Maxon Epos Library Documentation

Release 0.1

Bruno Tibério

February 21, 2017

Contents

Date 28 Dec 2016

Version 0.1

Author Bruno Tibério

Contact bruno.tiberio@tecnico.ulisboa.pt

This documentation describes the class Epos of developed for Matlab(tm) to control with the Maxon Motors EPOS 70/10 device.

Contents:

CHAPTER 1

Epos Class

class Epos (*debug_flag*)

EPOS Constructor

If debug flag is active, it reports communications between PC and Epos device

Args:

debug_flag [optional] A boolean. If true, hexadecimal messages are displayed.

Returns: An object of the class Epos.

Examples:

```
epos = Epos();

epos = Epos(1);
```

If debug flag is used the format is changed for 'hex' for easier understanding. See *help format*.

Methods

begin

begin (*devname*, *Baud*)

Connects to Epos device

Establish the connection to EPOS via RS232 connection Sets connected if configuration was sucessfull or not.

Args:

devname Portname for the device (example: '/dev/ttyUSB0').

Baud [optional] baudrate for the communication (default 115200).

Returns:

OK a boolean if all resquests were sucessfull or not.

disconnect

disconnect ()

Disconnect device

Closes epos port and sets format to short (default matlab) if debug flag was used.

writeBYTE

writeBYTE(*myByte*)

Send a byte to epos

Args:

myByte byte to be sent to epos device

Returns:

OK a boolean if write was sucessfull or not

writeWORD

writeWORD(*myWord*)

Send a word (2bytes) to Epos device

Args:

myWord word to be sent to epos device

Returns:

OK a boolean if write was sucessfull or not

readBYTE

readBYTE()

read a byte from epos

Returns:

myByte byte read from epos

OK a boolean if write was sucessfull or not

readWORD

readWORD()

read a word from epos.

Returns:

myWord word read from epos

OK a boolean if write was sucessfull or not

readAnswer

readAnswer()

read an answer from a request

Returns:

answer answer from previous request.

NumWords number of words in answer.

CRCcalc

CRCcalc(*DataArray*, *CRCnumberOfWords*)
calculate 16 bit CRC checksum

CRCcalc calculates the CRC of frame message, wich is made of: [header][DATA][CRC = 0]

For correct crc calculation, the last word (CRC field) must be zero.

Args:

DataArray frame to be checked

CRCnumberOfWords number of words (word = 2 bytes) present in frame

Returns:

CRC_OK a boolean if crc is match or not

CRCCheck

CRCCheck(*DataArray*)
check if crc is correct

CRCCheck extracts the CRC received on message (last word of array) replaces it to zero and calculates the new crc over all array. After it compares value received with the new one calculated.

Args:

DataArray frame to be checked.

Returns:

CRC_OK a boolean if crc is match or not.

sendCom

sendCom(*DataArray*, *numWords*)
send command to EPOS

Send command to EPOS, taking care of all necessary 'ack' and checksum tests.

Args:

DataArray frame to be sent.

numWords number of words present in the frame

Returns:

OK boolean if all went ok or not

readObject

readObject(*index*, *subindex*)
reads an object from dictionary

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

Args:

index reference of dictionary object index

subindex reference of dictionary object subindex

Returns:

answer message returned by EPOS or empty if unsucessfull

OK boolean if all went ok or not

writeObject

writeObject(*index*, *subindex*, *data*)

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

Args:

index reference of dictionary object index

subindex reference of dictionary object subindex

data array to be stored in object

Returns:

answer message returned by EPOS or empty if unsucessfull

OK boolean if all went ok or not

checkError

checkError(*E_error*)

Check if any error occurred in message received

When you send a request to EPOS, the returned response frame, contains a data field wich stores information of errors if any. The corresponding message of error explaining it is printed.

Args:

E_error error data field from EPOS

Returns:

anyError boolean representing if any error happened.

checkEposError

checkEposError()

check if EPOS device is with any fault

Request current ErrorHistory object and list the errors if any present.

Returns:

listErrors cellstr containing errors found or "No Errors"

anyError boolean representing if any error happened.

OK boolean if request was sucessfull or not.

checkEposState

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:

state string with current EPOS state.

ID numeric identification of the state

OK boolean if corrected received status word or not

changeEposState

changeEposState(*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

Args:

state string with state witch we want to switch.

Returns:

OK boolean if all went ok and no error was received.

readStatusWord

readStatusWord()
reads current status word object

Ask Epos device for the current status word object. If a correct request is made, the status word is placed in answer.

Returns:

answer Corresponding status word, 'error' if request was sucessful but an error was returned or empty if request was not sucessfull.

OK A boolean if all requests went ok or not.

printStatusWord

printStatusWord()
Print the meaning of the current status word.

readControlWord

readControlWord()
reads current control word object

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

Returns:

answer Corresponding control word, 'error' if request was sucessful but an error was returned or empty if request was not sucessfull.

OK A boolean if all requests went ok or not.

printControlWord

printControlWord()
Print the meaning of the current control word.

readSWversion

readSWversion()
Reads Software version object

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

Returns:

answer Corresponding software version, 'error' if request was sucessful but an error was returned or empty if request was not sucessfull.

OK A boolean if all requests went ok or not.

readPositionModeSetting

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:

position the demanded position value [qc].

OK A boolean if all requests went ok or not.

setPositionModeSetting

setPositionModeSetting(*position*)
Sets the desired Position

Ask Epos device to define position mode setting object.

Args:

position the demanded position value [qc]

Returns:

OK A boolean if all requests went ok or not.

readVelocityModeSetting

readVelocityModeSetting()
reads the setted desired velocity

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

Returns:

velocity Corresponding device name, 'error' if request was sucessful but an error was returned or empty if request was not sucessfull.

OK A boolean if all requests went ok or not.

setVelocityModeSetting

setVelocityModeSetting(*velocity*)
Sets the desired velocity

Ask Epos device to set velocity mode setting object.

Returns:

OK A boolean if all requests went ok or not.

readCurrentModeSetting

readCurrentModeSetting()
Reads the setted desired current

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

Returns:

current Corresponding device name, 'error' if request was sucessful but an error was returned or empty if request was not sucessfull.

OK A boolean if all requests went ok or not.

setCurrentModeSetting

setCurrentModeSetting(*current*)
Sets the desired current

Ask Epos device to store current mode setting object.

Args:

current current value to be set [mA]

Returns:

OK A boolean if all requests went ok or not.

readOpMode

readOpMode()
Reads the operation mode object

Returns:

opMode current opMode of EPOS.

OK A boolean if all requests went ok or not.

setOpMode

setOpMode(*opMode*)
Set the 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

Args:

opMode the desired opMode.

Returns:

OK A boolean if all requests went ok or not.

printOpMode

printOpMode()
Prints the current operation mode.

setMotorConfig

setMotorConfig(*motorType*, *currentLimit*, *maximumSpeed*, *polePairNumber*)
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.

Args:

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:

OK A boolean if all requests went ok or not.

readMotorConfig

readMotorConfig()
Read the current motor configuration

Requests from EPOS the current motor type and motor data. The motorConfig is an struture 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, an empty structure is returned. Any error inside any field requests are marked with 'error'.

Returns:

motorConfig A structure with the current configuration of motor

OK A boolean if all went as expected or not.

printMotorConfig

printMotorConfig()
Print current Motor configuration

setSensorConfig

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 1: inverted (or encoder mounted on motor shaft side)

Args:

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:

OK A boolean if all went as expected or not.

readSensorConfig

readSensorConfig()
Read the current 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:

sensorConfig A structure with the current configuration of the sensor

OK A boolean if all went as expected or not.

printSensorConfig

printSensorConfig()
Prints the current sensor config.

readCurrentControlParam

readCurrentControlParam()
Read the PI gains used in current control mode

Returns:

currentControlPIgains a structure with P and I gains.

OK A boolean if all went as expected or not.

setCurrentControlParam

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

Args:

pGain Proportional gain.

iGain Integral gain.

Returns:

OK A boolean if all went as expected or not.

printCurrentControlParam

printCurrentControlParam()
Print actual current control mode gains.

readSoftwarePosLimit

readSoftwarePosLimit()
Reads the limits of the software position

Returns:

pos A structure with fields minPos and maxPos

OK A boolean if all requests went ok or not.

setSoftwarePosLimit

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

range : [-2147483648|2147483647]

Args:

minPos minimum limit.

maxPos maximum limit.

Returns:

OK A boolean if all requests went ok or not.

printSoftwarePosLimit

printSoftwarePosLimit()
Prints software position limits.

readMaxProfileVelocity

readMaxProfileVelocity()
Reads the maximum velocity of Profile modes.

This value is used as velocity limit in a position (or velocity) profile mode

Returns:

maxProfileVelocity the value of maximum velocity.

OK A boolean if all requests went ok or not.

setMaxProfileVelocity

setMaxProfileVelocity(*maxProfileVelocity*)
Set the maximum velocity of Profile modes.

This value is used as velocity limit in a position (or velocity) profile mode

Args:

maxProfileVelocity the value of maximum velocity.

Returns:

OK A boolean if all requests went ok or not.

readProfileVelocity

readProfileVelocity()
Read the profile velocity.

The profile velocity is the velocity normally attained at the end of the acceleration ramp during a profiled move [Velocity units]

Returns:

profileVelocity The value of velocity.

OK A boolean if all requests went ok or not.

setProfileVelocity

setProfileVelocity(*profileVelocity*)
Set the profile velocity.

The profile velocity is the velocity normally attained at the end of the acceleration ramp during a profiled move [Velocity units]

Args:

profileVelocity The value of velocity.

Returns:

OK A boolean if all requests went ok or not.

readProfileAcceleration

readProfileAcceleration()
Read the profile acceleration.

Defines the acceleration ramp during a movement.

Returns:

profileAcceleration The value of acceleration.

OK A boolean if all requests went ok or not.

setProfileAcceleration

setProfileAcceleration(*profileAcceleration*)
Set the profile acceleration.

Defines the acceleration ramp during a movement.

Args:

profileVelocity The value of acceleration.

Returns:

OK A boolean if all requests went ok or not.

readProfileDeceleration

readProfileDeceleration()
Read the profile deceleration.

The profile deceleration defines the deceleration ramp during a movement.

Returns:

profileDeceleration The value of deceleration.

OK A boolean if all requests went ok or not.

setProfileDeceleration

setProfileDeceleration(*profileDeceleration*)
Set the profile deceleration.

The profile deceleration defines the deceleration ramp during a movement.

Args:

profileDeceleration The value of deceleration.

Returns:

OK A boolean if all requests went ok or not.

readQuickstopDeceleration

readQuickstopDeceleration()
Read the quickstop deceleration.

Deceleration used in fault reaction state.

Returns:

quickstopDeceleration The value of deceleration.

OK A boolean if all requests went ok or not.

setQuickstopDeceleration

setQuickstopDeceleration(*quickstopDeceleration*)
Set the quickstop deceleration.

The quickstop deceleration defines the deceleration during a fault reaction.

Args:

quickstopDeceleration The value of deceleration.

Returns:

OK A boolean if all requests went ok or not.

readMotionProfileType

readMotionProfileType()
Read the motion profile type.

Motion profile type describes the type of trajectories used in profile modes to generate the paths.

Returns:

motionProfileType 0 if linear ramp, 1 if sin^2 ramp.

OK A boolean if all requests went ok or not.

setMotionProfileType

setMotionProfileType(*motionProfileType*)
Set the motion profile type.

Motion profile type describes the type of trajectories used in profile modes to generate the paths.

Args:

motionProfileType 0 if linear ramp, 1 if sin^2 ramp.

Returns:

OK A boolean if all requests went ok or not.

readPositionProfileConfig

readPositionProfileConfig()
Read all parameters related to position profile configuration mode.

The parameters are stored in a structure with:

- maxFollowingError
- softwarePositionLimit
- maxProfileVelocity
- profileVelocity
- profileAcceleration
- profileDeceleration
- quickstopDeceleration
- motionProfileType

Returns:

positionProfileConfig Struture with all parameters.

OK A boolean if all requests went ok or not.

setPositionProfileConfig

setPositionProfileConfig(*maxFollowingError*, *minPos*, *maxPos*, *maxProfileVelocity*, *profileVelocity*, *profileAcceleration*, *profileDeceleration*, *quickstopDeceleration*, *motionProfileType*)

Set all parameters related to position profile configuration mode.

Args:

maxFollowingError max permissible following error

minPos software limit minimum position

maxPos software limit maximum position

maxProfileVelocity max velocity allowed in profile mode

profileVelocity velocity at end of acceleration ramps

profileAcceleration acceleration value at ramps up

profileDeceleration deceleration value at ramps down

quickstopDeceleration deceleration value at fault reaction

motionProfile type of motion profiles to be generated

Returns:

OK A boolean if all requests went ok or not.

printPositionProfileConfig

printPositionProfileConfig()

Print position profile configuration parameters

readTargetPosition

readTargetPosition()

Read target position value.

The target position if the value in quadrature counts of desired value to be achieved.

Returns:

position Target position value in quadrature counts.

OK A boolean if all requests went ok or not.

setTargetPosition

setTargetPosition(*position*)

Set target position value.

The target position if the value in quadrature counts of desired value to be achieved.

Args:

position Target position value in quadrature counts.

Returns:

OK A boolean if all requests went ok or not.

setPositioningControlOptions

setPositioningControlOptions(*isRelativePos*, *changeNow*, *newSetpoint*)

Set position control options. Position control options change how epos should react to a change in a new target value. The flags are passed to the Controlword. The behavior is described in the following table:

Name	Value	Description
isRelativePos	0	Target position is an absolute value
	1	Target position is a relative value
changeNow	0	Finish the actual positioning and then start next positioning
	1	Interrupt the actual positioning and start the next positioning
newSetpoint	0	Does not assume Target position
	1	Assume Target position

Args:

isRelativePos A boolean if position is relative or absolute.

changeNow A boolean if epos should wait for current movement to end or start changing for the new position.

newSetpoint A boolean if epos should assume target position or not

Returns:

OK A boolean if all requests went ok or not.

haltOperation

haltOperation()

Stop current movement with halt deceleration.

Returns:

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

resumeHaltOpereation

resumeHaltOpereation()

Resumes previous operation before an halt command was issued.

Returns:

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

readVelocityControlParam

readVelocityControlParam()

Reads the parameters PI of the velocity control

Returns:

velocityControlPIgains A structure with pGain and iGain.

OK A boolean if all requests went ok or not.

setVelocityControlParam

setVelocityControlParam(*pGain*, *iGain*)
Set the parameters PI of the velocity control

Args:

pGain the proportional gain.

iGain the integral gain.

Returns:

OK A boolean if all requests went ok or not.

printVelocityControlParam

printVelocityControlParam()
Prints the velocity control parameters PI gains

readPositionControlParam

readPositionControlParam()
Read position control PID gains and feedfoward velocity and acceleration values.

Returns:

positionControlPIDgains A structure with PID gains and feedfoward velocity and acceleration values.

OK A boolean if all requests went ok or not

setPositionControlParam

setPositionControlParam(*pGain*, *iGain*, *dGain*, *vFeed*, *aFeed*)
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 – 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.

Args:

pGain Proportional gain value

iGain Integral gain value

dGain Derivative gain value

vFeed velocity feed foward gain value

aFeed acceleration feed foward gain value

Returns:

OK A boolean if all requests went ok or not

printPositionControlParam

printPositionControlParam()
Print position control PID gains.

readFollowingError

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

Returns:

followingError value of actual following error.

OK A boolean if all requests went ok or not.

readMaxFollowingError

readMaxFollowingError()
Reads the maximum 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.

Returns:

maxFollowingError The value of maximum following error.

OK A boolean if all requests went ok or not.

setMaxFollowingError

setMaxFollowingError(*maxFollowingError*)
Set the maximum 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.

Args:

maxFollowingError The value of maximum following error.

Returns:

OK A boolean if all requests went ok or not.

readPositionValue

readPositionValue()
Read current position value

Returns:

position current position in quadrature counts

OK A boolean if all requests went ok or not.

readPositionWindow

readPositionWindow()
Read current position Window value

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

Returns:

positionWindow current position window in quadrature counts

OK A boolean if all requests went ok or not.

setPositionWindow

setPositionWindow(*positionWindow*)
Set position Window value

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

Args:

positionWindow current position window in quadrature counts

Returns:

OK A boolean if all requests went ok or not.

readPositionWindowTime

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:

positionWindowTime current position window time in milliseconds.

OK A boolean if all requests went ok or not.

setPositionWindowTime

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.

Args:

positionWindowTime current position window time in milliseconds.

Returns:

OK A boolean if all requests went ok or not.

readVelocityValue

readVelocityValue()
Read current velocity value.

Returns:

velocity velocity in rpm.

OK A boolean if all requests went ok or not.

readVelocityValueAveraged

readVelocityValueAveraged()
Read current velocity averege value.

Returns:

velocity velocity in rpm.

OK A boolean if all requests went ok or not.

readCurrentValue

readCurrentValue()
Read current value.

Returns:

current the value of current in mA.

OK A boolean if all requests went ok or not.

readCurrentValueAveraged

readCurrentValueAveraged()
Read current average value.

Returns:

current the value of current in mA.

OK A boolean if all requests went ok or not.

readHomeOffset

readHomeOffset()
Read home offset position value.

Returns:

homeOffset position offset for home value.

OK A boolean if all requests went ok or not.

setHomeOffset

setHomeOffset(*homeOffset*)
Set home offset position value.

Args:

homeOffset position offset for home value.

Returns:

OK A boolean if all requests went ok or not.

save

save()
All parameters of device are stored in non volatile memory. For that, the code "save" is written to this object.

Returns:

OK a boolean if write was sucessfull or not.

W