Definition
CAN	Controller Area Network
CiA	CAN in Automation e. V.
COB	Communication Object (CAN message). A unit of transportation in a CAN network. Data must be sent across a network inside a COB.
COB-ID	COB-Identifier. Identifies a COB uniquely in a network. The identifier determines the priority of that COB in the MAC sub-layer too.
PDO	Process Data Object. Object for data exchange between several devices.
SDO	Service Data Object. Peer-to-peer communication with access to the object dictionary of a device.
pp	Profile Position Mode
pv	Profile Velocity Mode
vl	Velocity Mode
hm	Homing Mode
ip	Interpolated Position Mode
tq	Profile Torque Mode
all	Mandatory for all modes
ce	Common entries in the object dictionary
dc	Device Control
pc	Position Control Function

A.6 Object Dictionary Entries

All information follows the CiA® 402-CANopen Drives and Motion Control Profile.

Object $603F_h$: Error code

The Error code captures the code of the last error that occurred in the drive. It corresponds to the value of the lower 16 bits of object 1003_h pre-defined error field.

The device specific bit in the error register is used by the CANopen Device Profile for Drives and Motion Control. The error code can be read from the predefined error field at object 1003h and to be compatible with device profiles for drives available for other field bus systems from object $603F_h$ as well.

OBJECT DESCRIPTION:

INDEX	$607C_h$
Name	Home offset
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	0

Object 6040_h: Controlword

The controlword consist of bits for the controlling of the state, the controlling of operating modes and manufacturer specific options.

OBJECT DESCRIPTION:

INDEX	6040_h
Name	Controlword
Object Code	VAR
Data Type	UNSIGNED16
Category	Mandatory

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

DATA DESCRIPTION:

The bits of the controlword are defined as follows.

15 11	10 9	8	7	6 4	3	2	1	0
manufacturer specific	reserved	halt	Fault reset	Operation mode specific	Enable operation	Quick stop	Enable voltage	Switch on
O	O	O	M	O	M	M	M	M

MSB LSB

* O: Optional; M: Mandatory

BITS 0 – 3 AND 7:

Device control commands are triggered by the following bit patterns in the controlword:

Command	Bit of the controlword					Transitions
	Fault reset	Enable operation	Quick stop	Enable voltage	Switch on	
Shutdown	0	X	1	1	0	2, 6, 8
Switch on	0	0	1	1	1	3*
Switch on	0	1	1	1	1	3**
Disable voltage	0	X	X	0	X	7, 9, 10, 12
Quick stop	0	X	0	1	X	7, 10, 11
Disable operation	0	0	1	1	1	5
Enable operation	0	1	1	1	1	4, 16
Fault reset	-	X	X	X	X	15

Device control commands (bits marked X are irrelevant, * ... In the state SWITCHED ON the drive executes the functionality of this state., ** ... It exists no functionality in the state SWITCHED ON. The drive does not do any in this state.)

BITS 4, 5, 6 AND 8:

These bits are operation mode specific. The description is situated in the chapter of the special mode. The following table gives an overview:

Bit	Operation mode					
	Velocity mode	Profile position mode	Profile velocity mode	Profile torque mode	Homing mode	Interpolation position mode
4	rfg enable	New set-point	reserved	reserved	Homing operation start	Enable ip mode
5	rfg unlock	Change set immediately	reserved	reserved	reserved	reserved
6	rfg use ref	abs / rel	reserved	reserved	reserved	reserved
8	Halt	Halt	Halt	Halt	Halt	Halt

BITS 9, 10:

These bits are reserved for further use. They are inactive by setting to zero. If they have no special function, they must be set to zero.

BITS 11, 12, 13, 14 AND 15:

These bits are manufacturer specific.

Object 6041_h: Statusword

The statusword indicates the current state of the drive. No bits are latched. The statusword consist of bits for the current state of the drive, the operating state of the mode and manufacturer specific options.

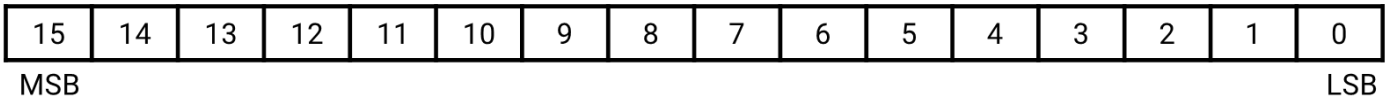
OBJECT DESCRIPTION:

INDEX	6041_h
Name	Statusword
Object Code	VAR
Data Type	UNSIGNED16
Category	Mandatory

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

DATA DESCRIPTION:



Bits in the statusword:

Bit	Description	M/O
0	Ready to switch on	M
1	Switched on	M
2	Operation enabled	M
3	Fault	M
4	Voltage enabled	M
5	Quick stop	M
6	Switch on disabled	M
7	Warning	O
8	Manufacturer specific	O
9	Remote	M
10	Target reached	M
11	Internal limit active	M
12 – 13	Operation mode specific	O
14 – 15	Manufacturer specific	O

* M = Mandatory; O = Optional.

BITS 0 – 3, 5 AND 6:

The following bits indicate the status of the device:

Value (binary)	State
xxxx xxxx x0xx 0000	Not ready to switch on
xxxx xxxx x1xx 0000	Switch on disabled
xxxx xxxx x01x 0001	Ready to switch on
xxxx xxxx x01x 0011	Switched on
xxxx xxxx x01x 0111	Operation enabled
xxxx xxxx x00x 0111	Quick stop active
xxxx xxxx x0xx 1111	Fault reaction active
xxxx xxxx x0xx 1000	Fault

Device state bits (x ... irrelevant for this state).

BIT 4: VOLTAGE ENABLED

High voltage is applied to the drive when this bit is set to 1.

BIT 5: QUICK STOP

When reset, this bit indicates that the drive is reacting on a quick stop request. Bits 0, 1 and 2 of the *statusword* must be set to 1 to indicate that the drive is capable to regenerate. The setting of the other bits indicates the status of the drive (e.g. the drive is performing a quick stop as result of a reaction to a non-fatal fault. The fault bit is set as well as bits 0, 1 and 2).

BIT 7: WARNING

A drive warning is present if bit 7 is set. The cause means no error but a state that has to be mentioned, e.g. temperature limit, job refused. The status of the drive does not change. The cause of this warning may be found by reading the fault code parameter. The bit is set and reset by the device.

BIT 8:

This bit may be used by a drive manufacturer to implement any manufacturer specific functionality.

BIT 9: REMOTE

If bit 9 is set, then parameters may be modified via the CAN-network, and the drive executes the content of a command message. If the bit remote is reset, then the drive is in local mode and will not execute the command message. The drive may transmit messages containing valid actual values like a *position actual value*, depending on the actual drive configuration. The drive will accept accesses via SDO in local mode.

BIT 10: TARGET REACHED

If bit 10 is set by the drive, then a set-point has been reached. The set-point is dependent on the operating mode. The description is situated in the chapter of the special mode. The change of a target value by software alters this bit.
If *quick stop option code* is 5, 6, 7 or 8, this bit must be set, when the quick stop operation is finished and the drive is halted.
If halt occurred and the drive has halted then this bit is set too.

BIT 11: INTERNAL LIMIT ACTIVE

This bit set by the drive indicates, that an internal limitation is active (e.g. *position range limit*).

BIT 12 AND 13:

These bits are operation mode specific. The description is situated in the chapter of the special mode. The following table gives an overview:

Bit	Operation mode					
	vl	**pp**	**pv**	**tq**	**hm**	**Ip**
12	reserved	Set-point acknowledge	Speed	reserved	Homing attained	ip mode active
13	reserved	Following error	Max slippage error	reserved	Homing error	reserved

Mode specific bits in the *statusword*.

BIT 14 AND 15:

These bits may be used by a drive manufacturer to implement any manufacturer specific functionality.

Object 605A$_h$: Quick stop option code

The parameter quick stop option code determines what action should be taken if the Quick Stop Function is executed.

OBJECT DESCRIPTION:

INDEX	605A$_h$
Name	Quick stop option code
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	No
Value Range	INTEGER16
Default Value	2

DATA DESCRIPTION:

Value	**Description**
-32768 … -1	Manufacturer specific
0	disable drive function
1	slow down on slow down ramp
2	slow down on quick stop ramp
3	slow down on the current limit
4	slow down on the voltage limit
5	slow down on slow down ramp and stay in QUICK STOP
6	slow down on quick stop ramp and stay in QUICK STOP
7	slow down on the current limit and stay in QUICK STOP
8	slow down on the voltage limit and stay in QUICK STOP
9 ... 32767	Reserved

Object $605B_h$: Shutdown option code

The parameter *shutdown option code* determines what action should be taken if there is a transition in **OPERATION ENABLE ⇒ READY TO SWITCH ON**.

OBJECT DESCRIPTION:

INDEX	$605B_h$
Name	Shutdown option code
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	No
Value Range	INTEGER16
Default Value	0

DATA DESCRIPTION:

Value	Description
-32768 … -1	Manufacturer specific
0	Disable drive function
1	Slow down with slow down ramp; disable of the drive function
2 ... 32767	Reserved

Object 605C$_h$: Disable operation option code

The parameter disable operation option code determines what action should be taken if there is a transition in **OPERATION ENABLE ⇒ SWITCHED ON**.

OBJECT DESCRIPTION:

INDEX	605C$_h$
Name	Disable operation option code
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	No
Value Range	INTEGER16
Default Value	1

DATA DESCRIPTION:

Value	Description
-32768 … -1	Manufacturer specific
0	Disable drive function
1	Slow down with slow down ramp and then disabling of the drive function
2 ... 32767	Reserved

Object 605D$_h$: Halt option code

The parameter halt option code determines what action should be taken if the bit 8 (halt) in the controlword is active.

OBJECT DESCRIPTION:

INDEX	605D$_h$
Name	Halt option code
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	No
Value Range	INTEGER16
Default Value	1

DATA DESCRIPTION:

Value	Description
-32768 … -1	Manufacturer specific modes of operation.
0	disable drive, motor is free to rotate
1	slow down on slow down ramp
2	slow down on quick stop ramp
3	slow down on the current limit
4	slow down on the voltage limit
5 ... 32767	Reserved

Object 605E$_h$: Fault reaction option code

The parameter *fault reaction option code* determines what action should be taken if a fault occurs in the drive.

OBJECT DESCRIPTION:

INDEX	605E$_h$
Name	Fault reaction option code
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	No
Value Range	INTEGER16
Default Value	2

DATA DESCRIPTION:

Value	Description
-32768 … -1	Manufacturer specific modes of operation.
0	disable drive, motor is free to rotate
1	slow down on slow down ramp
2	slow down on quick stop ramp
3	slow down on the current limit
4	slow down on the voltage limit
5 ... 32767	Reserved

Object 6060_h: Modes of operation

The parameter modes of operation switches the actually choosen operation mode.

OBJECT DESCRIPTION:

INDEX	6060_h
Name	Modes of operation
Object Code	VAR
Data Type	INTEGER8
Category	Mandatory

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER8
Default Value	No

DATA DESCRIPTION:

Value	Description
-1 … -128	Manufacturer specific modes of operation.
0	Reserved
1	Profile Position Mode
2	Velocity Mode
3	Profile Velocity Mode
4	Torque Profile Mode
5	Reserved
6	Homing Mode
7	Interpolated Position Mode
8 … 127	Reserved

Note:
A read of *modes of operation* shows only the value of *modes of operation*. The actual mode of the drive is reflected in the object *modes of operation display*. It may be changed by writing to *modes of operation.*

Object 6061_h: Modes of operation display

The modes of operation display shows the current mode of operation. The meaning of the returned value corresponds to that of the modes of operation option code (index 6060_h).

OBJECT DESCRIPTION:

INDEX	6061_h
Name	Modes of operation display
Object Code	VAR
Data Type	INTEGER8
Category	Mandatory

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER8
Default Value	No

DATA DESCRIPTION:
Same as for object 6060_h *modes of operation*.

Note:
The actual mode is reflected in the *modes of operation display* (index 6061h), and not in the m*odes of operation* (index 6060h).

Object 6062_h: Position demand value

The position demand value is given in position units.

OBJECT DESCRIPTION:

INDEX	6062_h
Name	Position demand value
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 6063_h: Position actual value*

The actual value of the position measurement device is one of the two input values of the closed loop position control. The data unit is defined as increments. If necessary, the data unit must be transformed with the *position factor* defined in chapter 11 from user defined units to increments.

OBJECT DESCRIPTION:

INDEX	6063_h
Name	Position actual value*
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 6064_h: Position actual value

This object represents the actual value of the position measurement device in user defined units.

OBJECT DESCRIPTION:

INDEX	6064_h
Name	Position actual value
Object Code	VAR
Data Type	INTEGER32
Category	Conditional; Mandatory, if pc supported Optional, if pp, ip, hm or tq supported

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 6065_h: Following error window

The following error window defines a range of tolerated position values symmetrically to the position demand value. As it is in most cases used with user defined units, a transformation into increments with the position factor is necessary. If the position actual value is out of the following error window, a following error occurs.

A following error might occur when a drive is blocked, unreachable profile velocity occurs, or at wrong closed loop coefficients.
If the value of the following error window is 2^{32}-1, the following control is switched off.

OBJECT DESCRIPTION:

INDEX	6065_h
Name	Following error window
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6067_h: Position window

The position window defines a symmetrical range of accepted positions relatively to the target position.
If the actual value of the position encoder is within the position window, this target position is regarded as reached. As the user mostly prefers to specify the position window in his application in user defined units, the position factor of chapter 11 must be used to transform this value into increments. The target position has to be handled in the same manner as in the Trajectory Generator concerning limiting functions and transformation into internal machine units before it can be used with this function.
If the value of the position window is 2^{32}-1, the position window control is switched off.

OBJECT DESCRIPTION:

INDEX	6067_h
Name	Position window
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6068_h: Position window time

When the actual position is within the position window during the defined position window time which is given in multiples of milliseconds, the corresponding bit 10 target reached in the statusword will be set to one.

OBJECT DESCRIPTION:

INDEX	6068_h
Name	Position window time
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

Object $606B_h$: Velocity demand value

The output value of the trajectory generator may be corrected by the output value of the position control function. It is then provided as a demand value for the velocity controller and given in the velocity units.

OBJECT DESCRIPTION:

INDEX	$606B_h$
Name	Velocity demand value
Object Code	VAR
Data Type	INTEGER32
Category	Conditional; Mandatory, if pv supported

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 606C$_h$: Velocity actual value

The velocity actual value is also represented in velocity units and is coupled to the velocity used as input to the velocity controller.

OBJECT DESCRIPTION:

INDEX	606C$_h$
Name	Velocity actual value
Object Code	VAR
Data Type	INTEGER32
Category	Conditional; Mandatory, if pv supported

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 606D$_h$: Velocity window

The *velocity window* monitors whether the required process velocity has been achieved after an eventual acceleration or deceleration (braking) phase. It is given in velocity units.

OBJECT DESCRIPTION:

INDEX	606D$_h$
Name	Velocity window
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

Object 606E$_h$: Velocity window time

The corresponding bit 10 target reached is set in the statusword when the difference between the target velocity and the velocity actual value is within the velocity window longer than the velocity window time. The value of the velocity window time is given in multiples of milliseconds.

OBJECT DESCRIPTION:

INDEX	606E$_h$
Name	Velocity window time
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

Object 606F$_h$: Velocity threshold

As soon as the velocity actual value exceeds the velocity threshold longer than the velocity threshold time bit 12 velocity = 0 is reset in the statusword. Below this threshold the bit is set and indicates that the axle is stationary. The value is given in velocity units.

OBJECT DESCRIPTION:

INDEX	606F$_h$
Name	Velocity threshold
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

Object 6071_h: Target torque

This parameter is the input value for the torque controller in profile torque mode and the value is given per thousand of rated torque.

OBJECT DESCRIPTION:

INDEX	$607C_h$
Name	Home offset
Object Code	VAR
Data Type	INTEGER16
Category	Conditional; Mandatory, if tq supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object 6072_h: Max torque

This value represents the maximum permissible torque in the motor and is given per thousand of rated torque.

OBJECT DESCRIPTION:

INDEX	6072_h
Name	Max torque
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	0

Object 6074_h: Torque demand value

This parameter is the output value of the torque limit function (if the torque control and power-stage function are available). The value is given per thousand of rated torque.

OBJECT DESCRIPTION:

INDEX	6074_h
Name	Torque demand value
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object 6075_h: Motor rated current

This value is taken from the motor name plate and is entered as multiples of milliamp. Depending on the motor and drive technology this current may be either DC, peak or rms (root-mean-square) current. All relative current data refers to this value.

OBJECT DESCRIPTION:

INDEX	6075_h
Name	Motor rated current
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

Object 6076_h: Motor rated torque

This value is taken from the motor name plate and is entered as multiples of mNm (mill Newtonmeter). All relative torque data refer to this value.
For linear motors, the object name is not changed, but the motor rated force value must be entered as multiples of mN (mill Newton).

OBJECT DESCRIPTION:

INDEX	6076_h
Name	Motor rated torque
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

Object 6077_h: Torque actual value

The *torque actual value* corresponds to the instantaneous torque in the drive motor. The value is given per thousand of rated torque.

OBJECT DESCRIPTION:

INDEX	6077_h
Name	Torque actual value
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object 6078_h: Current actual value

The *current actual value* refers to the instantaneous current in the drive motor. The value is given per thousand of rated current.

OBJECT DESCRIPTION:

INDEX	6078_h
Name	Current actual value
Object Code	VAR
Data Type	INTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object $607A_h$: Target position

The target position is the position that the drive should move to in position profile mode using the current settings of motion control parameters such as velocity, acceleration, deceleration, motion profile type etc. The target position is given in user defined position units. It is converted to position increments using the position factor (see chapter 11). The target position will be interpreted as absolute or relative depending on the “abs / rel” flag in the controlword.

OBJECT DESCRIPTION:

INDEX	$607A_h$
Name	Target position
Object Code	VAR
Data Type	INTEGER32
Category	Conditional; Mandatory, if pp or pc supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 607C$_h$: Home offset

The home offset object is the difference between the zero position for the application and the machine home position (found during homing), it is measured in position units. During homing the machine home position is found and once the homing is completed the zero position is offset from the home position by adding the home offset to the home position. All subsequent absolute moves shall be taken relative to this new zero position.

This is illustrated in the following diagram.

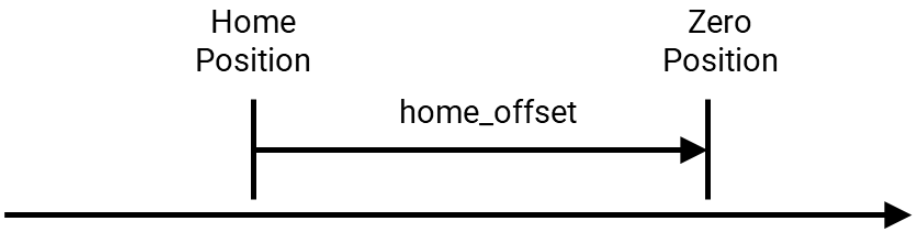


Figure 12: Home offset

If the home offset is not implemented then it shall be zero.

OBJECT DESCRIPTION:

INDEX	607C$_h$
Name	Home offset
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	0

Object $607D_h$: Software position limit

Software position limit contains the sub-parameters min position limit and max position limit. These parameters define the absolute position limits for the position demand value and the position actual value. Every new target position must be checked against these limits. The limit positions are specified in position units (same as target position) and are always relative to the machine home position.

Before being compared with the target position they must be corrected internally by the home offset as follows:

corrected min position limit = min position limit - home offset
corrected max position limit = max position limit - home offset

This calculation needs only be performed when *home offset* or *software position limit* is changed.

OBJECT DESCRIPTION:

INDEX	$607D_h$
Name	Software position limit
Object Code	ARRAY
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Sub-Index	0
Description	Number of entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	2
Default Value	2

Sub-Index	1
Description	Min position limit
Entry Category	Mandatory
Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	-2^{31}

Sub-Index	2
Description	Max position limit
Entry Category	Mandatory
Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	2^{31}-1

Object 607E$_h$: Polarity

Position demand value and position actual value are multiplied by 1 or -1 depending on the value of the polarity flag.

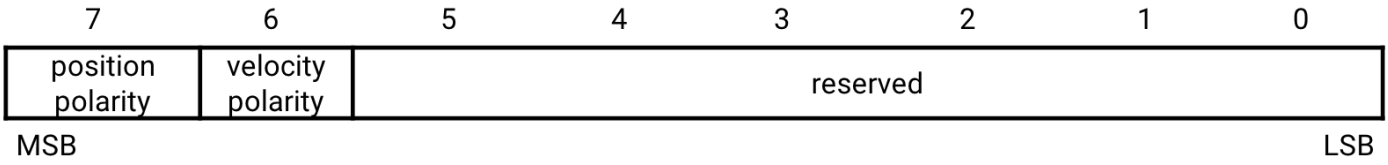
OBJECT DESCRIPTION:

INDEX	607E$_h$
Name	Polarity
Object Code	VAR
Data Type	UNSIGNED8
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED8
Default Value	0

DATA DESCRIPTION:



Value	Description
0	Multiply by 1
1	Multiply by -1

Object 607F$_h$: Max profile velocity

The *max profile velocity* is the maximum allowed speed in either direction during a profiled move. It is given in the same units as *profile velocity*.

OBJECT DESCRIPTION:

INDEX	607F$_h$
Name	Max profile velocity
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6080_h: Max motor speed

The *max motor speed* is the maximum allowable speed for the motor in either direction and is given in rpm. This is used to protect the motor and can be taken from the motor data sheet.

OBJECT DESCRIPTION:

INDEX	6080_h
Name	Max motor speed
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6081_h: Profile velocity

The profile velocity is the velocity normally attained at the end of the acceleration ramp during a profiled move and is valid for both directions of motion. The profile velocity is given in user defined speed units. It is converted to position increments per second using the velocity encoder factor.

OBJECT DESCRIPTION:

INDEX	6081_h
Name	Profile velocity
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory, if pp or pv supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6083_h: Profile acceleration

The profile acceleration is given in user defined acceleration units. It is converted to position increments per second2 using the normalizing factors.

OBJECT DESCRIPTION:

INDEX	6083_h
Name	Profile acceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory, if pp or pv supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6084_h: Profile deceleration

The profile deceleration is given in the same units as profile acceleration. If this parameter is not supported, then the profile acceleration value is also used for deceleration.

OBJECT DESCRIPTION:

INDEX	6084_h
Name	Profile deceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6085_h: Quick stop deceleration

The quick stop deceleration is the deceleration used to stop the motor if the 'Quick Stop' command is given and the quick stop option code (see 605Ah) is set to 2. The quick stop deceleration is given in the same units as the profile acceleration.

OBJECT DESCRIPTION:

INDEX	6085_h
Name	Quick stop deceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object 6086_h: Motion profile type

The motion profile type is used to select the type of motion profile used to perform a profiled move.

Value	Description
-32768 … -1	Manufacturer specific
0	Linear ramp (trapezoidal profile)
1	sin^2 ramp
2	Jerk-free ramp
3	Jerk-limited ramp
4 … 32767	reserved

OBJECT DESCRIPTION:

INDEX	6086_h
Name	Motion profile type
Object Code	VAR
Data Type	INTEGER16
Category	Conditional; Mandatory, if pp or pv supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object 6087_h: Torque slope

This parameter describes the rate of change of torque in units of per thousand of rated torque per second.

OBJECT DESCRIPTION:

INDEX	6087_h
Name	Torque slope
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory, if tq supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

Object 6088_h: Torque profile type

The *torque profile type* is used to select the type of torque profile used to perform a torque change.

OBJECT DESCRIPTION:

INDEX	6088_h
Name	Torque profile type
Object Code	VAR
Data Type	INTEGER16
Category	Conditional; Mandatory, if tq supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER16
Default Value	0

Object 6098_h: Homing method

The homing method object determines the method that will be used during homing.

OBJECT DESCRIPTION:

INDEX	6098_h
Name	Homing method
Object Code	VAR
Data Type	INTEGER8
Category	Conditional; Mandatory, if hm supported.

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER8
Default Value	0

DATA DESCRIPTION:

Value	Description
-128 .. -1	Manufacturer specific
0	No homing operation required
1 .. 35	Methods 1 to 35 (see the functional description)
36 .. 127	reserved

Object 6099_h: Homing speeds

This entry in the object dictionary defines the speeds used during homing and is given velocity units.

OBJECT DESCRIPTION:

INDEX	6099_h
Name	Homing speeds
Object Code	ARRAY
Data Type	UNSIGNED32
Category	Mandatory, if hm supported

ENTRY DESCRIPTION:

Sub-Index	0
Description	Number of entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	2
Default Value	2

Sub-Index	1
Description	Speed during search for switch
Entry Category	Mandatory
Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

Sub-Index	2
Description	Speed during search for zero
Entry Category	Mandatory
Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

Object 609A$_h$: Homing acceleration

The homing acceleration establishes the acceleration to be used for all accelerations and decelerations with the standard homing modes and is given in acceleration units.

OBJECT DESCRIPTION:

INDEX	609A$_h$
Name	Homing acceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object $60B0_h$: Position offset

This object sets the velocity loop feed forward scaled in CANopen velocity units. The fieldbus controller manages jumps in position command due to this feed-forward when starting, stopping, switching modes, and all other transitions.

OBJECT DESCRIPTION:

INDEX	$60B0_h$
Name	Position Offset
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	0

Object $60B1_h$: Velocity Offset

This object provides the offset of the velocity value in cyclic synchronous position mode.

OBJECT DESCRIPTION:

INDEX	$60B1_h$
Name	Velocity Offset
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	0

Object $60B2_h$: Torque Offset

This object provides the offset of the commanded torque from a bus network connected to the drive. Scaling is 1/1000 of rated torque.

OBJECT DESCRIPTION:

INDEX	$60B2_h$
Name	Torque Offset
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	0

Object $60C5_h$: Max acceleration

To prevent the motor and the application from being destroyed, the *max acceleration* can be used to limit the acceleration to an acceptable value.
The *max acceleration* is given in user defined *acceleration units* ($608D_h$, $608E_h$). It is converted to position increments per second2 using the *acceleration factor* (6097_h).

OBJECT DESCRIPTION:

INDEX	$60C5_h$
Name	Max acceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object $60C6_h$: Max deceleration

To prevent the motor and the application from being destroyed, the *max deceleration* can be used to limit the deceleration to an acceptable value.
The *max deceleration* is given in the same units as the *max acceleration* ($60C5_h$). If this parameter is not supported, then the *max acceleration* value is also used for deceleration.

OBJECT DESCRIPTION:

INDEX	$60C6_h$
Name	Max deceleration
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

Object $60E0_h$: Positive Torque Limit Value

The object gives the configured maximum motor torque in positive direction. The value is given per thousand (1 ‰) of rated torque.

OBJECT DESCRIPTION:

INDEX	$60E0_h$
Name	Positive Torque Limit Value
Object Code	VAR
Data Type	UINTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UINTEGER16
Default Value	0

Object $60E1_h$: Negative Torque Limit Value

The object gives the configured maximum motor torque in negative direction. The value is given per thousand (1 ‰) of rated torque.

OBJECT DESCRIPTION:

INDEX	$60E1_h$
Name	Negative Torque Limit Value
Object Code	VAR
Data Type	UINTEGER16
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UINTEGER16
Default Value	0

Object $60E4_h$: Additional position actual value

This object indicates the actual position in user defined units for additional feedbacks. Each sub-index corresponds to a different feedback device.

Objects 60E8h, 60E9h, 60EDh and 60EEh, 68EEh Additional feed constant - driving shaft revolutions - AxisX are used to scale this value. Object 60F1h, 68F1h Additional encoder inversion - AxisX permits inverting the position provided in the corresponding sub-index of this value.

OBJECT DESCRIPTION:

INDEX	$60E4_h$
Name	Additional position actual value
Object Code	VAR
Data Type	INTEGER32
Category	Optional

Object $60EF_h$: Motor resolution

Motor resolution. This is used to protect the motor and can be taken from the motor data sheet.

OBJECT DESCRIPTION:

INDEX	$60EF_h$
Name	Motor resolution
Object Code	VAR
Data Type	UINTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UINTEGER32
Default Value	No

Object $60F4_h$: Following error actual value

This object represents the actual value of the following error, it is given in user defined position units.

OBJECT DESCRIPTION:

INDEX	$60F4_h$
Name	Following error actual value
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object $60FC_h$: Position demand value*

This output of the trajectory generator in profile position mode is an internal value using increments as unit what is expressed with an *. To save calculation time for some applications, this object is additionally introduced to the *position demand value* (6062_h).

OBJECT DESCRIPTION:

INDEX	$60FC_h$
Name	Position demand value*
Object Code	VAR
Data Type	INTEGER32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 60FD$_h$: *Digital inputs*

This index defines simple digital inputs for drives. The user may apply any signals to these inputs for special purposes like limit or reference switches.

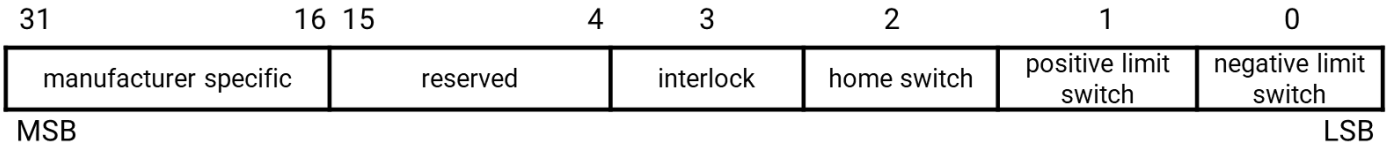
OBJECT DESCRIPTION:

INDEX	60FD$_h$
Name	Digital inputs
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	0

DATA DESCRIPTION:



The switch have to be "active high".

Object 60FF$_h$: Target velocity

The target velocity is the input for the trajectory generator and the value is given in velocity units.

OBJECT DESCRIPTION:

INDEX	60FF$_h$
Name	Target velocity
Object Code	VAR
Data Type	INTEGER32
Category	Conditional; Mandatory, if pv supported

ENTRY DESCRIPTION:

Access	rw
PDO Mapping	Possible
Value Range	INTEGER32
Default Value	No

Object 6502_h: Supported drive modes

A drive can support more than one and several distinct modes of operation. This object gives an overview of the implemented operating modes in the device. This object is read only.

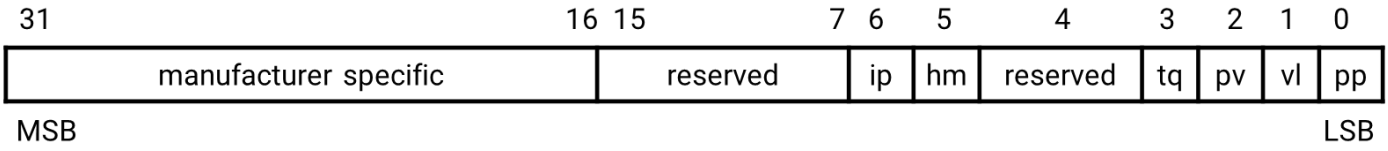
OBJECT DESCRIPTION:

INDEX	6502_h
Name	Supported drive modes
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION:

Access	ro
PDO Mapping	Possible
Value Range	UNSIGNED32
Default Value	No

DATA DESCRIPTION:



A.7 Homing Methods

All information follows the CiA® 402-CANopen Drives and Motion Control Profile.

By choosing a method of homing by writing a value to homing method will clearly establish:

- the homing signal (positive limit switch, negative limit switch, home switch).
- the direction of actuation and where appropriate.
- the position of the index pulse.

The home position and the zero position are offset by the home offset, see the definition of home offset for how this offset is used.

Various homing positions are illustrated in the following diagrams. An encircled number indicates the code for selection of this homing position. The direction of movement is also indicated. Further homing methods may be defined by the manufacturer using the negative values of homing method.

There are four sources of homing signal available, these are the negative and positive limit switches, the home switch and the index pulse from an encoder.

In the diagrams of homing sequences shown below, the encoder count increases as the axle's position moves to the right, in other words the left is the minimum position and the right is the maximum position.

For the operation of positioning drives, an exact knowledge of the absolute position is normally required. Since for cost reasons, drives often do not have an absolute encoder, a homing operation is necessary. There are several, application-specific methods. The homing method is used for selection.

The exact sequence of the homing operation is clearly described by the method. In some circumstances, a device has several methods to choose from, using the homing method.

The following sub-sections describe the details of how each of the homing modes shall function.

Method 1: Homing on the negative limit switch and index pulse

Using this method the initial direction of movement is leftward if the negative limit switch is inactive (here shown as low). The home position is at the first index pulse to the right of the position where the negative limit switch becomes inactive.

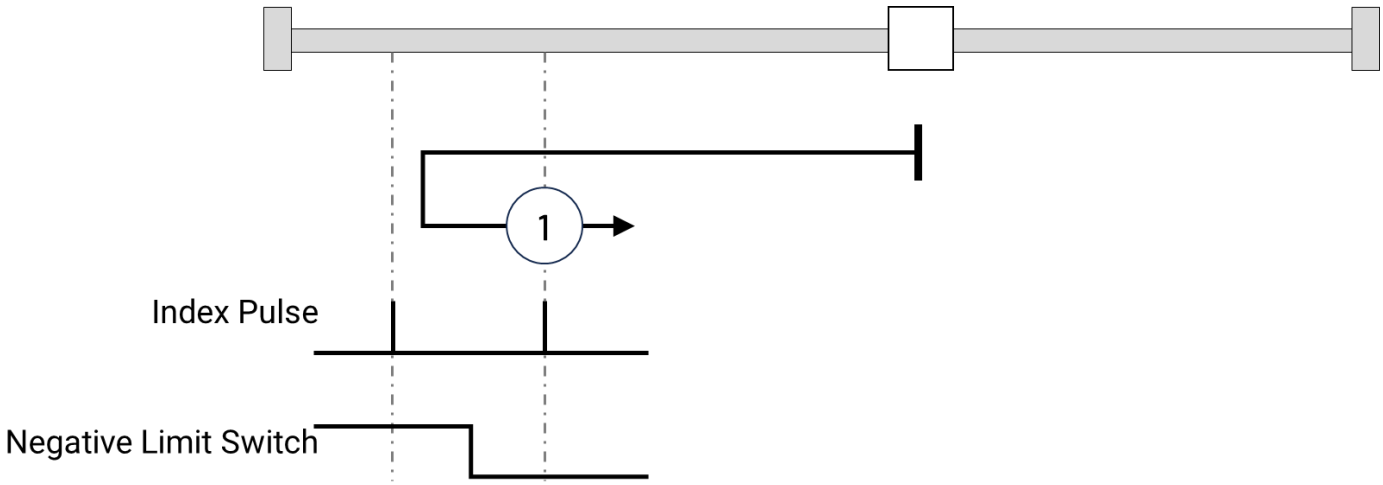


Figure 13: Homing on the negative limit switch and index pulse

Method 2: Homing on the positive limit switch and index pulse

Using this method the initial direction of movement is rightward if the positive limit switch is inactive (here shown as low). The position of home is at the first index pulse to the left of the position where the positive limit switch becomes inactive.

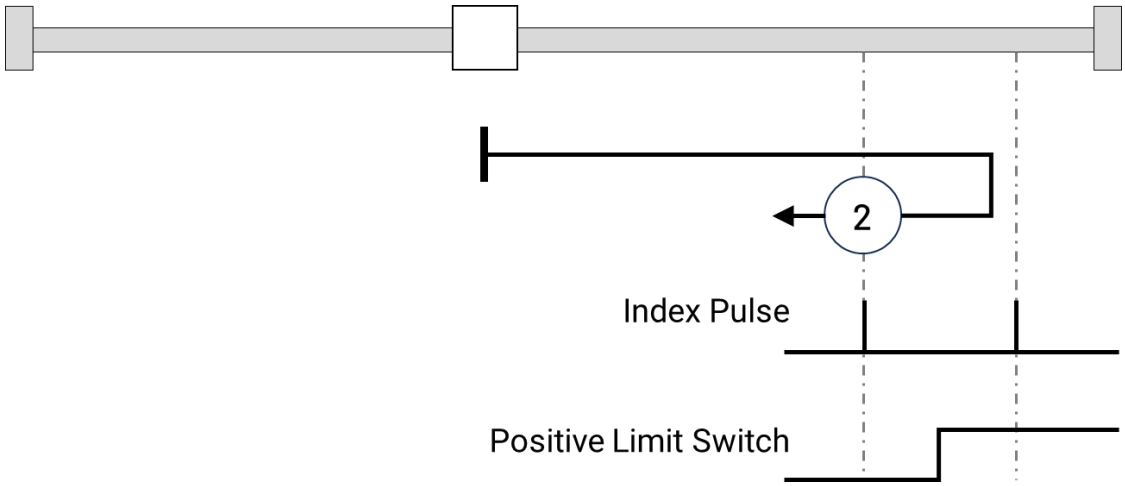


Figure 14: Homing on the positive limit switch and index pulse

Methods 3 and 4: Homing on the positive home switch and index pulse

Using methods 3 or 4 the initial direction of movement is dependent on the state of the home switch.
The home position is at the index pulse to either to the left or the right of the point where the home switch changes state. If the initial position is sited so that the direction of movement must reverse during homing, the point at which the reversal takes place is anywhere after a change of state of the home switch.

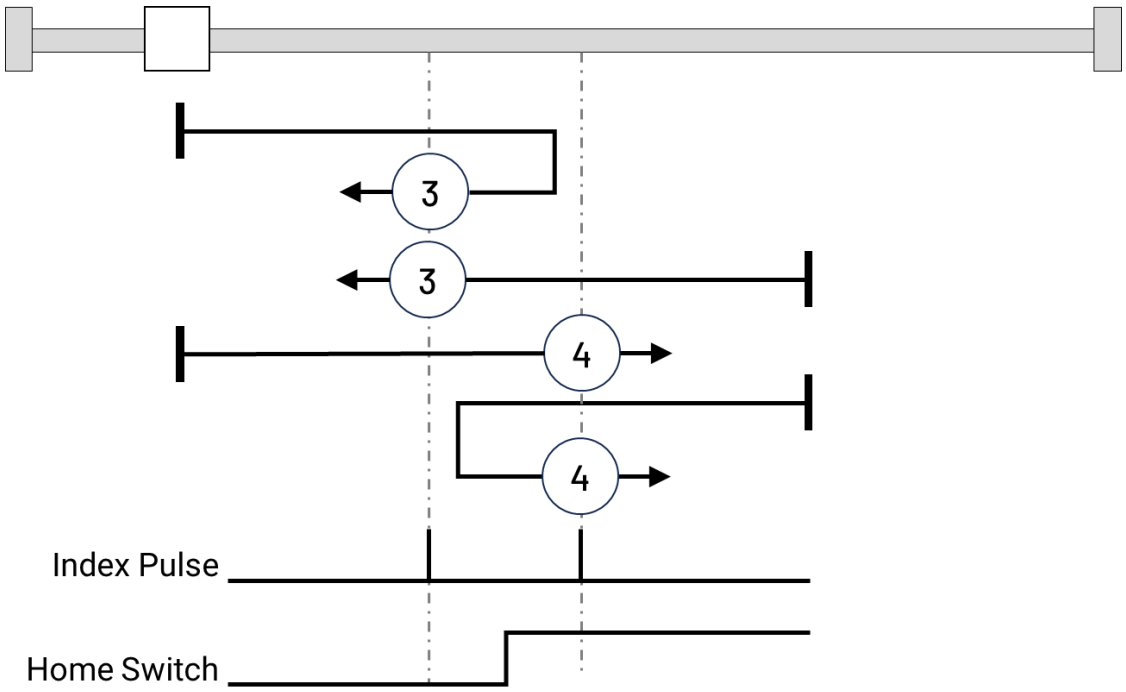


Figure 15: Homing on the positive home switch and index pulse

Methods 5 and 6: Homing on the negative home switch and index pulse

Using methods 5 or 6 the initial direction of movement is dependent on the state of the home switch.
The home position is at the index pulse to either to the left or the right of the point where the home switch changes state. If the initial position is sited so that the direction of movement must reverse during homing, the point at which the reversal takes place is anywhere after a change of state of the home switch.

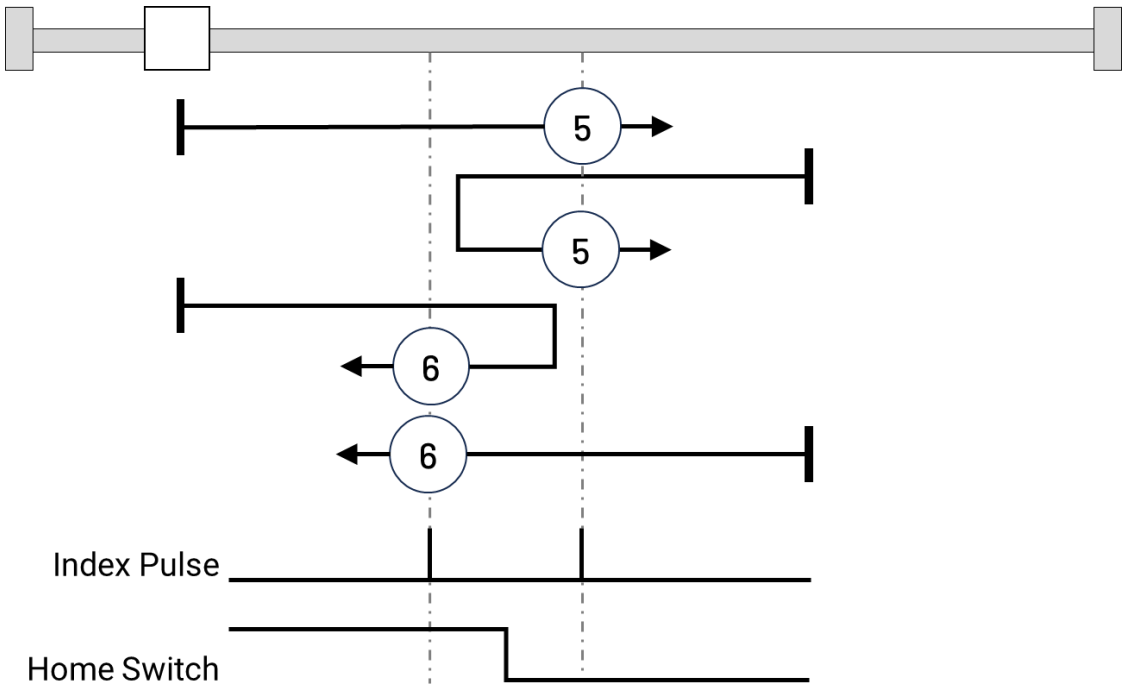


Figure 16: Homing on the negative home switch and index pulse

Methods 7 to 14: Homing on the home switch and index pulse

These methods use a home switch which is active over only portion of the travel, in effect the switch has a “momentary” action as the axle's position sweeps past the switch.
Using methods 7 to 10 the initial direction of movement is to the right, and using methods 11 to 14 the initial direction of movement is to the left except if the home switch is active at the start of the motion.
In this case the initial direction of motion is Dependent on the edge being sought. The home position is at the index pulse on either side of the rising or falling edges of the home switch, as shown in the following two diagrams. If the initial direction of movement leads away from the home switch, the drive must reverse on encountering the relevant limit switch.

Methods 7 to 10:

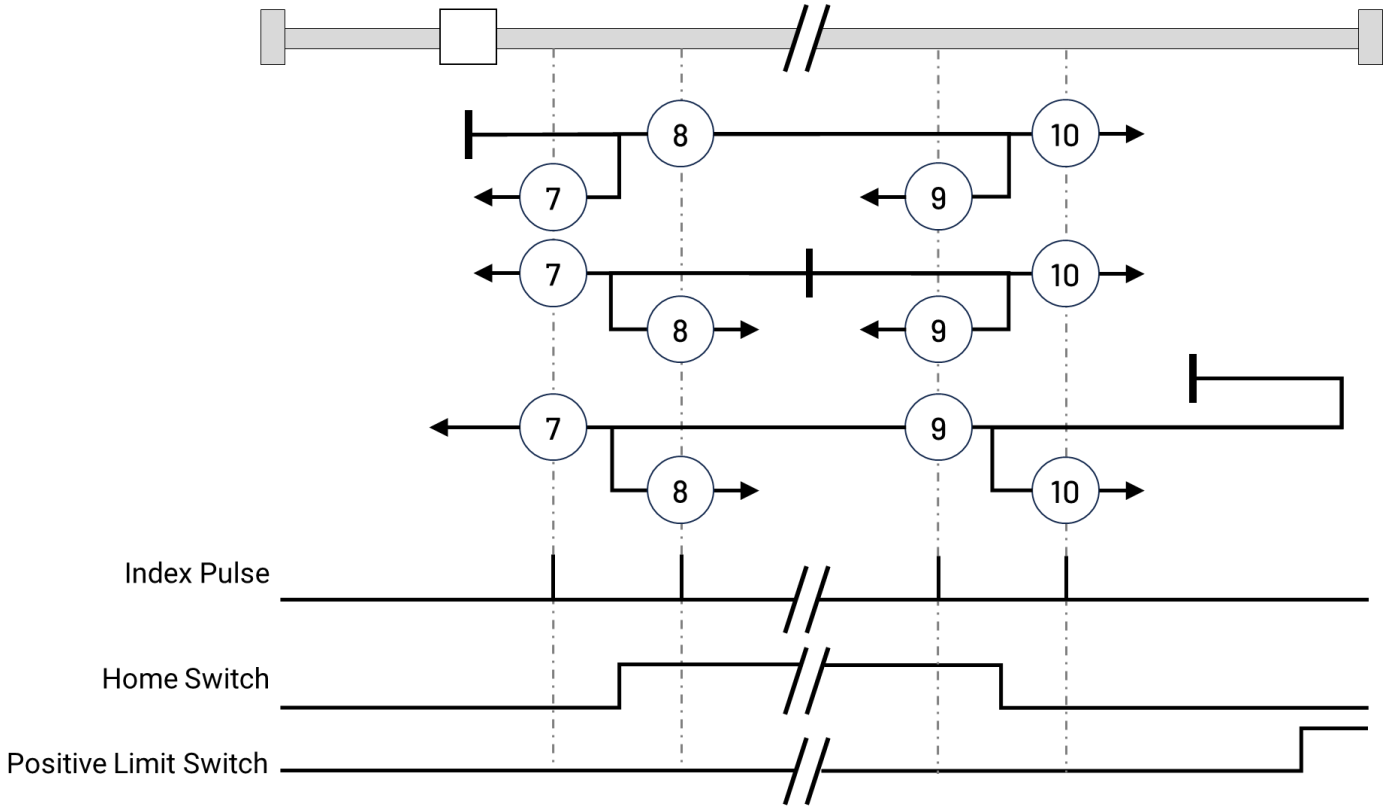


Figure 17: Homing on the home switch and index pulse - positive initial move

Methods 11 to 14:

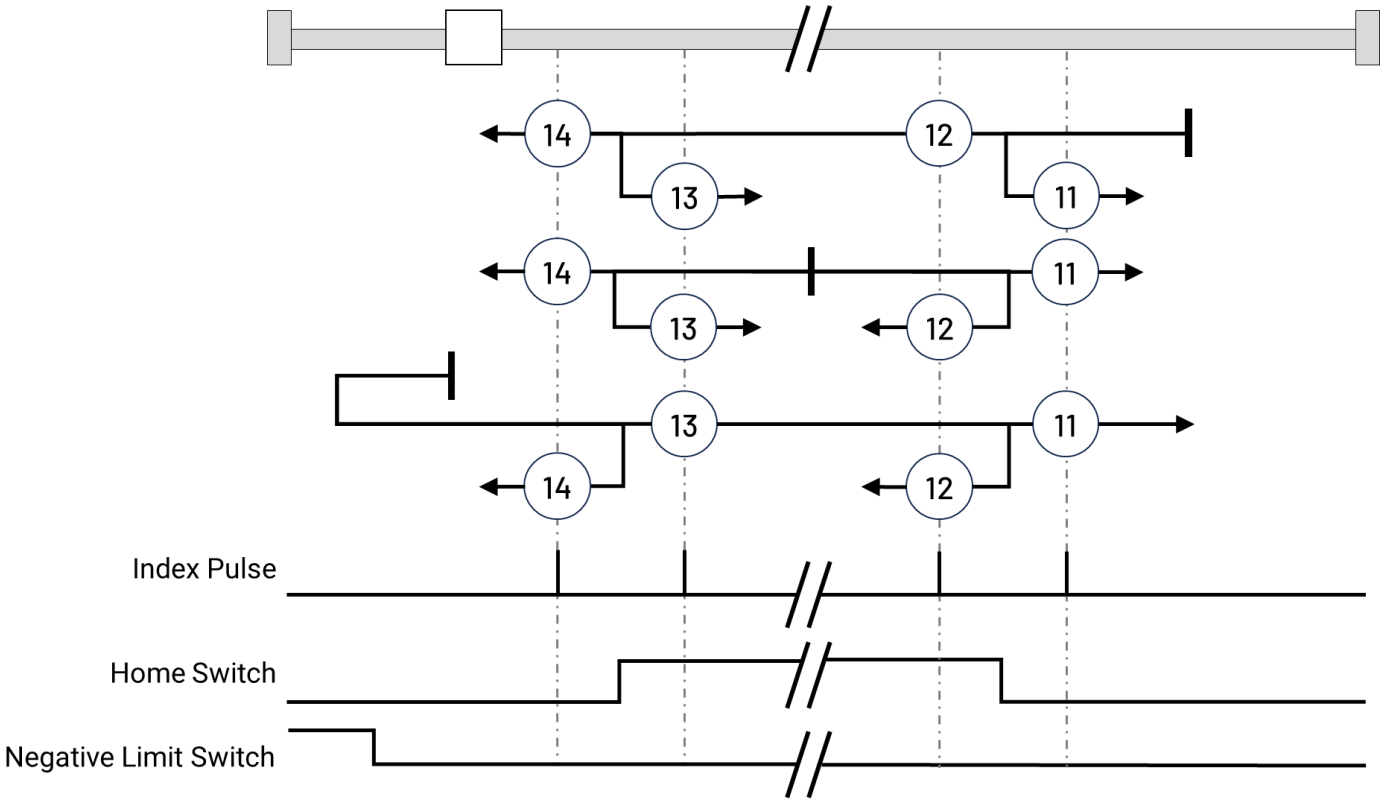


Figure 18: Homing on the home switch and index pulse - negative initial move

Methods 15 and 16: Reserved

These methods are reserved for future expansion of the homing mode.

Methods 17 to 30: Homing without an index pulse

These methods are similar to methods 1 to 14 except that the home position is not dependent on the index pulse but only dependent on the relevant home or limit switch transitions. For example methods 19 and 20 are similar to methods 3 and 4 as shown in the following diagram.

Methods 19 and 20:

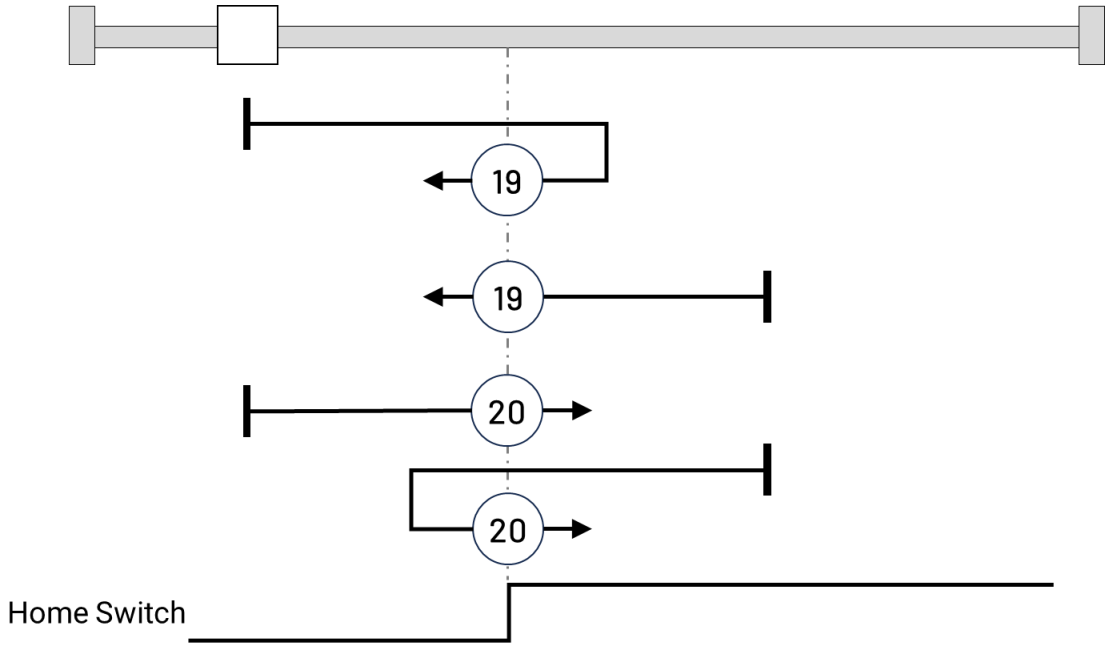


Figure 19: Homing on the positive home switch

Methods 20 and 21:

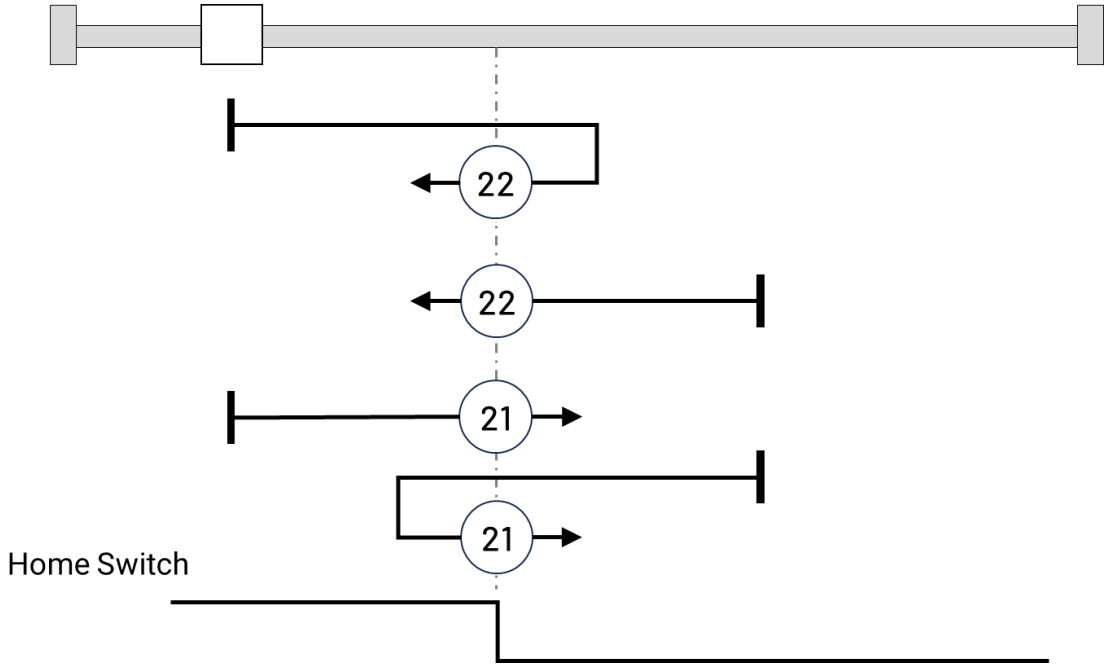


Figure 20: Homing on the positive home switch

Methods 31 and 32: Reserved

These methods are reserved for future expansion of the homing mode.

Methods 33 to 34: Homing on the index pulse

Using methods 33 or 34 the direction of homing is negative or positive respectively. The home position is at the index pulse found in the selected direction.

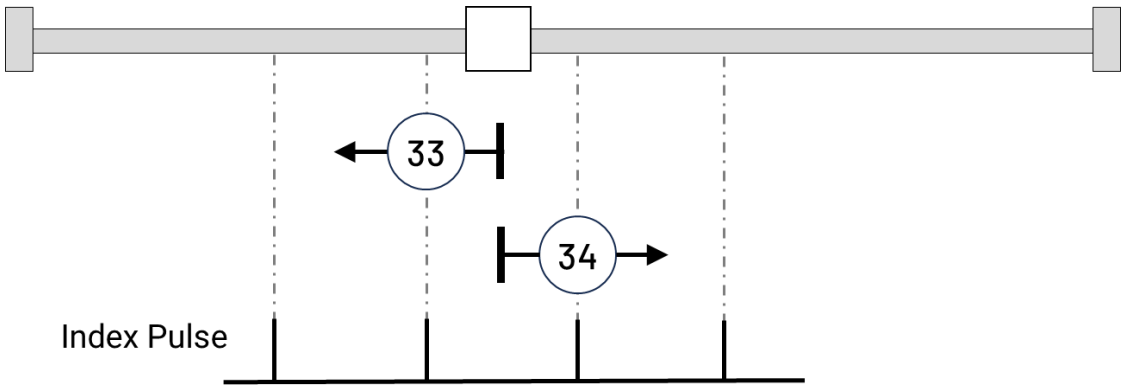


Figure 21: Homing on the index pulse

Method 35 and 37: Homing on the current position

In method 35 and method 37: the current position is taken to be the home position.

References

1. CiA Draft Standard 402: CiA® 402 - CANopen Drives and Motion Control Profile
   (Available at: CiA Website)
   Version 2.0, 26 July 2002.

2. ETG.6010 Implementation Directive for CiA402 Drive Profile
   (ETG6010_V1i1i0_D_R_CiA402_ImplDirective), ETG internal use only, 19 November 2014.
   (Available at: EtherCAT Website)

3. 86Duino EtherCAT Library Documentation: Programming Guide for 86Duino EtherCAT Library, Version 1.5, August 2024. Internal Document (Confidential).

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.