Bioscara

Generated by Doxygen 1.9.8

Chapter 1

Documentation

This documentation currently documents how the robot controller communicates with the joint controllers, this includes:

- The joint firmware in the `/Arduino` directory
- The interfacing library used for communicating with the joints in the `/ROS2` directory.

1.1 Usage

the joint_communication library is structured as a ROS2 package but can also be used in another build toolchain. If that is the case ensure the include paths are still correct.

Chapter 2

README

This package contains all launch and config files for the robot to work.

Chapter 3

README

The packages are structured according to this guide: RTW Package Structure

When compiling the package is installed in the share/ directory. Also the URDF is stored there. The bioscara.launch.py file expects to find the urdf there. This is done in the packages cmake file

```
install(
  DIRECTORY hardware/include/
  DESTINATION include/ros2_control_demo_example_1
)
install(
  DIRECTORY description/launch description/ros2_control description/urdf
  DESTINATION share/ros2_control_demo_example_1
)
install(
  DIRECTORY bringup/launch bringup/config
  DESTINATION share/ros2_control_demo_example_1
)
install(TARGETS ros2_control_demo_example_1
  EXPORT export_ros2_control_demo_example_1
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION bin
)
```

TODO:

- [ ] Format and rework this content

Chapter 4

Todo List

- **Member bioscara_hardware_interface::BioscaraHardwareInterface::on_init (const hardware_interface::HardwareInfo &info) override**

  Implement sensors and GPIO

- **Member bioscara_hardware_interface::BioscaraHardwareInterface::on_shutdown (const rclcpp_lifecycle::State &previous_state) override**

  Research in ROS2_control source code if this is ever called from any state other than UNCONFIGURED

- **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 std::string name, const int address, const float reduction, const float offset)**

  Measure joint ranges

  - Investigate if possible to make independent of homing

- **Member Joint_comms::checkOrientations (float angle=10.0)**

  Only execute if not performed before

  - save in private flag and inhibit movement if this has not been executed.

Chapter 5

Namespace Index

5.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Index

7.1 Class List

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

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

Chapter 9

Namespace Documentation

9.1 bioscara Namespace Reference

Functions

- generate_launch_description ()

9.1.1 Function Documentation

9.1.1.1 generate_launch_description()

```
bioscara.generate_launch_description ( )
```

9.2 bioscara_hardware_interface Namespace Reference

Classes

- class BioscaraHardwareInterface

9.3 display Namespace Reference

Functions

- generate_launch_description ()

9.3.1 Function Documentation

9.3.1.1 generate_launch_description()

```
display.generate_launch_description ( )
```

9.4 gazebo Namespace Reference

Functions

- generate_launch_description ()

9.4.1 Function Documentation

9.4.1.1 generate_launch_description()

```
gazebo.generate_launch_description ( )
```

9.5 setup Namespace Reference

Variables

- str package_name = 'bioscara_description'
- name
- version
- packages
- data_files
- install_requires
- zip_safe
- maintainer
- maintainer_email
- description
- license
- tests_require
- entry_points

9.5.1 Variable Documentation

9.5.1.1 data_files

```
setup.data_files
```

9.5.1.2 description

```
setup.description
```

9.5.1.3 entry_points

```
setup.entry_points
```

9.5.1.4 install_requires

```
setup.install_requires
```

9.5.1.5 license

```
setup.license
```

9.5.1.6 maintainer

```
setup.maintainer
```

9.5.1.7 maintainer_email

```
setup.maintainer_email
```

9.5.1.8 name

```
setup.name
```

9.5.1.9 package_name

```
str setup.package_name = 'bioscara_description'
```

9.5.1.10 packages

```
setup.packages
```

9.5.1.11 tests_require

```
setup.tests_require
```

9.5.1.12 version

```
setup.version
```

9.5.1.13 zip_safe

```
setup.zip_safe
```

9.6 test_copyright Namespace Reference

Functions

- test_copyright ()

9.6.1 Function Documentation

9.6.1.1 test_copyright()

```
test_copyright.test_copyright ( )
```

9.7 test_flake8 Namespace Reference

Functions

- test_flake8 ()

9.7.1 Function Documentation

9.7.1.1 test_flake8()

```
test_flake8.test_flake8 ( )
```

9.8 test_forward_position_controller Namespace Reference

Functions

- generate_launch_description ()

9.8.1 Function Documentation

9.8.1.1 generate_launch_description()

```
test_forward_position_controller.generate_launch_description ( )
```

9.9 test_joint_trajectory_controller Namespace Reference

Functions

- generate_launch_description ()

9.9.1 Function Documentation

9.9.1.1 generate_launch_description()

```
test_joint_trajectory_controller.generate_launch_description ( )
```

9.10 test_pep257 Namespace Reference

Functions

- test_pep257 ()

9.10.1 Function Documentation

9.10.1.1 test_pep257()

```
test_pep257.test_pep257 ( )
```

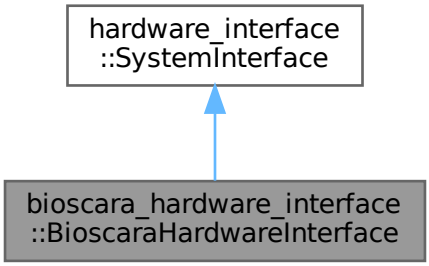
Chapter 10

Class Documentation

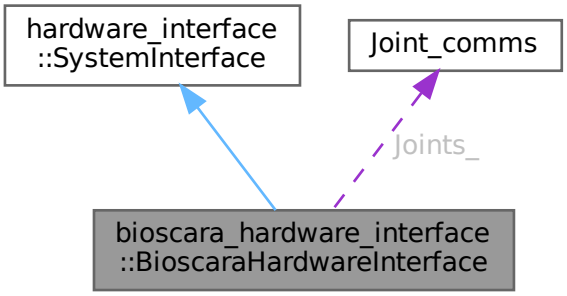
10.1 bioscara_hardware_interface::BioscaraHardwareInterface Class Reference

`#include <bioscara_hardware.hpp>`

Inheritance diagram for bioscara_hardware_interface::BioscaraHardwareInterface:



Collaboration diagram for bioscara_hardware_interface::BioscaraHardwareInterface:



Public Member Functions

- hardware_interface::CallbackReturn on_init (const hardware_interface::HardwareInfo &info) override
- hardware_interface::CallbackReturn on_shutdown (const rclcpp_lifecycle::State &previous_state) override

  Called on transition to FINALIZED state.
- hardware_interface::CallbackReturn on_configure (const rclcpp_lifecycle::State &previous_state) override
- hardware_interface::CallbackReturn on_cleanup (const rclcpp_lifecycle::State &previous_state) override
- hardware_interface::CallbackReturn on_activate (const rclcpp_lifecycle::State &previous_state) override
- hardware_interface::CallbackReturn on_deactivate (const rclcpp_lifecycle::State &previous_state) override
- hardware_interface::return_type read (const rclcpp::Time &time, const rclcpp::Duration &period) override
- hardware_interface::return_type write (const rclcpp::Time &time, const rclcpp::Duration &period) override

Private Attributes

- Joint_comms Joints_

10.1.1 Member Function Documentation

10.1.1.1 on_activate()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_activate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

10.1.1.2 on_cleanup()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_cleanup (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

Disconnect from the joints and throw error if it fails

10.1.1.3 on_configure()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_configure (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

Connect to the joints and throw error if it fails

10.1.1.4 on_deactivate()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_deactivate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

10.1.1.5 on_init()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_init (
            const hardware_interface::HardwareInfo & info )  [override]
```

Loop over all joints decribed in the hardware description file, check if they have only the position command and state interface defined and finally add them to the internal Joints_ list

Todo • Implement sensors and GPIO

10.1.1.6 on_shutdown()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↩
::on_shutdown (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

Called on transition to FINALIZED state.

Removes all joints from the com object.

Todo Research in ROS2_control source code if this is ever called from any state other than UNCONFIGURED

10.1.1.7 read()

```
hardware_interface::return_type bioscara_hardware_interface::BioscaraHardwareInterface::read (
            const rclcpp::Time & time,
            const rclcpp::Duration & period )  [override]
```

10.1.1.8 write()

```
hardware_interface::return_type bioscara_hardware_interface::BioscaraHardwareInterface::write
(
            const rclcpp::Time & time,
            const rclcpp::Duration & period )  [override]
```

10.1.2 Member Data Documentation

10.1.2.1 Joints_

```
Joint_comms bioscara_hardware_interface::BioscaraHardwareInterface::Joints_  [private]
```

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

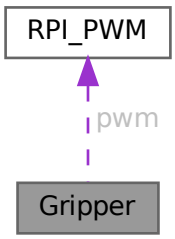
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/include/bioscara_hardware_↩ interface/bioscara_hardware.hpp
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/src/bioscara_hardware.cpp

10.2 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

10.2.1 Detailed Description

Gripper object to interact with the robot gripper.

This class is a wrapper function to interact with a PWM servo gripper. An example application is shown below. Note that depending on the build toolchain the include path can differ. This example assumes the bioscara_hardware_driver package is built with ROS2.

```
#include "bioscara_hardware_driver/mGripper.h"
int main(int argc, char **argv)
{
    Gripper gripper;
    gripper.init();
```

```
    if(gripper.enable() != 0){
        cerr << "Failed to engage gripper" << endl;
        return -1;
    }

    if (gripper.setPosition(40) != 0)
    {
        cerr << "setting position failed" << endl;
        return -1;
    }

    if(gripper.disable() != 0){
        cerr << "Failed to disengage gripper" << endl;
        return -1;
    }

    gripper.deinit();
    return 0;
}
```

10.2.2 Constructor & Destructor Documentation

10.2.2.1 Gripper()

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

10.2.3 Member Function Documentation

10.2.3.1 deinit()

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

Placeholder, does nothing.

Returns

0

10.2.3.2 disable()

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

Disables the servo.

Stops the servo and disables the PWM generation.

Returns

non-zero error code.

10.2.3.3 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.

10.2.3.4 init()

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

Placeholder, does nothing.

Returns

0

10.2.3.5 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

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

10.2.4 Member Data Documentation

10.2.4.1 pwm

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

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

- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mGripper.h
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mGripper.cpp

10.3 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 reduction, const float offset)
- int init (void)
- int deinit (void)
- int printInfo (void)
- int getPosition (float &pos)

  get the current joint position in radians or m for cylindrical and prismatic joints respectively.
- int setPosition (float pos)

  get the current joint position in radians or m for cylindrical and prismatic joints respectively.
- int moveSteps (int32_t steps)

  Move full steps.
- int getVelocity (float &vel)

  get the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.
- int setVelocity (float vel)

  Set the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.
- int checkOrientation (float angle=10.0)

  Calls the checkOrientation method of the motor. Checks in which direction the motor is turning.
- int enable (u_int8_t driveCurrent, u_int8_t holdCurrent)

  Initialize the joint and engages motor.
- int disable (void)

  disenganges 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 setMaxAcceleration (float maxAccel)

  Set the maximum permitted joint acceleration (and deceleration) in rad/s^2 or m/s^2 for cylindrical and prismatic joints respectively.
- int setMaxVelocity (float maxVel)

  Set the maximum permitted joint velocity in rad/s or m/s for cylindrical and prismatic joints respectively.
- int enableStallguard (u_int8_t sensitivity)

  Enable encoder stall detection. A detected stall can be reset by homeing.
- bool isHomed (void)

  Checks the state if the motor is homed.
- bool isEnabled (void)

  Checks the state if the motor is enabled.
- bool isStalled (void)

  Checks if the motor is stalled.
- 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.
- int address

  I2C adress.
- float reduction = 1

  Joint to actuator reduction ratio.
- float offset = 0

  Joint position offset.
- int handle = -1

  I2C bus handle.

10.3.1 Detailed Description

Representing a single joint on the I2C bus.

10.3.2 Member Enumeration Documentation

10.3.2.1 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].

10.3.3 Constructor & Destructor Documentation

10.3.3.1 Joint()

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

10.3.4 Member Function Documentation

10.3.4.1 checkCom()

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

10.3.4.2 checkOrientation()

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

Calls the checkOrientation method of the motor. Checks in which direction the motor is turning.

As the orientation check is blocking on the motor, this this function returns when the isBusy flag is clear again.

Parameters

angle	degrees how much the motor should turn. A few degrees is sufficient.

Returns

0 on success, -1 on communication error, -3 when the motor is not enabled, -4 when the motor is stalled.

10.3.4.3 deinit()

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

10.3.4.4 disable()

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

disenganges the joint motor without closing i2c handle

Returns

error code.

10.3.4.5 disableCL()

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

Disables the Closed-Loop PID Controller.

Returns

error code.

10.3.4.6 enable()

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

Initialize the joint and engages motor.

Parameters

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

Returns

0 on success, -1 on communication error, -3 when the motor is not enabled.

10.3.4.7 enableStallguard()

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

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

Parameters

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

10.3.4.8 getFlags()

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

get driver state flags

Returns

flags.

10.3.4.9 getPosition()

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

get the current joint position in radians or m for cylindrical and prismatic joints respectively.

Parameters

pos	

Returns

error code

10.3.4.10 getVelocity()

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

get the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.

Parameters

vel	

Returns

error code

10.3.4.11 home()

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

Homes the motor.

Parameters

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

Returns

0 on success, -1 on communication error, -2 when not homed succesfull (isHomed flag still not set), -3 when the motor is not enabled.

10.3.4.12 init()

```
int Joint::init (
```

```
            void  )
```

10.3.4.13 isEnabled()

```
bool Joint::isEnabled (
            void  )
```

Checks the state if the motor is enabled.

Reads the internal state flags from the last transmission. If an update is neccessary call getFlags() before invoking this function. If the motor actually can move depends on the state of the STALLED flag which can be checked using isStalled().

Returns

true if the motor is enabled, false if not.

10.3.4.14 isHomed()

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

Checks the state if the motor is homed.

Reads the internal state flags from the last transmission. If an update is neccessary call getFlags() before invoking this function.

Returns

true if the motor is homed, false if not.

10.3.4.15 isStalled()

```
bool Joint::isStalled (
            void  )
```

Checks if the motor is stalled.

Reads the internal state flags from the last transmission. If an update is neccessary call getFlags() before invoking this function.

Returns

true if the motor is stalled, false if not.

10.3.4.16 moveSteps()

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

Move full steps.

This function can be called even when not homed.

Parameters

steps	number of full steps

Returns

0 on success, -1 on communication error, -3 when the motor is not enabled, -4 when the motor is stalled.

10.3.4.17 printInfo()

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

10.3.4.18 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

T	Datatype of value to be transmitted

Parameters

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

Returns

0 on OK, negative on error

10.3.4.19 setBrakeMode()

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

Set Brake Mode.

Parameters

mode	Freewheel: 0, Coolbrake: 1, Hardbrake: 2

Returns

error code.

10.3.4.20 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

current	0% - 100% of driver current

Returns

error code.

10.3.4.21 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

current	0% - 100% of driver current

Returns

error code.

10.3.4.22 setMaxAcceleration()

```
int Joint::setMaxAcceleration (
            float maxAccel )
```

Set the maximum permitted joint acceleration (and deceleration) in rad/s^2 or m/s^2 for cylindrical and prismatic joints respectively.

Parameters

maxAccel	maximum joint acceleration.

Returns

error code

10.3.4.23 setMaxVelocity()

```
int Joint::setMaxVelocity (
            float maxVel )
```

Set the maximum permitted joint velocity in rad/s or m/s for cylindrical and prismatic joints respectively.

Parameters

maxVel	maximum joint velocity.

Returns

error code

10.3.4.24 setPosition()

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

get the current joint position in radians or m for cylindrical and prismatic joints respectively.

Parameters

pos	in rad or m

Returns

0 on success, -1 on communication error, -2 when not homed, -3 when the motor is not enabled, -4 when the motor is stalled.

10.3.4.25 setVelocity()

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

Set the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.

Parameters

vel	

Returns

0 on success, -1 on communication error, -2 when not homed, -3 when the motor is not enabled, -4 when the motor is stalled.

10.3.4.26 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. Continously checking the busy flag also might interfere with the stop() function on the controller side.

Parameters

mode	Hard: 0, Soft: 1

Returns

error code.

10.3.4.27 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

T	Datatype of value to be transmitted

Parameters

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

Returns

0 on OK, negative on error

10.3.5 Member Data Documentation

10.3.5.1 address

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

I2C adress.

10.3.5.2 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	ENABLED	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 or the Stallguard enabled.
BUSY is set if the slave is busy processing a previous command.
HOMED is cleared if the joint is homed. Movement is only allowed if this flag is clear
ENABLED is cleared if the joint is enabled after calling Joint::enable()

10.3.5.3 handle

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

I2C bus handle.

10.3.5.4 name

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

10.3.5.5 offset

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

Joint position offset.

10.3.5.6 reduction

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

Joint to actuator reduction ratio.

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

- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mJoint.h
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mJoint.hpp
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mJoint.cpp

10.4 Joint_comms Class Reference

Communication object for all joints.

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

Public Member Functions

- Joint_comms (void)
- ~Joint_comms ()
- int init (void)

  Connects to all joints.
- int deinit (void)

  Disconnects all joints from the I2C bus.
- void addJoint (const std::string name, const int address, const float reduction, const float offset)

  add a Joint.
- void removeJoint (const std::string name)

  removes a joint.
- void removeJoints (void)

  removes all joints from the communication object.
- int enable (const std::string name, const u_int8_t driveCurrent, const u_int8_t holdCurrent)

  Engage a joint by name.
- int disables (void)

  Disenganges all joints without closing i2c handle.
- int home (const std::string name, const u_int8_t direction, const u_int8_t rpm, const u_int8_t sensitivity, const u_int8_t current)

Executes the homing sequence of a joint.

- int getPosition (const std::string name, float &angle)

  Get the position of the joint by name.
- int setPosition (const std::string name, const float angle)

  Set the position of the joint by name.
- int getVelocity (const std::string name, float &degps)

  Get the velocity of a joint by name.
- int setVelocity (const std::string name, float degps)

  Set the velocity of a joint by name.
- int checkOrientations (float angle=10.0)

  Sequentially checks the orientations of each joint.
- int checkOrientation (const std::string name, float angle=10.0)

  Checks the orientations of the specified joint. This function is automatically called when homing a joint.
- int stops (bool mode)

  Stops the motors.
- int enableStallguard (const std::string name, const u_int8_t threshold)

  Enable encoder stall detection of specified joint.
- int setMaxAcceleration (const std::string name, float maxAccel)

  Set the maximum permitted joint acceleration (and deceleration) in deg/s^2 or mm/s^2 for cylindrical and prismatic joints respectively.
- int setMaxVelocity (const std::string name, float maxVel)

  Set the maximum permitted joint velocity in deg/s or mm/s for cylindrical and prismatic joints respectively.

Public Attributes

- std::unordered_map< std::string, Joint > joints

  unordered map storing the Joint objects.

10.4.1 Detailed Description

Communication object for all joints.

Class handling interfacing with the joints.

The methods of this class are optimized for easy use in the ROS2_control hardware interface methods.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 Joint_comms()

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

10.4.2.2 ~Joint_comms()

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

10.4.3 Member Function Documentation

10.4.3.1 addJoint()

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

add a Joint.

Appends a joint to internal map.

Parameters

name	string device name for output logs
address	1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4
reduction	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;
offset	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

10.4.3.2 checkOrientation()

```
int Joint_comms::checkOrientation (
            const std::string name,
            float angle = 10.0 )
```

Checks the orientations of the specified joint. This function is automatically called when homing a joint.

When checking the orientation the motor moves a few degrees and compares the encoder output. It then internally saves the direction it is wired. This function should only be called after the joint has just been powered up. This function must be called after the joint has been enabled with enable() and before any movement.

Parameters

name	name of the joint to check the orientation.
angle	degrees in motor units to rotate to check the orientation. Should be small values of a few degrees.

Returns

error code.

10.4.3.3 checkOrientations()

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

Sequentially checks the orientations of each joint.

This function should only be called after the joint has just been powered up. This function must be called after the joint has been enabled with enables() and before any movement.

Parameters

angle	degrees in motor units 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.

10.4.3.4 deinit()

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

Disconnects all joints from the I2C bus.

Deinitializes all joints by removing them from the I2C bus.

Returns

0 on success, non-zero otherwise

10.4.3.5 disables()

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

Disenganges all joints without closing i2c handle.

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

Returns

error code.

10.4.3.6 enable()

```
int Joint_comms::enable (
            const std::string name,
            const u_int8_t driveCurrent,
            const u_int8_t holdCurrent )
```

Engage a joint by name.

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

Parameters

driveCurrent	drive current 0-100%.
holdCurrent	hold current 0-100%.

Returns

error code.

10.4.3.7 enableStallguard()

```
int Joint_comms::enableStallguard (
            const std::string name,
            const u_int8_t threshold )
```

Enable encoder stall detection of specified joint.

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

Parameters

name	name of joint to enable stall detection
thresholds	value of threshold. 0 - 255 where lower is more sensitive.

10.4.3.8 getPosition()

```
int Joint_comms::getPosition (
            const std::string name,
            float & angle )
```

Get the position of the joint by name.

The current positions of the joint specified by the name is returned. The units are degrees and mm for revolute and prismatic joints respectively. Joint::getPosition()

Parameters

angle	Reference to allocated variable to hold the joint position.

Returns

error code.

10.4.3.9 getVelocity()

```
int Joint_comms::getVelocity (
            const std::string name,
            float & degps )
```

Get the velocity of a joint by name.

The current velocity of the specified joint is returned. The units are degrees/s and mm/s for revolute and prismatic joints respectively.

Parameters

degps	Reference to variable to hold all the joint velocity.

Returns

error code.

10.4.3.10 home()

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

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

Returns

error code.

10.4.3.11 init()

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

Connects to all joints.

Iterates over all joints and connects to them on the I2C bus and tests if they are responsive.

Warning

Add some joints using addJoint() before calling this function.

Returns

0 on success, non-zero otherwise

10.4.3.12 removeJoint()

```
void Joint_comms::removeJoint (
            const std::string name )
```

removes a joint.

removes a joint from the internal map by name.

Parameters

name	string device name of the joint to remove

10.4.3.13 removeJoints()

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

removes all joints from the communication object.

10.4.3.14 setMaxAcceleration()

```
int Joint_comms::setMaxAcceleration (
            const std::string name,
            float maxAccel )
```

Set the maximum permitted joint acceleration (and deceleration) in deg/s^2 or mm/s^2 for cylindrical and prismatic joints respectively.

Parameters

maxAccel	maximum joint acceleration.

Returns

error code

10.4.3.15 setMaxVelocity()

```
int Joint_comms::setMaxVelocity (
```

```
            const std::string name,
            float maxVel )
```

Set the maximum permitted joint velocity in deg/s or mm/s for cylindrical and prismatic joints respectively.

Parameters

maxVel	maximum joint velocity.

Returns

error code

10.4.3.16 setPosition()

```
int Joint_comms::setPosition (
            const std::string name,
            const float angle )
```

Set the position of the joint by name.

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

Parameters

angle	value of new target positions.

Returns

error code.

10.4.3.17 setVelocity()

```
int Joint_comms::setVelocity (
            const std::string name,
            float degps )
```

Set the velocity of a joint by name.

Set the new target velocity of the specified joint. The units are degrees/s and mm/s for revolute and prismatic joints respectively.

Parameters

degps	New target velocity.

Returns

error code.

10.4.3.18 stops()

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

Stops the motors.

Stops all motors either soft or hard.

Parameters

mode	Hard: 0, Soft: 1

Returns

error code.

10.4.4 Member Data Documentation

10.4.4.1 joints

```
std::unordered_map<std::string, Joint> Joint_comms::joints
```

unordered map storing the Joint objects.

an unordered map is chosen to simplify acces via the joint name, as this conforms well with the ROS2_control hardware interface The map does not need to be ordered. Search, insertion, and removal of elements have average constant-time complexity.

A Joint can be added by invoking addJoint() A joint can be removed by invoking remvoveJoint()

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

- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mJointCom.h
- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mJointCom.cpp

10.5 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

10.5.1 Detailed Description

PWM class for the Raspberry PI 4 and 5.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 ~RPI_PWM()

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

10.5.3 Member Function Documentation

10.5.3.1 disable()

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

10.5.3.2 enable()

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

10.5.3.3 setDutyCycle()

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

Sets the duty cycle.

Parameters

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

10.5.3.4 setDutyCycleNS()

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

10.5.3.5 setPeriod()

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

10.5.3.6 start()

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

Starts the PWM

Parameters

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

10.5.3.7 stop()

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

Stops the PWM

10.5.3.8 writeSYS()

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

10.5.4 Member Data Documentation

10.5.4.1 chippath

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

10.5.4.2 per

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

10.5.4.3 pwmpath

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

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

- ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/uPWM.h

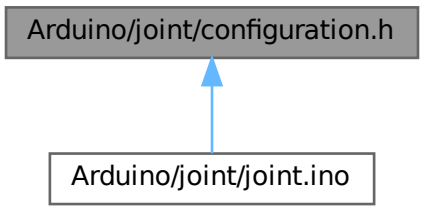
Chapter 11

File Documentation

11.1 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/s^2. 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.

11.1.1 Detailed Description

Configuration definitions for Joint 1 to Joint 4.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

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

11.1.2 Macro Definition Documentation

11.1.2.1 ADR

```
#define ADR 0x11
```

I2C adress of joint n is 0x1n.

11.1.2.2 MAXACCEL

```
#define MAXACCEL 10000
```

Maximum acceleration in steps/s^2. Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.

11.1.2.3 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.

11.2 configuration.h

Go to the documentation of this file.

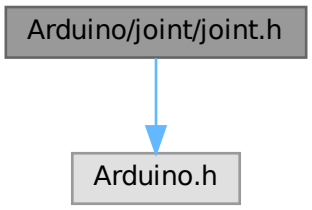
```
00001
00014 #ifndef CONFIGURATION_H
00015 #define CONFIGURATION_H
00016
00017 #if defined(J1)
00019 #define ADR 0x11
00020 #define MAXACCEL 10000
00021 #define MAXVEL 800
00022
00023 #elif defined(J2)
00024 #define ADR 0x12
00025 #define MAXACCEL 10000
00026 #define MAXVEL 800
00027
00028 #elif defined(J3)
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
```

11.3 Arduino/joint/joint.h File Reference

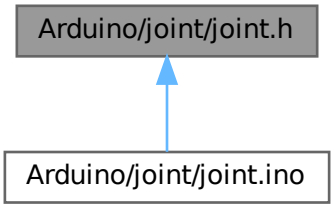
joint firmware header

`#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.

11.3.1 Detailed Description

joint firmware header

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

This file contains definitions and macros for the joint firmware.

11.3.2 Macro Definition Documentation

11.3.2.1 ACK

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

11.3.2.2 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

buffer	pointer to a buffer to dump to the console
size	number of bytes to dump

11.3.2.3 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

11.3.2.4 NACK

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

11.3.2.5 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

11.3.3 Enumeration Type Documentation

11.3.3.1 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	W; Size: 4; [(float) deg/s^2].
SETMAXDECELERATION	
SETMAXVELOCITY	W; Size: 4; [(float) deg/s].
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].

11.3.4 Function Documentation

11.3.4.1 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

val	Reference to output variable
rxBuf	Buffer to read value from
rx_length	Length of the buffer

11.3.4.2 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

val	Reference to input variable
txBuf	pointer to tx buffer
tx_length	Length of the buffer returne

Returns

0 On success

11.4 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)
```

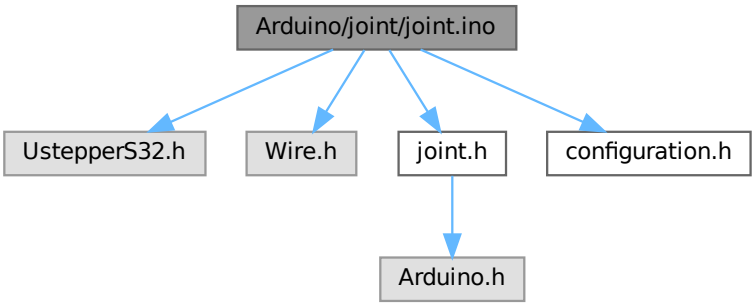
```
00105 {
00106   memcpy(&val, rxBuf, rx_length);
00107 }
00108
00116 template <typename T>
00117 int writeValue(const T val, uint8_t *txBuf, size_t &tx_length)
00118 {
00119   tx_length = sizeof(T);
00120   memcpy(txBuf, &val, tx_length);
00121   return 0;
00122 }
00123
00124 #endif
```

11.5 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 J4

  Define either joint that is to be flashed.

Functions

- void blocking_handler (uint8_t reg)

  Handles commands received via I2C.
- void non_blocking_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

11.5.1 Detailed Description

joint firmware

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains the joint firmware.

11.5.2 Macro Definition Documentation

11.5.2.1 J4

```
#define J4
```

Define either joint that is to be flashed.

Define either J1, J2, J3 or J4 and subsequently include configuration.h

11.5.3 Function Documentation

11.5.3.1 blocking_handler()

```
void blocking_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.

The registers handled in this handler are those whose implementation can take time and can thereby not be called directly from the request handler.

Parameters

reg	command that should be executed.

11.5.3.2 loop()

```
void loop (
            void  )
```

Main loop.

Executes the following:
1) if isStallguardEnabled: compares stepper.getPidError() with stallguardThreshold and sets isStalled flag.
2) sets/clears notHomed flag if the joint is homed or not.
3) sets/clears notSetup if the joint is setup or not.
4) if rx_data_ready: set isBusy flag to indicate device is busy. Invoke blocking_handler. Clear isBusy flag to indicate device is no longer busy

11.5.3.3 non_blocking_handler()

```
void non_blocking_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.

Parameters

reg	command to execute/register to read.

11.5.3.4 receiveEvent()

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

I2C receive event Handler.

Reads the content of the received message. Saves the register 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. Every I2C transaction starts with a receive event when the command is sent and is immediatly followed by a request since at minimum the flags need to be transmitted back. This means that the receive handler and request handler are always executed sequentially. The main loop is not executed since both handlers are ISRs. 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.

Parameters

n	the number of bytes read from the controller device: MAX_BUFFER

11.5.3.5 requestEvent()

```
void requestEvent ( )
```

I2C request event Handler.

Sends the response data to the master. Every transaction begins with a receive event. The request event is always triggered since at a minimum the status flags are returned to the master. Hence this function is only invoked after the receiveEvent() handler has been called. The function calls the non_blocking_handler() which is non-blocking. Since most Ustepper functions are non-blocking as they just read/write registers to the stepper driver/encoder they can be handled directly in the ISR. The non_blocking_handler() populates the tx_buf with relevant data, the current state flags are appended to the tx_buf and then it is send to the master.

11.5.3.6 setup()

```
void setup (
            void  )
```

Setup Peripherals.

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

11.5.4 Variable Documentation

11.5.4.1 reg

```
uint8_t reg = 0
```

11.5.4.2 rx_buf

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

11.5.4.3 rx_data_ready

```
bool rx_data_ready = 0
```

11.5.4.4 rx_length

```
size_t rx_length = 0
```

11.5.4.5 stepper

```
UstepperS32 stepper
```

11.5.4.6 tx_buf

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

11.5.4.7 tx_data_ready

```
bool tx_data_ready = 0
```

11.5.4.8 tx_length

```
size_t tx_length = 0
```

11.6 docs/DOCS_README.md File Reference

11.7 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ bringup/launch/bioscara.launch.py File Reference

Namespaces

- namespace bioscara

Functions

- bioscara.generate_launch_description ()

11.8 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ bringup/launch/test_forward_position_controller.launch.py File Reference

Namespaces

- namespace test_forward_position_controller

Functions

- test_forward_position_controller.generate_launch_description ()

11.9 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ bringup/launch/test_joint_trajectory_controller.launch.py File Reference

Namespaces

- namespace test_joint_trajectory_controller

Functions

- test_joint_trajectory_controller.generate_launch_description ()

11.10 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/↩ README.md File Reference

11.11 ROS2/ros2_scara_ws/src/dalsa_bioscara/README.md File Reference

11.12 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ description/bioscara_description/__init__.py File Reference

11.13 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ description/launch/display.launch.py File Reference

Namespaces

- namespace display

Functions

- display.generate_launch_description ()

11.14 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_↩ description/launch/gazebo.launch.py File Reference

Namespaces

- namespace gazebo

Functions

- gazebo.generate_launch_description ()

11.15 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/setup.py File Reference

Namespaces

- namespace setup

Variables

- str setup.package_name = 'bioscara_description'
- setup.name
- setup.version
- setup.packages
- setup.data_files
- setup.install_requires
- setup.zip_safe
- setup.maintainer
- setup.maintainer_email
- setup.description
- setup.license
- setup.tests_require
- setup.entry_points

11.16 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_copyright.py File Reference

Namespaces

- namespace test_copyright

Functions

- test_copyright.test_copyright ()

11.17 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_flake8.py File Reference

Namespaces

- namespace test_flake8

Functions

- test_flake8.test_flake8 ()

11.18 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_pep257.py File Reference

Namespaces

- namespace test_pep257

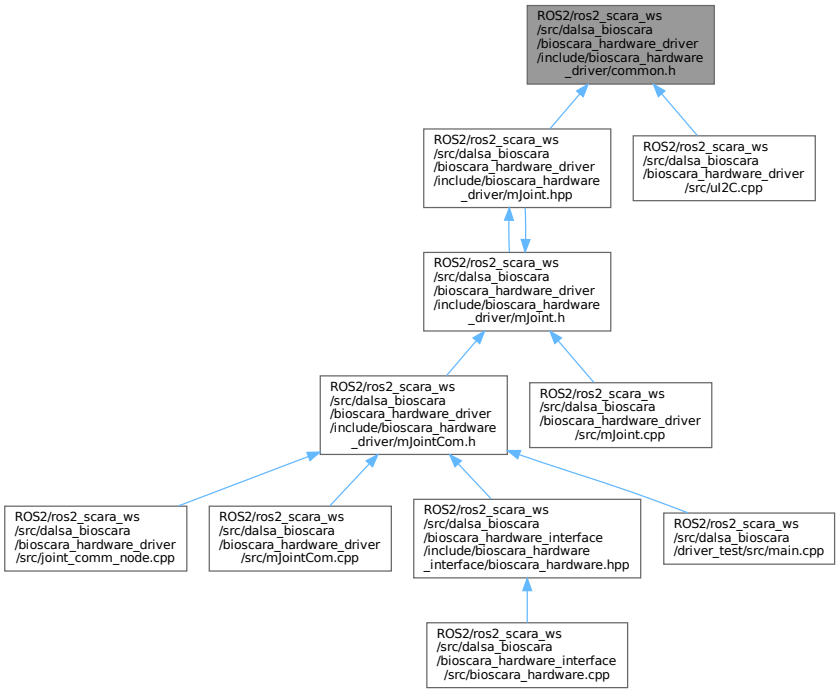
Functions

- test_pep257.test_pep257 ()

11.19 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/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.

11.19.1 Detailed Description

A file containing utility macros and functions.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

11.19.2 Macro Definition Documentation

11.19.2.1 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

buffer	pointer to a buffer to dump to the console
size	number of bytes to dump

11.20 common.h

Go to the documentation of this file.

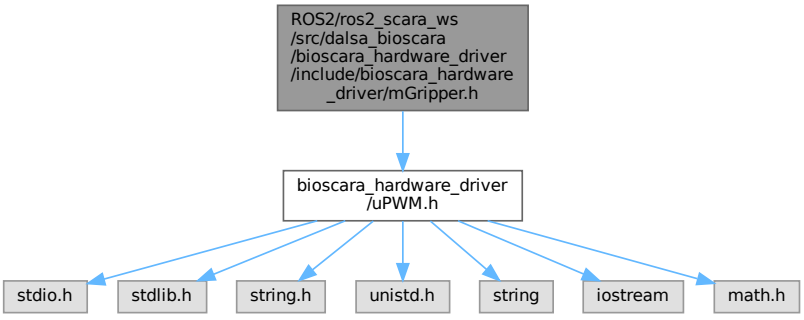
```
00001
00011 #ifndef COMMON_H
00012 #define COMMON_H
```

```
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
```

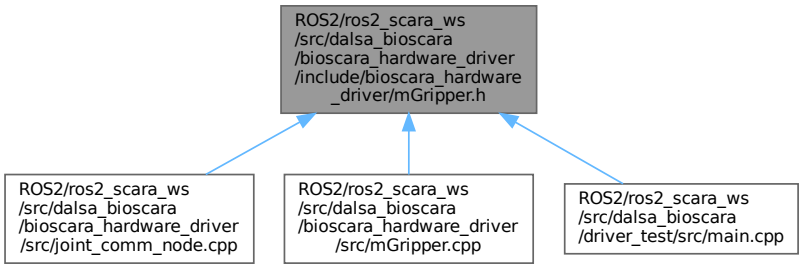
11.21 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/include/bioscara_hardware_driver/mGripper.h File Reference

File containing the Gripper class.

`#include "bioscara_hardware_driver/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.

11.21.1 Detailed Description

File containing the Gripper class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

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

11.22 mGripper.h

Go to the documentation of this file.

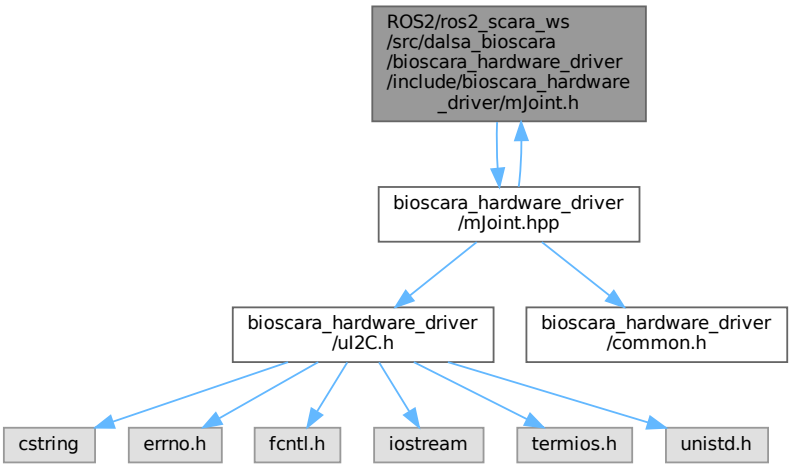
```
00001
00013 #ifndef MGRIPPER_H
00014 #define MGRIPPER_H
00015 #include "bioscara_hardware_driver/uPWM.h"
00016
00054 class Gripper
00055 {
00056 public:
00057     Gripper(void);
00058
00064     int init(void);
00065
00071     int deinit(void);
00072
00081     int enable(void);
00082
00090     int disable(void);
00091
00098     int setPosition(float width);
00099
00100 private:
00101     RPI_PWM pwm;
00102 };
00103 #endif // MGRIPPER_H
```

11.23 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/include/bioscara_hardware_driver/mJoint.h File Reference

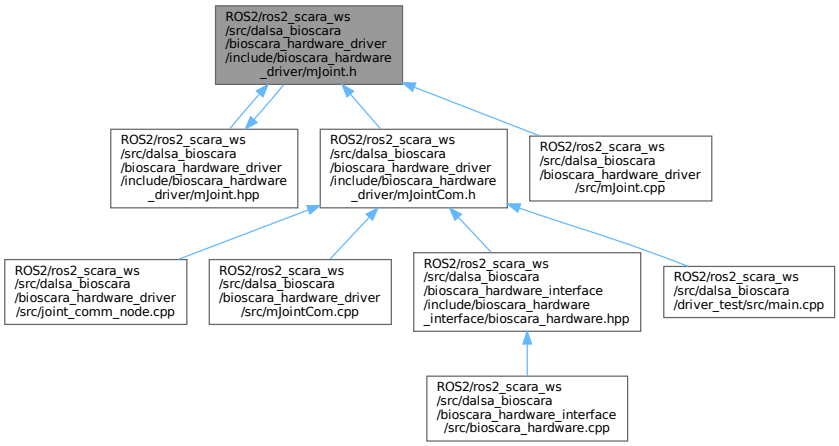
File including the Joint class.

```
#include "bioscara_hardware_driver/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 JOINT2ACTUATOR(in, reduction, offset) (reduction ∗ (in - offset))

  Macro for a simple transmission from joint units to actuator units.
- #define ACTUATOR2JOINT(in, reduction, offset) (in / reduction + offset)

  Macro for a simple transmission from actuator units to joint units.
- #define M_PI 3.14159265358979323846

  pi
- #define RAD2DEG(rad) (rad / M_PI ∗ 180)

  Macro to convert radians to degree.
- #define DEG2RAD(deg) (deg ∗ M_PI / 180)

  Macro to convert degree to radians.

11.23.1 Detailed Description

File including the Joint class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

11.23.2 Macro Definition Documentation

11.23.2.1 ACTUATOR2JOINT

```
#define ACTUATOR2JOINT(
              in,
              reduction,
              offset ) (in / reduction + offset)
```

Macro for a simple transmission from actuator units to joint units.

The translation is based on the ros2_control transmission interface, simple transmission. For position reduction and offset need to be used.
For velocity and acceleration only use reduction and NO offset
For effort/torque use 1/reduction and NO offset

11.23.2.2 DEG2RAD

```
#define DEG2RAD(
              deg ) (deg * M_PI / 180)
```

Macro to convert degree to radians.

11.23.2.3 JOINT2ACTUATOR

```
#define JOINT2ACTUATOR(
              in,
              reduction,
              offset ) (reduction * (in - offset))
```

Macro for a simple transmission from joint units to actuator units.

The translation is based on the ros2_control transmission interface, simple transmission. For position reduction and offset need to be used.
For velocity and acceleration only use reduction and NO offset
For effort/torque use 1/reduction and NO offset

11.23.2.4 M_PI

```
#define M_PI 3.14159265358979323846
```

pi

11.23.2.5 RAD2DEG

```
#define RAD2DEG(
              rad ) (rad / M_PI * 180)
```

Macro to convert radians to degree.

11.24 mJoint.h

Go to the documentation of this file.

```
00001
00012 #ifndef MJOINT_H
00013 #define MJOINT_H
00014
00023 #define JOINT2ACTUATOR(in, reduction, offset) (reduction * (in - offset))
00024
00033 #define ACTUATOR2JOINT(in, reduction, offset) (in / reduction + offset)
00034
00039 #define M_PI 3.14159265358979323846
00040
00045 #define RAD2DEG(rad) (rad / M_PI * 180)
00046
00051 #define DEG2RAD(deg) (deg * M_PI / 180)
00052
00057 class Joint
00058 {
00059 public:
00060   Joint(const int address, const std::string name, const float reduction, const float offset);
```

```
00061   // ~Joint();
00062
00063   int init(void);
00064   int deinit(void);
00065   int printInfo(void);
00066
00074   int getPosition(float &pos);
00075
00087   int setPosition(float pos);
00088
00100   int moveSteps(int32_t steps);
00101
00109   int getVelocity(float &vel);
00110
00122   int setVelocity(float vel);
00123
00135   int checkOrientation(float angle = 10.0);
00136
00145   int enable(u_int8_t driveCurrent, u_int8_t holdCurrent);
00146
00151   int disable(void);
00152
00165   int home(u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current);
00166
00175   int stop(bool mode);
00180   int disableCL(void);
00181
00188   int setDriveCurrent(u_int8_t current);
00189
00196   int setHoldCurrent(u_int8_t current);
00197
00203   int setBrakeMode(u_int8_t mode);
00204
00212   int setMaxAcceleration(float maxAccel);
00213
00221   int setMaxVelocity(float maxVel);
00222
00227   int enableStallguard(u_int8_t sensitivity);
00228
00237   bool isHomed(void);
00238
00248   bool isEnabled(void);
00249
00257   bool isStalled(void);
00258
00259   int checkCom(void);
00260
00265   u_int8_t getFlags(void);
00266
00267   std::string name;
00268
00269 protected:
00270 private:
00280   enum stp_reg_t
00281   {
00282     PING = 0x0f,
00283     SETUP = 0x10,
00284     SETRPM = 0x11,
00285     GETDRIVERRPM = 0x12,
00286     MOVESTEPS = 0x13,
00287     MOVEANGLE = 0x14,
00288     MOVETOANGLE = 0x15,
00289     GETMOTORSTATE = 0x16,
00290     RUNCOTINOUS = 0x17,
00291     ANGLEMOVED = 0x18,
00292     SETCURRENT = 0x19,
00293     SETHOLDCURRENT = 0x1A,
00294     SETMAXACCELERATION = 0x1B,
00295     SETMAXDECELERATION = 0x1C,
00296     SETMAXVELOCITY = 0x1D,
00297     ENABLESTALLGUARD = 0x1E,
00298     DISABLESTALLGUARD = 0x1F,
00299     CLEARSTALL = 0x20,
00300     ISSTALLED = 0x21,
00301     SETBRAKEMODE = 0x22,
00302     ENABLEPID = 0x23,
00303     DISABLEPID = 0x24,
00304     ENABLECLOSEDLOOP = 0x25,
00305     DISABLECLOSEDLOOP = 0x26,
00306     SETCONTROLTHRESHOLD = 0x27,
00307     MOVETOEND = 0x28,
00308     STOP = 0x29,
00309     GETPIDERROR = 0x2A,
00310     CHECKORIENTATION = 0x2B,
00311     GETENCODERRPM = 0x2C,
00312     HOME = 0x2D,
00313     ISHOMED = 0x2E,
```

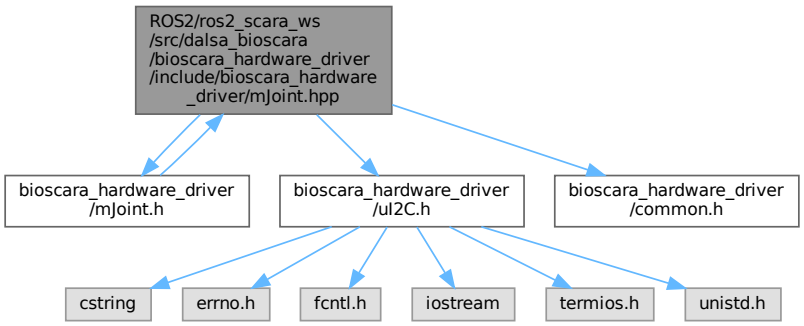
```
00314     ISSETUP = 0x2F
00315   };
00316
00317   template <typename T>
00318   int read(const stp_reg_t reg, T &data, u_int8_t &flags);
00319
00320   template <typename T>
00321   int write(const stp_reg_t reg, T data, u_int8_t &flags);
00322
00341   u_int8_t flags = 0x00;
00342
00343   int address;
00344   float reduction = 1;
00345   float offset = 0;
00346
00347   int handle = -1;
00348 };
00349
00350 #include "bioscara_hardware_driver/mJoint.hpp"
00351
00352 #endif
```

11.25 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/include/bioscara_hardware_driver/mJoint.hpp File Reference

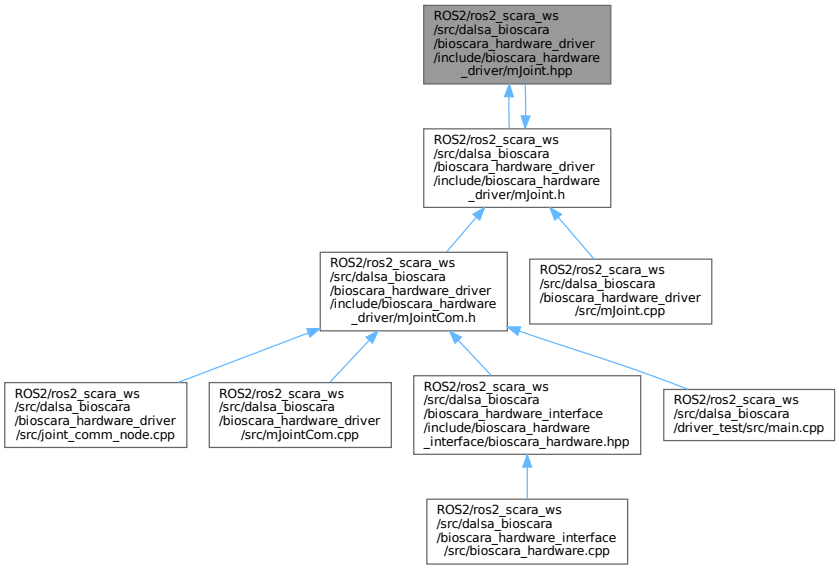
Templated functions for the Joint class.

```
#include "bioscara_hardware_driver/mJoint.h"
#include "bioscara_hardware_driver/uI2C.h"
#include "bioscara_hardware_driver/common.h"
```

Include dependency graph for mJoint.hpp:



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



11.25.1 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.

11.26 mJoint.hpp

Go to the documentation of this file.

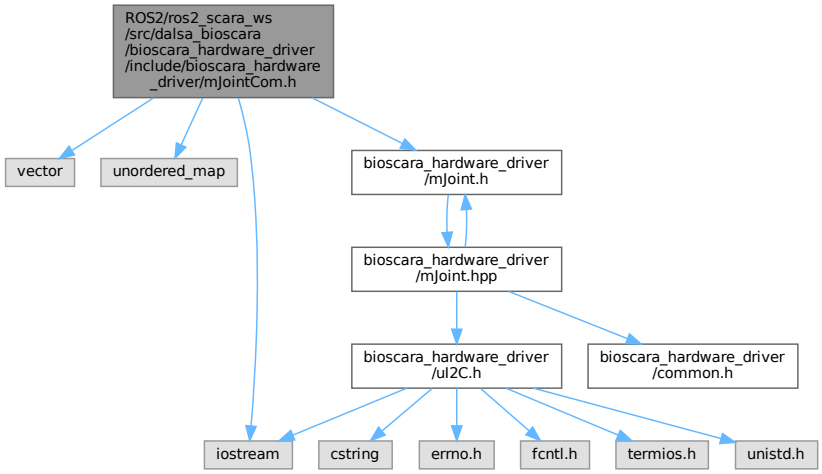
```
00001
00012 #include "bioscara_hardware_driver/mJoint.h"
00013 #include "bioscara_hardware_driver/uI2C.h"
00014 #include "bioscara_hardware_driver/common.h"
00015
00031 template <typename T>
00032 int Joint::read(const stp_reg_t reg, T &data, u_int8_t &flags)
00033 {
00034     size_t size = sizeof(T) + 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 }
```

11.27 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/include/bioscara_hardware_driver/mJointCom.h File Reference

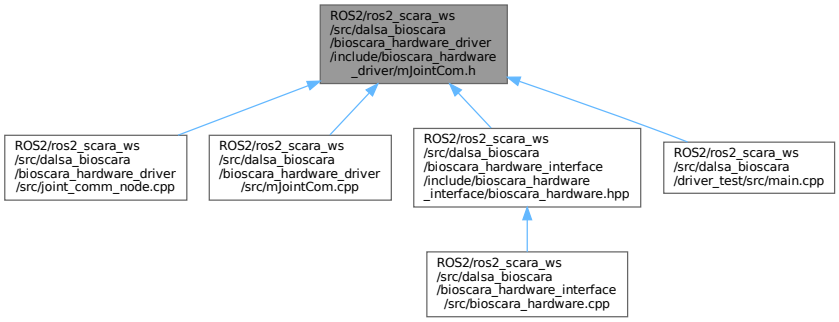
File containing the Joint_comms class.

```
#include <vector>
#include <unordered_map>
#include <iostream>
#include "bioscara_hardware_driver/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.

11.27.1 Detailed Description

File containing the Joint_comms class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

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

11.28 mJointCom.h

Go to the documentation of this file.

```
00001
00014 #ifndef MJOINTCOM_H
00015 #define MJOINTCOM_H
00016
00017 #include <vector>
00018 #include <unordered_map>
00019 #include <iostream>
00020 #include "bioscara_hardware_driver/mJoint.h"
00021
00031 class Joint_comms
00032 {
00033 public:
00034   Joint_comms(void);
00035   ~Joint_comms();
00036
00046   int init(void);
00047
00055   int deinit(void);
00056
00073   void addJoint(const std::string name, const int address, const float reduction, const float offset);
00074
00081   void removeJoint(const std::string name);
00082
00086   void removeJoints(void);
00087
00099   int enable(const std::string name, const u_int8_t driveCurrent, const u_int8_t holdCurrent);
00100
00107   int disables(void);
00108
00123   int home(const std::string name, const u_int8_t direction, const u_int8_t rpm, const u_int8_t
      sensitivity, const u_int8_t current);
00124
00135   int getPosition(const std::string name, float &angle);
00136
00146   int setPosition(const std::string name, const float angle);
00147
00157   int getVelocity(const std::string name, float &degps);
00158
00168   int setVelocity(const std::string name, float degps);
00169
00182   int checkOrientations(float angle = 10.0);
00183
00196   int checkOrientation(const std::string name, float angle = 10.0);
00197
00205   int stops(bool mode);
00206
00207   // /**
00208   //  * @brief Disables the Closed-Loop PID Controllers
00209   //  * @return error code.
00210   //  */
00211   // int disableCLs(void);
00212
00213   // /**
00214   //  * @brief Set the drive Currents.
00215   //  * @warning This function is unreliable and not well tested. Use enables() instead!
00216   //  *
```

```
00217   //  *
00218   //  * @param current 0% - 100% of driver current
00219   //  * @return error code.
00220   //  */
00221   // int setDriveCurrents(std::vector<u_int8_t> current);
00222
00223   // /**
00224   //  * @brief Overload to set all drive currents to the same value
00225   //  * @warning This function is unreliable and not well tested. Use enables() instead!
00226
00227   //  * @param current 0% - 100% of driver current
00228   //  * @return error code.
00229   //  */
00230   // int setDriveCurrents(u_int8_t current);
00231
00232   // /**
00233   //  * @brief Set the Hold Currents
00234   //  * @warning This function is unreliable and not well tested. Use enables() instead!
00235
00236   //  * @param current 0% - 100% of driver current
00237   //  * @return error code.
00238   //  */
00239   // int setHoldCurrents(std::vector<u_int8_t> current);
00240
00241   // /**
00242   //  * @brief Overload to set all hold currents to the same value
00243   //  * @warning This function is unreliable and not well tested. Use enables() instead!
00244   //  *
00245   //  * @param current 0% - 100% of driver current
00246   //  * @return error code.
00247   //  */
00248   // int setHoldCurrents(u_int8_t current);
00249
00250   // /**
00251   //  * @brief Set Brake Modes.
00252   //  *
00253   //  * Applies the same brake modes to all joints. usefull to disengage all motors.
00254   //  * @param mode Freewheel: 0, Coolbrake: 1, Hardbrake: 2
00255   //  * @return error code.
00256   //  */
00257   // int setBrakeModes(u_int8_t mode);
00258
00259   // /**
00260   //  * @brief Enable encoder stall detection.
00261   //  *
00262   //  * If the PID error exceeds the set threshold a stall is triggered and the motor disabled.
00263   //  * A detected stall can be reset by homeing.
00264   //  * @param thresholds Vector of thresholds. 0 - 255 where lower is more sensitive.
00265   //  */
00266   // int enableStallguards(std::vector<u_int8_t> thresholds);
00267
00276   int enableStallguard(const std::string name, const u_int8_t threshold);
00277
00285   int setMaxAcceleration(const std::string name, float maxAccel);
00286
00294   int setMaxVelocity(const std::string name, float maxVel);
00295
00305   std::unordered_map<std::string, Joint> joints;
00306
00307 protected:
00308 private:
00309 };
00310
00311 #endif
```

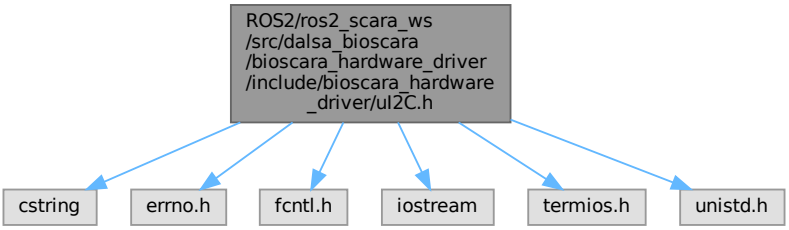
11.29 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/ul2C.h File Reference

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

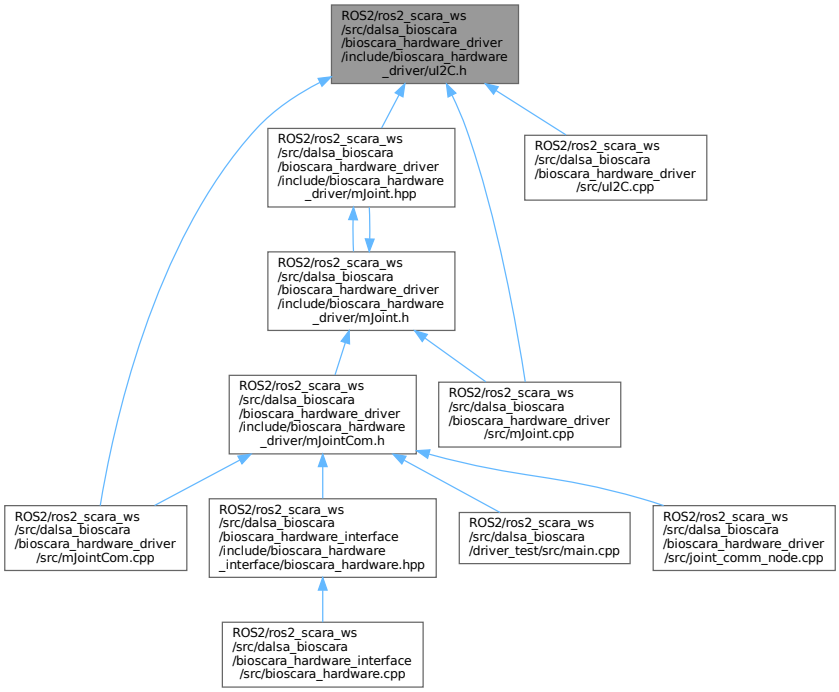
```
#include <cstring>
#include <errno.h>
#include <fcntl.h>
#include <iostream>
#include <termios.h>
```

```
#include <unistd.h>
```

Include dependency graph for uI2C.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

11.29.1 Detailed Description

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

Author

Sebastian Storz

Version

0.1

Date

2025-05-28

Copyright

Copyright (c) 2025

lgpio needs to be installed and linked! Installation:

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

bash

11.29.2 Macro Definition Documentation

11.29.2.1 ACK

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

11.29.2.2 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

11.29.2.3 NACK

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

11.29.2.4 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

11.29.3 Function Documentation

11.29.3.1 closeI2CDevHandle()

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

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

11.29.3.2 openI2CDevHandle()

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

Initiates an I2C device on the bus.

Parameters

dev_addr	7-bit device adress [0 - 0x7F]

Returns

the device handle, negative on error.

11.29.3.3 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

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

Returns

number of bytes read, negative on error.

11.29.3.4 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

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

Returns

0 on OK, negative on error.

11.30 uI2C.h

Go to the documentation of this file.

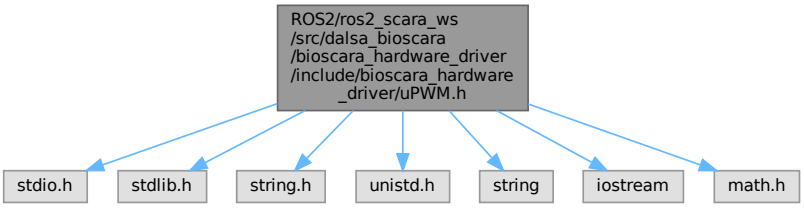
```
00001
00028 #ifndef USERIAL_H
00029 #define USERIAL_H
00030 #include <cstring>
00031 #include <errno.h>
00032 #include <fcntl.h>
00033 #include <iostream>
00034 #include <termios.h>
00035 #include <unistd.h>
00036
00037 #define ACK 'O'
00038 #define NACK 'N'
00039
00043 #define RFLAGS_SIZE 1
00044
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
```

11.31 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/include/bioscara_hardware_driver/uPWM.h File Reference

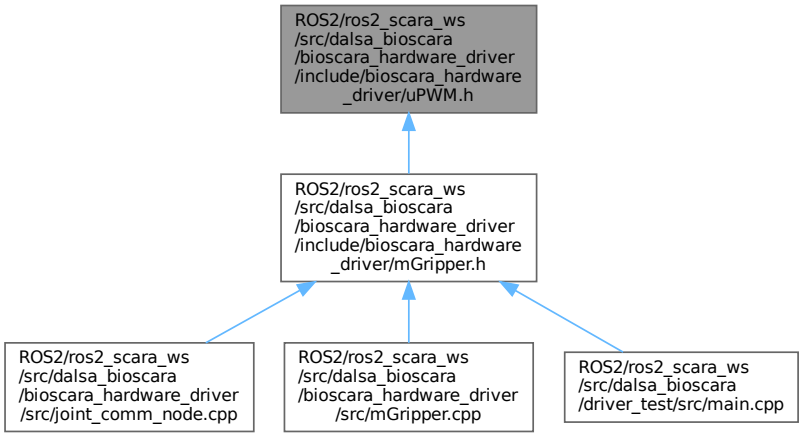
Includes source code for Hardware PWM generation on Raspberry Pi 4.

```
#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.

11.31.1 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.

lgpio, 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

11.32 uPWM.h

Go to the documentation of this file.

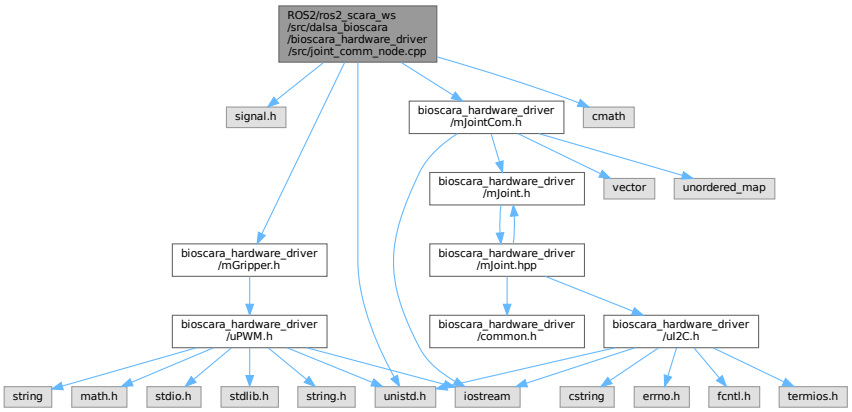
```
00001
00019 #ifndef __RPIPWM
00020 #define __RPIPWM
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <unistd.h>
00026 #include <string>
00027 #include <iostream>
00028 #include <math.h>
00029
00033 class RPI_PWM
00034 {
00035 public:
00044     int 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
```

11.33 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/joint_comm_node.cpp File Reference

```
#include <signal.h>
#include <unistd.h>
#include "bioscara_hardware_driver/mJointCom.h"
#include "bioscara_hardware_driver/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

11.33.1 Function Documentation

11.33.1.1 INT_handler()

```
void INT_handler (
            int s )
```

11.33.1.2 main()

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

11.33.2 Variable Documentation

11.33.2.1 _Gripper

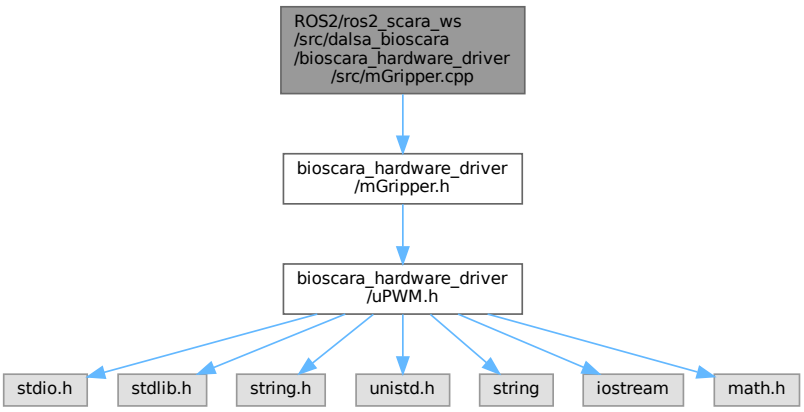
```
Gripper _Gripper
```

11.33.2.2 _Joints

```
Joint_comms _Joints
```

11.34 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/src/mGripper.cpp File Reference

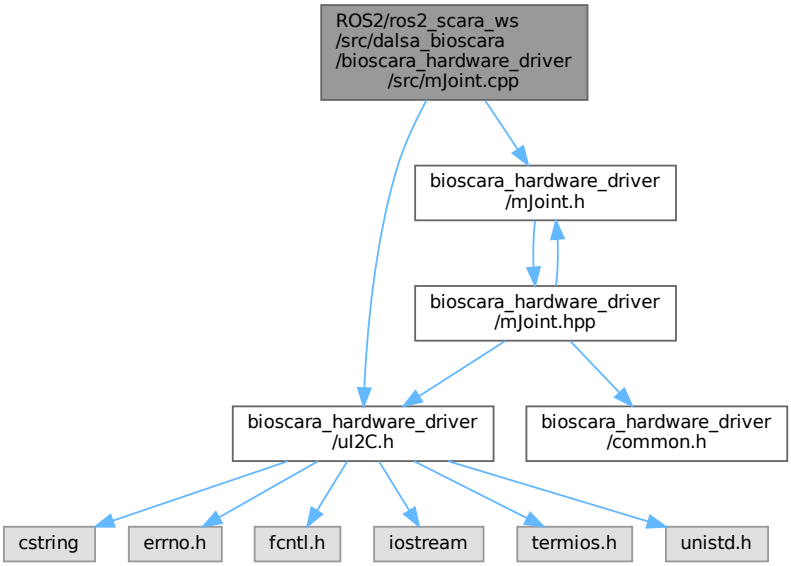
```
#include "bioscara_hardware_driver/mGripper.h"
```
Include dependency graph for mGripper.cpp:



11.35 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/src/mJoint.cpp File Reference

```
#include "bioscara_hardware_driver/uI2C.h"
#include "bioscara_hardware_driver/mJoint.h"
```

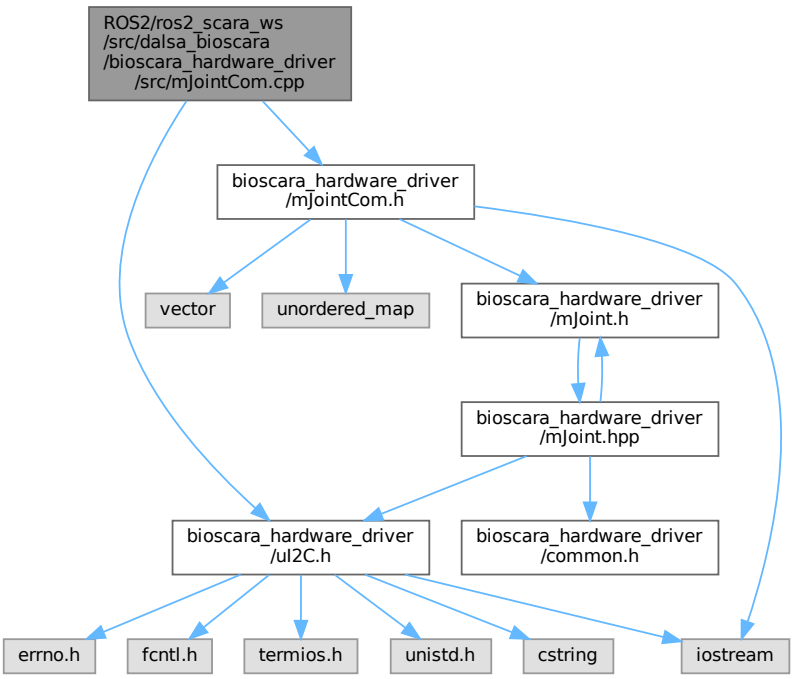
Include dependency graph for mJoint.cpp:



11.36 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ driver/src/mJointCom.cpp File Reference

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

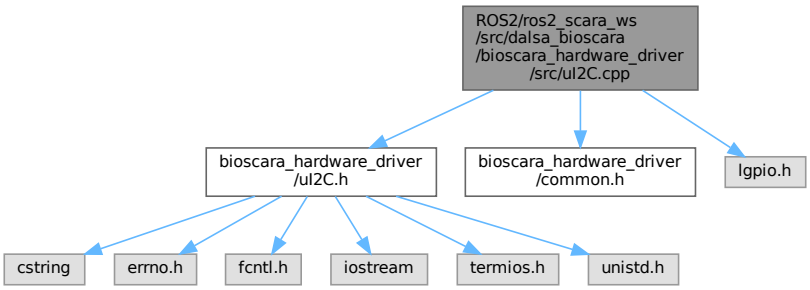
Include dependency graph for mJointCom.cpp:



11.37 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/uI2C.cpp File Reference

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

Include dependency graph for uI2C.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

11.37.1 Function Documentation

11.37.1.1 closeI2CDevHandle()

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

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

11.37.1.2 openI2CDevHandle()

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

Initiates an I2C device on the bus.

Parameters

dev_addr	7-bit device adress [0 - 0x7F]

Returns

the device handle, negative on error.

11.37.1.3 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

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

Returns

number of bytes read, negative on error.

11.37.1.4 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

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

Returns

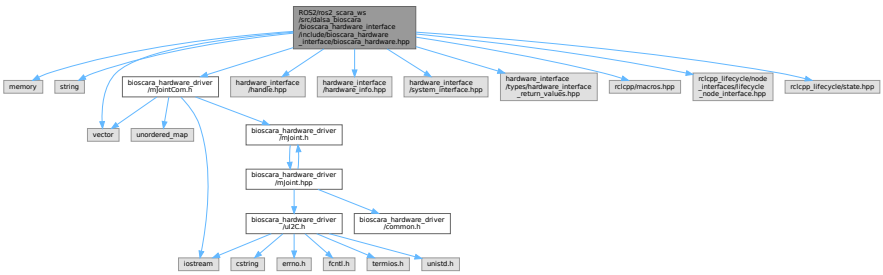
0 on OK, negative on error.

11.38 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ interface/include/bioscara_hardware_interface/bioscara_↩ hardware.hpp File Reference

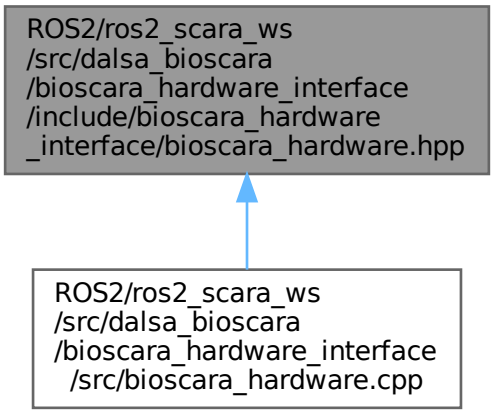
```
#include <memory>
#include <string>
#include <vector>
#include "bioscara_hardware_driver/mJointCom.h"
#include "hardware_interface/handle.hpp"
#include "hardware_interface/hardware_info.hpp"
#include "hardware_interface/system_interface.hpp"
#include "hardware_interface/types/hardware_interface_return_values.hpp"
#include "rclcpp/macros.hpp"
#include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
```

```
#include "rclcpp_lifecycle/state.hpp"
```

Include dependency graph for bioscara_hardware.hpp:



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



Classes

- class bioscara_hardware_interface::BioscaraHardwareInterface

Namespaces

- namespace bioscara_hardware_interface

11.39 bioscara_hardware.hpp

Go to the documentation of this file.

```
00001 // Copyright 2023 ros2_control Development Team
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
```

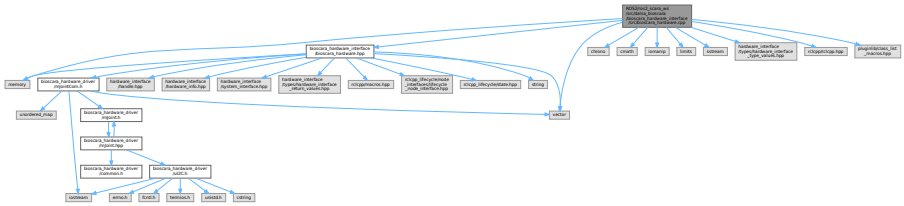
```
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00015 #ifndef BIOSCARA_HARDWARE_INTERFACE_HPP_
00016 #define BIOSCARA_HARDWARE_INTERFACE_HPP_
00017
00018 #include <memory>
00019 #include <string>
00020 #include <vector>
00021
00022 #include "bioscara_hardware_driver/mJointCom.h"
00023
00024 #include "hardware_interface/handle.hpp"
00025 #include "hardware_interface/hardware_info.hpp"
00026 #include "hardware_interface/system_interface.hpp"
00027 #include "hardware_interface/types/hardware_interface_return_values.hpp"
00028 #include "rclcpp/macros.hpp"
00029 #include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
00030 #include "rclcpp_lifecycle/state.hpp"
00031
00032 namespace bioscara_hardware_interface
00033 {
00034   class BioscaraHardwareInterface : public hardware_interface::SystemInterface
00035   {
00036   public:
00037     RCLCPP_SHARED_PTR_DEFINITIONS(BioscaraHardwareInterface)
00038
00039     hardware_interface::CallbackReturn on_init(
00040         const hardware_interface::HardwareInfo &info) override;
00041
00042     hardware_interface::CallbackReturn on_shutdown(
00043         const rclcpp_lifecycle::State &previous_state) override;
00044
00045     hardware_interface::CallbackReturn on_configure(
00046         const rclcpp_lifecycle::State &previous_state) override;
00047
00048     hardware_interface::CallbackReturn on_cleanup(
00049         const rclcpp_lifecycle::State &previous_state) override;
00050
00051     hardware_interface::CallbackReturn on_activate(
00052         const rclcpp_lifecycle::State &previous_state) override;
00053
00054     hardware_interface::CallbackReturn on_deactivate(
00055         const rclcpp_lifecycle::State &previous_state) override;
00056
00057     hardware_interface::return_type read(
00058         const rclcpp::Time &time, const rclcpp::Duration &period) override;
00059
00060     hardware_interface::return_type write(
00061         const rclcpp::Time &time, const rclcpp::Duration &period) override;
00062
00063   private:
00064     /*
00065      * Communication object containing all joints of the robot
00066      */
00067     Joint_comms Joints_;
00068
00069     double deg2rad(float )
00070   };
00071
00072 } // namespace bioscara_hardware_interface
00073
00074 #endif // BIOSCARA_HARDWARE_INTERFACE_HPP_
```

11.40 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_↩ interface/src/bioscara_hardware.cpp File Reference

```
#include "bioscara_hardware_interface/bioscara_hardware.hpp"
#include <chrono>
#include <cmath>
#include <iomanip>
#include <limits>
```

```
#include <memory>
#include <sstream>
#include <vector>
#include "hardware_interface/types/hardware_interface_type_values.hpp"
#include "rclcpp/rclcpp.hpp"
#include "pluginlib/class_list_macros.hpp"
```

Include dependency graph for bioscara_hardware.cpp:



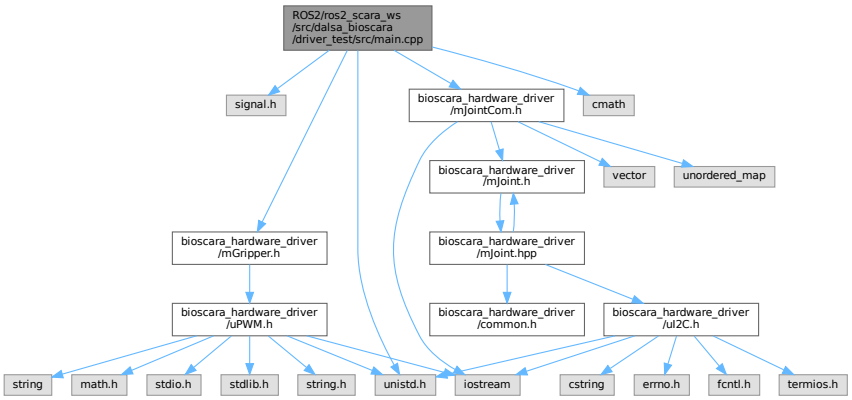
Namespaces

- namespace bioscara_hardware_interface

11.41 ROS2/ros2_scara_ws/src/dalsa_bioscara/driver_test/src/main.cpp File Reference

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

Include dependency graph for main.cpp:



Functions

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

Variables

- Joint_comms _Joints
- Gripper _Gripper

11.41.1 Function Documentation

11.41.1.1 INT_handler()

```
void INT_handler (
            int s )
```

11.41.1.2 main()

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

11.41.2 Variable Documentation

11.41.2.1 _Gripper

```
Gripper _Gripper
```

11.41.2.2 _Joints

```
Joint_comms _Joints
```

Index