

Bioscara

Generated by Doxygen 1.9.8

1 Documentation	1
2 Todo List	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 Gripper Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 Gripper()	10
5.1.3 Member Function Documentation	10
5.1.3.1 deinit()	10
5.1.3.2 disable()	10
5.1.3.3 enable()	11
5.1.3.4 init()	11
5.1.3.5 setPosition()	11
5.1.4 Member Data Documentation	11
5.1.4.1 pwm	11
5.2 Joint Class Reference	12
5.2.1 Detailed Description	13
5.2.2 Member Enumeration Documentation	14
5.2.2.1 stp_reg_t	14
5.2.3 Constructor & Destructor Documentation	15
5.2.3.1 Joint()	15
5.2.4 Member Function Documentation	15
5.2.4.1 checkCom()	15
5.2.4.2 checkOrientation()	15
5.2.4.3 deinit()	15
5.2.4.4 disable()	15
5.2.4.5 disableCL()	15
5.2.4.6 enable()	15
5.2.4.7 enableStallguard()	16
5.2.4.8 getFlags()	16
5.2.4.9 getIsHomed() [1/2]	16
5.2.4.10 getIsHomed() [2/2]	17
5.2.4.11 getIsSetup() [1/2]	17
5.2.4.12 getIsSetup() [2/2]	17
5.2.4.13 getPosition()	17
5.2.4.14 getStall()	17

5.2.4.15 getVelocity()	18
5.2.4.16 home()	18
5.2.4.17 init()	18
5.2.4.18 isHomed()	18
5.2.4.19 isSetup()	19
5.2.4.20 moveSteps()	19
5.2.4.21 printInfo()	19
5.2.4.22 read()	19
5.2.4.23 setBrakeMode()	20
5.2.4.24 setDriveCurrent()	20
5.2.4.25 setHoldCurrent()	20
5.2.4.26 setPosition()	21
5.2.4.27 setVelocity()	21
5.2.4.28 stop()	21
5.2.4.29 write()	21
5.2.5 Member Data Documentation	22
5.2.5.1 address	22
5.2.5.2 flags	22
5.2.5.3 gearRatio	23
5.2.5.4 handle	23
5.2.5.5 ishomed	23
5.2.5.6 issetup	23
5.2.5.7 name	23
5.2.5.8 offset	23
5.3 Joint_comms Class Reference	23
5.3.1 Detailed Description	25
5.3.2 Constructor & Destructor Documentation	25
5.3.2.1 Joint_comms()	25
5.3.2.2 ~Joint_comms()	25
5.3.3 Member Function Documentation	25
5.3.3.1 addJoint()	25
5.3.3.2 checkOrientations() [1/2]	26
5.3.3.3 checkOrientations() [2/2]	26
5.3.3.4 deinit()	26
5.3.3.5 disableCLs()	27
5.3.3.6 disables()	27
5.3.3.7 enables() [1/2]	27
5.3.3.8 enables() [2/2]	27
5.3.3.9 enableStallguards()	28
5.3.3.10 getPositions()	28
5.3.3.11 getVelocities()	29
5.3.3.12 home()	29

5.3.3.13 init()	29
5.3.3.14 setBrakeModes()	30
5.3.3.15 setDriveCurrents() [1/2]	30
5.3.3.16 setDriveCurrents() [2/2]	30
5.3.3.17 setHoldCurrents() [1/2]	31
5.3.3.18 setHoldCurrents() [2/2]	31
5.3.3.19 setPositions()	32
5.3.3.20 setVelocities()	32
5.3.3.21 stops()	32
5.3.4 Member Data Documentation	33
5.3.4.1 joints	33
5.4 RPI_PWM Class Reference	33
5.4.1 Detailed Description	34
5.4.2 Constructor & Destructor Documentation	34
5.4.2.1 ~RPI_PWM()	34
5.4.3 Member Function Documentation	34
5.4.3.1 disable()	34
5.4.3.2 enable()	34
5.4.3.3 setDutyCycle()	34
5.4.3.4 setDutyCycleNS()	34
5.4.3.5 setPeriod()	34
5.4.3.6 start()	35
5.4.3.7 stop()	35
5.4.3.8 writeSYS()	35
5.4.4 Member Data Documentation	35
5.4.4.1 chippath	35
5.4.4.2 per	35
5.4.4.3 pwmpath	35
6 File Documentation	37
6.1 Arduino/joint/configuration.h File Reference	37
6.1.1 Detailed Description	38
6.1.2 Macro Definition Documentation	38
6.1.2.1 ADR	38
6.1.2.2 MAXACCEL	38
6.1.2.3 MAXVEL	38
6.2 configuration.h	39
6.3 Arduino/joint/joint.h File Reference	39
6.3.1 Detailed Description	41
6.3.2 Macro Definition Documentation	41
6.3.2.1 ACK	41
6.3.2.2 DUMP_BUFFER	41

6.3.2.3 MAX_BUFFER	42
6.3.2.4 NACK	42
6.3.2.5 RFLAGS_SIZE	42
6.3.3 Enumeration Type Documentation	42
6.3.3.1 stp_reg_t	42
6.3.4 Function Documentation	43
6.3.4.1 readValue()	43
6.3.4.2 writeValue()	43
6.4 joint.h	44
6.5 Arduino/joint/joint.ino File Reference	45
6.5.1 Detailed Description	46
6.5.2 Macro Definition Documentation	46
6.5.2.1 J1	46
6.5.3 Function Documentation	47
6.5.3.1 loop()	47
6.5.3.2 receiveEvent()	47
6.5.3.3 requestEvent()	47
6.5.3.4 setup()	48
6.5.3.5 stepper_receive_handler()	48
6.5.3.6 stepper_request_handler()	48
6.5.4 Variable Documentation	48
6.5.4.1 reg	48
6.5.4.2 rx_buf	49
6.5.4.3 rx_data_ready	49
6.5.4.4 rx_length	49
6.5.4.5 stepper	49
6.5.4.6 tx_buf	49
6.5.4.7 tx_data_ready	49
6.5.4.8 tx_length	49
6.6 docs/DOCS_README.md File Reference	49
6.7 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/common.h File Reference	49
6.7.1 Detailed Description	50
6.7.2 Macro Definition Documentation	51
6.7.2.1 DUMP_BUFFER	51
6.8 common.h	51
6.9 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mGripper.h File Reference	51
6.9.1 Detailed Description	52
6.10 mGripper.h	53
6.11 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.h File Reference	53
6.11.1 Detailed Description	55
6.11.2 Macro Definition Documentation	55
6.11.2.1 ENCODER2JOINTANGLE	55

6.11.2.2 JOINT2ENCODERANGLE	55
6.12 mJoint.h	55
6.13 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.hpp File Reference	57
6.13.1 Detailed Description	58
6.14 mJoint.hpp	59
6.15 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJointCom.h File Reference	59
6.15.1 Detailed Description	60
6.16 mJointCom.h	61
6.17 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/ul2C.h File Reference	62
6.17.1 Detailed Description	64
6.17.2 Macro Definition Documentation	64
6.17.2.1 ACK	64
6.17.2.2 MAX_BUFFER	64
6.17.2.3 NACK	64
6.17.2.4 RFLAGS_SIZE	65
6.17.3 Function Documentation	65
6.17.3.1 closeI2CDevHandle()	65
6.17.3.2 openI2CDevHandle()	65
6.17.3.3 readFromI2CDev()	65
6.17.3.4 writeToI2CDev()	66
6.18 ul2C.h	66
6.19 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/uPWM.h File Reference	67
6.19.1 Detailed Description	68
6.20 uPWM.h	69
6.21 ROS2/ros2_scara_ws/src/joint_communication/src/joint_comm_node.cpp File Reference	70
6.21.1 Function Documentation	70
6.21.1.1 INT_handler()	70
6.21.1.2 main()	71
6.21.2 Variable Documentation	71
6.21.2.1 _Gripper	71
6.21.2.2 _Joints	71
6.22 ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp File Reference	71
6.23 ROS2/ros2_scara_ws/src/joint_communication/src/mJoint.cpp File Reference	72
6.24 ROS2/ros2_scara_ws/src/joint_communication/src/mJointCom.cpp File Reference	72
6.25 ROS2/ros2_scara_ws/src/joint_communication/src/ul2C.cpp File Reference	73
6.25.1 Function Documentation	74
6.25.1.1 closeI2CDevHandle()	74
6.25.1.2 openI2CDevHandle()	74
6.25.1.3 readFromI2CDev()	74
6.25.1.4 writeToI2CDev()	76

Chapter 1

Documentation

This documentation currently documents the following:

- The joint firmware in the [/Arduino](#) directory
- The interfacing library used for communicating with the joints in the [/ROS2](#) directory.

Chapter 2

Todo List

- Member **Joint::read** (const stp_reg_t reg, T &data, u_int8_t &flags) .
 - Implement a return code for read only functions
 - Implement clearStall function
- Member **Joint_comms::addJoint** (const int address, const std::string name, const float gearRatio, const float offset) .
 - Measure joint ranges
 - Investigate if possible to make independent of homing
- Member **Joint_comms::checkOrientations** (std::vector< float > angle_v) .
 - Only execute if not performed before
 - save in private flag and inhibit movement if this has not been executed.
- Member **loop** (void) .
 - why are BIT2 and BIT3 constantly checked and set? Would it be sufficient to do this only invoking the actual functions?

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Gripper	Gripper object to interact with the robot gripper	9
Joint	Representing a single joint on the I2C bus	12
Joint_comms	Communication object for all joints	23
RPI_PWM	PWM class for the Raspberry PI 4 and 5	33

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Arduino/joint/ configuration.h	
Configuration definitions for Joint 1 to Joint 4	37
Arduino/joint/ joint.h	
Joint firmware header	39
Arduino/joint/ joint.ino	
Joint firmware	45
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ common.h	
A file containing utility macros and functions	49
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ mGripper.h	
File containing the Gripper class	51
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ mJoint.h	
File including the Joint class	53
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ mJoint.hpp	
Templated functions for the Joint class	57
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ mJointCom.h	
File containing the Joint_comms class	59
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ ul2C.h	
Low level utility for I2C communication on Raspberry Pi using I2C library	62
ROS2/ros2_scarl_ws/src/joint_communication/include/joint_communication/ uPWM.h	
Includes source code for Hardware PWM generation on Raspberry Pi 4	67
ROS2/ros2_scarl_ws/src/joint_communication/src/ joint_comm_node.cpp	70
ROS2/ros2_scarl_ws/src/joint_communication/src/ mGripper.cpp	71
ROS2/ros2_scarl_ws/src/joint_communication/src/ mJoint.cpp	72
ROS2/ros2_scarl_ws/src/joint_communication/src/ mJointCom.cpp	72
ROS2/ros2_scarl_ws/src/joint_communication/src/ ul2C.cpp	73

Chapter 5

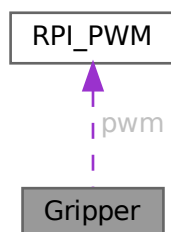
Class Documentation

5.1 Gripper Class Reference

[Gripper](#) object to interact with the robot gripper.

```
#include <mGripper.h>
```

Collaboration diagram for Gripper:



Public Member Functions

- [Gripper](#) (void)
- int [init](#) (void)
Placeholder, does nothing.
- int [deinit](#) (void)
Placeholder, does nothing.
- int [enable](#) (void)
Prepares the servo for use.
- int [disable](#) (void)
Disables the servo.
- int [setPosition](#) (float width)
Sets the gripper width in mm from the closed position.

Private Attributes

- [RPI_PWM pwm](#)

5.1.1 Detailed Description

[Gripper](#) object to interact with the robot gripper.

The gripper must be a PWM controlled servo.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Gripper()

```
Gripper::Gripper (  
    void )
```

5.1.3 Member Function Documentation

5.1.3.1 deinit()

```
int Gripper::deinit (  
    void )
```

Placeholder, does nothing.

Returns

0

5.1.3.2 disable()

```
int Gripper::disable (  
    void )
```

Disables the servo.

Stops the servo and disables the PWM generation.

Returns

non-zero error code.

5.1.3.3 enable()

```
int Gripper::enable (
    void )
```

Prepares the servo for use.

Starts the PWM generation but does not set a position. Must be called before a position is set. The PWM pin is GPIO18. PWM chip is 0, channel 0. *

Returns

non-zero error code.

5.1.3.4 init()

```
int Gripper::init (
    void )
```

Placeholder, does nothing.

Returns

0

5.1.3.5 setPosition()

```
int Gripper::setPosition (
    float width )
```

Sets the gripper width in mm from the closed position.

Arguments outside the allowed range are bounded to limit.

Parameters

<i>width</i>	width in mm. 30 - 85 mm are currently allowed. With a new gripper this should be changed.
--------------	---

5.1.4 Member Data Documentation

5.1.4.1 pwm

```
RPI_PWM Gripper::pwm [private]
```

The documentation for this class was generated from the following files:

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mGripper.h
- ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp

5.2 Joint Class Reference

Representing a single joint on the I2C bus.

```
#include <mJoint.h>
```

Public Member Functions

- [Joint](#) (const int [address](#), const std::string [name](#), const float [gearRatio](#), const float [offset](#))
- int [init](#) (void)
- int [deinit](#) (void)
- int [printInfo](#) (void)
- int [getPosition](#) (float &angle)
- int [setPosition](#) (float angle)
- int [getVelocity](#) (float °ps)
- int [setVelocity](#) (float degps)
- int [checkOrientation](#) (float angle=10.0)
- int [enable](#) (u_int8_t driveCurrent, u_int8_t holdCurrent)
Initialize the joint and engages motor.
- int [disable](#) (void)
disengages the joint motor without closing i2c handle
- int [home](#) (u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current)
Homes the motor.
- int [stop](#) (bool mode)
Stops the motor.
- int [disableCL](#) (void)
Disables the Closed-Loop PID Controller.
- int [setDriveCurrent](#) (u_int8_t current)
Set the Drive Current.
- int [setHoldCurrent](#) (u_int8_t current)
Set the Hold Current.
- int [setBrakeMode](#) (u_int8_t mode)
Set Brake Mode.
- int [getStall](#) (u_int8_t &stall)
checks if the motor is stalled
- int [enableStallguard](#) (u_int8_t sensitivity)
Enable encoder stall detection. A detected stall can be reset by homing.
- int [getIsHomed](#) (u_int8_t &homed)
retrieves the status flags from the joint and checks if the joint is homed.
- int [getIsHomed](#) (void)
retrieves the status flags from the joint.
- bool [isHomed](#) (void)
Get the isHomed state variable saved locally.
- int [getIsSetup](#) (u_int8_t &setup)
checks if the joint is setup from the joint
- int [getIsSetup](#) (void)
checks if the joint is setup from the joint
- bool [isSetup](#) (void)
- int [moveSteps](#) (int32_t steps)
- int [checkCom](#) (void)
- u_int8_t [getFlags](#) (void)

Public Attributes

- `std::string name`

Private Types

- enum `stp_reg_t` {
`PING` = 0x0f , `SETUP` = 0x10 , `SETRPM` = 0x11 , `GETDRIVERRPM` = 0x12 ,
`MOVESTEPS` = 0x13 , `MOVEANGLE` = 0x14 , `MOVETOANGLE` = 0x15 , `GETMOTORSTATE` = 0x16 ,
`RUNCOTINOUS` = 0x17 , `ANGLEMOVED` = 0x18 , `SETCURRENT` = 0x19 , `SETHOLDCURRENT` = 0x1A ,
`SETMAXACCELERATION` = 0x1B , `SETMAXDECELERATION` = 0x1C , `SETMAXVELOCITY` = 0x1D ,
`ENABLESTALLGUARD` = 0x1E ,
`DISABLESTALLGUARD` = 0x1F , `CLEARSTALL` = 0x20 , `ISSTALLED` = 0x21 , `SETBRAKEMODE` = 0x22 ,
`ENABLEPID` = 0x23 , `DISABLEPID` = 0x24 , `ENABLECLOSEDLOOP` = 0x25 , `DISABLECLOSEDLOOP` = 0x26 ,
`SETCONTROLTHRESHOLD` = 0x27 , `MOVETOEND` = 0x28 , `STOP` = 0x29 , `GETPIDERROR` = 0x2A ,
`CHECKORIENTATION` = 0x2B , `GETENCODERRPM` = 0x2C , `HOME` = 0x2D , `ISHOMED` = 0x2E ,
`ISSETUP` = 0x2F }

register and command definitions

Private Member Functions

- template<typename T >
`int read` (const `stp_reg_t reg`, T &data, `u_int8_t &flags`)
Wrapper function to request data from the I2C slave.
- template<typename T >
`int write` (const `stp_reg_t reg`, T data, `u_int8_t &flags`)
Wrapper function to send command to the I2C slave.

Private Attributes

- `u_int8_t flags` = 0x00
State flags transmitted with every I2C transaction.
- `u_int8_t ishomed` = 0
flag if homed
- `u_int8_t issetup` = 0
flag is setup
- `int address`
I2C adress.
- `float gearRatio` = 1
gear ratio from encoder units to joint units
- `float offset` = 0
offset in degrees or mm from encoder zero to joint zero.
- `int handle` = -1
I2C bus handle.

5.2.1 Detailed Description

Representing a single joint on the I2C bus.

5.2.2 Member Enumeration Documentation

5.2.2.1 stp_reg_t

```
enum Joint::stp_reg_t [private]
```

register and command definitions

a register can be read (R) or written (W), each register has a size in bytes. The payload can be split into multiple values or just be a single value. Note that not all functions are implemented.

Enumerator

PING	R; Size: 1; [(char) ACK].
SETUP	W; Size: 2; [(uint8) holdCurrent, (uint8) driveCurrent].
SETRPM	W; Size: 4; [(float) RPM].
GETDRIVERRPM	
MOVESTEPS	W; Size: 4; [(int32) steps].
MOVEANGLE	
MOVETOANGLE	W; Size: 4; [(float) degrees].
GETMOTORSTATE	
RUNCOTINOUS	
ANGLEMOVED	R; Size: 4; [(float) degrees].
SETCURRENT	W; Size: 1; [(uint8) driveCurrent].
SETHOLDCURRENT	W; Size: 1; [(uint8) holdCurrent].
SETMAXACCELERATION	
SETMAXDECELERATION	
SETMAXVELOCITY	
ENABLESTALLGUARD	W; Size: 1; [(uint8) threshold].
DISABLESTALLGUARD	
CLEARSTALL	
ISSTALLED	R; Size: 1; [(uint8) isStalled].
SETBRAKEMODE	W; Size: 1; [(uint8) mode].
ENABLEPID	
DISABLEPID	
ENABLECLOSEDLOOP	
DISABLECLOSEDLOOP	W; Size: 1; [(uint8) 0].
SETCONTROLTHRESHOLD	
MOVETOEND	
STOP	W; Size: 1; [(uint8) mode].
GETPIDERROR	
CHECKORIENTATION	W; Size: 4; [(float) degrees].
GETENCODERRPM	R; Size: 4; [(float) RPM].
HOME	W; Size: 4; [(uint8) current, (int8) sensitivity, (uint8) speed, (uint8) direction].
ISHOMED	R; Size: 1; [(uint8) isStalled].
ISSETUP	R; Size: 1; [(uint8) isStalled].

5.2.3 Constructor & Destructor Documentation

5.2.3.1 Joint()

```
Joint::Joint (
    const int address,
    const std::string name,
    const float gearRatio,
    const float offset )
```

5.2.4 Member Function Documentation

5.2.4.1 checkCom()

```
int Joint::checkCom (
    void )
```

5.2.4.2 checkOrientation()

```
int Joint::checkOrientation (
    float angle = 10.0 )
```

5.2.4.3 deinit()

```
int Joint::deinit (
    void )
```

5.2.4.4 disable()

```
int Joint::disable (
    void )
```

disengages the joint motor without closing i2c handle

Returns

error code.

5.2.4.5 disableCL()

```
int Joint::disableCL (
    void )
```

Disables the Closed-Loop PID Controller.

Returns

error code.

5.2.4.6 enable()

```
int Joint::enable (
    u_int8_t driveCurrent,
    u_int8_t holdCurrent )
```

Initialize the joint and engages motor.

Parameters

<i>driveCurrent</i>	drive current in 0-100 % of 2.5A output (check uStepper doc.)
<i>holdCurrent</i>	hold current in 0-100 % of 2.5A output (check uStepper doc.)

Returns

error code.

5.2.4.7 enableStallguard()

```
int Joint::enableStallguard (
    u_int8_t sensitivity )
```

Enable encoder stall detection. A detected stall can be reset by homing.

Parameters

<i>sensitivity</i>	Encoder stalldetect sensitivity - From -100 to 10 where lower number is less sensitive and higher is more sensitive
--------------------	---

5.2.4.8 getFlags()

```
u_int8_t Joint::getFlags (
    void )
```

get driver state flags

Returns

flags.

5.2.4.9 getIsHomed() [1/2]

```
int Joint::getIsHomed (
    u_int8_t & homed )
```

retrieves the status flags from the joint and checks if the joint is homed.

Parameters

<i>homed</i>	not homed: 0, homed: 1
--------------	------------------------

Returns

error code.

5.2.4.10 `getIsHomed()` [2/2]

```
int Joint::getIsHomed (
    void )
```

retrieves the status flags from the joint.

This overload does not return the value of isHomed variable, use [Joint::isHomed\(\)](#) instead.

Returns

error code.

5.2.4.11 `getIsSetup()` [1/2]

```
int Joint::getIsSetup (
    u_int8_t & setup )
```

checks if the joint is setup from the joint

Parameters

<i>setup</i>	not setup: 0, setup: 1
--------------	------------------------

Returns

error code.

5.2.4.12 `getIsSetup()` [2/2]

```
int Joint::getIsSetup (
    void )
```

checks if the joint is setup from the joint

Returns

error code.

5.2.4.13 `getPosition()`

```
int Joint::getPosition (
    float & angle )
```

5.2.4.14 `getStall()`

```
int Joint::getStall (
    u_int8_t & stall )
```

checks if the motor is stalled

Parameters

<i>stall</i>	not stalled: 0, stalled: 1
--------------	----------------------------

Returns

error code.

5.2.4.15 getVelocity()

```
int Joint::getVelocity (
    float & degps )
```

5.2.4.16 home()

```
int Joint::home (
    u_int8_t direction,
    u_int8_t rpm,
    u_int8_t sensitivity,
    u_int8_t current )
```

Homes the motor.

Parameters

<i>direction</i>	CCW: 0, CW: 1.
<i>rpm</i>	speed of motor in rpm > 10.
<i>sensitivity</i>	Encoder pid error threshold 0 to 255.
<i>current</i>	homeing current, determines how easy it is to stop the motor and thereby provoke a stall

Returns

error code.

5.2.4.17 init()

```
int Joint::init (
    void )
```

5.2.4.18 isHomed()

```
bool Joint::isHomed (
    void )
```

Get the isHomed state variable saved locally.

To retrieve the actual state call [Joint::getIsHomed\(\)](#)

Returns

local isHomed state variable.

5.2.4.19 isSetup()

```
bool Joint::isSetup (
    void )
```

Returns

the isSetup state variable.

5.2.4.20 moveSteps()

```
int Joint::moveSteps (
    int32_t steps )
```

5.2.4.21 printInfo()

```
int Joint::printInfo (
    void )
```

5.2.4.22 read()

```
template<typename T >
int Joint::read (
    const stp_reg_t reg,
    T & data,
    u_int8_t & flags ) [private]
```

Wrapper function to request data from the I2C slave.

Allocates a buffer of size `sizeof(T) + RFLAGS_SIZE`. invokes [readFromI2CDev\(\)](#), and copies the received payload to *data* and the transmisison flags to *flags*. See [Joint::flags](#) for details.

- Todo**
- Implement a return code for read only functions
 - Implement clearStall function

Template Parameters

<i>T</i>	Datatype of value to be transmitted
----------	-------------------------------------

Parameters

<i>reg</i>	stp_reg_t register to read
<i>data</i>	reference to store payload.
<i>flags</i>	reference to a byte which stores the return flags

Returns

0 on OK, negative on error

5.2.4.23 setBrakeMode()

```
int Joint::setBrakeMode (
    u_int8_t mode )
```

Set Brake Mode.

Parameters

<i>mode</i>	Freewheel: 0, Coolbrake: 1, Hardbrake: 2
-------------	--

Returns

error code.

5.2.4.24 setDriveCurrent()

```
int Joint::setDriveCurrent (
    u_int8_t current )
```

Set the Drive Current.

Warning

This function is unreliable and not well tested. Use [enable\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.2.4.25 setHoldCurrent()

```
int Joint::setHoldCurrent (
    u_int8_t current )
```

Set the Hold Current.

Warning

This function is unreliable and not well tested. Use [enable\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.2.4.26 setPosition()

```
int Joint::setPosition (
    float angle )
```

5.2.4.27 setVelocity()

```
int Joint::setVelocity (
    float degps )
```

5.2.4.28 stop()

```
int Joint::stop (
    bool mode )
```

Stops the motor.

Note

When stopping the motor in soft mode, wait sufficiently long until the motor has stopped. Since the [stop\(\)](#) function in the motor controller is blocking. Continuously checking the busy flag also might interfere with the [stop\(\)](#) function on the controller side.

Parameters

<i>mode</i>	Hard: 0, Soft: 1
-------------	------------------

Returns

error code.

5.2.4.29 write()

```
template<typename T >
int Joint::write (
    const stp_reg_t reg,
    T data,
    u_int8_t & flags ) [private]
```

Wrapper function to send command to the I2C slave.

Allocates a buffer of size `sizeof(T) + RFLAGS_SIZE`. Copyies *data* to the buffer and invokes `writeToI2CDev()`. The flags received from the transaction are copied to *flags*. The flags are described in `Joint::read()`.

Template Parameters

<i>T</i>	Datatype of value to be transmitted
----------	-------------------------------------

Parameters

<i>reg</i>	stp_reg_t command to execute
<i>data</i>	payload to transmit. It is the users responsibility to populate the right amount of data for the relevant register
<i>flags</i>	reference to a byte which stores the return flags

Returns

0 on OK, negative on error

5.2.5 Member Data Documentation

5.2.5.1 address

```
int Joint::address [private]
```

I2C adress.

5.2.5.2 flags

```
u_int8_t Joint::flags = 0x00 [private]
```

State flags transmitted with every I2C transaction.

The transmission flags purpose are to transmit the joints current state. Note: They can not be used as error indication of the execution of a transmitted write command, since commands are executed after the I2C transaction is completed. The status flags are one byte with following structure:

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
reserved	reserved	reserved	reserved	SETUP	HOMED	BUSY	STALL

STALL is set if a stall from the stall detection is sensed and the joint is stopped. The flag is cleared when the joint is homed.

BUSY is set if the slave is busy processing a previous command.

HOMED is set if the joint is homed. Movement is only allowed if this flag is clear

SETUP is set if the joint is setup after calling `Joint::enable()`

5.2.5.3 gearRatio

```
float Joint::gearRatio = 1 [private]
```

gear ratio from encoder units to joint units

5.2.5.4 handle

```
int Joint::handle = -1 [private]
```

I2C bus handle.

5.2.5.5 ishomed

```
u_int8_t Joint::ishomed = 0 [private]
```

flag if homed

5.2.5.6 issetup

```
u_int8_t Joint::issetup = 0 [private]
```

flag is setup

5.2.5.7 name

```
std::string Joint::name
```

5.2.5.8 offset

```
float Joint::offset = 0 [private]
```

offset in degrees or mm from encoder zero to joint zero.

The documentation for this class was generated from the following files:

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/[mJoint.h](#)
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/[mJoint.hpp](#)
- ROS2/ros2_scara_ws/src/joint_communication/src/[mJoint.cpp](#)

5.3 Joint_comms Class Reference

Communication object for all joints.

```
#include <mJointCom.h>
```

Public Member Functions

- [Joint_comms](#) (void)
- [~Joint_comms](#) ()
- int [init](#) (void)
Initializes all joints.
- int [deinit](#) (void)
Frees all joints from the I2C bus.
- void [addJoint](#) (const int address, const std::string name, const float gearRatio, const float offset)
add Joints.
- int [enables](#) (std::vector< u_int8_t > driveCurrent_v, std::vector< u_int8_t > holdCurrent_v)
Engages the joints.
- int [enables](#) (u_int8_t driveCurrent, u_int8_t holdCurrent)
Engages the joints with the same current settings for all joints.
- int [disables](#) (void)
Disengages the joint without closing i2c handle.
- int [home](#) (std::string name, u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current)
Executes the homing sequence of a joint.
- int [getPosition](#) (std::vector< float > &angle_v)
Get the positions of all joints.
- int [setPosition](#) (std::vector< float > angle_v)
Set the positions of all joints.
- int [getVelocities](#) (std::vector< float > °ps_v)
Get the velocities of all joints.
- int [setVelocities](#) (std::vector< float > degps_v)
Set the velocities of all joints.
- int [checkOrientations](#) (std::vector< float > angle_v)
Sequentially checks the orientations of each joint.
- int [checkOrientations](#) (float angle=10.0)
Overload to use standard angle of 10 degrees.
- int [stops](#) (bool mode)
Stops the motors.
- int [disableCLs](#) (void)
Disables the Closed-Loop PID Controllers.
- int [setDriveCurrents](#) (std::vector< u_int8_t > current)
Set the drive Currents.
- int [setDriveCurrents](#) (u_int8_t current)
Overload to set all drive currents to the same value.
- int [setHoldCurrents](#) (std::vector< u_int8_t > current)
Set the Hold Currents.
- int [setHoldCurrents](#) (u_int8_t current)
Overload to set all hold currents to the same value.
- int [setBrakeModes](#) (u_int8_t mode)
Set Brake Modes.
- int [enableStallguards](#) (std::vector< u_int8_t > thresholds)
Enable encoder stall detection.

Public Attributes

- std::vector< [Joint](#) > [joints](#)
Internal vector storing the [Joint](#) objects.

5.3.1 Detailed Description

Communication object for all joints.

Class handling interfacing with the joints.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Joint_comms()

```
Joint_comms::Joint_comms (
    void )
```

5.3.2.2 ~Joint_comms()

```
Joint_comms::~~Joint_comms ( )
```

5.3.3 Member Function Documentation

5.3.3.1 addJoint()

```
void Joint_comms::addJoint (
    const int address,
    const std::string name,
    const float gearRatio,
    const float offset )
```

add Joints.

Appends a joint to internal vector.

Parameters

<i>addresses</i>	1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4
<i>names</i>	string device name for output logs
<i>gearRatio</i>	gear ratio of joint. This is used to transform position and velocity commands in joint units to the stepper units. Signed: sign depends if homed CW or CCW. J1: 35; J2: -360/4 (4 mm per revolution); J3: 24; J4: 12;
<i>offset</i>	offset between encoder zero and joint zero (in joint units). J1: TBD; J2: -TBD (negative because homed at top); J3: TBD; J4: TBD;

- Todo**
- Measure joint ranges
 - Investigate if possible to make independent of homing

5.3.3.2 checkOrientations() [1/2]

```
int Joint_comms::checkOrientations (
    float angle = 10.0 )
```

Overload to use standard angle of 10 degrees.

Returns

error code.

5.3.3.3 checkOrientations() [2/2]

```
int Joint_comms::checkOrientations (
    std::vector< float > angle_v )
```

Sequentially checks the orientations of each joint.

This function must only be called after the joint has been powered down. This function must be called after the joint has been enabled with [enables\(\)](#) and before any movement.

Parameters

<i>angle_v</i>	vector of degrees to rotate to check the orientation. Should be small values of a few degrees.
----------------	--

Returns

error code.

- Todo**
- Only execute if not performed before
 - save in private flag and inhibit movement if this has not been executed.

5.3.3.4 deinit()

```
int Joint_comms::deinit (
    void )
```

Frees all joints from the I2C bus.

Deinitializes all joints by removing them from the I2C bus.

Returns

0 on success, non-zero otherwise

5.3.3.5 disableCLs()

```
int Joint_comms::disableCLs (
    void )
```

Disables the Closed-Loop PID Controllers.

Returns

error code.

5.3.3.6 disables()

```
int Joint_comms::disables (
    void )
```

Disengages the joint without closing i2c handle.

Call this function when the joint should be in freedrive mode.

Returns

error code.

5.3.3.7 enables() [1/2]

```
int Joint_comms::enables (
    std::vector< u_int8_t > driveCurrent_v,
    std::vector< u_int8_t > holdCurrent_v )
```

Engages the joints.

Sets the drive and hold currents for each joint and engages the motor. Currents are in percent of driver max. output (2.5A, check with datasheet)

Parameters

<i>driveCurrent_v</i>	vector of drive currents 0-100. the i'th vector entry sets the current for the i'th added joint.
<i>holdCurrent_v</i>	vector of hold currents 0-100. the i'th vector entry sets the current for the i'th added joint.

Returns

error code.

5.3.3.8 enables() [2/2]

```
int Joint_comms::enables (
```

```

    u_int8_t driveCurrent,
    u_int8_t holdCurrent )

```

Engages the joints with the same current settings for all joints.

In this overload the same drive and hold currents are written to every joint.

Parameters

<i>driveCurrent</i>	drive current 0-100.
<i>holdCurrent</i>	hold current 0-100.

Returns

error code.

5.3.3.9 enableStallguards()

```

int Joint_comms::enableStallguards (
    std::vector< u_int8_t > thresholds )

```

Enable encoder stall detection.

If the PID error exceeds the set threshold a stall is triggered and the motor disabled. A detected stall can be reset by homing.

Parameters

<i>thresholds</i>	Vector of thresholds. 0 - 255 where lower is more sensitive.
-------------------	--

5.3.3.10 getPositions()

```

int Joint_comms::getPositions (
    std::vector< float > & angle_v )

```

Get the positions of all joints.

The current positions of all joints are returned. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

<i>angle_v</i>	Reference to allocated vector of appropriate size to hold all joint positions.
----------------	--

Returns

error code.

5.3.3.11 getVelocities()

```
int Joint_comms::getVelocities (
    std::vector< float > & degps_v )
```

Get the velocities of all joints.

The current velocities of all joints are returned. The units are degrees/s and mm/s for revolute and prismatic joints respectively.

Parameters

<i>degps_v</i>	Reference to allocated vector of appropriate size to hold all joint velocities.
----------------	---

Returns

error code.

5.3.3.12 home()

```
int Joint_comms::home (
    std::string name,
    u_int8_t direction,
    u_int8_t rpm,
    u_int8_t sensitivity,
    u_int8_t current )
```

Executes the homing sequence of a joint.

The joint will drive in the specified direction with the specified speed until a resistance which drives the current above the specified threshold is encountered. At this point the stepper stops and zeros the encoder.

Parameters

<i>name</i>	joint name.
<i>direction</i>	CCW: 0, CW: 1.
<i>rpm</i>	speed of motor in rpm > 10
<i>sensitivity</i>	PID error threshold, 0 to 255.
<i>current</i>	homeing current, determines how easy it is to stop the motor and thereby provoke a stall

Returns

error code.

5.3.3.13 init()

```
int Joint_comms::init (
    void )
```

Initializes all joints.

Warning

Add some joints using [addJoint\(\)](#) before calling this function. Iterates over all joints and initializes them on the I2C bus and tests if they are responsive.

Returns

0 on success, non-zero otherwise

5.3.3.14 **setBrakeModes()**

```
int Joint_comms::setBrakeModes (
    u_int8_t mode )
```

Set Brake Modes.

Applies the same brake modes to all joints. usefull to disengage all motors.

Parameters

<i>mode</i>	Freewheel: 0, Coolbrake: 1, Hardbrake: 2
-------------	--

Returns

error code.

5.3.3.15 **setDriveCurrents()** [1/2]

```
int Joint_comms::setDriveCurrents (
    std::vector< u_int8_t > current )
```

Set the drive Currents.

Warning

This function is unreliable and not well tested. Use [enables\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.3.3.16 **setDriveCurrents()** [2/2]

```
int Joint_comms::setDriveCurrents (
    u_int8_t current )
```

Overload to set all drive currents to the same value.

Warning

This function is unreliable and not well tested. Use [enables\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.3.3.17 setHoldCurrents() [1/2]

```
int Joint_comms::setHoldCurrents (
    std::vector< u_int8_t > current )
```

Set the Hold Currents.

Warning

This function is unreliable and not well tested. Use [enables\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.3.3.18 setHoldCurrents() [2/2]

```
int Joint_comms::setHoldCurrents (
    u_int8_t current )
```

Overload to set all hold currents to the same value.

Warning

This function is unreliable and not well tested. Use [enables\(\)](#) instead!

Parameters

<i>current</i>	0% - 100% of driver current
----------------	-----------------------------

Returns

error code.

5.3.3.19 setPositions()

```
int Joint_comms::setPositions (
    std::vector< float > angle_v )
```

Set the positions of all joints.

Set new target positons of all joints. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

<i>angle</i> ↔ _v	Vector of new target positions.
----------------------	---------------------------------

Returns

error code.

5.3.3.20 setVelocities()

```
int Joint_comms::setVelocities (
    std::vector< float > degps_v )
```

Set the velocities of all joints.

Set new target positons of all joints. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

<i>degps</i> ↔ _v	Vector of new target velocities.
----------------------	----------------------------------

Returns

error code.

5.3.3.21 stops()

```
int Joint_comms::stops (
    bool mode )
```

Stops the motors.

Stops all motors either soft or hard.

Parameters

<i>mode</i>	Hard: 0, Soft: 1
-------------	------------------

Returns

error code.

5.3.4 Member Data Documentation

5.3.4.1 joints

```
std::vector<Joint> Joint_comms::joints
```

Internal vector storing the [Joint](#) objects.

A [Joint](#) can be added by invoking [addJoint\(\)](#)

The documentation for this class was generated from the following files:

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/[mJointCom.h](#)
- ROS2/ros2_scara_ws/src/joint_communication/src/[mJointCom.cpp](#)

5.4 RPI_PWM Class Reference

PWM class for the Raspberry PI 4 and 5.

```
#include <uPWM.h>
```

Public Member Functions

- int [start](#) (int channel, int frequency, float duty_cycle=0, int chip=2)
- void [stop](#) ()
- [~RPI_PWM](#) ()
- int [setDutyCycle](#) (float v) const

Private Member Functions

- void [setPeriod](#) (int ns) const
- int [setDutyCycleNS](#) (int ns) const
- void [enable](#) () const
- void [disable](#) () const
- int [writeSYS](#) (std::string filename, int value) const

Private Attributes

- int [per](#) = 0
- std::string [chippath](#)
- std::string [pwmpath](#)

5.4.1 Detailed Description

PWM class for the Raspberry PI 4 and 5.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ~RPI_PWM()

```
RPI_PWM::~~RPI_PWM ( ) [inline]
```

5.4.3 Member Function Documentation

5.4.3.1 disable()

```
void RPI_PWM::disable ( ) const [inline], [private]
```

5.4.3.2 enable()

```
void RPI_PWM::enable ( ) const [inline], [private]
```

5.4.3.3 setDutyCycle()

```
int RPI_PWM::setDutyCycle (
    float v ) const [inline]
```

Sets the duty cycle.

Parameters

<i>v</i>	The duty cycle in percent.
<i>return</i>	>0 on success and -1 after an error.

5.4.3.4 setDutyCycleNS()

```
int RPI_PWM::setDutyCycleNS (
    int ns ) const [inline], [private]
```

5.4.3.5 setPeriod()

```
void RPI_PWM::setPeriod (
    int ns ) const [inline], [private]
```

5.4.3.6 start()

```
int RPI_PWM::start (
    int channel,
    int frequency,
    float duty_cycle = 0,
    int chip = 2 ) [inline]
```

Starts the PWM

Parameters

<i>channel</i>	The GPIO channel which is 2 or 3 for the RPI5
<i>frequency</i>	The PWM frequency
<i>duty_cycle</i>	The initial duty cycle of the PWM (default 0)
<i>chip</i>	The chip number (for RPI5 it's 2)
<i>return</i>	>0 on success and -1 if an error has happened.

5.4.3.7 stop()

```
void RPI_PWM::stop ( ) [inline]
```

Stops the PWM

5.4.3.8 writeSYS()

```
int RPI_PWM::writeSYS (
    std::string filename,
    int value ) const [inline], [private]
```

5.4.4 Member Data Documentation

5.4.4.1 chippath

```
std::string RPI_PWM::chippath [private]
```

5.4.4.2 per

```
int RPI_PWM::per = 0 [private]
```

5.4.4.3 pwmpath

```
std::string RPI_PWM::pwmpath [private]
```

The documentation for this class was generated from the following file:

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/uPWM.h

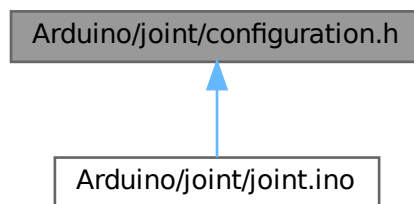
Chapter 6

File Documentation

6.1 Arduino/joint/configuration.h File Reference

Configuration definitions for [Joint 1](#) to [Joint 4](#).

This graph shows which files directly or indirectly include this file:



Macros

- `#define ADR 0x11`
I2C adress of joint n is 0x1n.
- `#define MAXACCEL 10000`
Maximum acceleration in steps/s². Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.
- `#define MAXVEL 800`
Maximum velocity in steps/s. Can be set for each joint. If set to high stalls might trigger since PID error grows too large.

6.1.1 Detailed Description

Configuration definitions for [Joint 1](#) to [Joint 4](#).

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file shall be included AFTER one of J1, J2, J3 or J4 have been defined.

6.1.2 Macro Definition Documentation

6.1.2.1 ADR

```
#define ADR 0x11
```

I2C adress of joint n is 0x1n.

6.1.2.2 MAXACCEL

```
#define MAXACCEL 10000
```

Maximum acceleration in steps/s². Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.

6.1.2.3 MAXVEL

```
#define MAXVEL 800
```

Maximum velocity in steps/s. Can be set for each joint. If set to high stalls might trigger since PID error grows too large.

6.2 configuration.h

[Go to the documentation of this file.](#)

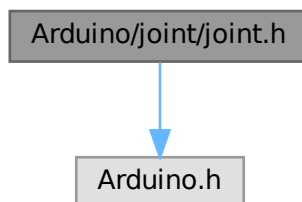
```
00001
00014 #ifndef CONFIGURATION_H
00015 #define CONFIGURATION_H
00016
00017 #if defined(J1)
00019 #define ADR 0x11
00020 #define MAXACCEL 10000
00021 #define MAXVEL 800
00022
00023 #elif defined(J2)
00024 #define ADR 0x12
00025 #define MAXACCEL 10000
00026 #define MAXVEL 800
00027
00028 #elif defined(J3)
00029 #define ADR 0x13
00030 #define MAXACCEL 10000
00031 #define MAXVEL 800
00032
00033 #elif defined(J4)
00034 #define ADR 0x14
00035 #define MAXACCEL 10000
00036 #define MAXVEL 800
00037 #else
00038
00039 /* Below only defined for documentation */
00043 #define ADR 0x11
00044
00049 #define MAXACCEL 10000
00050
00055 #define MAXVEL 800
00056 #error "No Joint has been defined. Define one of 'JX' where X 1,2,3,4"
00057 #endif
00058
00059 #endif
```

6.3 Arduino/joint/joint.h File Reference

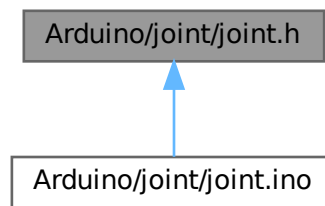
joint firmware header

```
#include <Arduino.h>
```

Include dependency graph for joint.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ACK 'O'`
- `#define NACK 'N'`
- `#define MAX_BUFFER 4`
Maximum size of I2C Payload in bytes.
- `#define RFLAGS_SIZE 1`
Size of the return flags in bytes.
- `#define DUMP_BUFFER(buffer, size)`
Macro to dump a buffer to the serial console.

Enumerations

- `enum stp_reg_t {`
`PING = 0x0f , SETUP = 0x10 , SETRPM = 0x11 , GETDRIVERRPM = 0x12 ,`
`MOVESTEPS = 0x13 , MOVEANGLE = 0x14 , MOVETOANGLE = 0x15 , GETMOTORSTATE = 0x16 ,`
`RUNCOTINOUS = 0x17 , ANGLEMOVED = 0x18 , SETCURRENT = 0x19 , SETHOLDCURRENT = 0x1A ,`
`SETMAXACCELERATION = 0x1B , SETMAXDECELERATION = 0x1C , SETMAXVELOCITY = 0x1D ,`
`ENABLESTALLGUARD = 0x1E ,`
`DISABLESTALLGUARD = 0x1F , CLEARSTALL = 0x20 , ISSTALLED = 0x21 , SETBRAKEMODE = 0x22 ,`
`ENABLEPID = 0x23 , DISABLEPID = 0x24 , ENABLECLOSEDLOOP = 0x25 , DISABLECLOSEDLOOP =`
`0x26 ,`
`SETCONTROLTHRESHOLD = 0x27 , MOVETOEND = 0x28 , STOP = 0x29 , GETPIDERROR = 0x2A ,`
`CHECKORIENTATION = 0x2B , GETENCODERRPM = 0x2C , HOME = 0x2D , ISHOMED = 0x2E ,`
`ISSETUP = 0x2F }`
register and command definitions

Functions

- `template<typename T >`
`void readValue (T &val, uint8_t *rxBuf, size_t rx_length)`
Reads a value from a buffer to a value of the specified type.
- `template<typename T >`
`int writeValue (const T val, uint8_t *txBuf, size_t &tx_length)`
Writes a value of the specified type to a buffer.

6.3.1 Detailed Description

joint firmware header

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains definitions and macros for the joint firmware.

6.3.2 Macro Definition Documentation

6.3.2.1 ACK

```
#define ACK 'O'
```

6.3.2.2 DUMP_BUFFER

```
#define DUMP_BUFFER(  
    buffer,  
    size )
```

Value:

```
{  
    Serial.print("Buffer dump: ");  
    for (size_t i = 0; i < size; i++)  
    {  
        Serial.print(buffer[i], HEX);  
        Serial.print(" ");  
    }  
    Serial.println();  
}
```

Macro to dump a buffer to the serial console.

Parameters

<i>buffer</i>	pointer to a buffer to dump to the console
<i>size</i>	number of bytes to dump

6.3.2.3 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

6.3.2.4 NACK

```
#define NACK 'N'
```

6.3.2.5 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

6.3.3 Enumeration Type Documentation

6.3.3.1 stp_reg_t

```
enum stp_reg_t
```

register and command definitions

a register can be read (R) or written (W), each register has a size in bytes. The payload can be split into multiple values or just be a single value. Note that not all functions are implemented.

Enumerator

PING	R; Size: 1; [(char) ACK].
SETUP	W; Size: 2; [(uint8) holdCurrent, (uint8) driveCurrent].
SETRPM	W; Size: 4; [(float) RPM].
GETDRIVERRPM	
MOVESTEPS	W; Size: 4; [(int32) steps].
MOVEANGLE	
MOVETOANGLE	W; Size: 4; [(float) degrees].
GETMOTORSTATE	
RUNCOTINOUS	
ANGLEMOVED	R; Size: 4; [(float) degrees].
SETCURRENT	W; Size: 1; [(uint8) driveCurrent].
SETHOLDCURRENT	W; Size: 1; [(uint8) holdCurrent].
SETMAXACCELERATION	
SETMAXDECELERATION	
SETMAXVELOCITY	
ENABLESTALLGUARD	W; Size: 1; [(uint8) threshold].

Enumerator

DISABLESTALLGUARD	
CLEARSTALL	
ISSTALLED	R; Size: 1; [(uint8) isStalled].
SETBRAKEMODE	W; Size: 1; [(uint8) mode].
ENABLEPID	
DISABLEPID	
ENABLECLOSEDLOOP	
DISABLECLOSEDLOOP	W; Size: 1; [(uint8) 0].
SETCONTROLTHRESHOLD	
MOVETOEND	
STOP	W; Size: 1; [(uint8) mode].
GETPIDERROR	
CHECKORIENTATION	W; Size: 4; [(float) degrees].
GETENCODERRPM	R; Size: 4; [(float) RPM].
HOME	W; Size: 4; [(uint8) current, (uint8) sensitivity, (uint8) speed, (uint8) direction].
ISHOMED	R; Size: 1; [(uint8) isStalled].
ISSETUP	R; Size: 1; [(uint8) isStalled].

6.3.4 Function Documentation

6.3.4.1 readValue()

```
template<typename T >
void readValue (
    T & val,
    uint8_t * rxBuf,
    size_t rx_length )
```

Reads a value from a buffer to a value of the specified type.

Parameters

<i>val</i>	Reference to output variable
<i>rxBuf</i>	Buffer to read value from
<i>rx_length</i>	Length of the buffer

6.3.4.2 writeValue()

```
template<typename T >
int writeValue (
    const T val,
    uint8_t * txBuf,
    size_t & tx_length )
```

Writes a value of the specified type to a buffer.

Parameters

<i>val</i>	Reference to input variable
<i>txBuf</i>	pointer to tx buffer
<i>tx_length</i>	Length of the buffer returned

Returns

0 On success

6.4 joint.h

[Go to the documentation of this file.](#)

```

00001
00014 #ifndef JOINT_H
00015 #define JOINT_H
00016 #include <Arduino.h>
00017
00018 #define ACK 'O'
00019 #define NACK 'N'
00020
00026 #define MAX_BUFFER 4 // Bytes
00027
00033 #define RFLAGS_SIZE 1
00034
00041 #define DUMP_BUFFER(buffer, size) \
00042 { \
00043     Serial.print("Buffer dump: "); \
00044     for (size_t i = 0; i < size; i++) \
00045     { \
00046         Serial.print(buffer[i], HEX); \
00047         Serial.print(" "); \
00048     } \
00049     Serial.println(); \
00050 }
00051
00060 enum stp_reg_t
00061 {
00062     PING = 0x0f,
00063     SETUP = 0x10,
00064     SETRPM = 0x11,
00065     GETDRIVERRPM = 0x12,
00066     MOVESTEPS = 0x13,
00067     MOVEANGLE = 0x14,
00068     MOVETOANGLE = 0x15,
00069     GETMOTORSTATE = 0x16,
00070     RUNCOTINOUS = 0x17,
00071     ANGLEMOVED = 0x18,
00072     SETCURRENT = 0x19,
00073     SETHOLDCURRENT = 0x1A,
00074     SETMAXACCELERATION = 0x1B,
00075     SETMAXDECELERATION = 0x1C,
00076     SETMAXVELOCITY = 0x1D,
00077     ENABLESTALLGUARD = 0x1E,
00078     DISABLESTALLGUARD = 0x1F,
00079     CLEARSTALL = 0x20,
00080     ISSTALLED = 0x21,
00081     SETBRAKEMODE = 0x22,
00082     ENABLEPID = 0x23,
00083     DISABLEPID = 0x24,
00084     ENABLECLOSEDLOOP = 0x25,
00085     DISABLECLOSEDLOOP = 0x26,
00086     SETCONTROLTHRESHOLD = 0x27,
00087     MOVETOEND = 0x28,
00088     STOP = 0x29,
00089     GETPIDERROR = 0x2A,
00090     CHECKORIENTATION = 0x2B,
00091     GETENCODERRPM = 0x2C,
00092     HOME = 0x2D,
00093     ISHOMED = 0x2E,
00094     ISSETUP = 0x2F
00095 };
00096
00103 template <typename T>
00104 void readValue(T &val, uint8_t *rxBuf, size_t rx_length)

```

```

00105 {
00106     memcpy(&val, rxBuf, rx_length);
00107 }
00108
00116 template <typename T>
00117 int writeValue(const T val, uint8_t *txBuf, size_t &tx_length)
00118 {
00119     tx_length = sizeof(T);
00120     memcpy(txBuf, &val, tx_length);
00121     return 0;
00122 }
00123
00124 #endif

```

6.5 Arduino/joint/joint.ino File Reference

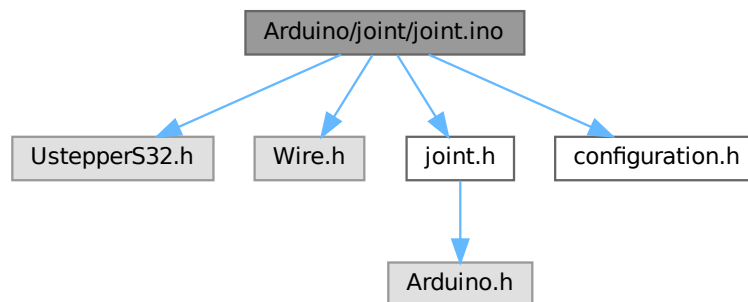
joint firmware

```

#include <UstepperS32.h>
#include <Wire.h>
#include "joint.h"
#include "configuration.h"

```

Include dependency graph for joint.ino:



Macros

- `#define J1`
Define either joint that is to be flashed.

Functions

- void `stepper_receive_handler` (uint8_t reg)
Handles commands received via I2C.
- void `stepper_request_handler` (uint8_t reg)
Handles read request received via I2C.
- void `receiveEvent` (int n)
I2C receive event Handler.
- void `requestEvent` ()
I2C request event Handler.
- void `setup` (void)
Setup Peripherals.
- void `loop` (void)
Main loop.

Variables

- UstepperS32 [stepper](#)
- uint8_t [reg](#) = 0
- uint8_t [rx_buf](#) [[MAX_BUFFER](#)] = { 0 }
- uint8_t [tx_buf](#) [[MAX_BUFFER](#)+[RFLAGS_SIZE](#)] = { 0 }
- bool [tx_data_ready](#) = 0
- bool [rx_data_ready](#) = 0
- size_t [tx_length](#) = 0
- size_t [rx_length](#) = 0

6.5.1 Detailed Description

joint firmware

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains the joint firmware.

6.5.2 Macro Definition Documentation

6.5.2.1 J1

```
#define J1
```

Define either joint that is to be flashed.

Define either J1, J2, J3 or J4 and subsequently include [configuration.h](#)

6.5.3 Function Documentation

6.5.3.1 loop()

```
void loop (
    void )
```

Main loop.

Executes the following:

- 1) if isStallguardEnabled: compares stepper.getPidError() with stallguardThreshold and sets BIT0 of the state byte.
- 2) sets/clears BIT2 of the state byte if the joint is homed or not.
- 3) sets/clears BIT3 of the state byte if the joint is setup or not.
- 4) if rx_data_ready: set BIT1 of the state byte to indicate device is busy. Invoke stepper_receive_handler. Clear BIT1 of the state byte to indicate device is no longer busy

Todo • why are BIT2 and BIT3 constantly checked and set? Would it be sufficient to do this only invoking the actual functions?

6.5.3.2 receiveEvent()

```
void receiveEvent (
    int n )
```

I2C receive event Handler.

Reads the content of the received message. Saves the reg so it can be used in the main loop. If the master invokes the read() function the message contains only the register byte and no payload. If the master invokes the write() the message has a payload of appropriate size for the command. For a read request the message looks like this:

```
< [REG]
> [TXBUFn]...[TXBUF2][TXBUF1][TXBUF0][FLAGS]
```

For a command the message looks like this:

```
< [REG][RXBUFn]...[RXBUF2][RXBUF1][RXBUF0]
> [FLAGS]
```

The payload is read into the rx_buf, rx_length is set to the payload length and the rx_data_ready flag is set.

Parameters

<i>n</i>	the number of bytes read from the controller device: MAX_BUFFER
----------	---

6.5.3.3 requestEvent()

```
void requestEvent ( )
```

I2C request event Handler.

Sends the response data to the master. Every transaction begins with a receive event. This function is only called when the master calls the read() function. Hence this function is only invoked after the [receiveEvent\(\)](#) handler has been called. The function calls the [stepper_request_handler\(\)](#) which is non-blocking. [stepper_request_handler\(\)](#) populates the tx_buf, the current state flags are appended to the tx_buf and then it is send to the master.

6.5.3.4 setup()

```
void setup (
    void )
```

Setup Peripherals.

Setup I2C with the address ADR, and begin Serial for debugging with baudrate 9600.

6.5.3.5 stepper_receive_handler()

```
void stepper_receive_handler (
    uint8_t reg )
```

Handles commands received via I2C.

Warning

This is a blocking function which may take some time to execute. This function must not be called from an ISR or callback! Call from main loop instead.

All the registers that are inside this handler are considered command which require an action.

Parameters

<i>reg</i>	command that should be executed.
------------	----------------------------------

6.5.3.6 stepper_request_handler()

```
void stepper_request_handler (
    uint8_t reg )
```

Handles read request received via I2C.

Can be invoked from the I2C ISR since reads from the stepper are non-blocking. Also Handling reads and the subsequent `wire.write()`, did not work from the main loop.

All registers inside this function are regarded as read only.

Parameters

<i>reg</i>	register to read.
------------	-------------------

6.5.4 Variable Documentation

6.5.4.1 reg

```
uint8_t reg = 0
```


6.5.4.2 rx_buf

```
uint8_t rx_buf[MAX_BUFFER] = { 0 }
```

6.5.4.3 rx_data_ready

```
bool rx_data_ready = 0
```

6.5.4.4 rx_length

```
size_t rx_length = 0
```

6.5.4.5 stepper

```
UstepperS32 stepper
```

6.5.4.6 tx_buf

```
uint8_t tx_buf[MAX_BUFFER+RFLAGS_SIZE] = { 0 }
```

6.5.4.7 tx_data_ready

```
bool tx_data_ready = 0
```

6.5.4.8 tx_length

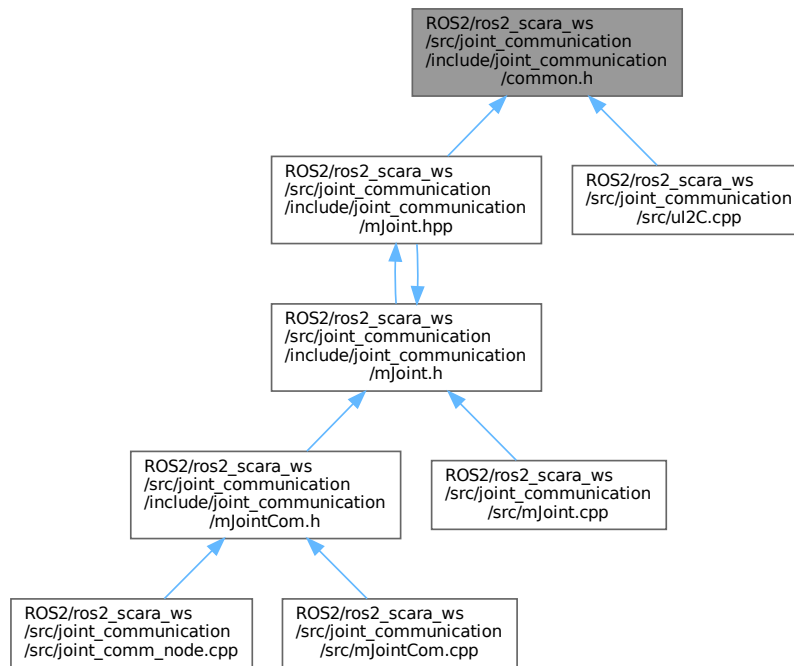
```
size_t tx_length = 0
```

6.6 docs/DOCS_README.md File Reference

6.7 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/common.h File Reference

A file containing utility macros and functions.

This graph shows which files directly or indirectly include this file:



Macros

- `#define DUMP_BUFFER(buffer, size)`
Macro to dump a buffer to cout.

6.7.1 Detailed Description

A file containing utility macros and functions.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

6.7.2 Macro Definition Documentation

6.7.2.1 DUMP_BUFFER

```
#define DUMP_BUFFER(  
    buffer,  
    size )
```

Value:

```
{  
    std::cout << "Buffer dump: ";  
    for (size_t i = 0; i < size; i++)  
    {  
        printf("%#x ", buffer[i]);  
    }  
    std::cout << std::endl;  
}
```

Macro to dump a buffer to cout.

Parameters

<i>buffer</i>	pointer to a buffer to dump to the console
<i>size</i>	number of bytes to dump

6.8 common.h

[Go to the documentation of this file.](#)

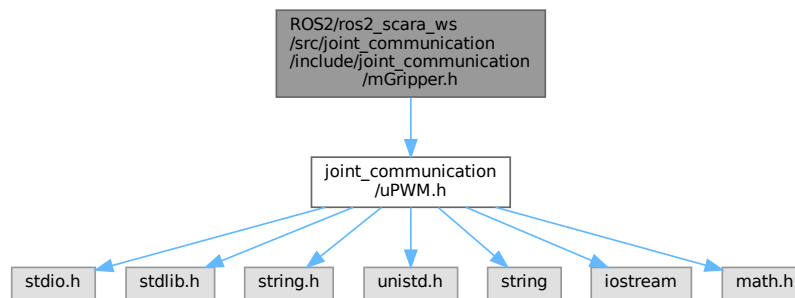
```
00001  
00011 #ifndef COMMON_H  
00012 #define COMMON_H  
00013  
00014  
00021 #define DUMP_BUFFER(buffer, size) \\  
00022 { \\  
00023     std::cout << "Buffer dump: "; \\  
00024     for (size_t i = 0; i < size; i++) \\  
00025     { \\  
00026         printf("%#x ", buffer[i]); \\  
00027     } \\  
00028     std::cout << std::endl; \\  
00029 } \\  
00030  
00031 #endif // COMMON_H
```

6.9 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/mGripper.h File Reference

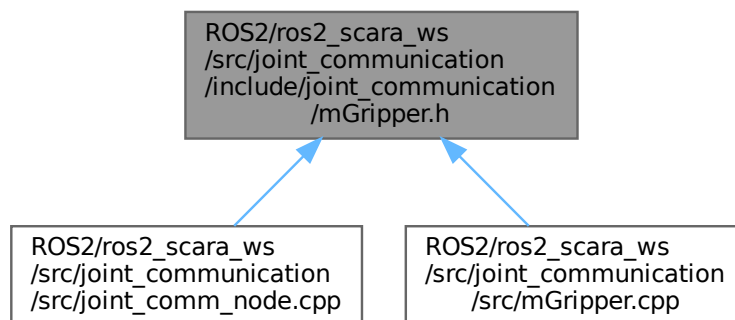
File containing the [Gripper](#) class.

```
#include "joint_communication/uPWM.h"
```

Include dependency graph for mGripper.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Gripper](#)
[Gripper](#) object to interact with the robot gripper.

6.9.1 Detailed Description

File containing the [Gripper](#) class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

Include this file for API functions to interact with the gripper.

6.10 mGripper.h

[Go to the documentation of this file.](#)

```

00001
00013 #ifndef MGRIPPER_H
00014 #define MGRIPPER_H
00015 #include "joint_communication/uPWM.h"
00016
00023 class Gripper
00024 {
00025 public:
00026     Gripper(void);
00027
00033     int init(void);
00034
00040     int deinit(void);
00041
00050     int enable(void);
00051
00059     int disable(void);
00060
00067     int setPosition(float width);
00068
00069 private:
00070     RPI_PWM pwm;
00071 };
00072 #endif // MGRIPPER_H

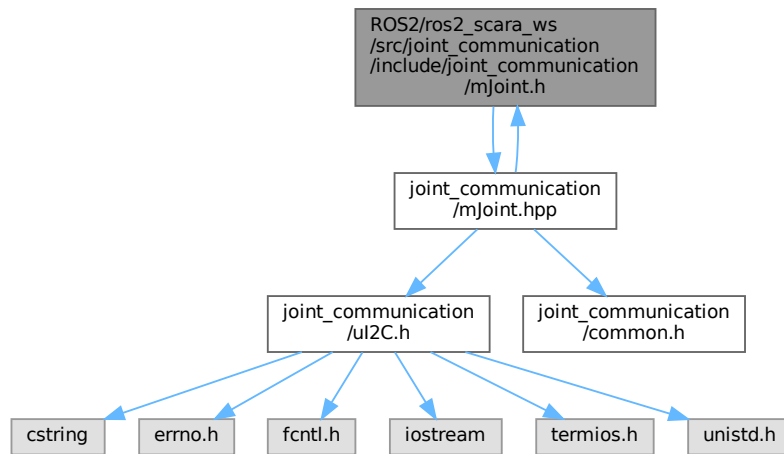
```

6.11 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/mJoint.h File Reference

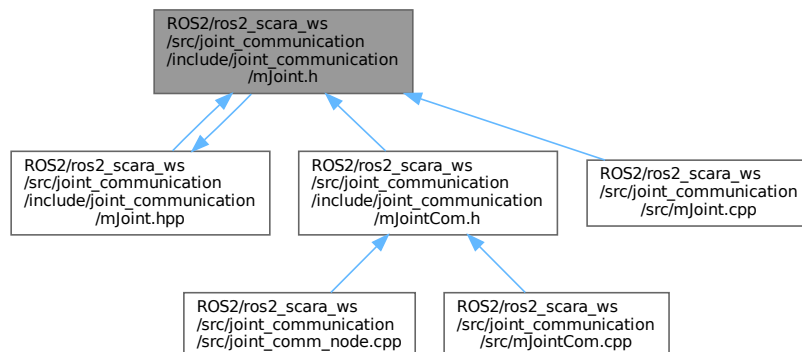
File including the [Joint](#) class.

```
#include "joint_communication/mJoint.hpp"
```

Include dependency graph for mJoint.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Joint](#)

Representing a single joint on the I2C bus.

Macros

- #define [JOINT2ENCODERANGLE](#)(jointAngle, gearRatio, offset) (gearRatio * (jointAngle + offset))
- #define [ENCODER2JOINTANGLE](#)(encoderAngle, gearRatio, offset) (encoderAngle / gearRatio - offset)

6.11.1 Detailed Description

File including the [Joint](#) class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

6.11.2 Macro Definition Documentation

6.11.2.1 ENCODER2JOINTANGLE

```
#define ENCODER2JOINTANGLE(  
    encoderAngle,  
    gearRatio,  
    offset ) (encoderAngle / gearRatio - offset)
```

6.11.2.2 JOINT2ENCODERANGLE

```
#define JOINT2ENCODERANGLE(  
    jointAngle,  
    gearRatio,  
    offset ) (gearRatio * (jointAngle + offset))
```

6.12 mJoint.h

[Go to the documentation of this file.](#)

```
00001  
00012 #ifndef MJOINT_H  
00013 #define MJOINT_H  
00014  
00015 #define JOINT2ENCODERANGLE(jointAngle, gearRatio, offset) (gearRatio * (jointAngle + offset))  
00016 #define ENCODER2JOINTANGLE(encoderAngle, gearRatio, offset) (encoderAngle / gearRatio - offset)  
00017  
00022 class Joint  
00023 {  
00024 public:  
00025     Joint(const int address, const std::string name, const float gearRatio, const float offset);  
00026     // ~Joint();  
00027  
00028     int init(void);  
00029     int deinit(void);
```

```

00030     int printInfo(void);
00031     int getPosition(float &angle);
00032     int setPosition(float angle);
00033     int getVelocity(float &degps);
00034     int setVelocity(float degps);
00035     int checkOrientation(float angle = 10.0);
00036
00043     int enable(u_int8_t driveCurrent, u_int8_t holdCurrent);
00044
00049     int disable(void);
00050
00060     int home(u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current);
00061
00070     int stop(bool mode);
00075     int disableCL(void);
00076
00083     int setDriveCurrent(u_int8_t current);
00084
00091     int setHoldCurrent(u_int8_t current);
00092
00098     int setBrakeMode(u_int8_t mode);
00099
00105     int getStall(u_int8_t &stall);
00106
00111     int enableStallguard(u_int8_t sensitivity);
00112
00118     int getIsHomed(u_int8_t &homed);
00119
00126     int getIsHomed(void);
00127
00134     bool isHomed(void);
00135
00141     int getIsSetup(u_int8_t &setup);
00142
00147     int getIsSetup(void);
00148
00152     bool isSetup(void);
00153
00154     int moveSteps(int32_t steps);
00155     int checkCom(void);
00156
00161     u_int8_t getFlags(void);
00162
00163     std::string name;
00164
00165 protected:
00166 private:
00176     enum stp_reg_t
00177     {
00178         PING = 0x0f,
00179         SETUP = 0x10,
00180         SETRPM = 0x11,
00181         GETDRIVERRPM = 0x12,
00182         MOVESTEPS = 0x13,
00183         MOVEANGLE = 0x14,
00184         MOVETOANGLE = 0x15,
00185         GETMOTORSTATE = 0x16,
00186         RUNCOTINOUS = 0x17,
00187         ANGLEMOVED = 0x18,
00188         SETCURRENT = 0x19,
00189         SETHOLDCURRENT = 0x1A,
00190         SETMAXACCELERATION = 0x1B,
00191         SETMAXDECELERATION = 0x1C,
00192         SETMAXVELOCITY = 0x1D,
00193         ENABLESTALLGUARD = 0x1E,
00194         DISABLESTALLGUARD = 0x1F,
00195         CLEARSTALL = 0x20,
00196         ISSTALLED = 0x21,
00197         SETBRAKEMODE = 0x22,
00198         ENABLEPID = 0x23,
00199         DISABLEPID = 0x24,
00200         ENABLECLOSEDLOOP = 0x25,
00201         DISABLECLOSEDLOOP = 0x26,
00202         SETCONTROLTHRESHOLD = 0x27,
00203         MOVETOEND = 0x28,
00204         STOP = 0x29,
00205         GETPIDERROR = 0x2A,
00206         CHECKORIENTATION = 0x2B,
00207         GETENCODERRPM = 0x2C,
00208         HOME = 0x2D,
00209         ISHOMED = 0x2E,
00210         ISSETUP = 0x2F
00211     };
00212
00213     template <typename T>
00214     int read(const stp_reg_t reg, T &data, u_int8_t &flags);
00215

```



```

00216     template <typename T>
00217     int write(const stp_reg_t reg, T data, u_int8_t &flags);
00218
00237     u_int8_t flags = 0x00;
00238
00239     u_int8_t ishomed = 0;
00240     u_int8_t issetup = 0;
00241
00242     int address;
00243     float gearRatio = 1;
00244     float offset = 0;
00245
00246     int handle = -1;
00247 };
00248
00249 #include "joint_communication/mJoint.hpp"
00250
00251 #endif

```

6.13 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.hpp File Reference

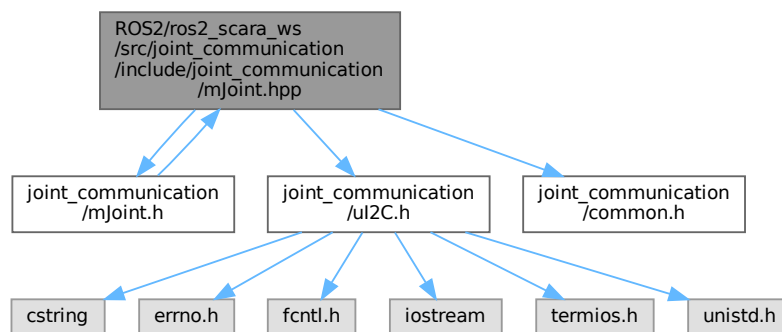
Templated functions for the [Joint](#) class.

```

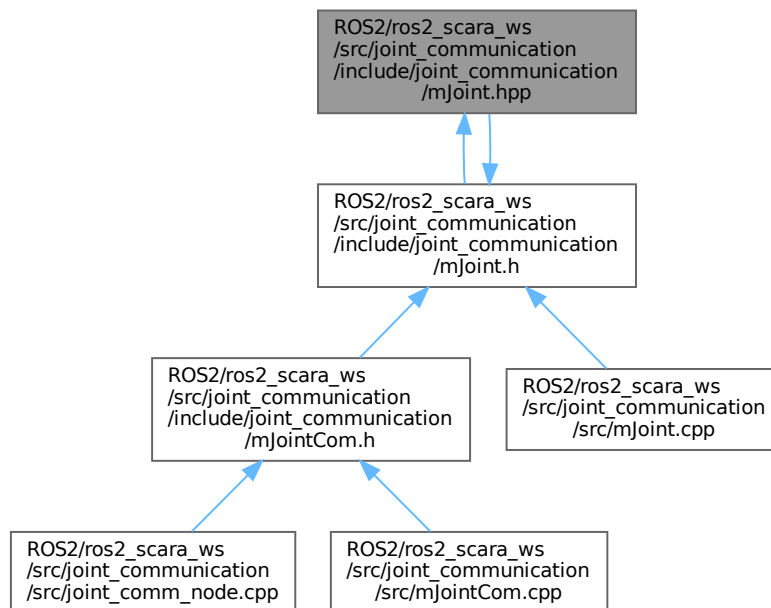
#include "joint_communication/mJoint.h"
#include "joint_communication/uI2C.h"
#include "joint_communication/common.h"

```

Include dependency graph for mJoint.hpp:



This graph shows which files directly or indirectly include this file:



6.13.1 Detailed Description

Templated functions for the [Joint](#) class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

This header must be included at the END of the [mJoint.h](#) file.

6.14 mJoint.hpp

[Go to the documentation of this file.](#)

```

00001
00012 #include "joint_communication/mJoint.h"
00013 #include "joint_communication/uI2C.h"
00014 #include "joint_communication/common.h"
00015
00031 template <typename T>
00032 int Joint::read(const stp_reg_t reg, T &data, u_int8_t &flags)
00033 {
00034     size_t size = sizeof(T) + RFLAGS_SIZE;
00035     char *buf = new char[size];
00036     int n = readFromI2CDev(this->handle, reg, buf, size);
00037     if (n != static_cast<int>(size))
00038     {
00039         delete[] buf;
00040         return -1;
00041     }
00042     memcpy(&data, buf, size - RFLAGS_SIZE);
00043     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00044     delete[] buf;
00045     return 0;
00046 }
00047
00063 template <typename T>
00064 int Joint::write(const stp_reg_t reg, T data, u_int8_t &flags)
00065 {
00066     size_t size = sizeof(T) + RFLAGS_SIZE;
00067     char *buf = new char[size];
00068     memcpy(buf, &data, size - RFLAGS_SIZE);
00069     int rc = writeToI2CDev(this->handle, reg, buf, size - RFLAGS_SIZE, buf + size - RFLAGS_SIZE);
00070     rc = rc > 0 ? 0 : rc;
00071
00072     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00073     delete[] buf;
00074     return rc;
00075 }

```

6.15 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/mJointCom.h File Reference

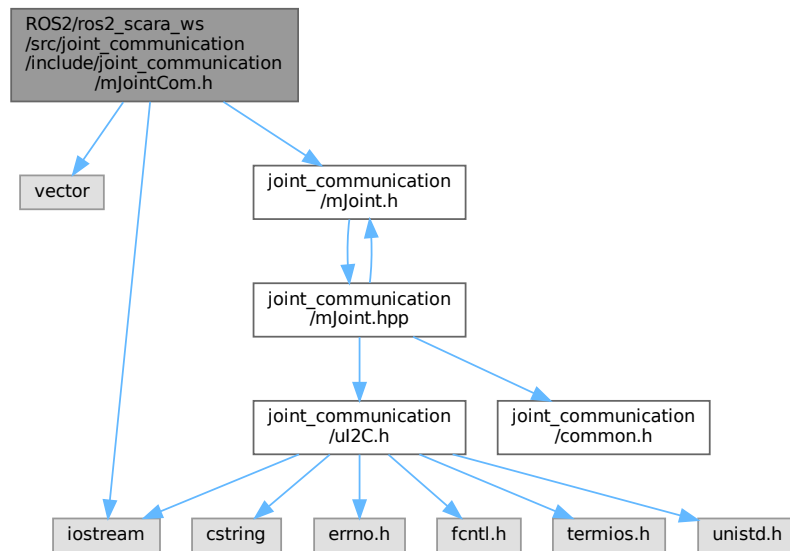
File containing the [Joint_comms](#) class.

```

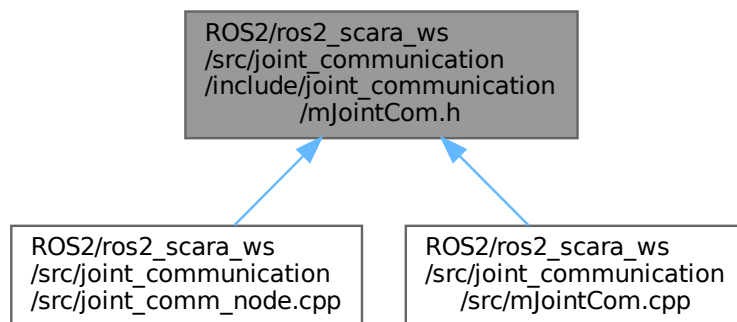
#include <vector>
#include <iostream>
#include "joint_communication/mJoint.h"

```

Include dependency graph for mJointCom.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Joint_comms](#)
Communication object for all joints.

6.15.1 Detailed Description

File containing the [Joint_comms](#) class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

Include this file for API functions to interact with the stepper motors.

6.16 mJointCom.h

[Go to the documentation of this file.](#)

```

00001
00014 #ifndef MJOINTCOM_H
00015 #define MJOINTCOM_H
00016
00017 #include <vector>
00018 #include <iostream>
00019 #include "joint_communication/mJoint.h"
00020
00027 class Joint_comms
00028 {
00029 public:
00030     Joint_comms(void);
00031     ~Joint_comms();
00032
00041     int init(void);
00042
00050     int deinit(void);
00051
00068     void addJoint(const int address, const std::string name, const float gearRatio, const float offset);
00069
00083     int enables(std::vector<u_int8_t> driveCurrent_v, std::vector<u_int8_t> holdCurrent_v);
00084
00093     int enables(u_int8_t driveCurrent, u_int8_t holdCurrent);
00094
00101     int disables(void);
00102
00117     int home(std::string name, u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t
current);
00118
00128     int getPositions(std::vector<float> &angle_v);
00129
00139     int setPositions(std::vector<float> angle_v);
00140
00150     int getVelocities(std::vector<float> &degps_v);
00151
00161     int setVelocities(std::vector<float> degps_v);
00162
00175     int checkOrientations(std::vector<float> angle_v);
00176
00182     int checkOrientations(float angle = 10.0);
00183
00191     int stops(bool mode);
00192
00197     int disableCLs(void);
00198
00207     int setDriveCurrents(std::vector<u_int8_t> current);
00208
00216     int setDriveCurrents(u_int8_t current);
00217

```

```

00225   int setHoldCurrents(std::vector<u_int8_t> current);
00226
00234   int setHoldCurrents(u_int8_t current);
00235
00243   int setBrakeModes(u_int8_t mode);
00244
00252   int enableStallguards(std::vector<u_int8_t> thresholds);
00253
00259   std::vector<Joint> joints;
00260
00261 protected:
00262 private:
00263 };
00264
00265 #endif

```

6.17 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/ul2C.h File Reference

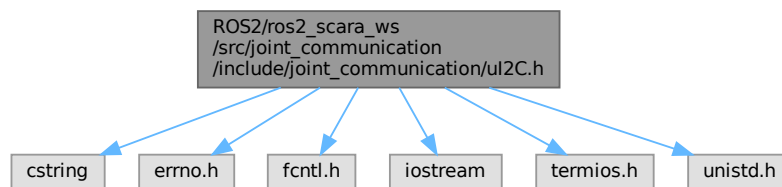
Low level utility for I2C communication on Raspberry Pi using lgpio library.

```

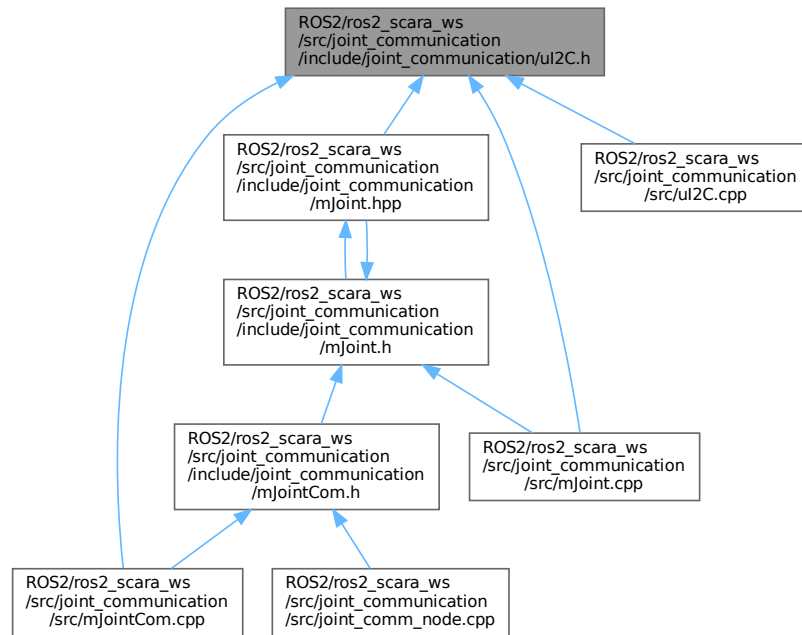
#include <cstring>
#include <errno.h>
#include <fcntl.h>
#include <iostream>
#include <termios.h>
#include <unistd.h>

```

Include dependency graph for ul2C.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ACK 'O'`
- `#define NACK 'N'`
- `#define RFLAGS_SIZE 1`
Size of the return flags in bytes.
- `#define MAX_BUFFER 4`
Maximum size of I2C Payload in bytes.

Functions

- `int openI2CDevHandle (const int dev_addr)`
Initiates an I2C device on the bus.
- `int readFromI2CDev (const int dev_handle, const int reg, char *buffer, const int data_length)`
reads block of bytes from device to buffer
- `int writeToI2CDev (const int dev_handle, const int reg, char *tx_buffer, const int data_length, char *RFLAGS_buffer)`
writes block of bytes from buffer to device
- `int closeI2CDevHandle (const int dev_handle)`
close an I2C device on the bus

6.17.1 Detailed Description

Low level utility for I2C communication on Raspberry Pi using I2C library.

Author

Sebastian Storz

Version

0.1

Date

2025-05-28

Copyright

Copyright (c) 2025

I2C needs to be installed and linked! Installation:

```
cd ~
sudo apt update
sudo apt install -y swig
wget https://github.com/joan2937/i2c/archive/master.zip
unzip master.zip
cd i2c-master
make
sudo make install
cd ..
sudo rm -rf i2c-master
rm master.zip
```

bash

6.17.2 Macro Definition Documentation

6.17.2.1 ACK

```
#define ACK 'O'
```

6.17.2.2 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

6.17.2.3 NACK

```
#define NACK 'N'
```


6.17.2.4 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

6.17.3 Function Documentation

6.17.3.1 closeI2CDevHandle()

```
int closeI2CDevHandle (
    const int dev_handle )
```

close an I2C device on the bus

Parameters

<i>dev_handle</i>	device handle obtained from openI2CDevHandle
-------------------	--

Returns

0 on OK, negative on error.

6.17.3.2 openI2CDevHandle()

```
int openI2CDevHandle (
    const int dev_addr )
```

Initiates an I2C device on the bus.

Parameters

<i>dev_addr</i>	7-bit device address [0 - 0x7F]
-----------------	---------------------------------

Returns

the device handle, negative on error.

6.17.3.3 readFromI2CDev()

```
int readFromI2CDev (
    const int dev_handle,
    const int reg,
    char * buffer,
    const int data_length )
```

reads block of bytes from device to buffer

Parameters

<i>dev_handle</i>	device handle obtained from <code>openI2CDevHandle</code>
<i>reg</i>	the command/data register
<i>buffer</i>	pointer to data buffer to hold received values
<i>data_length</i>	number of bytes to read

Returns

number of bytes read, negative on error.

6.17.3.4 writeToI2CDev()

```
int writeToI2CDev (
    const int dev_handle,
    const int reg,
    char * tx_buffer,
    const int data_length,
    char * RFLAGS_buffer )
```

writes block of bytes from buffer to device

Parameters

<i>dev_handle</i>	device handle obtained from <code>openI2CDevHandle</code>
<i>reg</i>	the command/data register
<i>tx_buffer</i>	pointer to data buffer holding the data to send
<i>data_length</i>	number of bytes to send
<i>RFLAGS_buffer</i>	buffer to hold returned flags

Returns

0 on OK, negative on error.

6.18 ul2C.h

[Go to the documentation of this file.](#)

```
00001
00028 #ifndef SERIAL_H
00029 #define SERIAL_H
00030 #include <cstring>
00031 #include <errno.h>
00032 #include <fcntl.h>
00033 #include <iostream>
00034 #include <termios.h>
00035 #include <unistd.h>
00036
00037 #define ACK 'O'
00038 #define NACK 'N'
00039
00043 #define RFLAGS_SIZE 1
00044
00048 #define MAX_BUFFER 4 // Bytes
00049
00055 int openI2CDevHandle(const int dev_addr);
```

```

00056
00065 int readFromI2CDev(const int dev_handle, const int reg, char *buffer, const int data_length);
00066
00076 int writeToI2CDev(const int dev_handle, const int reg, char *tx_buffer, const int data_length, char
    *RFLAGS_buffer);
00077
00083 int closeI2CDevHandle(const int dev_handle);
00084
00085
00086 #endif

```

6.19 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↵ communication/uPWM.h File Reference

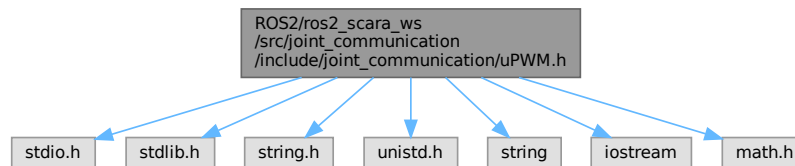
Includes source code for Hardware PWM generation on Raspberry Pi 4.

```

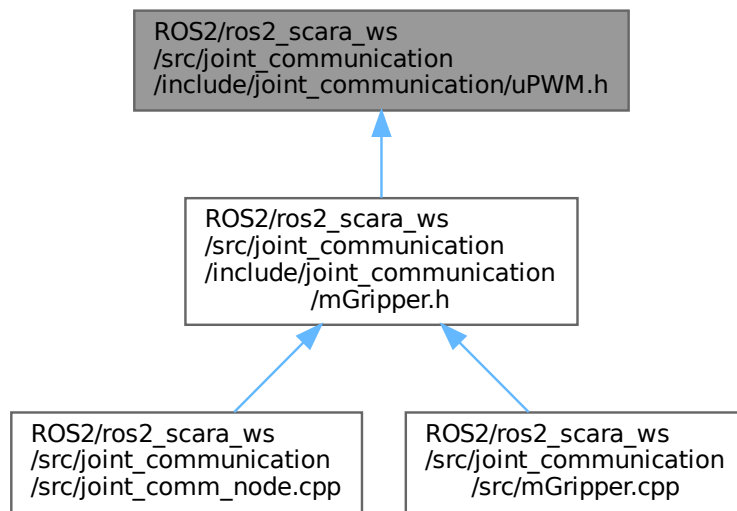
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <string>
#include <iostream>
#include <math.h>

```

Include dependency graph for uPWM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [RPI_PWM](#)
PWM class for the Raspberry PI 4 and 5.

6.19.1 Detailed Description

Includes source code for Hardware PWM generation on Raspberry Pi 4.

Author

Sebastian Storz and Bernd Porr, bernd.porr@glasgow.ac.uk

Version

0.1

Date

2025-05-27

I copied this from: https://github.com/berndporr/rpi_pwm/blob/main/rpi_pwm.h and slightly modified it.

Igpiio, the library used for I2C access can only generate soft PWM, The timing jitter will cause the servo to fidget. This may cause it to overheat and wear out prematurely.

Copyright

Copyright (c) 2025

6.20 uPWM.h

[Go to the documentation of this file.](#)

```

00001
00019 #ifndef __RPIPWM
00020 #define __RPIPWM
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <unistd.h>
00026 #include <string>
00027 #include <iostream>
00028 #include <math.h>
00029
00033 class RPI_PWM
00034 {
00035 public:
00044     int start(int channel, int frequency, float duty_cycle = 0, int chip = 2)
00045     {
00046         chippath = "/sys/class/pwm/pwmchip" + std::to_string(chip);
00047         pwmpath = chippath + "/pwm" + std::to_string(channel);
00048         std::string p = chippath + "/export";
00049         FILE *const fp = fopen(p.c_str(), "w");
00050         if (NULL == fp)
00051         {
00052             std::cerr << "PWM device does not exist. Make sure to add 'dtoverlay=pwm-2chan' to
/boot/firmware/config.txt.\n";
00053             return -1;
00054         }
00055         const int r = fprintf(fp, "%d", channel);
00056         fclose(fp);
00057         if (r < 0)
00058             return r;
00059         usleep(100000); // it takes a while till the PWM subdir is created
00060         per = (int)1E9 / frequency;
00061         setPeriod(per);
00062         setDutyCycle(duty_cycle);
00063         enable();
00064         return r;
00065     }
00066
00070     void stop()
00071     {
00072         disable();
00073     }
00074
00075     ~RPI_PWM()
00076     {
00077         disable();
00078     }
00079
00085     inline int setDutyCycle(float v) const
00086     {
00087         const int dc = (int)round((float)per * (v / 100.0));
00088         const int r = setDutyCycleNS(dc);
00089         return r;
00090     }
00091
00092 private:
00093     void setPeriod(int ns) const
00094     {
00095         writeSYS(pwmpath + "/" + "period", ns);
00096     }
00097
00098     inline int setDutyCycleNS(int ns) const
00099     {
00100         const int r = writeSYS(pwmpath + "/" + "duty_cycle", ns);
00101         return r;
00102     }
00103
00104     void enable() const
00105     {
00106         writeSYS(pwmpath + "/" + "enable", 1);
00107     }
00108
00109     void disable() const
00110     {
00111         writeSYS(pwmpath + "/" + "enable", 0);
00112     }
00113
00114     int per = 0;
00115
00116     std::string chippath;
00117     std::string pwmpath;

```

```

00118
00119     inline int writeSYS(std::string filename, int value) const
00120     {
00121         FILE *const fp = fopen(filename.c_str(), "w");
00122         if (NULL == fp)
00123         {
00124             return -1;
00125         }
00126         const int r = fprintf(fp, "%d", value);
00127         fclose(fp);
00128         return r;
00129     }
00130 };
00131
00132 #endif

```

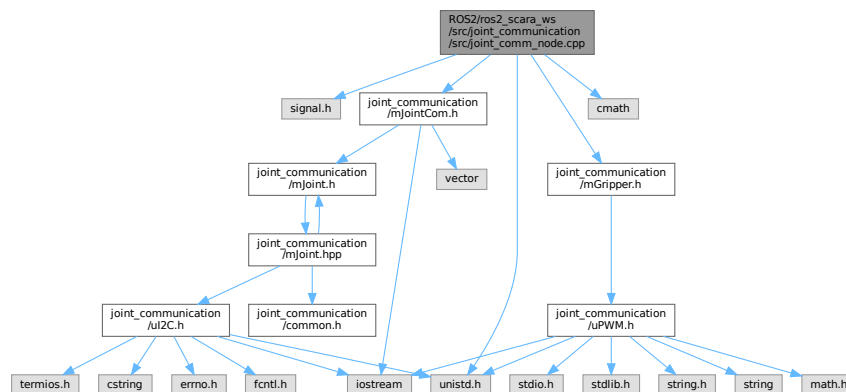
6.21 ROS2/ros2_scara_ws/src/joint_communication/src/joint_comm_node.cpp File Reference

```

#include <signal.h>
#include <unistd.h>
#include "joint_communication/mJointCom.h"
#include "joint_communication/mGripper.h"
#include <cmath>

```

Include dependency graph for joint_comm_node.cpp:



Functions

- void [INT_handler](#) (int s)
- int [main](#) (int argc, char **argv)

Variables

- [Joint_comms_Joints](#)
- [Gripper_Gripper](#)

6.21.1 Function Documentation

6.21.1.1 INT_handler()

```

void INT_handler (
    int s )

```

6.21.1.2 main()

```
int main (
    int argc,
    char ** argv )
```

6.21.2 Variable Documentation

6.21.2.1 _Gripper

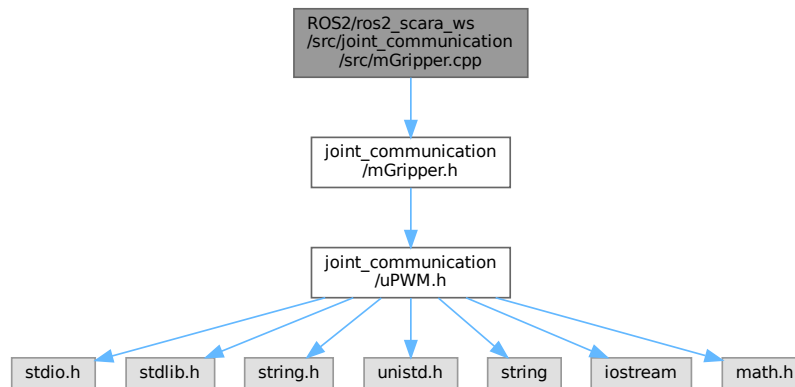
`Gripper` `_Gripper`

6.21.2.2 _Joints

`Joint_comms` `_Joints`

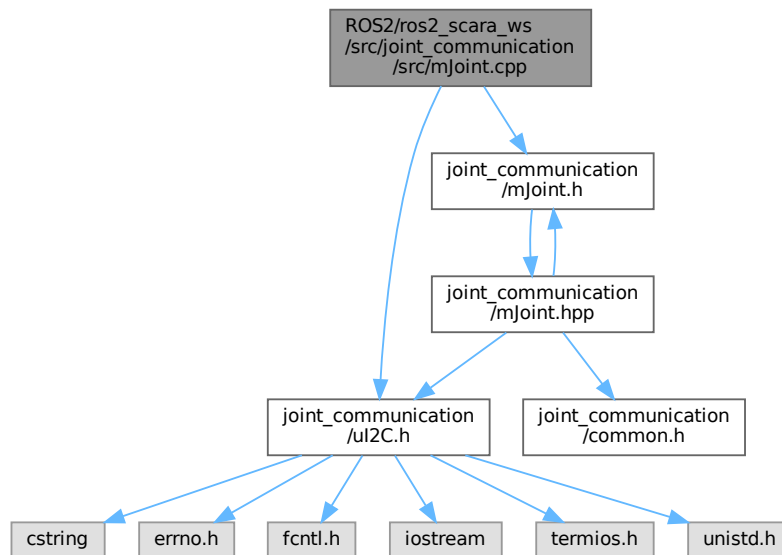
6.22 ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp File Reference

```
#include "joint_communication/mGripper.h"
Include dependency graph for mGripper.cpp:
```



6.23 ROS2/ros2_scara_ws/src/joint_communication/src/mJoint.cpp File Reference

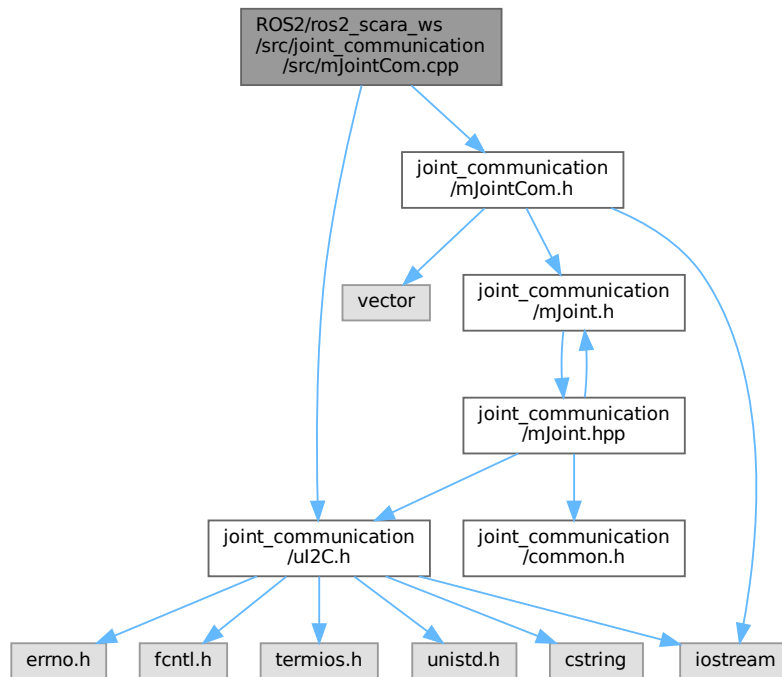
```
#include "joint_communication/uI2C.h"
#include "joint_communication/mJoint.h"
Include dependency graph for mJoint.cpp:
```



6.24 ROS2/ros2_scara_ws/src/joint_communication/src/mJointCom.cpp File Reference

```
#include "joint_communication/uI2C.h"
#include "joint_communication/mJointCom.h"
```

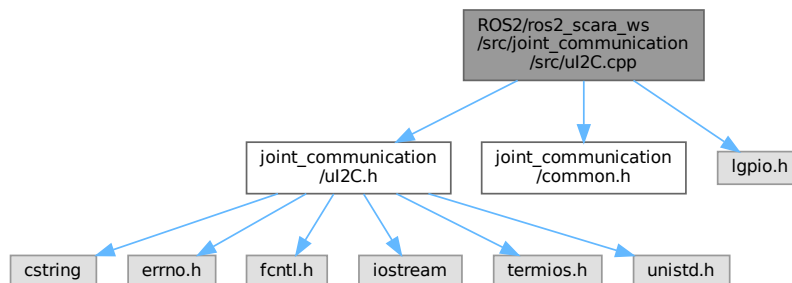

Include dependency graph for mJointCom.cpp:



6.25 ROS2/ros2_scara_ws/src/joint_communication/src/ul2C.cpp File Reference

```
#include "joint_communication/uI2C.h"
#include "joint_communication/common.h"
#include <lgpio.h>
```

Include dependency graph for ul2C.cpp:



Functions

- int [openI2CDevHandle](#) (const int dev_addr)

Initiates an I2C device on the bus.

- int `readFromI2CDev` (const int dev_handle, const int `reg`, char *buffer, const int data_length)
reads block of bytes from device to buffer
- int `writeToI2CDev` (const int dev_handle, const int `reg`, char *tx_buffer, const int data_length, char *RFLAGS_buffer)
writes block of bytes from buffer to device
- int `closeI2CDevHandle` (const int dev_handle)
close an I2C device on the bus

6.25.1 Function Documentation

6.25.1.1 `closeI2CDevHandle()`

```
int closeI2CDevHandle (
    const int dev_handle )
```

close an I2C device on the bus

Parameters

<code>dev_handle</code>	device handle obtained from <code>openI2CDevHandle</code>
-------------------------	---

Returns

0 on OK, negative on error.

6.25.1.2 `openI2CDevHandle()`

```
int openI2CDevHandle (
    const int dev_addr )
```

Initiates an I2C device on the bus.

Parameters

<code>dev_addr</code>	7-bit device address [0 - 0x7F]
-----------------------	---------------------------------

Returns

the device handle, negative on error.

6.25.1.3 `readFromI2CDev()`

```
int readFromI2CDev (
    const int dev_handle,
    const int reg,
```

```
char * buffer,  
const int data_length )
```

reads block of bytes from device to buffer

Parameters

<i>dev_handle</i>	device handle obtained from <code>openI2CDevHandle</code>
<i>reg</i>	the command/data register
<i>buffer</i>	pointer to data buffer to hold received values
<i>data_length</i>	number of bytes to read

Returns

number of bytes read, negative on error.

6.25.1.4 writeToI2CDev()

```
int writeToI2CDev (  
    const int dev_handle,  
    const int reg,  
    char * tx_buffer,  
    const int data_length,  
    char * RFLAGS_buffer )
```

writes block of bytes from buffer to device

Parameters

<i>dev_handle</i>	device handle obtained from <code>openI2CDevHandle</code>
<i>reg</i>	the command/data register
<i>tx_buffer</i>	pointer to data buffer holding the data to send
<i>data_length</i>	number of bytes to send
<i>RFLAGS_buffer</i>	buffer to hold returned flags

Returns

0 on OK, negative on error.

Index

- [_Gripper](#)
 - [joint_comm_node.cpp, 71](#)
 - [_Joints](#)
 - [joint_comm_node.cpp, 71](#)
 - [~Joint_comms](#)
 - [Joint_comms, 25](#)
 - [~RPI_PWM](#)
 - [RPI_PWM, 34](#)
- [ACK](#)
 - [joint.h, 41](#)
 - [ul2C.h, 64](#)
- [addJoint](#)
 - [Joint_comms, 25](#)
- [address](#)
 - [Joint, 22](#)
- [ADR](#)
 - [configuration.h, 38](#)
- [ANGLEMOVED](#)
 - [Joint, 14](#)
 - [joint.h, 42](#)
- [Arduino/joint/configuration.h, 37, 39](#)
- [Arduino/joint/joint.h, 39, 44](#)
- [Arduino/joint/joint.ino, 45](#)
- [checkCom](#)
 - [Joint, 15](#)
- [CHECKORIENTATION](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [checkOrientation](#)
 - [Joint, 15](#)
- [checkOrientations](#)
 - [Joint_comms, 25, 26](#)
- [chippath](#)
 - [RPI_PWM, 35](#)
- [CLEARSTALL](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [closeI2CDevHandle](#)
 - [ul2C.cpp, 74](#)
 - [ul2C.h, 65](#)
- [common.h](#)
 - [DUMP_BUFFER, 51](#)
- [configuration.h](#)
 - [ADR, 38](#)
 - [MAXACCEL, 38](#)
 - [MAXVEL, 38](#)
- [deinit](#)
 - [Gripper, 10](#)
 - [Joint, 15](#)
 - [Joint_comms, 26](#)
- [disable](#)
 - [Gripper, 10](#)
 - [Joint, 15](#)
 - [RPI_PWM, 34](#)
- [disableCL](#)
 - [Joint, 15](#)
- [DISABLECLOSEDLOOP](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [disableCLs](#)
 - [Joint_comms, 26](#)
- [DISABLEPID](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [disables](#)
 - [Joint_comms, 27](#)
- [DISABLESTALLGUARD](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [docs/DOCS_README.md, 49](#)
- [Documentation, 1](#)
- [DUMP_BUFFER](#)
 - [common.h, 51](#)
 - [joint.h, 41](#)
- [enable](#)
 - [Gripper, 10](#)
 - [Joint, 15](#)
 - [RPI_PWM, 34](#)
- [ENABLECLOSEDLOOP](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [ENABLEPID](#)
 - [Joint, 14](#)
 - [joint.h, 43](#)
- [enables](#)
 - [Joint_comms, 27](#)
- [ENABLESTALLGUARD](#)
 - [Joint, 14](#)
 - [joint.h, 42](#)
- [enableStallguard](#)
 - [Joint, 16](#)
- [enableStallguards](#)
 - [Joint_comms, 28](#)
- [ENCODER2JOINTANGLE](#)
 - [mJoint.h, 55](#)

- flags
 - Joint, [22](#)
- gearRatio
 - Joint, [22](#)
- GETDRIVERRPM
 - Joint, [14](#)
 - joint.h, [42](#)
- GETENCODERRPM
 - Joint, [14](#)
 - joint.h, [43](#)
- getFlags
 - Joint, [16](#)
- getIsHomed
 - Joint, [16](#)
- getIsSetup
 - Joint, [17](#)
- GETMOTORSTATE
 - Joint, [14](#)
 - joint.h, [42](#)
- GETPIDERROR
 - Joint, [14](#)
 - joint.h, [43](#)
- getPosition
 - Joint, [17](#)
- getPositions
 - Joint_comms, [28](#)
- getStall
 - Joint, [17](#)
- getVelocities
 - Joint_comms, [28](#)
- getVelocity
 - Joint, [18](#)
- Gripper, [9](#)
 - deinit, [10](#)
 - disable, [10](#)
 - enable, [10](#)
 - Gripper, [10](#)
 - init, [11](#)
 - pwm, [11](#)
 - setPosition, [11](#)
- handle
 - Joint, [23](#)
- HOME
 - Joint, [14](#)
 - joint.h, [43](#)
- home
 - Joint, [18](#)
 - Joint_comms, [29](#)
- init
 - Gripper, [11](#)
 - Joint, [18](#)
 - Joint_comms, [29](#)
- INT_handler
 - joint_comm_node.cpp, [70](#)
- ISHOMED
 - Joint, [14](#)
- joint.h, [43](#)
- isHomed
 - Joint, [18](#)
- ishomed
 - Joint, [23](#)
- ISSETUP
 - Joint, [14](#)
 - joint.h, [43](#)
- isSetup
 - Joint, [18](#)
- issetup
 - Joint, [23](#)
- ISSTALLED
 - Joint, [14](#)
 - joint.h, [43](#)
- J1
 - joint.ino, [46](#)
- Joint, [12](#)
 - address, [22](#)
 - ANGLEMOVED, [14](#)
 - checkCom, [15](#)
 - CHECKORIENTATION, [14](#)
 - checkOrientation, [15](#)
 - CLEARSTALL, [14](#)
 - deinit, [15](#)
 - disable, [15](#)
 - disableCL, [15](#)
 - DISABLECLOSEDLOOP, [14](#)
 - DISABLEPID, [14](#)
 - DISABLESTALLGUARD, [14](#)
 - enable, [15](#)
 - ENABLECLOSEDLOOP, [14](#)
 - ENABLEPID, [14](#)
 - ENABLESTALLGUARD, [14](#)
 - enableStallguard, [16](#)
 - flags, [22](#)
 - gearRatio, [22](#)
 - GETDRIVERRPM, [14](#)
 - GETENCODERRPM, [14](#)
 - getFlags, [16](#)
 - getIsHomed, [16](#)
 - getIsSetup, [17](#)
 - GETMOTORSTATE, [14](#)
 - GETPIDERROR, [14](#)
 - getPosition, [17](#)
 - getStall, [17](#)
 - getVelocity, [18](#)
 - handle, [23](#)
 - HOME, [14](#)
 - home, [18](#)
 - init, [18](#)
 - ISHOMED, [14](#)
 - isHomed, [18](#)
 - ishomed, [23](#)
 - ISSETUP, [14](#)
 - isSetup, [18](#)
 - issetup, [23](#)
 - ISSTALLED, [14](#)

- Joint, [15](#)
- MOVEANGLE, [14](#)
- MOVESTEPS, [14](#)
- moveSteps, [19](#)
- MOVETOANGLE, [14](#)
- MOVETOEND, [14](#)
- name, [23](#)
- offset, [23](#)
- PING, [14](#)
- printInfo, [19](#)
- read, [19](#)
- RUNCOTINOUS, [14](#)
- SETBRAKEMODE, [14](#)
- setBrakeMode, [20](#)
- SETCONTROLTHRESHOLD, [14](#)
- SETCURRENT, [14](#)
- setDriveCurrent, [20](#)
- SETHOLDCURRENT, [14](#)
- setHoldCurrent, [20](#)
- SETMAXACCELERATION, [14](#)
- SETMAXDECELERATION, [14](#)
- SETMAXVELOCITY, [14](#)
- setPosition, [21](#)
- SETRPM, [14](#)
- SETUP, [14](#)
- setVelocity, [21](#)
- STOP, [14](#)
- stop, [21](#)
- stp_reg_t, [14](#)
- write, [21](#)
- joint.h
 - ACK, [41](#)
 - ANGLEMOVED, [42](#)
 - CHECKORIENTATION, [43](#)
 - CLEARSTALL, [43](#)
 - DISABLECLOSEDLOOP, [43](#)
 - DISABLEPID, [43](#)
 - DISABLESTALLGUARD, [43](#)
 - DUMP_BUFFER, [41](#)
 - ENABLECLOSEDLOOP, [43](#)
 - ENABLEPID, [43](#)
 - ENABLESTALLGUARD, [42](#)
 - GETDRIVERRPM, [42](#)
 - GETENCODERRPM, [43](#)
 - GETMOTORSTATE, [42](#)
 - GETPIDERROR, [43](#)
 - HOME, [43](#)
 - ISHOMED, [43](#)
 - ISSETUP, [43](#)
 - ISSTALLED, [43](#)
 - MAX_BUFFER, [42](#)
 - MOVEANGLE, [42](#)
 - MOVESTEPS, [42](#)
 - MOVETOANGLE, [42](#)
 - MOVETOEND, [43](#)
 - NACK, [42](#)
 - PING, [42](#)
 - readValue, [43](#)
 - RFLAGS_SIZE, [42](#)
 - RUNCOTINOUS, [42](#)
 - SETBRAKEMODE, [43](#)
 - SETCONTROLTHRESHOLD, [43](#)
 - SETCURRENT, [42](#)
 - SETHOLDCURRENT, [42](#)
 - SETMAXACCELERATION, [42](#)
 - SETMAXDECELERATION, [42](#)
 - SETMAXVELOCITY, [42](#)
 - SETRPM, [42](#)
 - SETUP, [42](#)
 - STOP, [43](#)
 - stp_reg_t, [42](#)
 - writeValue, [43](#)
- joint.ino
 - J1, [46](#)
 - loop, [47](#)
 - receiveEvent, [47](#)
 - reg, [48](#)
 - requestEvent, [47](#)
 - rx_buf, [48](#)
 - rx_data_ready, [49](#)
 - rx_length, [49](#)
 - setup, [47](#)
 - stepper, [49](#)
 - stepper_receive_handler, [48](#)
 - stepper_request_handler, [48](#)
 - tx_buf, [49](#)
 - tx_data_ready, [49](#)
 - tx_length, [49](#)
- JOINT2ENCODERANGLE
 - mJoint.h, [55](#)
- joint_comm_node.cpp
 - _Gripper, [71](#)
 - _Joints, [71](#)
 - INT_handler, [70](#)
 - main, [70](#)
- Joint_comms, [23](#)
 - ~Joint_comms, [25](#)
 - addJoint, [25](#)
 - checkOrientations, [25, 26](#)
 - deinit, [26](#)
 - disableCLs, [26](#)
 - disables, [27](#)
 - enables, [27](#)
 - enableStallguards, [28](#)
 - getPositions, [28](#)
 - getVelocities, [28](#)
 - home, [29](#)
 - init, [29](#)
 - Joint_comms, [25](#)
 - joints, [33](#)
 - setBrakeModes, [30](#)
 - setDriveCurrents, [30](#)
 - setHoldCurrents, [31](#)
 - setPositions, [32](#)
 - setVelocities, [32](#)
 - stops, [32](#)

- joints
 - Joint_comms, 33
- loop
 - joint.ino, 47
- main
 - joint_comm_node.cpp, 70
- MAX_BUFFER
 - joint.h, 42
 - ul2C.h, 64
- MAXACCEL
 - configuration.h, 38
- MAXVEL
 - configuration.h, 38
- mJoint.h
 - ENCODER2JOINTANGLE, 55
 - JOINT2ENCODERANGLE, 55
- MOVEANGLE
 - Joint, 14
 - joint.h, 42
- MOVESTEPS
 - Joint, 14
 - joint.h, 42
- moveSteps
 - Joint, 19
- MOVETOANGLE
 - Joint, 14
 - joint.h, 42
- MOVETOEND
 - Joint, 14
 - joint.h, 43
- NACK
 - joint.h, 42
 - ul2C.h, 64
- name
 - Joint, 23
- offset
 - Joint, 23
- openI2CDevHandle
 - ul2C.cpp, 74
 - ul2C.h, 65
- per
 - RPI_PWM, 35
- PING
 - Joint, 14
 - joint.h, 42
- printInfo
 - Joint, 19
- pwm
 - Gripper, 11
- pwmpath
 - RPI_PWM, 35
- read
 - Joint, 19
- readFromI2CDev
 - ul2C.cpp, 74
 - ul2C.h, 65
- readValue
 - joint.h, 43
- receiveEvent
 - joint.ino, 47
- reg
 - joint.ino, 48
- requestEvent
 - joint.ino, 47
- RFLAGS_SIZE
 - joint.h, 42
 - ul2C.h, 64
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 49, 51
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 51, 53
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 53, 55
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 57, 59
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 59, 61
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 62, 66
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication
 - 67, 69
- ROS2/ros2_scara_ws/src/joint_communication/src/joint_comm_node.cpp,
 - 70
- ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp,
 - 71
- ROS2/ros2_scara_ws/src/joint_communication/src/mJoint.cpp,
 - 72
- ROS2/ros2_scara_ws/src/joint_communication/src/mJointCom.cpp,
 - 72
- ROS2/ros2_scara_ws/src/joint_communication/src/ul2C.cpp,
 - 73
- RPI_PWM, 33
 - ~RPI_PWM, 34
 - chippath, 35
 - disable, 34
 - enable, 34
 - per, 35
 - pwmpath, 35
 - setDutyCycle, 34
 - setDutyCycleNS, 34
 - setPeriod, 34
 - start, 34
 - stop, 35
 - writeSYS, 35
- RUNCOTINOUS
 - Joint, 14
 - joint.h, 42
- rx_buf
 - joint.ino, 48
- rx_data_ready
 - joint.ino, 49
- rx_length

- joint.ino, 49
- SETBRAKEMODE
 - Joint, 14
 - joint.h, 43
- setBrakeMode
 - Joint, 20
- setBrakeModes
 - Joint_comms, 30
- SETCONTROLTHRESHOLD
 - Joint, 14
 - joint.h, 43
- SETCURRENT
 - Joint, 14
 - joint.h, 42
- setDriveCurrent
 - Joint, 20
- setDriveCurrents
 - Joint_comms, 30
- setDutyCycle
 - RPI_PWM, 34
- setDutyCycleNS
 - RPI_PWM, 34
- SETHOLDCURRENT
 - Joint, 14
 - joint.h, 42
- setHoldCurrent
 - Joint, 20
- setHoldCurrents
 - Joint_comms, 31
- SETMAXACCELERATION
 - Joint, 14
 - joint.h, 42
- SETMAXDECELERATION
 - Joint, 14
 - joint.h, 42
- SETMAXVELOCITY
 - Joint, 14
 - joint.h, 42
- setPeriod
 - RPI_PWM, 34
- setPosition
 - Gripper, 11
 - Joint, 21
- setPositions
 - Joint_comms, 32
- SETRPM
 - Joint, 14
 - joint.h, 42
- SETUP
 - Joint, 14
 - joint.h, 42
- setup
 - joint.ino, 47
- setVelocities
 - Joint_comms, 32
- setVelocity
 - Joint, 21
- start
 - RPI_PWM, 34
- stepper
 - joint.ino, 49
- stepper_receive_handler
 - joint.ino, 48
- stepper_request_handler
 - joint.ino, 48
- STOP
 - Joint, 14
 - joint.h, 43
- stop
 - Joint, 21
 - RPI_PWM, 35
- stops
 - Joint_comms, 32
- stp_reg_t
 - Joint, 14
 - joint.h, 42
- Todo List, 3
- tx_buf
 - joint.ino, 49
- tx_data_ready
 - joint.ino, 49
- tx_length
 - joint.ino, 49
- ul2C.cpp
 - closeI2CDevHandle, 74
 - openI2CDevHandle, 74
 - readFromI2CDev, 74
 - writeToI2CDev, 76
- ul2C.h
 - ACK, 64
 - closeI2CDevHandle, 65
 - MAX_BUFFER, 64
 - NACK, 64
 - openI2CDevHandle, 65
 - readFromI2CDev, 65
 - RFLAGS_SIZE, 64
 - writeToI2CDev, 66
- write
 - Joint, 21
- writeSYS
 - RPI_PWM, 35
- writeToI2CDev
 - ul2C.cpp, 76
 - ul2C.h, 66
- writeValue
 - joint.h, 43