# Maxon EPOS CANopen Library Documentation

***Release alpha***

**Bruno Tibério**

**Sep 11, 2018**

# Contents:

**Date** Sep 11, 2018

**Version** 0.1

**Author** Bruno Tibério

**Contact** bruno.tiberio@tecnico.ulisboa.pt

This documentation describes the class Epos developed in Python using CANopen to control the Maxon Motors EPOS 70/10 device.

# CHAPTER 1

## Epos Class description

**class** epos.**Epos**(*_network=None*, *debug=False*)

**begin**(*nodeID*, *_channel='can0'*, *_bustype='socketcan'*, *objectDictionary=None*)

Initialize Epos device

Configure and setup Epos device.

**Parameters**

- **nodeID** – Node ID of the device.
- **channel** (*optional*) – Port used for communication. Default can0
- **bustype** (*optional*) – Port type used. Default socketcan.
- **objectDictionary** (*optional*) – Name of EDS file, if any available.

**Returns** A boolean if all went ok.

**Return type** bool

**changeEposState**(*newState*)

Change EPOS state

Change Epos state using controlWord object

To change Epos state, a write to controlWord object is made. The bit change in controlWord is made as shown in the following table:

| State | LowByte of Controlword [binary] |
|---|---|
| shutdown | 0xxx x110 |
| switch on | 0xxx x111 |
| disable voltage | 0xxx xx0x |
| quick stop | 0xxx x01x |
| disable operation | 0xxx 0111 |
| enable operation | 0xxx 1111 |
| fault reset | 1xxx xxxx |

see section 8.1.3 of firmware for more information

**Parameters** **newState** – string with state witch user want to switch.

**Returns** boolean if all went ok and no error was received.

**Return type** bool

**checkEposState**()

Check current state of Epos

Ask the StatusWord of EPOS and parse it to return the current state of EPOS.

| State | ID | Statusword [binary] |
|---|---|---|
| Start | 0 | x0xx xxx0 x000 0000 |
| Not Ready to Switch On | 1 | x0xx xxx1 x000 0000 |
| Switch on disabled | 2 | x0xx xxx1 x100 0000 |
| ready to switch on | 3 | x0xx xxx1 x010 0001 |
| switched on | 4 | x0xx xxx1 x010 0011 |
| refresh | 5 | x1xx xxx1 x010 0011 |
| measure init | 6 | x1xx xxx1 x011 0011 |
| operation enable | 7 | x0xx xxx1 x011 0111 |
| quick stop active | 8 | x0xx xxx1 x001 0111 |
| fault reaction active (disabled) | 9 | x0xx xxx1 x000 1111 |
| fault reaction active (enabled) | 10 | x0xx xxx1 x001 1111 |
| Fault | 11 | x0xx xxx1 x000 1000 |

see section 8.1.1 of firmware manual for more details.

**Returns** numeric identification of the state or -1 in case of fail.

**Return type** int

**loadConfig**()

Load all configurations

**logDebug**(*message=None*)

Log a message

A wrap around logging. The log message will have the following structure: [class name : function name ] message

the function name will be the caller function retrieved automatically by using sys._getframe(1).f_code.co_name

**Parameters** **message** – a string with the message.

**logInfo**(*message=None*)

Log a message

A wrap around logging. The log message will have the following structure: [class name : function name ] message

**Parameters** **message** – a string with the message.

**printControlWord**(*controlword=None*)

Print the meaning of controlword

Check the meaning of current controlword of device or check the meaning of your own controlword. Usefull to check your own controlword before actually sending it to device.

**Parameters** **controlword** (*optional*) – If None, request the controlword of device.

**printCurrentControlParameters**()
Print the current mode control PI gains

Request current mode control parameter gains from device and print.

**printMotorConfig**()
Print current motor config

Request current motor config and print it

**printOpMode**()
Print current operation mode

**printPositionControlParameters**()
Print position control mode parameters

Request device for the position control mode parameters and prints it.

**printSensorConfig**()
Print current sensor configuration

**printSoftwarePosLimit**()
Print current software position limits

**readControlWord**()
Read ControlWord

Request current controlword from device.

**Returns**

A tupple containing:

**controlword** the current controlword or None if any error.

**Ok** A boolean if all went ok.

**Return type** tupple

**readCurrentControlParameters**()
Read the PI gains used in current control mode

**Returns**

A tupple containing:

**gains** A dictionary with the current pGain and iGain

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readCurrentModeSetting**()
Read current value setted

Asks EPOS for the current value setted in current control mode.

**Returns**

A tupple containing:

**current** value setted.

**Ok** a boolean if sucessfull or not.

**Return type** tupple

**readCurrentValue**()
Read current value

**Returns**

a tupple containing:

**current** current in mA.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readCurrentValueAveraged**()
Read current averaged value

**Returns**

a tupple containing:

**current** current averaged in mA.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readFollowingError**()
Returns the current following error

Read the current following error value which is the difference between atual value and desired value.

**Returns**

a tupple containing:

**followingError** value of actual following error.

**OK** A boolean if all requests went ok or not.

**Return type** tupple

**readMaxFollowingError**()
Read the Max following error

Read the max following error value which is the maximum allowed difference between atual value and desired value in modulus.

**Returns**

a tupple containing:

**maxFollowingError** value of max following error.

**OK** A boolean if all requests went ok or not.

**Return type** tupple

**readMotorConfig**()
Read motor configuration

Read the current motor configuration

Requests from EPOS the current motor type and motor data. The motorConfig is a dictionary containing the following information:

- **motorType** describes the type of motor.
- **currentLimit** - describes the maximum continuous current limit.

- **maxCurrentLimit** - describes the maximum allowed current limit. Usually is set as two times the continuous current limit.
- **polePairNumber** - describes the pole pair number of the rotor of the brushless DC motor.
- **maximumSpeed** - describes the maximum allowed speed in current mode.
- **thermalTimeConstant** - describes the thermal time constant of motor winding is used to calculate the time how long the maximal output current is allowed for the connected motor [100 ms].

If unable to request the configuration or unsucessfull, None and false is returned .

**Returns**

A tupple with:

**motorConfig** A structure with the current configuration of motor

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readObject**(*index*, *subindex*)
Reads an object

Request a read from dictionary object referenced by index and subindex.

**Parameters**

- **index** – reference of dictionary object index
- **subindex** – reference of dictionary object subindex

**Returns** message returned by EPOS or empty if unsucessfull

**Return type** bytes

**readOpMode**()
Read current operation mode

**Returns**

A tupple containing:

**opMode** current opMode or None if request fails

**Ok** A boolean if sucessfull or not

**Return type** tupple

**readPositionControlParameters**()
Read position mode control parameters

Read position mode control PID gains and and feedfoward and acceleration values

**Returns**

A tupple containing:

**posModeParameters** a dictionary containg pGain, iGain, dGain, vFeed and aFeed.

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readPositionModeSetting**()
Reads the setted desired Position

Ask Epos device for demand position object. If a correct request is made, the position is placed in answer. If not, an answer will be empty

**Returns**

A tupple containing:

**position** the demanded position value.

**OK** A boolean if all requests went ok or not.

**Return type** tupple

**readPositionValue**()
Read current position value

**Returns**

a tupple containing:

**position** current position in quadrature counts.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readPositionWindow**()
Read current position Window value.

Position window is the modulus threashold value in which the output is considerated to be achieved.

**Returns**

a tupple containing:

**postionWindow** current position window in quadrature counts.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readPositionWindowTime**()
Read current position Window time value.

Position window time is the minimum time in milliseconds in which the output must be inside the position window for the target is considerated to have been reached.

**Returns**

a tupple containing:

**postionWindowTime** current position window time in milliseconds.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readQuickStopDeceleration**()
Read the quick stop deceleration.

Read deceleration used in fault reaction state.

**Returns**

A tupple containing:

**quickstopDeceleration** The value of deceleration in rpm/s.

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readSensorConfig**()
Read sensor configuration

Requests from EPOS the current sensor configuration. The sensorConfig is an struture containing the following information:

- sensorType - describes the type of sensor.
- pulseNumber - describes the number of pulses per revolution in one channel.
- sensorPolarity - describes the of each sensor.

If unable to request the configuration or unsucessfull, an empty structure is returned. Any error inside any field requests are marked with 'error'.

**Returns**

A tupple containing:

**sensorConfig** A dictionary with the current configuration of the sensor

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readSoftwarePosLimit**()
Read the software position limit

**Returns**

A tupple containing:

**limits** a dictionary containing minPos and maxPos

**OK** A boolean if all went as expected or not.

**Return type** tupple

**readStatusWord**()
Read StatusWord

Request current statusword from device.

**Returns**

A tupple containing:

**statusword** the current statusword or None if any error.

**Ok** A boolean if all went ok.

**Return type** tupple

**readVelocityModeSetting**()
Reads the setted desired velocity

Asks EPOS for the desired velocity value in velocity control mode

**Returns**

A tupple containing:

**velocity** Value setted or None if any error.

**Ok** A boolean if sucessfull or not.

**Return type** tupple

**readVelocityValue**()
Read current velocity value

**Returns**

a tupple containing:

**velocity** current velocity in rpm.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**readVelocityValueAveraged**()
Read current velocity averaged value

**Returns**

a tupple containing:

**velocity** current velocity in rpm.

**Ok** A boolean if all requests went ok or not.

**Return type** tupple

**saveConfig**()
Save all configurrations

**setCurrentControlParameters**(*pGain*, *iGain*)
Set the PI gains used in current control mode

**Parameters**

- **pGain** – Proportional gain.
- **iGain** – Integral gain.

**Returns** A boolean if all went as expected or not.

**Return type** bool

**setCurrentModeSetting**(*current*)
Set desired current

Set the value for desired current in current control mode

**Parameters** **current** – the value to be set [mA]

**Returns** a boolean if sucessfull or not

**Return type** bool

**setMaxFollowingError**(*maxFollowingError*)
Set the Max following error

The Max Following Error is the maximum permissible difference between demanded and actual position at any time of evaluation. It serves as a safety and motion-supervising feature. If the following error becomes too high, this is a sign of something going wrong. Either the drive cannot reach the required speed or it is even blocked.

**Parameters** **maxFollowingError** – The value of maximum following error.

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setMotorConfig**(*motorType*, *currentLimit*, *maximumSpeed*, *polePairNumber*)
Set motor configuration

Sets the configuration of the motor parameters. The valid motor type is:

| motorType | value | Description |
|---|---|---|
| DC motor | 1 | brushed DC motor |
| Sinusoidal PM BL motor | 10 | EC motor sinus commutated |
| Trapezoidal PM BL motor | 11 | EC motor block commutated |

The current limit is the current limit is the maximal permissible continuous current of the motor in mA. Minimum value is 0 and max is hardware dependent.

The output current limit is recommended to be 2 times the continuous current limit.

The pole pair number refers to the number of magnetic pole pairs (number of poles / 2) from rotor of a brushless DC motor.

The maximum speed is used to prevent mechanical destroys in current mode. It is possible to limit the velocity [rpm]

Thermal winding not changed, using default 40ms.

**Parameters**

- **motorType** – value of motor type. see table behind.
- **currentLimit** – max continuous current limit [mA].
- **maximumSpeed** – max allowed speed in current mode [rpm].
- **polePairNumber** – number of pole pairs for brushless DC motors.

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setOpMode**(*opMode*)
Set Operation mode

Sets the operation mode of Epos. OpMode is described as:

| OpMode | Description |
|---|---|
| 6 | Homing Mode |
| 3 | Profile Velocity Mode |
| 1 | Profile Position Mode |
| -1 | Position Mode |
| -2 | Velocity Mode |
| -3 | Current Mode |
| -4 | Diagnostic Mode |
| -5 | MasterEncoder Mode |
| -6 | Step/Direction Mode |

**Parameters** **opMode** – the desired opMode.

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setPositionControlParameters**(*pGain*, *iGain*, *dGain*, *vFeed=0*, *aFeed=0*)
Set position mode control parameters

Set position control PID gains and feedfoward velocity and acceleration values.

**Feedback and Feed Forward**

*PID feedback amplification*

PID stands for Proportional, Integral and Derivative control parameters. They describe how the error signal e is amplified in order to produce an appropriate correction. The goal is to reduce this error, i.e. the deviation between the set (or demand) value and the measured (or actual) value. Low values of control parameters will usually result in a sluggish control behavior. High values will lead to a stiffer control with the risk of overshoot and at too high an amplification, the system may start oscillating.

*Feed-forward*

With the PID algorithms, corrective action only occurs if there is a deviation between the set and actual values. For positioning systems, this means that there always is â€“ in fact, there has to be a position error while in motion. This is called following error. The objective of the feedforward control is to minimize this following error by taking into account the set value changes in advance. Energy is provided in an open-loop controller set-up to compensate friction and for the purpose of mass inertia acceleration. Generally, there are two parameters available in feed-forward. They have to be determined for the specific application and motion task:

- Speed feed-forward gain: This component is multiplied by the demanded speed and compensates for speed-proportional friction.
- Acceleration feed-forward correction: This component is related to the mass inertia of the system and provides sufficient current to accelerate this inertia.

Incorporating the feed forward features reduces the average following error when accelerating and decelerating. By combining a feed-forward control and PID, the PID controller only has to correct the residual error remaining after feed-forward, thereby improving the system response and allowing very stiff control behavior.

According to Position Regulation with Feed Forward the acceleration and velocity feed forward take effect in Profile Position Mode and Homing Mode. There is no influence to all the other operation modes like Position Mode, Profile Velocity Mode, Velocity Mode and Current Mode

**Parameters**

- **pGain** – Proportional gain value
- **iGain** – Integral gain value
- **dGain** – Derivative gain value
- **vFeed** – velocity feed foward gain value. Default to 0
- **aFeed** – acceleration feed foward gain value. Default to 0

**Returns** A boolean if all requests went ok or not

**Return type** OK

**setPositionModeSetting**(*position*)
Sets the desired Position

Ask Epos device to define position mode setting object.

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setPositionWindow**(*positionWindow*)
Set position Window value

Position window is the modulos threashold value in which the output is considerated to be achieved.

**Parameters** **positionWindow** – position window in quadrature counts

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setPositionWindowTime**(*positionWindowTime*)
Set position Window Time value

Position window time is the minimum time in milliseconds in which the output must be inside the position window for the target is considerated to have been reached.

**Parameters** **positionWindowTime** – position window time in milliseconds.

**Returns** A boolean if all requests went ok or not.

**Return type** bool

**setQuickStopDeceleration**(*quickstopDeceleration*)
Set the quick stop deceleration.

The quick stop deceleration defines the deceleration during a fault reaction.

**Parameters** **quicstopDeceleration** – the value of deceleration in rpm/s

**Returns** A boolean if all went as expected or not.

**Return type** bool

**setSensorConfig**(*pulseNumber*, *sensorType*, *sensorPolarity*)
Change sensor configuration

Change the sensor configuration of motor. **Only possible if in disable state** The encoder pulse number should be set to number of counts per revolution of the connected incremental encoder. range : [16 - 7500]

sensor type is described as:

| value | description |
|---|---|
| 1 | Incremental Encoder with index (3-channel) |
| 2 | Incremental Encoder without index (2-channel) |
| 3 | Hall Sensors (Remark: consider worse resolution) |

sensor polarity is set by setting the corresponding bit from the word:

| Bit | description |
|---|---|
| 15-2 | Reserved (0) |
| 1 | Hall sensors polarity 0: normal / 1: inverted |
| 0 | Encoder polarity 0: normal<br>1: inverted (or encoder mounted on motor shaft side) |

**Parameters**

- **pulseNumber** – Number of pulses per revolution.

- **sensorType** – 1,2 or 3 according to the previous table.
- **sensorPolarity** – a value between 0 and 3 describing the polarity of sensors as stated before.

**Returns** A boolean if all went as expected or not.

**Return type** bool

**setSoftwarePosLimit**(*minPos*, *maxPos*)
Set the software position limits

Use encoder readings as limit position for extremes range = [-2147483648 | 2147483647]

**Parameters**

- **minPos** – minimum possition limit
- **maxPos** – maximum possition limit

**Returns** A boolean if all went as expected or not.

**Return type** bool

**setVelocityModeSetting**(*velocity*)
Set desired velocity

Set the value for desired velocity in velocity control mode.

**Parameters** **velocity** – value to be setted.

**Returns** a boolean if sucessfull or not.

**Return type** bool

**writeControlWord**(*controlword*)
Send controlword to device

**Parameters** **controlword** – word to be sent.

**Returns** a boolean if all went ok.

**Return type** bool

**writeObject**(*index*, *subindex*, *data*)
Write an object

Request a write to dictionary object referenced by index and subindex.

**Parameters**

- **index** – reference of dictionary object index
- **subindex** – reference of dictionary object subindex
- **data** – data to be stored

**Returns** boolean if all went ok or not

**Return type** bool

CHAPTER 2

# Indices and tables

- genindex
- search

# Python Module Index

## e

# Index