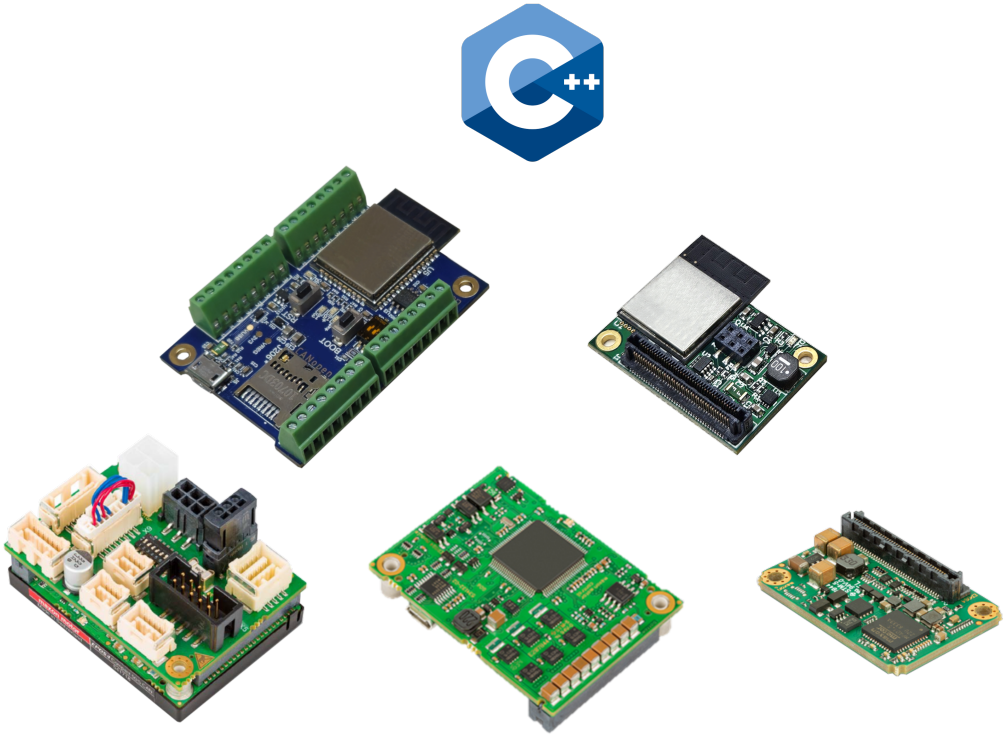
maxon

EPOS4

Class Library Documentation



CANopen

EPOS4
Class Library Documentation
MMAU | Edition 2024-02

TABLE OF CONTENTS

READ THIS FIRST

These instructions are intended for qualified technical personnel. Prior commencing with any activities...

- *you must carefully read and understand this manual and*
- *you must follow the instructions given therein.*

EPOS4 positioning controllers *are considered as partly completed machinery according to EU Directive 2006/42/EC, Article 2, Clause (g) and* ***is intended to be incorporated into or assembled with other machinery or other partly completed machinery or equipment.***

Therefore, you must not put the device into service...

- *unless you have made completely sure that the other machinery fully complies with the EU directive's requirements!*
- *unless the other machinery fulfills all relevant health and safety aspects!*
- *unless all respective interfaces have been established and fulfill the herein stated requirements!*

1 ABOUT

1.1 About this Document

1.1.1 Intended Purpose

The present document provides instructions on the implemented data structures and programming functions of the ”EPOS4 Class” Library implementation to be used to control the EPOS4 device using the MiniMaster LT and MicroMaster LT CANopen Master Controllers.

1.1.2 Target Audience

The present document is intended for trained and skilled personnel. It conveys information on how to understand and fulfill the respective work and duties.
This document is a reference book. It does require particular knowledge and expertise specific to the equipment described.

1.1.3 Trademarks and Brand Names

For easier legibility, registered brand names are listed below and will not be further tagged with their respective trademark. It must be understood that the brands (the list below is not necessarily concluding) are protected by copyright and/or other intellectual property rights even if their legal trademarks are omitted in the later course of this document.

Notation	Meaning
CANopen® CiA®	© CiA CAN in Automation e.V, DE-Nuremberg

Table 1-1 Brand names and trademark owners

2 Introduction

2.1 Document Structure

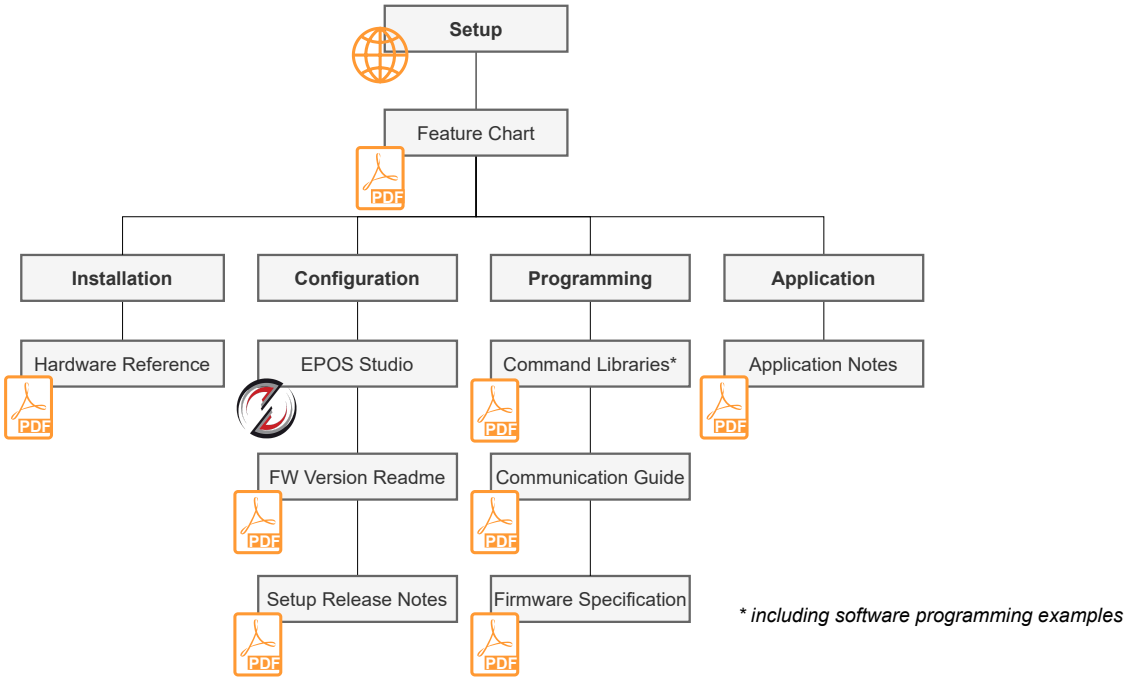


Figure 2-1 EPOS4 - Documentation structure

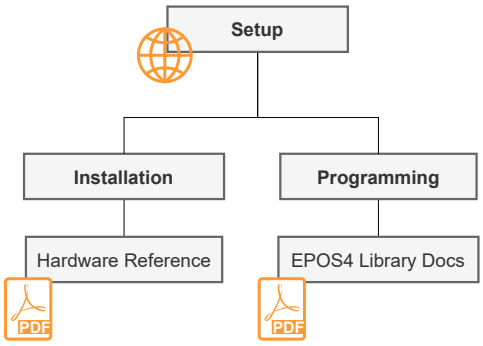


Figure 2-2 MicroMaster LT/MiniMaster LT - Documentation structure

2.2 General Information

The EPOS4 Class encapsulates the control of an EPOS4 digital positioning controller as a C++ class with member functions and variables. The class-based structure provides an intuitive approach to the control of an EPOS4 and attached motor hardware. The library has defined enumerator types to improve readability and general programming ease.
The class can be included in the MiniMaster LT/MicroMaster LT main application code, which is written in C/C++. The library provides control of EPOS4 CAN devices only and can be used in real-time applications with SDO and PDO support.
The library can be categorised into three broad areas, addressing the general CANopen specification, the EPOS4-specific CANopen implementation, and the core motion control implementation.

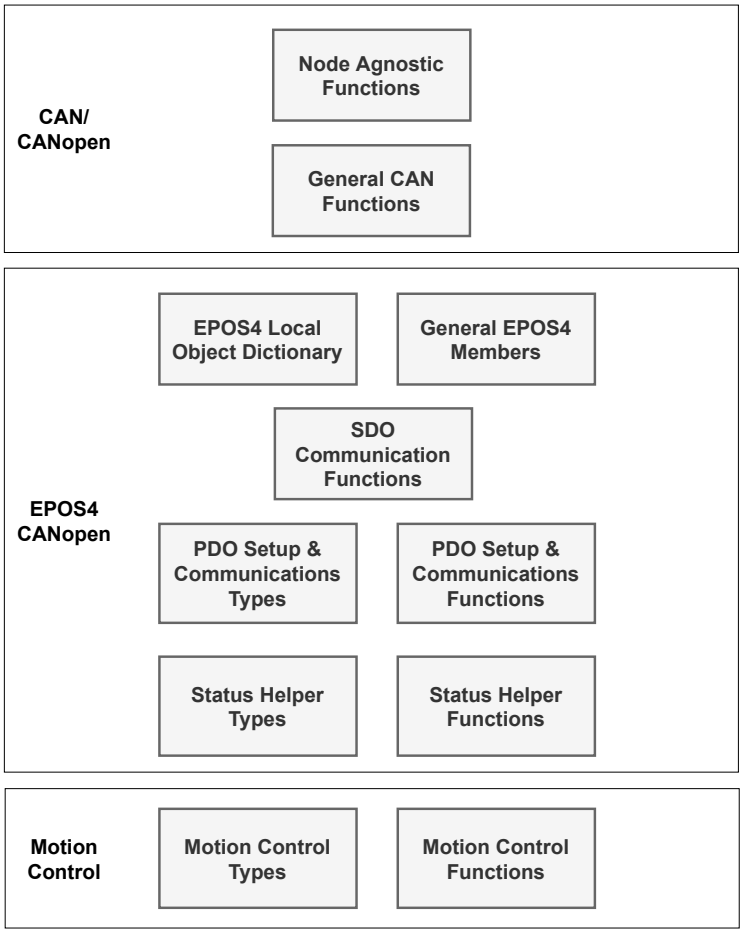


Figure 2-3 Software Structure - EPOS4 Class

Online code repositories have been set-up to host the library and an example project structure to help get started. These can be accessed using the following links:

- Master LT EPOS4 Library Repository - contains the header file and compiled library file that can be imported into a custom C++ project.
- Master LT EPOS4 Demo Repository - contains a sample project structure containing an example program.

3 EPOS4Class Members

3.1 Object Dictionary indices

Definition of EPOS_OD_t type.

3.1.1 EPOS_OD_t

DESCRIPTION

Index, Subindex, and Length of each EPOS4 object dictionary entry.

ENUMERATORS

Total enumerators (including undocumented): 223

3.2 General EPOS4 Members

Storage of local Object Dictionary and NMT state.

3.2.1 EPOS4()

FUNCTION

```
EPOS4(int32_t nodeID)
```

DESCRIPTION

Construct a new EPOS4 object.

PARAMETERS

nodeID	`int32_t`	The node-ID the corresponding EPOS4 is configured with.

3.2.2 getODpair()

FUNCTION

```
std::pair<int32_t, uint8_t> getODpair(EPOS_OD_t index)
```

DESCRIPTION

Gets an up to date value of the chosen Object Dictionary entry.

If the entry is not automatically updated though an Asynchronous TxPDO, it will be requested using an SDO. In this case, the function will block execution until the requested value has been returned from the EPOS4.

The status output is the same as would be returned by EPOS4::localODstatus().

PARAMETERS

index	`EPOS_OD_t`	The index of the Object.

RETURN PARAMETER

`std::pair <int32_t, uint8_t>`	An up to date value of the chosen object, and the status of the object.

3.2.3 getODvalue()

FUNCTION

```
uint32_t getODvalue(EPOS_OD_t index)
```

DESCRIPTION

Gets an up to date value of the chosen Object Dictionary entry.

If the entry is not automatically updated though an Asynchronous TxPDO, it will be requested using an SDO. In this case, the function will block execution until the requested value has been returned from the EPOS4.

It is recommended to check for SDO communication errors if using this function, by monitoring the return of the EPOS4::receiver() function. Otherwise, validity can be checked afterwards by using EPOS4::localODstatus().

PARAMETERS

index	`EPOS_OD_t`	The index of the Object.

RETURN PARAMETER

`uint32_t`	An up to date value of the chosen object.

3.2.4 localOD()

FUNCTION

```
int32_t localOD(EPOS_OD_t index)
```

DESCRIPTION

Returns the locally stored value of an Object Dictionary entry.

The local value is the most recent value sent by the Master or received from the EPOS4, and may not be valid or up to date!

Validity can be determined using EPOS4::localODstatus()

PARAMETERS

index	`EPOS_OD_t`	The index of the Object.

RETURN PARAMETER

`int32_t`	The locally stored value for the chosen object.

3.2.5 localODstatus()

FUNCTION

```
uint8_t localODstatus(EPOS_OD_t index)
```

DESCRIPTION

Returns the status of an Object Dictionary entry's local value.

Whether a value is up to date is not able to be determined from the status, and it is up to the user to define.

The bits of the status are defined as follows:

- bit 0: Most recent SDO read request response was valid.
- bit 1: Most recent SDO write request response was valid.
- bit 2: Bad SDO read or write, **ENTRY IS LIKELY INVALID**.
- bits 3 - 6: Reserved for future use.
- bit 7: OD entry mapped to an asynchronous TxPDO, and will be automatically updated to the latest value from the EPOS4.

If bits 0, 1, 2 are all zero, then the object is waiting for an SDO reply. More detailed errors are passed by the EPOS4::receiver() function.

PARAMETERS

index	`EPOS_OD_t`	The index of the Object.

RETURN PARAMETER

`uint8_t`	The status of the chosen local object.

3.3 Node Agnostic Functions

Functions which can affect multiple nodes at once, or are part of the Master's setup.

3.3.1 TWAISetup()

FUNCTION

```
static ERROR_CODE_t TWAISetup(twai_timing_config_t timingConfig = TWAI_TIMING_
CONFIG_125KBITS(), bool ABRSetup = false, gpio_num_t TxIONum = CAN_TX_PIN,
gpio_num_t RxIONum = CAN_RX_PIN)
```

DESCRIPTION

Runs the initialisation routine for the CAN bus Driver.

PARAMETERS

timingConfig	`twai_timing_config_t`	The timing configuration for the CAN driver. Transmission rates can be 20, 50, 125, 250, 500, 800 kbit/s, or 1 Mbit/s. Default 125 kbit/s.
ABRSetup	`bool`	If **all** nodes on the CAN bus are set to Auto Bit Rate, set this to true. Default False.
TxIONum	`gpio_num_t`	IO pin to used for TX. Default CAN_TX_PIN = 21 (MiniMasterLT), 36 (MicroMasterLT)
RxIONum	`gpio_num_t`	IO pin to used for RX. Default CAN_RX_PIN = 22 (MiniMasterLT), 37 (MicroMasterLT)

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.3.2 broadcastSync()

FUNCTION

```
static ERROR_CODE_t broadcastSync()
```

DESCRIPTION

Broadcasts a SYNC Object onto the CAN bus.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.3.3 changeNMTState()

FUNCTION

```
static ERROR_CODE_t changeNMTState(NMT_COMMAND_t NMTCommand, uint8_t nodeID =
0x00)
```

DESCRIPTION

Function used to control the Network Management (NMT) state for devices on the CAN bus.

PARAMETERS

NMTCommand	`NMT_COMMAND_t`	The NMT command to execute.
nodeID	`uint8_t`	Node to send the NMT command to. Default 0 (All nodes)

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.3.4 sendHeartbeat()

FUNCTION

```
static ERROR_CODE_t sendHeartbeat(int32_t heartbeatNodeID)
```

DESCRIPTION

Broadcast a Heartbeat frame onto the CAN bus.

PARAMETERS

heartbeatNodeID	`int32_t`	The node-ID to be broadcast with the heartbeat frame. It should match the HBnodeID configured for the heartbeat consumer.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.4 General CAN Functions

Perform tasks required to configure.

3.4.1 getIndexFromNoLength()

FUNCTION

```
EPOS_OD_t getIndexFromNoLength(int32_t testIndex, int32_t testSubindex = 0x00)
```

DESCRIPTION

Convert an index and subindex into a valid EPOS_OD_t value.

PARAMETERS

testIndex	`int32_t`	Index (0xssiiii) value. 'ss' represent subindex byte, 'iiii' represent index bytes.
testSubindex	`int32_t`	if a subindex is not packed in the index, this value will be used.

RETURN PARAMETER

`EPOS_OD_t`	EPOS_OD_t: A valid EPOS4 index entry. Equal to EPOS_OD_INVALID_ENTRY if no entry matches the given index.

3.4.2 receiver()

FUNCTION

```
ERROR_CODE_t receiver(const twai_message_t message)
```

DESCRIPTION

Parses any frames on the CAN bus and updates the local object dictionary accordingly.

PARAMETERS

message	`const twai_message_t`	The frame received from the CAN bus.

RETURN PARAMETER

`ERROR_CODE_t`	EPOS_ERROR_CODE_t: If an EPOS4 or CAN SDO error was received, this will be the error type.

3.4.3 setHeartbeatConsumer()

FUNCTION

```
ERROR_CODE_t setHeartbeatConsumer(uint16_t HBnodeID, uint16_t HBperiodMS,
uint16_t consumerNum = 1)
```

DESCRIPTION

Configures the period and Node-ID from which the EPOS4 will expect heartbeats.

If heartbeats are not received in time, the EPOS4 will perform the actions specified by the abort connection option code and communication error behavior.

PARAMETERS

HBnodeID	`uint16_t`	The node ID of the heartbeat producer.
HBperiodMS	`uint16_t`	The expected period in milliseconds between heartbeats. It is recommended to set the consumer heartbeat time value at least 20 ms higher than the period of the producer to account for CAN bus latency jitter.
consumerNum	`uint16_t`	The EPOS4 can consume two different heartbeats, choose 1 or 2 to set the configuration for the corresponding consumer number.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.4.4 ticksSinceHeartbeat()

FUNCTION

```
TickType_t ticksSinceHeartbeat()
```

DESCRIPTION

Returns the number of ticks passed between now and the time the last heartbeat was received from the EPOS4.

RETURN PARAMETER

`TickType_t`	TickType_t: The number of ticks since the last heartbeat was received. Multiply by portTICK_RATE_MS to get milliseconds.

3.5 SDO Communication Functions

SDO communications are used for configuration and access to individual data objects.

3.5.1 requestSDO()

FUNCTION

```
ERROR_CODE_t requestSDO(EPOS_OD_t index, uint16_t waitForResponse = 0)
```

DESCRIPTION

Request to read EPOS4 Object Dictionary entry using SDO.

It is usually best to use EPOS4::getODvalue() or EPOS4::getODpair() to retrieve a value from the EPOS4.

To access the value after the request from the EPOS4, use the localOD functions:

```
ERROR_CODE_t error = EPOS4::requestSDO(objectIndex, 1);
int32_t value = EPOS4::localOD(objectIndex);
```

PARAMETERS

index	`EPOS_OD_t`	Index of the Object Dictionary entry to request from the EPOS4.
waitForResponse	`uint16_t`	If true, execution will block until a response is received from the EPOS4. If the response is not received within SDO_TIMEOUT_TICKS an error is returned. If the response is invalid (SDO Abort Error) an error is also returned. Default: 0 (non - blocking, fewer error warnings. Parse errors from the receiver if using)

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.5.2 sendSDO()

FUNCTION

```
ERROR_CODE_t sendSDO(EPOS_OD_t index, int32_t value, uint16_t waitForResponse =
0)
```

DESCRIPTION

Writes to the EPOS4 Object Dictionary.

PARAMETERS

index	`EPOS_OD_t`	Index of the Object Dictionary entry to write to in the EPOS4.
value	`int32_t`	The value to be written to the corresponding object.
waitForResponse	`uint16_t`	If true, execution will block until a response is received from the EPOS4. Default: 0 (non - blocking). If the response is not received within SDO_TIMEOUT_TICKS an error is returned. If the response is invalid (SDO Abort Error) an error is also returned.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.6 PDO Setup & Communication Functions

PDOs allow for faster, automatic, and synchronised communication of pre-defined objects.

3.6.1 configPDO()

FUNCTION

```
ERROR_CODE_t configPDO(std::string Name, const PDO_MAPPING_t PDOConfig, uint32_
t COBID = 0)
```

DESCRIPTION

Sends PDO mapping configurations to the EPOS4.
- The EPOS4 must be in pre-operational state for a PDO mapping to be accepted.
- Existing PDO entries should be cleared using EPOS4::resetNumPDOMapped() first, as any pre-configured entries will be unable to be parsed by the Master.

PARAMETERS

Name	`std::string`	The name to assign to the PDO, used when calling RX/TX functions: • EPOS4::sendRxPDO() • EPOS4::getTxPDO()
PDOConfig	`const PDO_MAPPING_t`	A PDO Map containing the configuration to send to the EPOS4
COBID	`uint32_t`	COB-ID for this PDO. Default 0: Automatic ID (PDO_TYPE_t + Node-ID).

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.6.2 getTxPDO()

FUNCTION

```
ERROR_CODE_t getTxPDO(std::string PDOName)
```

DESCRIPTION

Request a TX PDO using a remote data frame (EPOS4 → Master).

PARAMETERS

PDOName	`std::string`	The name assigned to the TX PDO being requested.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.6.3 resetNumPDOMapped()

FUNCTION

```
ERROR_CODE_t resetNumPDOMapped()
```

DESCRIPTION

Set the number of PDOs mapped for all PDOs to zero.

Should be used before PDOs are written, as any pre-configured entries will be unable to be parsed by the Master.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.6.4 sendRxPDO()

FUNCTION

```
ERROR_CODE_t sendRxPDO(std::string PDOName, std::vector<int32_t> values)
```

DESCRIPTION

Send data to the EPOS4 using an RxPDO (Master → EPOS4)

PARAMETERS

PDOName	`std::string`	The name assigned to the RX PDO being sent.
values	`std::vector <int32_t>`	A vector of values to be sent through using the RxPDO. They must be given in the same order as the indices were set in the PDO_MAPPING_t configuration.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.7 PDO Setup & Communication Types

Types used to simplify PDO setup and usage.

3.7.1 PDO_MAPPING_t

DESCRIPTION

PDO mapping structure.

MEMBERS

PDONumber	`PDO_TYPE_t`	TX/RX PDO number to configure
mode	`PDO_TRANSMISSION_MODE_t`	The transmission type, which determines the PDO scheduling.
targetIndexes	`std::vector <EPOS_OD_t>`	Object Dictionary indices of each mapped object
targetLengths	`std::vector <int32_t>`	Byte length of each mapped object. **Must total 8 or less**. Leave uninitialised to determine automatically from targetIndexes

3.7.2 PDO_TRANSMISSION_MODE_t

DESCRIPTION

PDO transmission modes.

ENUMERATORS

Total enumerators (including undocumented): 3

`PDO_TRANSMISSION_MODE_SYNC`	1	Synchronous: • TXPDO: Transmits from EPOS4 after a SYNC object • RXPDO: Only writes values to EPOS4 object dictionary after a SYNC object
`PDO_TRANSMISSION_MODE_ASYNC_RTR`	253	Asynchronous RTR: • **TXPDO only**: Transmits from EPOS4 only when requested by the Master
`PDO_TRANSMISSION_MODE_ASYNC`	255	Asynchronous: • TXPDO: Transmitted from EPOS4 when a mapped value changes, with a defined minimum period • RXPDO: Writes values to EPOS4 object dictionary immediately

3.7.3 PDO_TYPE_t

DESCRIPTION

PDO Type, Number, and corresponding COB-ID base.

ENUMERATORS

Total enumerators (including undocumented): 8

TXPDO1	0x180	TxPDO Number 1, COB-ID Base: 0x180
TXPDO2	0x280	TxPDO Number 2, COB-ID Base: 0x280
TXPDO3	0x380	TxPDO Number 3, COB-ID Base: 0x380
TXPDO4	0x480	TxPDO Number 4, COB-ID Base: 0x480
RXPDO1	0x200	RxPDO Number 1, COB-ID Base: 0x200
RXPDO2	0x300	RxPDO Number 2, COB-ID Base: 0x300
RXPDO3	0x400	RxPDO Number 3, COB-ID Base: 0x400
RXPDO4	0x500	RxPDO Number 4, COB-ID Base: 0x500

3.8 Motion Control Functions

These functions control motor Torque, Velocity, and Position.

3.8.1 disable()

FUNCTION

```
ERROR_CODE_t disable()
```

DESCRIPTION

Disables the power stage.
Changes to the 'Ready to Switch on' axis state.
Torque will no longer be applied to hold motor position.
Does not reset faults or quick stops, does not power up from 'Switch on Disabled' state.

RETURN PARAMETER

ERROR_CODE_t	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.8.2 enable()

FUNCTION

```
ERROR_CODE_t enable()
```

DESCRIPTION

Enables the power stage.
Changes to the 'Operation Enabled' axis state.
Torque will be applied to the motor to hold position.

RETURN PARAMETER

ERROR_CODE_t	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.8.3 halt()

FUNCTION

```
ERROR_CODE_t halt()
```

DESCRIPTION

Command the motor to halt motion.
Sets the halt bit of the ControlWord to true.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.8.4 isEnabled()

FUNCTION

```
char isEnabled()
```

DESCRIPTION

Checks if EPOS4 is in the Operation Enabled axis state.

RETURN PARAMETER

`char`	char • True: Enabled • False: Disabled

3.8.5 moveToTargetPosition()

FUNCTION

```
ERROR_CODE_t moveToTargetPosition(int32_t position, bool wantRelative, bool
waitForTarget)
```

DESCRIPTION

Sets the new Target Position and moves the motor.

PARAMETERS

position	`int32_t`	The target position, in encoder counts.
wantRelative	`bool`	If the target position is absolute or relative. • False: Absolute, The motor will move until the Position Actual Value matches the given position. • True: Relative, The motor will move until the Position Actual Value has changed by the target amount.
waitForTarget	`bool`	If the function should block execution until the motion is completed. • True: Wait.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.8.6 moveToTargetVelocity()

FUNCTION

```
ERROR_CODE_t moveToTargetVelocity(int32_t velocity)
```

DESCRIPTION

Sets new Target Velocity and moves the motor.

PARAMETERS

velocity	`int32_t`	The desired speed, in rpm. Speed before gearing.

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.8.7 setModeOfOperation()

FUNCTION

```
ERROR_CODE_t setModeOfOperation(EPOS_OPERATION_MODE_t mode)
```

DESCRIPTION

Changes the EPOS4's mode of operation.

Available Modes:

- Profile Position Mode (PPM), Drive follows a trajectory to a target position
  - EPOS4::moveToTargetPosition()
- Profile Velocity Mode (PVM), Drive follows a trajectory to reach a target velocity
  - EPOS4::halt()
  - EPOS4::moveToTargetVelocity()
- Homing Mode (HMM), For various methods to find the home position.
  - HOMING_METHOD_t

Cyclic Synchronous modes are available but do not have helper functions:

- Position Mode (CSP)
- Velocity Mode (CSV)
- Torque Mode (CST)

PARAMETERS

mode	`EPOS_ OPERATION_ MODE_t`	The desired mode

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.9 Motion Control Types

Types used for motion control.

3.9.1 CW_BITS_t

DESCRIPTION

EPOS4 ControlWord bits.
Bits 0, 1, 2, 3, and 7 can be controlled using EPOS4::sendAxisCommand()

ENUMERATORS

Total enumerators (including undocumented): 17

CW_BITS_SWITCH_ON	0	Switch On
CW_BITS_ENABLE_VOLTAGE	1	Enable Voltage
CW_BITS_QUICK_STOP	2	Quick Stop
CW_BITS_ENABLE_OPERATION	3	Enable Operation
CW_BITS_NEW_SET_POINT	4	PPM: New Setpoint
CW_BITS_HOMING_OPERATION_START	4	HMM: Start Homing operation
CW_BITS_CHANGE_SET_IMMEDIATELY	5	PPM: Change Setpoint immediately
CW_BITS_ABS_OR_RELATIVE	6	PPM: Absolute or Relative position
CW_BITS_FAULT_RESET	7	Fault reset
CW_BITS_HALT	8	PPM, PVM, HMM: Halt
CW_BITS_ENDLESS_MOVEMENT	15	PPM: Endless movement

3.9.2 EPOS_OPERATION_MODE_t

DESCRIPTION

EPOS4 motion operation mode

ENUMERATORS

Total enumerators (including undocumented): 6

EPOS_OPERATION_MODE_PPM	1	1 Profile Position Mode (PPM)
EPOS_OPERATION_MODE_PVM	3	3 Profile Velocity Mode (PVM)
EPOS_OPERATION_MODE_HMM	6	6 Homing Mode (HMM)
EPOS_OPERATION_MODE_CSP	8	8 Cyclic Synchronous Position Mode (CSP)
EPOS_OPERATION_MODE_CSV	9	9 Cyclic Synchronous Velocity Mode (CSV)
EPOS_OPERATION_MODE_CST	10	10 Cyclic Synchronous Torque Mode (CST)

3.9.3 HOMING_METHOD_t

DESCRIPTION

Homing Methods. See EPOS4 Firmware Specification 3.5.3 Homing Modes.

ENUMERATORS

Total enumerators (including undocumented): 15

3.9.4 SW_BITS_t

DESCRIPTION

EPOS4 StatusWord bits.
Bits 0, 1, 2, 3, 5, and 6 are interpreted by EPOS4::getAxisState()

ENUMERATORS

Total enumerators (including undocumented): 20

SW_BITS_READY_TO_SWITCH_ON	0	Ready to switch on
SW_BITS_SWITCHED_ON	1	Switched on
SW_BITS_OPERATION_ENABLED	2	Operation enabled
SW_BITS_FAULT	3	Fault
SW_BITS_VOLTAGE_ENABLED	4	Voltage enabled (power stage on)
SW_BITS_QUICK_STOP	5	Quick stop
SW_BITS_SWITCH_ON_DISABLED	6	Switch on disabled
SW_BITS_WARNING	7	Warning
SW_BITS_REMOTE	9	Remote: Indicates NMT state is 'Operational'
SW_BITS_TARGET_REACHED	10	PPM, PVM, HMM: Target Reached
SW_BITS_INTERNAL_LIMIT_ACTIVE	11	I2t, Current, Velocity limit active
SW_BITS_SET_POINT_ACK	12	PPM: Setpoint Acknowledged
SW_BITS_HOMING_ATTAINED	12	HMM: Homing Attained
SW_BITS_SPEED_IS_ZERO	12	PVM: Speed is Zero
SW_BITS_DRIVE_FOLLOWS_COMMAND_VALUE	12	CSP, CSV, CST: Drive Following Command Value
SW_BITS_FOLLOWING_ERROR	13	PPM, CSP: Following Error
SW_BITS_HOMING_ERROR	13	HMM: Homing Error
SW_BITS_POSITION_REFERENCED_TO_HOME	15	Position Referenced to Home

3.10 Status Helper Functions

These functions assist in determining and changing the state of the drive.

3.10.1 clearError()

FUNCTION

```
ERROR_CODE_t clearError()
```

DESCRIPTION

Attempt to clear any errors on the EPOS4.
This is done by toggling the ControlWord's CW_BITS_FAULT_RESET bit.

RETURN PARAMETER

ERROR_CODE_t	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.10.2 getAxisState()

FUNCTION

```
EPOS_AXIS_STATE_t getAxisState(int32_t StatusWord = -1)
```

DESCRIPTION

Decodes the Axis State from the Status Word.
See EPOS4 Firmware Specification 2.2, Device Control for information on Axis States.

PARAMETERS

StatusWord	`int32_t`	If the StatusWord is not passed to the function, the up to date value will be requested from the EPOS4 using EPOS4::getOD().

RETURN PARAMETER

`EPOS_AXIS_STATE_t`	EPOS_AXIS_STATE_t: The current Axis state

3.10.3 getBitFromErrorRegister()

FUNCTION

```
int16_t getBitFromErrorRegister(ER_BITS_t bit)
```

DESCRIPTION

Extract a particular bit from the local ErrorRegister object.
Note that the error register is automatically updated by CAN emergency frames.

PARAMETERS

bit	`ER_BITS_t`	ErrorRegister bit to be extracted

RETURN PARAMETER

`int16_t`	uint16_t: 1 or 0 representing the selected bit

3.10.4 getBitFromStatusWord()

FUNCTION

```
int16_t getBitFromStatusWord(SW_BITS_t bit)
```

DESCRIPTION

Extract a particular bit from the local StatusWord object.
The StatusWord is not retrieved from the EPOS4 in this function, It should be either configured in an asynchronous TxPDO mapping or retrieved via SDO before being used.

```
motor.requestSDO(EPOS_OD_STATUSWORD, 1);
motor.getBitFromStatusWord(SW_BITS_t bit);
```

PARAMETERS

bit	`SW_BITS_t`	StatusWord bit to be extracted

RETURN PARAMETER

`int16_t`	uint16_t: 1 or 0 representing the selected bit

3.10.5 parseError()

FUNCTION

```
static uint8_t parseError(ERROR_CODE_t error)
```

DESCRIPTION

Determines the fault reaction a particular Error Code causes.

PARAMETERS

error	`ERROR_CODE_t`	The error code to parse.

RETURN PARAMETER

`uint8_t`	uint8_t: Byte indicating the reaction, in the format: `0b_DFAXWSM` • bit 0: Master Error (M) • bit 1: SDO Error (S) • bit 2: Warning (no effect on device status) (W) • bit 3: Position Clear (position value will be cleared on error reset) (X) • bit 4: Abort connection reaction performed (A) • bit 5: Fault reaction performed (F) • bit 6: Secure movement no longer possible (D) • bit 7: Reserved If = 0; Then there is No Error.

3.10.6 sendAxisCommand()

FUNCTION

```
ERROR_CODE_t sendAxisCommand(EPOS_AXIS_COMMAND_t command)
```

DESCRIPTION

Sends a Device Control Command to change the state of the drive.
See EPOS4 Firmware Specification 2.2, Device Control for information on Axis States.

PARAMETERS

command	`EPOS_AXIS_COMMAND_t`	The axis control command to send to the EPOS4

RETURN PARAMETER

`ERROR_CODE_t`	ERROR_CODE_t: ERROR_CODE_NOERROR when successful.

3.10.7 setControlWordBits()

FUNCTION

```
uint16_t setControlWordBits(std::vector<CW_BITS_t> bits, std::vector<uint16_t>
values)
```

DESCRIPTION

Returns a modified copy of the local ControlWord with the selected bits modified.

PARAMETERS

bits	`std::vector <CW_BITS_t>`	Vector of CW_BITS_t to change
values	`std::vector <uint16_t>`	Vector of values (0 or 1) corresponding to the selected bits

RETURN PARAMETER

`uint16_t`	uint16_t: The modified ControlWord. This should be sent to the EPOS4 using EPOS4::sendSDO() or EPOS4::sendRxPDO().

3.11 Status Helper Types

Types defining Errors, Drive Axis States, and CANopen NMT Services.

3.11.1 EPOS_AXIS_COMMAND_t

DESCRIPTION

Control Commands.

Bit masks are encoded in the upper byte.

See EPOS4 Firmware Specification 2.2.3 Device Control Commands.

ENUMERATORS

Total enumerators (including undocumented): 8

`AXIS_COMMAND_ SHUTDOWN`	0b1000’0111’ 0000’0110	State transitions 2, 6, 8. ’Shutdown’
`AXIS_COMMAND_ SWITCH_ON`	0b1000’0111’ 0000’0111	State transition 3 ’Switch On
`AXIS_COMMAND_ SWITCH_ON_ENABLE_ OPERATION`	0b1000’1111’ 0000’1111	State transitions 3, 4 ’Switch On & Enable Motion
`AXIS_COMMAND_ DISABLE_VOLTAGE`	0b1000’0010’ 0000’0000	State transitions 7, 9, 10, 12 ’Disable Voltage
`AXIS_COMMAND_ QUICK_STOP`	0b1000’0110’ 0000’0010	State transition 11 ’Quick Stop
`AXIS_COMMAND_ DISABLE_OPERATION`	0b1000’1111’ 0000’0111	State transition 5 ’Switch On, Disable Torque
`AXIS_COMMAND_ ENABLE_OPERATION`	0b1000’1111’ 0000’1111	State transitions 4, 16 ’Enable Motion
`AXIS_COMMAND_ FAULT_RESET`	0b1000’0000’ 1000’0000	State transitions 14, 15 ’Reset Fault, Disable Voltage

3.11.2 EPOS_AXIS_STATE_t

DESCRIPTION

State of the Drive Axis States.

Retrieved from the StatusWord using the mask 0b0110’1111. See EPOS4 Firmware Specification 2.2.1 State of the Drive.

ENUMERATORS

Total enumerators (including undocumented): 8

AXIS_STATE_NOT_READY_TO_SWITCH_ON	0b0000'0000	Drive function is disabled
AXIS_STATE_SWITCH_ON_DISABLED	0b0100'0000	Drive initialization is complete. Drive parameters may be changed. Drive function is disabled.
AXIS_STATE_READY_TO_SWITCH_ON	0b0010'0001	Drive parameters may be changed. Drive function is disabled.
AXIS_STATE_SWITCHED_ON	0b0010'0011	Drive function is disabled. Current offset calibration done
AXIS_STATE_OPERATION_ENABLED	0b0010'0111	No faults have been detected. Drive function is enabled and power is applied to the motor.
AXIS_STATE_QUICK_STOP_ACTIVE	0b0000'0111	'Quick stop' function is being executed. Drive function is enabled and power is applied to the motor.
AXIS_STATE_FAULT_REACTION_ACTIVE	0b0000'1111	A fault has occurred in the drive. Selected fault reaction is being executed.
AXIS_STATE_FAULT	0b0000'1000	A fault has occurred in the drive. Drive parameters may have changed. Drive function is disabled.

3.11.3 ERROR_CODE_t

DESCRIPTION

Error Codes, for Master, EPOS4, and SDO abort.
Can be parsed using EPOS4::parseError().

ENUMERATORS

Total enumerators (including undocumented): 107

MASTER_ERROR_CODE_GENERIC_ERROR	0x0100	Generic Error
MASTER_ERROR_CODE_CAN_ERROR	0x0101	The TWAI driver or CAN Bus are incorrectly configured
MASTER_ERROR_CODE_ARGUMENT_ERROR	0x0102	The function arguments are invalid
MASTER_ERROR_CODE_TIMEOUT_ERROR	0x0104	EPOS4 did not send the correct response in time, possibly a CAN error
MASTER_ERROR_CODE_EPOS4_ERROR	0x0108	Function cannot be used as EPOS4 is in an error state
MASTER_ERROR_CODE_SDO_ERROR	0x0110	SDO was not accepted
EPOS_ERROR_CODE_GENERIC_ERROR	0x1000	0x1000 to 0xFF22: EPOS4 Error Codes
SDO_ERROR_CODE_TOGGLE_BIT	0x0503'0000	0x0503'0000 to 0x0800'0023: SDO Error Codes

3.11.4 ER_BITS_t

DESCRIPTION

EPOS4 Error Register Bits.

ENUMERATORS

Total enumerators (including undocumented): 8

ER_BITS_GENERIC	0	Generic error
ER_BITS_CURRENT	1	Current error
ER_BITS_VOLTAGE	2	Voltage error
ER_BITS_ TEMPERATURE	3	Temperature error
ER_BITS_ COMMUNICATION	4	Communication error
ER_BITS_PROFILE_ SPECIFIC	5	Device Profile Specfic
ER_BITS_RESERVED	6	Reserved (always 0)
ER_BITS_MOTION	7	Motion Error

3.11.5 NMT_COMMAND_t

DESCRIPTION

Network Management (NMT) Commands.
On startup, the EPOS4 will automatically Transition from 'Initialisation' to 'Pre-Operational'.

ENUMERATORS

Total enumerators (including undocumented): 5

NMT_COMMAND_GOTO_ OPERATIONAL	0x1	Start Remote Node → Operational
NMT_COMMAND_GOTO_ STOPPED	0x02	Stop Remote Node → Stopped
NMT_COMMAND_GOTO_ PRE_OPERATIONAL	0x80	Enter Pre-Operational → Pre-Operational
NMT_COMMAND_GOTO_ RESET_NODE	0x81	Reset Node → Initialisation → Pre-Operational Same as a power restart: **All unsaved configurations will be lost!**
NMT_COMMAND_RESET_ COMMUNICATION	0x82	Reset Communication → Initialisation → Pre-Operational Recalculates SDO and PDO COB-IDs.

3.11.6 NMT_STATE_t

DESCRIPTION

Network Management (NMT) States

ENUMERATORS

Total enumerators (including undocumented): 4

NMT_STATE_BOOTUP	0x0	Initialisation. Will automatically transition to 'Pre-Operational', then send one heartbeat.
NMT_STATE_STOPPED	0x04	Stopped. No PDO or SDO communication allowed. Only NMT and Heartbeat frames can be used.
NMT_STATE_ OPERATIONAL	0x05	Operational. PDO communication is only allowed in 'Operational' state.
NMT_STATE_PRE_ OPERATIONAL	0x7f	Pre-Operational. PDO Mappings may only be configured while in 'Pre-Operational' state.

LIST OF FIGURES

LIST OF TABLES