

Bioscara

Generated by Doxygen 1.9.8



<b>1 Documentation</b>	<b>1</b>
1.1 Usage	1
<b>2 README</b>	<b>3</b>
<b>3 README</b>	<b>5</b>
<b>4 README</b>	<b>7</b>
<b>5 Todo List</b>	<b>9</b>
<b>6 Namespace Index</b>	<b>11</b>
6.1 Namespace List	11
<b>7 Hierarchical Index</b>	<b>13</b>
7.1 Class Hierarchy	13
<b>8 Class Index</b>	<b>15</b>
8.1 Class List	15
<b>9 File Index</b>	<b>17</b>
9.1 File List	17
<b>10 Namespace Documentation</b>	<b>19</b>
10.1 bioscara Namespace Reference	19
10.1.1 Function Documentation	19
10.1.1.1 generate_launch_description()	19
10.2 bioscara_hardware_interface Namespace Reference	19
10.2.1 Variable Documentation	19
10.2.1.1 HW_IF_HOME	19
10.3 display Namespace Reference	20
10.3.1 Function Documentation	20
10.3.1.1 generate_launch_description()	20
10.4 gazebo Namespace Reference	20
10.4.1 Function Documentation	20
10.4.1.1 generate_launch_description()	20
10.5 setup Namespace Reference	20
10.5.1 Variable Documentation	21
10.5.1.1 data_files	21
10.5.1.2 description	21
10.5.1.3 entry_points	21
10.5.1.4 install_requires	21
10.5.1.5 license	21
10.5.1.6 maintainer	21
10.5.1.7 maintainer_email	21
10.5.1.8 name	21

10.5.1.9 package_name	21
10.5.1.10 packages	21
10.5.1.11 tests_require	22
10.5.1.12 version	22
10.5.1.13 zip_safe	22
10.6 test_copyright Namespace Reference	22
10.6.1 Function Documentation	22
10.6.1.1 test_copyright()	22
10.7 test_flake8 Namespace Reference	22
10.7.1 Function Documentation	22
10.7.1.1 test_flake8()	22
10.8 test_joint_trajectory_controller Namespace Reference	22
10.8.1 Function Documentation	23
10.8.1.1 generate_launch_description()	23
10.9 test_pep257 Namespace Reference	23
10.9.1 Function Documentation	23
10.9.1.1 test_pep257()	23
<b>11 Class Documentation</b>	<b>25</b>
11.1 bioscara_hardware_interface::BioscaraHardwareInterface Class Reference	25
11.1.1 Detailed Description	27
11.1.2 Member Function Documentation	27
11.1.2.1 on_activate()	27
11.1.2.2 on_cleanup()	28
11.1.2.3 on_configure()	28
11.1.2.4 on_deactivate()	29
11.1.2.5 on_error()	29
11.1.2.6 on_init()	30
11.1.2.7 on_shutdown()	30
11.1.2.8 prepare_command_mode_switch()	31
11.1.2.9 read()	32
11.1.2.10 write()	32
11.1.3 Member Data Documentation	33
11.1.3.1 _joint_cfg	33
11.1.3.2 _joint_command_modes	33
11.1.3.3 _joints	34
11.2 Gripper Class Reference	34
11.2.1 Detailed Description	35
11.2.2 Constructor & Destructor Documentation	35
11.2.2.1 Gripper()	35
11.2.3 Member Function Documentation	35
11.2.3.1 deinit()	35

11.2.3.2 disable()	36
11.2.3.3 enable()	36
11.2.3.4 init()	36
11.2.3.5 setPosition()	36
11.2.4 Member Data Documentation	37
11.2.4.1 pwm	37
11.3 Joint Class Reference	37
11.3.1 Detailed Description	39
11.3.2 Member Enumeration Documentation	40
11.3.2.1 stp_reg_t	40
11.3.3 Constructor & Destructor Documentation	41
11.3.3.1 Joint()	41
11.3.3.2 ~Joint()	42
11.3.4 Member Function Documentation	42
11.3.4.1 _home()	42
11.3.4.2 checkCom()	43
11.3.4.3 checkOrientation()	43
11.3.4.4 deinit()	43
11.3.4.5 disable()	44
11.3.4.6 disableCL()	44
11.3.4.7 enable()	44
11.3.4.8 enableStallguard()	44
11.3.4.9 getCurrentBCmd()	45
11.3.4.10 getFlags()	45
11.3.4.11 getHomingOffset()	45
11.3.4.12 getPosition()	45
11.3.4.13 getVelocity()	46
11.3.4.14 home()	46
11.3.4.15 init()	46
11.3.4.16 isBusy()	47
11.3.4.17 isEnabled()	47
11.3.4.18 isHomed()	47
11.3.4.19 isStalled()	47
11.3.4.20 moveSteps()	48
11.3.4.21 postHoming()	49
11.3.4.22 printInfo()	49
11.3.4.23 read()	49
11.3.4.24 setBrakeMode()	50
11.3.4.25 setDriveCurrent()	50
11.3.4.26 setHoldCurrent()	50
11.3.4.27 setHomingOffset()	51
11.3.4.28 setMaxAcceleration()	51

11.3.4.29 setMaxVelocity()	51
11.3.4.30 setPosition()	52
11.3.4.31 setVelocity()	52
11.3.4.32 startHoming()	52
11.3.4.33 stop()	53
11.3.4.34 wait_while_busy()	53
11.3.4.35 write()	53
11.3.5 Member Data Documentation	54
11.3.5.1 address	54
11.3.5.2 current_b_cmd	54
11.3.5.3 flags	54
11.3.5.4 handle	55
11.3.5.5 max	55
11.3.5.6 min	55
11.3.5.7 name	55
11.3.5.8 offset	55
11.3.5.9 reduction	55
11.4 bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t Struct Reference	56
11.4.1 Detailed Description	56
11.4.2 Member Data Documentation	56
11.4.2.1 drive_current	56
11.4.2.2 hold_current	57
11.4.2.3 homing	57
11.4.2.4 i2c_address	57
11.4.2.5 max	57
11.4.2.6 max_acceleration	57
11.4.2.7 max_velocity	57
11.4.2.8 min	57
11.4.2.9 reduction	57
11.4.2.10 stall_threshold	57
11.5 bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t Struct Reference	58
11.5.1 Detailed Description	58
11.5.2 Member Data Documentation	58
11.5.2.1 acceleration	58
11.5.2.2 current	58
11.5.2.3 speed	58
11.5.2.4 threshold	58
11.6 RPI_PWM Class Reference	59
11.6.1 Detailed Description	59
11.6.2 Constructor & Destructor Documentation	59
11.6.2.1 ~RPI_PWM()	59
11.6.3 Member Function Documentation	59

11.6.3.1 disable()	59
11.6.3.2 enable()	59
11.6.3.3 setDutyCycle()	59
11.6.3.4 setDutyCycleNS()	60
11.6.3.5 setPeriod()	60
11.6.3.6 start()	60
11.6.3.7 stop()	60
11.6.3.8 writeSYS()	60
11.6.4 Member Data Documentation	61
11.6.4.1 chippath	61
11.6.4.2 per	61
11.6.4.3 pwmpath	61
<b>12 File Documentation</b>	<b>63</b>
12.1 Arduino/joint/configuration.h File Reference	63
12.1.1 Detailed Description	64
12.1.2 Macro Definition Documentation	64
12.1.2.1 ADR	64
12.1.2.2 MAXACCEL	64
12.1.2.3 MAXVEL	64
12.2 configuration.h	65
12.3 Arduino/joint/joint.h File Reference	65
12.3.1 Detailed Description	67
12.3.2 Macro Definition Documentation	67
12.3.2.1 ACK	67
12.3.2.2 DUMP_BUFFER	67
12.3.2.3 MAX_BUFFER	68
12.3.2.4 NACK	68
12.3.2.5 RFLAGS_SIZE	68
12.3.3 Enumeration Type Documentation	68
12.3.3.1 stp_reg_t	68
12.3.4 Function Documentation	69
12.3.4.1 readValue()	69
12.3.4.2 writeValue()	69
12.4 joint.h	70
12.5 Arduino/joint/joint.ino File Reference	71
12.5.1 Detailed Description	72
12.5.2 Macro Definition Documentation	72
12.5.2.1 J4	72
12.5.3 Function Documentation	72
12.5.3.1 blocking_handler()	72
12.5.3.2 loop()	73

12.5.3.3 non_blocking_handler()	73
12.5.3.4 receiveEvent()	73
12.5.3.5 requestEvent()	75
12.5.3.6 setup()	75
12.5.4 Variable Documentation	75
12.5.4.1 reg	75
12.5.4.2 rx_buf	75
12.5.4.3 rx_data_ready	75
12.5.4.4 rx_length	75
12.5.4.5 stepper	76
12.5.4.6 tx_buf	76
12.5.4.7 tx_data_ready	76
12.5.4.8 tx_length	76
12.6 docs/DOCS_README.md File Reference	76
12.7 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/launch/bioscara.launch.py File Reference	76
12.8 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/launch/test_joint_trajectory_controller.launch.py File Reference	76
12.9 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/README.md File Reference	77
12.10 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/README.md File Reference	77
12.11 ROS2/ros2_scara_ws/src/dalsa_bioscara/README.md File Reference	77
12.12 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/bioscara_description/___init___py File Reference	77
12.13 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/launch/display.launch.py File Reference	77
12.14 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/launch/gazebo.launch.py File Reference	77
12.15 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/setup.py File Reference	77
12.16 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_copyright.py File Reference	78
12.17 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_flake8.py File Reference	78
12.18 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/test_pep257.py File Reference	78
12.19 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/common.h File Reference	79
12.19.1 Detailed Description	79
12.19.2 Macro Definition Documentation	80
12.19.2.1 DUMP_BUFFER	80
12.20 common.h	80
12.21 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mGripper.h File Reference	81
12.21.1 Detailed Description	82
12.22 mGripper.h	82
12.23 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_driver/mJoint.h File Reference	83
12.23.1 Detailed Description	84
12.23.2 Macro Definition Documentation	84



12.23.2.1 ACTUATOR2JOINT . . . . .	84
12.23.2.2 DEG2RAD . . . . .	85
12.23.2.3 JOINT2ACTUATOR . . . . .	85
12.23.2.4 M_PI . . . . .	85
12.23.2.5 RAD2DEG . . . . .	85
12.24 mJoint.h . . . . .	85
12.25 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware↔ _driver/mJoint.hpp File Reference . . . . .	87
12.25.1 Detailed Description . . . . .	88
12.26 mJoint.hpp . . . . .	89
12.27 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware↔ _driver/ul2C.h File Reference . . . . .	89
12.27.1 Detailed Description . . . . .	91
12.27.2 Macro Definition Documentation . . . . .	91
12.27.2.1 ACK . . . . .	91
12.27.2.2 MAX_BUFFER . . . . .	92
12.27.2.3 NACK . . . . .	92
12.27.2.4 RFLAGS_SIZE . . . . .	92
12.27.3 Function Documentation . . . . .	92
12.27.3.1 closeI2CDevHandle() . . . . .	92
12.27.3.2 openI2CDevHandle() . . . . .	92
12.27.3.3 readFromI2CDev() . . . . .	93
12.27.3.4 writeToI2CDev() . . . . .	93
12.28 ul2C.h . . . . .	94
12.29 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware↔ _driver/uPWM.h File Reference . . . . .	94
12.29.1 Detailed Description . . . . .	95
12.30 uPWM.h . . . . .	96
12.31 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/joint_comm_node.cpp File Reference . . . . .	97
12.31.1 Function Documentation . . . . .	98
12.31.1.1 INT_handler() . . . . .	98
12.31.1.2 main() . . . . .	98
12.31.2 Variable Documentation . . . . .	99
12.31.2.1 J1 . . . . .	99
12.31.2.2 J2 . . . . .	99
12.31.2.3 J3 . . . . .	99
12.31.2.4 J4 . . . . .	99
12.32 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mGripper.cpp File Reference	99
12.33 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mJoint.cpp File Reference	100
12.34 ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/ul2C.cpp File Reference .	101
12.34.1 Function Documentation . . . . .	101
12.34.1.1 closeI2CDevHandle() . . . . .	101

12.34.1.2	<a href="#">openI2CDevHandle()</a>	102
12.34.1.3	<a href="#">readFromI2CDev()</a>	102
12.34.1.4	<a href="#">writeToI2CDev()</a>	102
12.35	<a href="#">ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/include/bioscara_↔ hardware_interface/bioscara_hardware.hpp File Reference</a>	103
12.36	<a href="#">bioscara_hardware.hpp</a>	104
12.37	<a href="#">ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/src/bioscara_hardware.cpp File Reference</a>	106
<b>Index</b>		<b>107</b>

# 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

All configuration parameters are stored in the `config/bioscara_parameters` file.





# Chapter 4

## 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 5

# Todo List

Member `bioscara_hardware_interface::BioscaraHardwareInterface::on_init` (`const hardware_interface::↵  
HardwareComponentInterfaceParams &params`) override

threshold and current are uint8\_t, if a number larger outside  $0 < n < 255$  is passed as a parameters it will overflow.

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



## Chapter 6

# Namespace Index

### 6.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">bioscara</a>	19
<a href="#">bioscara_hardware_interface</a>	19
<a href="#">display</a>	20
<a href="#">gazebo</a>	20
<a href="#">setup</a>	20
<a href="#">test_copyright</a>	22
<a href="#">test_flake8</a>	22
<a href="#">test_joint_trajectory_controller</a>	22
<a href="#">test_pep257</a>	23



## Chapter 7

# Hierarchical Index

### 7.1 Class Hierarchy

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

Gripper . . . . .	34
Joint . . . . .	37
bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t . . . . .	56
bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t . . . . .	58
RPI_PWM . . . . .	59
hardware_interface::SystemInterface	
bioscara_hardware_interface::BioscaraHardwareInterface . . . . .	25





## Chapter 8

# Class Index

### 8.1 Class List

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

<a href="#">bioscara_hardware_interface::BioscaraHardwareInterface</a>	
The bioscara hardware interface class . . . . .	25
<a href="#">Gripper</a>	
<a href="#">Gripper</a> object to interact with the robot gripper . . . . .	34
<a href="#">Joint</a>	
Representing a single joint on the I2C bus . . . . .	37
<a href="#">bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t</a>	
Configuration structure holding the passed paramters from the ros2_control urdf . . . . .	56
<a href="#">bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t</a>	
Configuration structure holding the passed homing paramters from the ros2_control urdf . . . . .	58
<a href="#">RPI_PWM</a>	
PWM class for the Raspberry PI 4 and 5 . . . . .	59



# Chapter 9

## File Index

### 9.1 File List

Here is a list of all files with brief descriptions:

Arduino/joint/ <a href="#">configuration.h</a>	
Configuration definitions for <a href="#">Joint 1</a> to <a href="#">Joint 4</a>	63
Arduino/joint/ <a href="#">joint.h</a>	
<a href="#">Joint</a> firmware header	65
Arduino/joint/ <a href="#">joint.ino</a>	
<a href="#">Joint</a> firmware	71
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/launch/ <a href="#">bioscara.launch.py</a>	76
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_bringup/launch/ <a href="#">test_joint_trajectory_controller.launch.py</a>	76
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/ <a href="#">setup.py</a>	77
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/bioscara_description/ <a href="#">__init__.py</a>	77
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/launch/ <a href="#">display.launch.py</a>	77
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/launch/ <a href="#">gazebo.launch.py</a>	77
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/ <a href="#">test_copyright.py</a>	78
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/ <a href="#">test_flake8.py</a>	78
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_description/test/ <a href="#">test_pep257.py</a>	78
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">common.h</a>	
A file containing utility macros and functions	79
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">mGripper.h</a>	
File containing the <a href="#">Gripper</a> class	81
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">mJoint.h</a>	
File including the <a href="#">Joint</a> class	83
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">mJoint.hpp</a>	
Templated functions for the <a href="#">Joint</a> class	87
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">ul2C.h</a>	
Low level utility for I2C communication on Raspberry Pi using I2C library	89
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/include/bioscara_hardware_↵ driver/ <a href="#">uPWM.h</a>	
Includes source code for Hardware PWM generation on Raspberry Pi 4	94
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/ <a href="#">joint_comm_node.cpp</a>	97

ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mGripper.cpp . . . . .	99
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/mJoint.cpp . . . . .	100
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_driver/src/ul2C.cpp . . . . .	101
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/include/bioscara_hardware_↵ interface/bioscara_hardware.hpp . . . . .	103
ROS2/ros2_scara_ws/src/dalsa_bioscara/bioscara_hardware_interface/src/bioscara_hardware.cpp . . .	106

## Chapter 10

# Namespace Documentation

### 10.1 bioscara Namespace Reference

#### Functions

- [generate\\_launch\\_description\(\)](#)

#### 10.1.1 Function Documentation

##### 10.1.1.1 generate\_launch\_description()

```
bioscara.generate_launch_description ( )
```

### 10.2 bioscara\_hardware\_interface Namespace Reference

#### Classes

- class [BioscaraHardwareInterface](#)  
*The bioscara hardware interface class.*

#### Variables

- constexpr char [HW\\_IF\\_HOME](#) [] = "home"

#### 10.2.1 Variable Documentation

##### 10.2.1.1 HW\_IF\_HOME

```
constexpr char bioscara_hardware_interface::HW_IF_HOME[] = "home" [constexpr]
```

## 10.3 display Namespace Reference

### Functions

- [generate\\_launch\\_description](#) ()

### 10.3.1 Function Documentation

#### 10.3.1.1 generate\_launch\_description()

`display.generate_launch_description ( )`

## 10.4 gazebo Namespace Reference

### Functions

- [generate\\_launch\\_description](#) ()

### 10.4.1 Function Documentation

#### 10.4.1.1 generate\_launch\_description()

`gazebo.generate_launch_description ( )`

## 10.5 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](#)

## 10.5.1 Variable Documentation

### 10.5.1.1 data\_files

`setup.data_files`

### 10.5.1.2 description

`setup.description`

### 10.5.1.3 entry\_points

`setup.entry_points`

### 10.5.1.4 install\_requires

`setup.install_requires`

### 10.5.1.5 license

`setup.license`

### 10.5.1.6 maintainer

`setup.maintainer`

### 10.5.1.7 maintainer\_email

`setup.maintainer_email`

### 10.5.1.8 name

`setup.name`

### 10.5.1.9 package\_name

`str setup.package_name = 'bioscara_description'`

### 10.5.1.10 packages

`setup.packages`

#### 10.5.1.11 tests\_require

`setup.tests_require`

#### 10.5.1.12 version

`setup.version`

#### 10.5.1.13 zip\_safe

`setup.zip_safe`

## 10.6 test\_copyright Namespace Reference

### Functions

- [test\\_copyright\(\)](#)

### 10.6.1 Function Documentation

#### 10.6.1.1 test\_copyright()

`test_copyright.test_copyright ( )`

## 10.7 test\_flake8 Namespace Reference

### Functions

- [test\\_flake8\(\)](#)

### 10.7.1 Function Documentation

#### 10.7.1.1 test\_flake8()

`test_flake8.test_flake8 ( )`

## 10.8 test\_joint\_trajectory\_controller Namespace Reference

### Functions

- [generate\\_launch\\_description\(\)](#)



### 10.8.1 Function Documentation

#### 10.8.1.1 generate\_launch\_description()

```
test_joint_trajectory_controller.generate_launch_description ( )
```

## 10.9 test\_pep257 Namespace Reference

### Functions

- [test\\_pep257](#) ( )

### 10.9.1 Function Documentation

#### 10.9.1.1 test\_pep257()

```
test_pep257.test_pep257 ( )
```



## Chapter 11

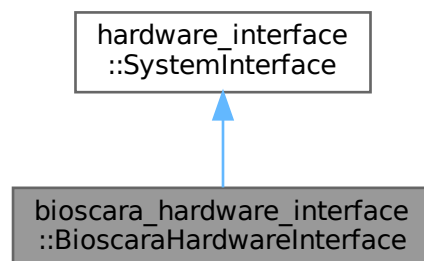
# Class Documentation

### 11.1 bioscara hardware\_interface::BioscaraHardwareInterface Class Reference

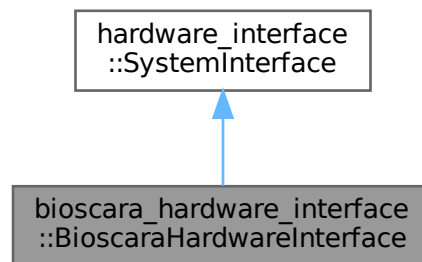
The bioscara hardware interface class.

```
#include <bioscara_hardware.hpp>
```

Inheritance diagram for bioscara\_hardware\_interface::BioscaraHardwareInterface:



Collaboration diagram for bioscara\_hardware\_interface::BioscaraHardwareInterface:



## Classes

- struct [joint\\_config\\_t](#)  
configuration structure holding the passed paramters from the `ros2_control urdf`
- struct [joint\\_homing\\_config\\_t](#)  
configuration structure holding the passed homing paramters from the `ros2_control urdf`

## Public Member Functions

- `hardware_interface::CallbackReturn` [on\\_init](#) (const `hardware_interface::HardwareComponentInterface`↔  
Params &params) override
- `hardware_interface::CallbackReturn` [on\\_shutdown](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called on the transistion from the `inactive`, `unconfigured` and `active` to the `finalized` state.*
- `hardware_interface::CallbackReturn` [on\\_configure](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called on the transistion from the `unconfigured` to the `inactive` state.*
- `hardware_interface::CallbackReturn` [on\\_cleanup](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called on the transistion from the `inactive` to the `unconfigured` state.*
- `hardware_interface::CallbackReturn` [on\\_activate](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called on the transistion from the `inactive` to the `active` state.*
- `hardware_interface::CallbackReturn` [on\\_deactivate](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called on the transistion from the `active` to the `inactive` state.*
- `hardware_interface::return_type` [read](#) (const `rcldcpp::Time` &time, const `rcldcpp::Duration` &period) override  
*Reads from the hardware and populates the state interfaces.*
- `hardware_interface::return_type` [write](#) (const `rcldcpp::Time` &time, const `rcldcpp::Duration` &period) override  
*Writes commands to the hardware from the command interfaces.*
- `hardware_interface::return_type` [prepare\\_command\\_mode\\_switch](#) (const `std::vector< std::string >` &start↔  
\_interfaces, const `std::vector< std::string >` &stop\_interfaces) override  
*Performs checks and book keeping of the active control mode when changing controllers.*
- `hardware_interface::CallbackReturn` [on\\_error](#) (const `rcldcpp_lifecycle::State` &previous\_state) override  
*Called when an error in any state or state transition is thrown.*

## Private Attributes

- `std::unordered_map< std::string, Joint > _joints`  
unordered map storing the *Joint* objects.
- `std::unordered_map< std::string, joint_config_t > _joint_cfg`  
unordered map storing the configuration struct of the joints.
- `std::unordered_map< std::string, std::set< std::string > > _joint_command_modes`  
unordered map of sets storing the active command interfaces for each joint.

### 11.1.1 Detailed Description

The bioscara hardware interface class.

The hardware interface serves to wrap custom hardware interaction in the standardized ros2\_control architecture.

#### Hardware Lifecycle

The hardware follows the ros2\_control hardware interface lifecycle which intern is following the *ROS2 managed node lifecycle*.

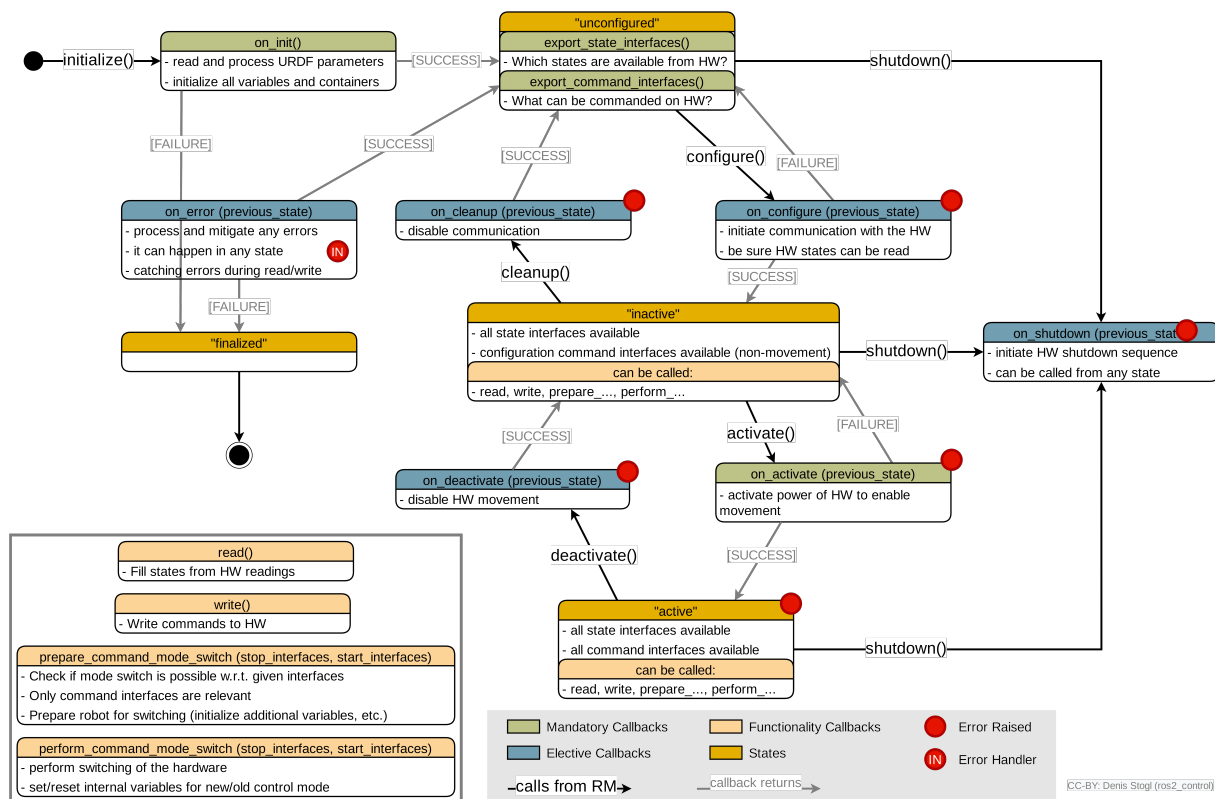


Figure 11.1 Hardware interface lifecycle

### 11.1.2 Member Function Documentation

#### 11.1.2.1 on\_activate()

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

Called on the transistion from the `inactive` to the `active` state.

Enables each joint, enables the stall detection and sets the maximum acceleration.

It is allowed to activate the hardware even if it is not homed. To home the joint the homing\_controller must be activated, but generally a hardware component must be active in order for controllers to become active.

To prohibit movement on activation the set point for each position command interface is set equal to the current measured position, and the velocity command is set to 0.0 for each command interface. The current values are obtained by calling the `read()` method once which populates the state interfaces with values.

#### Parameters

<code>previous_state</code>	
-----------------------------	--

#### Returns

`hardware_interface::CallbackReturn`

Below a workaround to force a read cycle of all joints to get initial values for the state interfaces. These will be copied to the command interface to prevent movement at startup.

#### 11.1.2.2 `on_cleanup()`

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

Called on the transistion from the `inactive` to the `unconfigured` state.

Disconnect from the joints.

#### Parameters

<code>previous_state</code>	
-----------------------------	--

#### Returns

`hardware_interface::CallbackReturn`

Disconnect from the joints and throw error if it fails

#### 11.1.2.3 `on_configure()`

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

Called on the transistion from the `unconfigured` to the `inactive` state.

Establish and test connection to each joint.

## Parameters

<i>previous_state</i>	
-----------------------	--

## Returns

hardware\_interface::CallbackReturn

## 11.1.2.4 on\_deactivate()

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

Called on the transision from the `active` to the `inactive` state.

Disables all joints and thereby allows backdriving. State interfaces continue to be updated.

## Parameters

<i>previous_state</i>	
-----------------------	--

## Returns

hardware\_interface::CallbackReturn

disable the joints and throw error if it fails

## 11.1.2.5 on\_error()

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

Called when an error in any state or state transition is thrown.

According to the [ros2\\_control documentation](#):

Error handling follows the node lifecycle. If successful `CallbackReturn::SUCCESS` is returned and hardware is again in `UNCONFIGURED` state, if any `ERROR` or `FAILURE` happens the hardware ends in `FINALIZED` state and can not be recovered. The only option is to reload the complete plugin, but there is currently no service for this in the Controller Manager.

Since the hardware will immediatly return to the `unconfigured` state ( [source](#)) if the error could be handled we manually call the transition functions which would normally be called to this state. Those are:

- **Previous state:** `active`
  - Deactivate hardware ([on\\_deactivate\(\)](#)) -> `inactive`

- Clean-Up hardware ([on\\_cleanup\(\)](#)) -> unconfigured
- **Previous state:** `inactive`
  - Deactivate hardware ([on\\_deactivate\(\)](#)) -> `inactive`
    - \* call the deactivate function anyway regardless if state was active or inactive. For example if the [on\\_activate\(\)](#) function fails on [Joint::enableStallguard\(\)](#) the joint will have been enabled, to disable it invoke [on\\_deactivate\(\)](#).
  - Clean-Up hardware ([on\\_cleanup\(\)](#)) -> unconfigured

In particular the deactivation is important. For example if a joint stalls the [read\(\)](#) or [write\(\)](#) methods throw an error, which will be handled here and allow the hardware to be deactivated, disabling the joints to allow backdriving.

#### Parameters

<code>previous_state</code>	
-----------------------------	--

#### Returns

`hardware_interface::CallbackReturn`

#### 11.1.2.6 on\_init()

```
hardware_interface::CallbackReturn bioscara_hardware_interface::BioscaraHardwareInterface↵
::on_init (
    const hardware_interface::HardwareComponentInterfaceParams & params ) [override]
```

Loop over all joints described in the hardware description file, check if they have the position and velocity command and state interface defined and finally add them to the internal `_joints` list

Loop over all GPIOs described in the hardware description file, check if they have the home command and state interface defined.

**Todo** threshold and current are `uint8_t`, if a number larger outside  $0 < n < 255$  is passed as a parameters it will overflow.

#### 11.1.2.7 on\_shutdown()

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

Called on the transition from the `inactive`, `unconfigured` and `active` to the `finalized` state.

When transitioning directly from `active` to `finalized` [on\\_deactivate\(\)](#) is automatically called before [Source Code](#) If the previous state is either `inactive` or `active` the [on\\_cleanup\(\)](#) method is called first. Then regardless of the previous state, the `_joints` map is cleared.



## Parameters

<i>previous_state</i>	
-----------------------	--

## Returns

hardware\_interface::CallbackReturn

## 11.1.2.8 prepare\_command\_mode\_switch()

```
hardware_interface::return_type bioscara_hardware_interface::BioscaraHardwareInterface::prepare↵
_command_mode_switch (
    const std::vector< std::string > & start_interfaces,
    const std::vector< std::string > & stop_interfaces ) [override]
```

Performs checks and book keeping of the active control mode when changing controllers.

For safe operation only one controller may interact with the hardware at the time. For example if the velocity JTC is active and has claimed the velocity command interfaces it is technically possible to activate the position JTC (or a homing controller, or others) that claim a different command interface (position in this case). However if both controllers are active they start writing to the hardware simultaneously which is to be avoided. For this reason a book keeping mechanism has been implemented which stores the currently active command interfaces for each joint in the `_joint_command_modes` member. Each joint has a set of active command interfaces. When a controller switch is performed the interfaces that should be stopped are removed from each joint set, then the one that should be started are added, if they are already present an error is thrown. Lastly a validation is performed. Currently the validation is simple since each joint may only have one command interface. The validation can be expanded for future use cases that require a combination of active command interfaces per joint for example.

The following basic checks are implemented:

- **On deactivation:**
  - [ERROR] Homing command interfaces may only be deactivated if no current homing process is ongoing (`Joint::getCurrentBCmd() != Joint::HOME`)
  - [WARN] Deactivating a velocity command interface if the velocity set point is 0.0.
  - [WARN] Deactivating a command interface that has not been started. This should not happen.
- **On activation:**
  - [ERROR] Activating a command interface that is already started. This should not happen.
  - [ERROR] Activating a second command interface for a joint.

## Parameters

<i>start_interfaces</i>	command interfaces that should be started in the form "joint/interface"
<i>stop_interfaces</i>	command interfaces that should be stopped in the form "joint/interface"

## Returns

hardware\_interface::return\_type

### 11.1.2.9 read()

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

Reads from the hardware and populates the state interfaces.

Iterates over all state interfaces and calls the corresponding [Joint](#) method.

- State interface "position" -> [Joint::getPosition\(\)](#)
- State interface "velocity" -> [Joint::getVelocity\(\)](#)
- State interface "home" -> [Joint::isHomed\(\)](#)
  - This does not actually trigger a communication, instead it relies on the return flags of the previous transmissions. Since position and velocity have been called immediately before the return flags are assumed to be valid.
  - If the homing of a joint has been activated through the command interface ([Joint::getCurrentBCmd\(\)](#) == [Joint::HOME](#)) the device signals BUSY ([Joint::isBusy\(\)](#)) as long as it is still homing. If the BUSY flag is reset while the current command is still [Joint::HOME](#) we can assume the homing has finished. Then the "home" command interface of the joint is reset to 0.0, which will stop the homing (perform cleanup tasks) at the next write cycle.

#### Parameters

<i>time</i>	
<i>period</i>	

#### Returns

hardware\_interface::return\_type

### 11.1.2.10 write()

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

Writes commands to the hardware from the command interfaces.

In contrast to the [read\(\)](#) method the [write\(\)](#) method only loops over the command interfaces that are currently active defined by the [BioscaraHardwareInterface::joint\\_command\\_modes](#) map. See [prepare\\_command\\_mode\\_switch\(\)](#) for a detailed reasoning why this approach has been chosen.

- Command interface "position" -> [Joint::setPosition\(\)](#)
- Command interface "velocity" -> [Joint::setVelocity\(\)](#)
- Command interface "home" -> [Joint::startHoming\(\)](#)

- If the commanded value in "home" is != 0.0 the and the joint is currently executing a blocking function, for example homing (`Joint::getCurrentBCmd() == Joint::NONE`), the homing sequence is started with the speed, sensitivity, current and acceleration defined in the `BioscaraHardwareInterface::_joint_cfg` which is polulated from the hardware description urdf. The direction of the homing is determined by the sign of the command interface value.
- If the commanded value in "home" is = 0.0 and the joint is currently executing homing, the homing is stopped. This can either happen prematurely through user input or when the homing is completed which is registered in `read()`.

#### Parameters

<i>time</i>	
<i>period</i>	

#### Returns

hardware\_interface::return\_type

### 11.1.3 Member Data Documentation

#### 11.1.3.1 \_joint\_cfg

```
std::unordered_map<std::string, joint_config_t> bioscara_hardware_interface::BioscaraHardwareInterface::_joint_cfg [private]
```

unordered map storing the configuration struct of the joints.

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.

#### 11.1.3.2 \_joint\_command\_modes

```
std::unordered_map<std::string, std::set<std::string> > bioscara_hardware_interface::BioscaraHardwareInterface::_joint_command_modes [private]
```

unordered map of sets storing the active command interfaces for each joint.

Each joint can have a set of active command interfaces. This type of structure is chosen to group interfaces by joint. In the `write()` function the interface name can simply be constructed by concatenating joint name with interface name. Although currently only one active command interface is allowed at the time, a set can be used to store multiple command interfaces that are acceptable to be combined, for example it would be acceptable to set velocity and driver current and hence that would be an allowable combination.

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.

### 11.1.3.3 \_joints

```
std::unordered_map<std::string, Joint> bioscara_hardware_interface::BioscaraHardwareInterface↔
::_joints [private]
```

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.

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

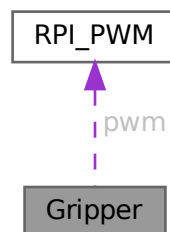
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/include/bioscara\_hardware\_↔ interface/[bioscara\\_hardware.hpp](#)
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/src/[bioscara\\_hardware.cpp](#)

## 11.2 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

## 11.2.1 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;
}
```

## 11.2.2 Constructor & Destructor Documentation

### 11.2.2.1 Gripper()

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

## 11.2.3 Member Function Documentation

### 11.2.3.1 deinit()

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

Placeholder, does nothing.

### Returns

0

#### 11.2.3.2 disable()

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

Disables the servo.

Stops the servo and disables the PWM generation.

##### Returns

non-zero error code.

#### 11.2.3.3 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.

#### 11.2.3.4 init()

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

Placeholder, does nothing.

##### Returns

0

#### 11.2.3.5 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

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

## 11.2.4 Member Data Documentation

### 11.2.4.1 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

## 11.3 Joint Class Reference

Representing a single joint on the I2C bus.

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

### Public Types

- enum `stp_reg_t` {  
`NONE` = 0x00 , `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 , `SETBRAKEMODE` = 0x22 ,  
`ENABLEPID` = 0x23 , `DISABLEPID` = 0x24 , `ENABLECLOSEDLOOP` = 0x25 , `DISABLECLOSEDLOOP` = 0x26 ,  
`SETCONTROLTHRESHOLD` = 0x27 , `MOVETOEND` = 0x28 , `STOP` = 0x29 , `GETPIDERROR` = 0x2A ,  
`CHECKORIENTATION` = 0x2B , `GETENCODERRPM` = 0x2C , `HOME` = 0x2D , `HOMEOFFSET` = 0x2E }  
*register and command definitions*

### Public Member Functions

- `Joint` (const std::string `name`, const int `address`, const float `reduction`, const float `min`, const float `max`)  
*Create a `Joint` object.*
- `~Joint` (void)
- int `init` (void)  
*Established connection to a joint via I2C.*
- int `deinit` (void)  
*Disconnects from a joint.*
- int `enable` (u\_int8\_t `driveCurrent`, u\_int8\_t `holdCurrent`)

- Setup the joint and engages motor.*

  - int `disable` (void)

*disengages the joint motor without closing i2c handle*
- int `home` (float velocity, u\_int8\_t sensitivity, u\_int8\_t current)

*Blocking implementation to home the joint.*
- int `startHoming` (float velocity, u\_int8\_t sensitivity, u\_int8\_t current)

*non-blocking implementation to home the joint.*
- int `postHoming` (void)

*perform tasks after a non-blocking homing.*
- 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 `stop` (void)
- 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<sup>2</sup> or m/s<sup>2</sup> 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 of the joint.*
- 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.*
- bool `isBusy` (void)
- Checks if the joint controller is busy processing a blocking command.*
- int `checkCom` (void)
- Check if communication to the joint is established.*
- u\_int8\_t `getFlags` (void)
- int `getHomingOffset` (float &offset)



- Retrieves the homing position from the last homing.
- `int setHomingOffset` (const float `offset`)  
Stores the homing position on the joint.
- `stp_reg_t getCurrentBCmd` (void)  
get the currently active blocking command

### Public Attributes

- `std::string name`

### 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.
- `void wait_while_busy` (const float `period_ms`)  
Blocking loop waiting for BUSY flag to reset.
- `int _home` (float `velocity`, `u_int8_t sensitivity`, `u_int8_t current`)  
Call to start the homing sequence of a joint.

### 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.
- `float min` = 0  
Joint lower limit.
- `float max` = 0  
Joint upper limit.
- `stp_reg_t current_b_cmd` = NONE  
Keeps track if a blocking command is being executed.
- `int handle` = -1  
I2C bus handle.

## 11.3.1 Detailed Description

Representing a single joint on the I2C bus.

## 11.3.2 Member Enumeration Documentation

### 11.3.2.1 stp\_reg\_t

enum [Joint::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

NONE	Used for signalling purposes.
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	
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].
HOMEOFFSET	R/W; Size: 4; [(float) -].

### 11.3.3 Constructor & Destructor Documentation

#### 11.3.3.1 Joint()

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

Create a [Joint](#) object.

The [Joint](#) object represents a single joint and its actuator. Each [Joint](#) has a transmission with the following relationship:

actuator position = (joint position - offset) \* reduction  
 joint position = actuator position / reduction + offset

#### Parameters

<i>name</i>	string device name for identification
<i>address</i>	1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4
<i>reduction</i>	gear reduction of the joint. This is used to transform position and velocity values between in joint units and actuator (stepper) units. The sign depends on the direction the motor is mounted and is turning. Adjust such that the joint moves in the positive direction on on positive joint commands. Cable polarity has no effect since the motors automatically adjust to always run in the 'right' direction from their point of view. J1: 35 J2: -2*pi/0.004 (4 mm linear movement per stepper revolution) J3: 24 J4: 12
<i>min</i>	lower joint limit in joint units. J1: -3.04647 J2: -0.0016 J3: -2.62672 J4: -3.01069
<i>max</i>	upper joint limit in joint units. J1: 3.04647 J2: 0.3380 J3: 2.62672 J4: 3.01069

#### 11.3.3.2 ~Joint()

```
Joint::~~Joint (
    void )
```

### 11.3.4 Member Function Documentation

#### 11.3.4.1 \_home()

```
int Joint::_home (
    float velocity,
    u_int8_t sensitivity,
    u_int8_t current ) [private]
```

Call to start the homing sequence of a joint.

First the joint will check the motor wiring by executing the checkOrientation internally. Then it will set the specified speed until a resistance which drives the PID error above the specified threshold is encountered. At this point the stepper stops and zeros the encoder.

#### Parameters

<i>velocity</i>	signed velocity in rad/s or m/s. Must be between $1.0 < \text{RAD2DEG}(\text{JOINT2ACTUATOR}(\text{velocity}, \text{reduction}, 0)) / 6 < 250.0$
<i>sensitivity</i>	Encoder pid error threshold 0 to 255.
<i>current</i>	homing current, determines how easy it is to stop the motor and thereby provoke a stall

**Returns**

0 on success, -1 on communication error, -3 when the motor is not enabled, -5 if the joint is not initialized, -101 if the velocity is zero, -102 if absolute value of the velocity is outside the specified limits.

**11.3.4.2 checkCom()**

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

Check if communication to the joint is established.

Sends a PING to and expects a ACK from the joint.

**Returns**

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.3 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**

<i>angle</i>	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, -5 if the joint is not initialized.

**11.3.4.4 deinit()**

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

Disconnects from a joint.

Removes the joint from the I2C bus.

**Returns**

0 on success, -1 when the joint could not be removed due to an I2C error, -5 if the joint is not initialized.

#### 11.3.4.5 disable()

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

disengages the joint motor without closing i2c handle

##### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

#### 11.3.4.6 disableCL()

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

Disables the Closed-Loop PID Controller.

##### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

#### 11.3.4.7 enable()

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

Setup the joint and engages motor.

This function prepares the motor for movement. After successful execution the joint is ready to accept [Joint::setPosition\(\)](#) and [Joint::setVelocity\(\)](#) commands.

The function sets the drive and hold current for the specified joint and engages the motor. The currents are in percent of driver max. output (2.5A, check with TMC5130 datasheet or Ustepper documentation)

##### Parameters

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

##### Returns

0 on success, -1 on communication error, -3 when the motor is not enabled, -5 if the joint is not initialized.

#### 11.3.4.8 enableStallguard()

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

Enable encoder stall detection of the joint.

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

#### Parameters

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

#### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

#### 11.3.4.9 `getCurrentBCmd()`

```
Joint::stp_reg_t Joint::getCurrentBCmd (
    void )
```

get the currently active blocking command

#### Returns

The the command of type `stp_reg_t`

#### 11.3.4.10 `getFlags()`

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

get driver state flags

#### Returns

flags  $\geq 0$  on success, -5 if the joint is not initialized.

#### 11.3.4.11 `getHomingOffset()`

```
int Joint::getHomingOffset (
    float & offset )
```

Retrieves the homing position from the last homing.

The homing position is stored on the joint to make it persistent as long as the joint is powered up.

#### Returns

0 on success, -1 on communication error, -2 when not homed, -5 if the joint is not initialized.

#### 11.3.4.12 `getPosition()`

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

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

#### Warning

If the joint is not homed this method does not return an error. Instead `pos` will be 0.0.

## Parameters

<i>pos</i>	
------------	--

## Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.13 getVelocity()**

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

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

## Parameters

<i>vel</i>	
------------	--

## Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.14 home()**

```
int Joint::home (
    float velocity,
    u_int8_t sensitivity,
    u_int8_t current )
```

Blocking implementation to home the joint.

A blocking implementation which only returns after the the joint is no longer BUSY. See [Joint::\\_home\(\)](#) for documentation.

Additionally this method returns:

## Returns

-2 when not homed succesfull (isHomed flag still not set), -109 if the joint is already currently homing (for example from a call to [Joint::startHoming\(\)](#)).

**11.3.4.15 init()**

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

Established connection to a joint via I2C.

Adds the joint to the I2C bus and tests if is responsive by sending a PING.

## Returns

0 on success, -1 on when no ACK is received from the joint, -2 if the I2C device could not be opened given the joint address.



#### 11.3.4.16 isBusy()

```
bool Joint::isBusy (
    void )
```

Checks if the joint controller is busy processing a blocking command.

Reads the internal state flags from the last transmission. If an update is necessary call [Joint::getFlags\(\)](#) before invoking this function.

##### Returns

true if a blocking command is currently executing, false if not.

#### 11.3.4.17 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 necessary call [Joint::getFlags\(\)](#) before invoking this function. If the motor actually can move depends on the state of the STALLED flag which can be checked using [Joint::isStalled\(\)](#).

##### Returns

true if the motor is enabled, false if not.

#### 11.3.4.18 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 necessary call [Joint::getFlags\(\)](#) before invoking this function.

##### Returns

true if the motor is homed, false if not.

#### 11.3.4.19 isStalled()

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

Checks if the motor is stalled.

Reads the internal state flags from the last transmission. If an update is necessary call [Joint::getFlags\(\)](#) before invoking this function.

##### Returns

true if the motor is stalled, false if not.

#### 11.3.4.20 moveSteps()

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

Move full steps.

This function can be called even when not homed.

## Parameters

<i>steps</i>	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, -5 if the joint is not initialized.

**11.3.4.21 postHoming()**

```
int Joint::postHoming (
    void )
```

perform tasks after a non-blocking homing.

This method resets the `current_b_cmd` to NONE, checks if the joint is homed, and saves the homing offset to the joint.

## Returns

0 on success, -109 if the `current_b_cmd` is not HOME, -1 on communication error, -2 when not homed, -5 if the joint is not initialized.

**11.3.4.22 printInfo()**

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

**11.3.4.23 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 transmission flags to `flags`. See [Joint::flags](#) for details.

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

## Template Parameters

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

**Parameters**

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

**Returns**

0 on OK, negative on error

**11.3.4.24 setBrakeMode()**

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

Set Brake Mode.

**Parameters**

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

**Returns**

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.25 setDriveCurrent()**

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

Set the Drive Current.

**Warning**

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

**Parameters**

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

**Returns**

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.26 setHoldCurrent()**

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

Set the Hold Current.

#### Warning

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

#### Parameters

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

#### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

#### 11.3.4.27 setHomingOffset()

```
int Joint::setHomingOffset (
    const float offset )
```

Stores the homing position on the joint.

The homing position is stored on the joint to make it persistent as long as the joint is powered up.

#### Returns

0 on success, -1 on communication error, -2 if not homed, -5 if the joint is not initialized.

#### 11.3.4.28 setMaxAcceleration()

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

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

#### Parameters

<i>maxAccel</i>	maximum joint acceleration.
-----------------	-----------------------------

#### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

#### 11.3.4.29 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**

<i>maxVel</i>	maximum joint velocity.
---------------	-------------------------

**Returns**

0 on success, -1 on communication error, -5 if the joint is not initialized.

**11.3.4.30 setPosition()**

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

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

**Parameters**

<i>pos</i>	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, -5 if the joint is not initialized.

**11.3.4.31 setVelocity()**

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

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

**Parameters**

<i>vel</i>	
------------	--

**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, -5 if the joint is not initialized.

**11.3.4.32 startHoming()**

```
int Joint::startHoming (
    float velocity,
    u_int8_t sensitivity,
    u_int8_t current )
```

non-blocking implementation to home the joint.

See [Joint::\\_home\(\)](#) for documentation. The `current_b_cmd` flag is set to HOME. This method returns immediately after starting the homing sequence. This should be used when the blocking implementation is not acceptable. For example in the update loop of the [bioscara\\_hardware\\_interface::BioscaraHardwareInterface::write\(\)](#).

Additionally this method returns:

#### Returns

-109 if the joint is already currently homing (for example from a call to [Joint::startHoming\(\)](#)).

### 11.3.4.33 stop()

```
int Joint::stop (
    void )
```

Stops the motor.

Stops the motor by setting the maximum velocity to zero and the position setpoint to the current position

#### Returns

0 on success, -1 on communication error, -5 if the joint is not initialized.

### 11.3.4.34 wait\_while\_busy()

```
void Joint::wait_while_busy (
    const float period_ms ) [private]
```

Blocking loop waiting for BUSY flag to reset.

#### Parameters

<i>period_ms</i>	time in ms between polls.
------------------	---------------------------

### 11.3.4.35 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`. Copies *data* to the buffer and invokes [writeToI2CDev\(\)](#). The flags received from the transaction are copied to *flags*. The flags are described in [Joint::read\(\)](#).

### Template Parameters

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

### Parameters

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

### Returns

0 on OK, negative on error

## 11.3.5 Member Data Documentation

### 11.3.5.1 address

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

I2C adress.

### 11.3.5.2 current\_b\_cmd

```
stp_reg_t Joint::current_b_cmd = NONE [private]
```

Keeps track if a blocking command is being executed.

### 11.3.5.3 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	NOTENABLED	NOTHOMED	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.



**NOTHOMED** is cleared if the joint is homed. Movement is only allowed if this flag is clear  
**NOTENABLED** is cleared if the joint is enabled after calling [Joint::enable\(\)](#)

#### 11.3.5.4 handle

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

I2C bus handle.

#### 11.3.5.5 max

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

[Joint](#) upper limit.

#### 11.3.5.6 min

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

[Joint](#) lower limit.

#### 11.3.5.7 name

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

#### 11.3.5.8 offset

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

[Joint](#) position offset.

#### 11.3.5.9 reduction

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

[Joint](#) to actuator reduction ratio.

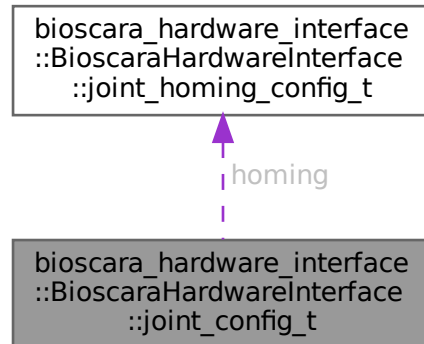
The documentation for this class was generated from the following files:

- 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](#)

## 11.4 bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_↔ config\_t Struct Reference

configuration structure holding the passed paramters from the ros2\_control urdf

Collaboration diagram for bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t:



### Public Attributes

- int `i2c_address`
- float `reduction` = 1
- float `min`
- float `max`
- u\_int8\_t `drive_current`
- u\_int8\_t `hold_current`
- u\_int8\_t `stall_threshold`
- float `max_velocity`
- float `max_acceleration`
- joint\_homing\_config\_t `homing`

### 11.4.1 Detailed Description

configuration structure holding the passed paramters from the ros2\_control urdf

Saving all parameters on initialization in a structure allows for quick access during runtime.

### 11.4.2 Member Data Documentation

#### 11.4.2.1 drive\_current

u\_int8\_t bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t::drive\_current

#### 11.4.2.2 hold\_current

```
u_int8_t bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::hold_current
```

#### 11.4.2.3 homing

```
joint_homing_config_t bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t↔  
::homing
```

#### 11.4.2.4 i2c\_address

```
int bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::i2c_address
```

#### 11.4.2.5 max

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::max
```

#### 11.4.2.6 max\_acceleration

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::max_acceleration
```

#### 11.4.2.7 max\_velocity

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::max_velocity
```

#### 11.4.2.8 min

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::min
```

#### 11.4.2.9 reduction

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::reduction = 1
```

#### 11.4.2.10 stall\_threshold

```
u_int8_t bioscara_hardware_interface::BioscaraHardwareInterface::joint_config_t::stall↔  
threshold
```

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

- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/include/bioscara\_hardware\_↔  
interface/[bioscara\\_hardware.hpp](#)

## 11.5 bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_homing\_config\_t Struct Reference

configuration structure holding the passed homing paramters from the ros2\_control urdf

### Public Attributes

- float [speed](#) = 0
- u\_int8\_t [threshold](#) = 10
- u\_int8\_t [current](#) = 10
- float [acceleration](#) = 0.01

### 11.5.1 Detailed Description

configuration structure holding the passed homing paramters from the ros2\_control urdf

Saving all parameters on initialization in a structure allows for quick access during runtime.

### 11.5.2 Member Data Documentation

#### 11.5.2.1 acceleration

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t::acceleration
= 0.01
```

#### 11.5.2.2 current

```
u_int8_t bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t::current
= 10
```


#### 11.5.2.3 speed

```
float bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t::speed = 0
```

#### 11.5.2.4 threshold

```
u_int8_t bioscara_hardware_interface::BioscaraHardwareInterface::joint_homing_config_t::threshold
= 10
```

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

- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/include/bioscara\_hardware\_   
interface/[bioscara\\_hardware.hpp](#)

## 11.6 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](#)

### 11.6.1 Detailed Description

PWM class for the Raspberry PI 4 and 5.

### 11.6.2 Constructor & Destructor Documentation

#### 11.6.2.1 [~RPI\\_PWM\(\)](#)

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

### 11.6.3 Member Function Documentation

#### 11.6.3.1 [disable\(\)](#)

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

#### 11.6.3.2 [enable\(\)](#)

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

#### 11.6.3.3 [setDutyCycle\(\)](#)

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

Sets the duty cycle.

## Parameters

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

**11.6.3.4 setDutyCycleNS()**

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

**11.6.3.5 setPeriod()**

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

**11.6.3.6 start()**

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

Starts the PWM

## Parameters

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

**11.6.3.7 stop()**

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

Stops the PWM

**11.6.3.8 writeSYS()**

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

## 11.6.4 Member Data Documentation

### 11.6.4.1 chippath

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

### 11.6.4.2 per

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

### 11.6.4.3 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](#)





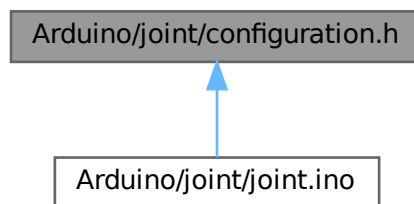
# Chapter 12

## File Documentation

### 12.1 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<sup>2</sup>. 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.*

### 12.1.1 Detailed Description

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

**Author**

Sebastian Storz

**Version**

0.1

**Date**

2025-05-27

**Copyright**

Copyright (c) 2025

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

### 12.1.2 Macro Definition Documentation

#### 12.1.2.1 ADR

```
#define ADR 0x11
```

I2C adress of joint n is 0x1n.

#### 12.1.2.2 MAXACCEL

```
#define MAXACCEL 10000
```

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

#### 12.1.2.3 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.

## 12.2 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
```

## 12.3 Arduino/joint/joint.h File Reference

joint firmware header

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

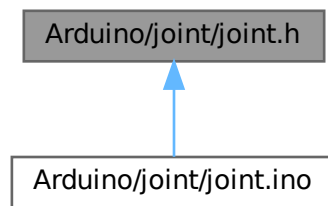
Include dependency graph for joint.h:

Arduino/joint/joint.h



Arduino.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 , SETBRAKEMODE = 0x22 , ENABLEPID = 0x23 ,`  
`DISABLEPID = 0x24 , ENABLECLOSEDLOOP = 0x25 , DISABLECLOSEDLOOP = 0x26 , SETCONTROLTHRESHOLD`  
`= 0x27 ,`  
`MOVETOEND = 0x28 , STOP = 0x29 , GETPIDERROR = 0x2A , CHECKORIENTATION = 0x2B ,`  
`GETENCODERRPM = 0x2C , HOME = 0x2D , HOMEOFFSET = 0x2E }`  
*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.*

### 12.3.1 Detailed Description

joint firmware header

#### Author

Sebastian Storz

#### Version

0.1

#### Date

2025-05-27

#### Copyright

Copyright (c) 2025

This file contains definitions and macros for the joint firmware.

### 12.3.2 Macro Definition Documentation

#### 12.3.2.1 ACK

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

#### 12.3.2.2 DUMP\_BUFFER

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

#### Value:

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

Macro to dump a buffer to the serial console.

#### Parameters

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

### 12.3.2.3 MAX\_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32\_t

### 12.3.2.4 NACK

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

### 12.3.2.5 RFLAGS\_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

## 12.3.3 Enumeration Type Documentation

### 12.3.3.1 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 <sup>2</sup> ].
SETMAXDECELERATION	
SETMAXVELOCITY	W; Size: 4; [(float) deg/s].
ENABLESTALLGUARD	W; Size: 1; [(uint8) threshold].

## Enumerator

DISABLESTALLGUARD	
CLEARSTALL	
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].
HOMEOFFSET	R/W; Size: 4; [(float) -].

## 12.3.4 Function Documentation

## 12.3.4.1 readValue()

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

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

## Parameters

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

## 12.3.4.2 writeValue()

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

Writes a value of the specified type to a buffer.

## Parameters

<i>val</i>	Reference to input variable
------------	-----------------------------

## Parameters

<i>txBuf</i>	pointer to tx buffer
<i>tx_length</i>	Length of the buffer returned

## Returns

0 On success

## 12.4 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     SETBRAKEMODE = 0x22,
00081     ENABLEPID = 0x23,
00082     DISABLEPID = 0x24,
00083     ENABLECLOSEDLOOP = 0x25,
00084     DISABLECLOSEDLOOP = 0x26,
00085     SETCONTROLTHRESHOLD = 0x27,
00086     MOVETOEND = 0x28,
00087     STOP = 0x29,
00088     GETPIDERROR = 0x2A,
00089     CHECKORIENTATION = 0x2B,
00090     GETENCODERRPM = 0x2C,
00091     HOME = 0x2D,
00092     HOMEOFFSET = 0x2E,
00093 };
00094
00101 template <typename T>
00102 void readValue(T &val, uint8_t *rxBuf, size_t rx_length)
00103 {
00104     memcpy(&val, rxBuf, rx_length);
00105 }
00106

```



```

00114 template <typename T>
00115 int writeValue(const T val, uint8_t *txBuf, size_t &tx_length)
00116 {
00117     tx_length = sizeof(T);
00118     memcpy(txBuf, &val, tx_length);
00119     return 0;
00120 }
00121
00122 #endif

```

## 12.5 Arduino/joint/joint.ino File Reference

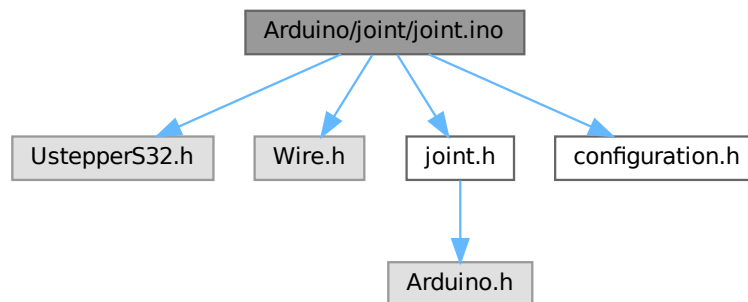
joint firmware

```

#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

**12.5.1 Detailed Description**

joint firmware

**Author**

Sebastian Storz

**Version**

0.1

**Date**

2025-05-27

**Copyright**

Copyright (c) 2025

This file contains the joint firmware.

**12.5.2 Macro Definition Documentation****12.5.2.1 J4**

```
#define J4
```

Define either joint that is to be flashed.

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

**12.5.3 Function Documentation****12.5.3.1 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**

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

Homing has been canceled from ISR (f.x. STOP)

**12.5.3.2 loop()**

```
void loop (
    void )
```

Main loop.

Executes the following:

- 1) if isStallguardEnabled: compares stepper.getPidError() with stallguardThreshold and sets isStalled flag.
- 2) 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

**12.5.3.3 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**

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

**12.5.3.4 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 immediately 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][RXBUF<sub>n</sub>]...[RXBUF<sub>2</sub>][RXBUF<sub>1</sub>][RXBUF<sub>0</sub>]

> [FLAGS]

The payload is read into the rx\_buf, rx\_length is set to the payload length.

### Parameters

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

#### 12.5.3.5 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.

#### 12.5.3.6 setup()

```
void setup (
    void )
```

Setup Peripherals.

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

### 12.5.4 Variable Documentation

#### 12.5.4.1 reg

```
uint8_t reg = 0
```

#### 12.5.4.2 rx\_buf

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

#### 12.5.4.3 rx\_data\_ready

```
bool rx_data_ready = 0
```

#### 12.5.4.4 rx\_length

```
size_t rx_length = 0
```

#### 12.5.4.5 stepper

```
UstepperS32 stepper
```

#### 12.5.4.6 tx\_buf

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

#### 12.5.4.7 tx\_data\_ready

```
bool tx_data_ready = 0
```

#### 12.5.4.8 tx\_length

```
size_t tx_length = 0
```

## 12.6 docs/DOCS\_README.md File Reference

## 12.7 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ bringup/launch/bioscara.launch.py File Reference

### Namespaces

- namespace [bioscara](#)

### Functions

- [bioscara.generate\\_launch\\_description\(\)](#)

## 12.8 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\(\)](#)

## 12.9 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_bringup/↔ README.md File Reference

### 12.10 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/↔ README.md File Reference

### 12.11 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/README.md File Reference

### 12.12 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/bioscara\_description/\_\_init\_\_.py File Reference

### 12.13 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/launch/display.launch.py File Reference

#### Namespaces

- namespace [display](#)

#### Functions

- [display.generate\\_launch\\_description\(\)](#)

### 12.14 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/launch/gazebo.launch.py File Reference

#### Namespaces

- namespace [gazebo](#)

#### Functions

- [gazebo.generate\\_launch\\_description\(\)](#)

### 12.15 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`

## 12.16 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/test/test\_copyright.py File Reference

### Namespaces

- namespace `test_copyright`

### Functions

- `test_copyright.test_copyright` ()

## 12.17 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/test/test\_flake8.py File Reference

### Namespaces

- namespace `test_flake8`

### Functions

- `test_flake8.test_flake8` ()

## 12.18 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_↔ description/test/test\_pep257.py File Reference

### Namespaces

- namespace `test_pep257`



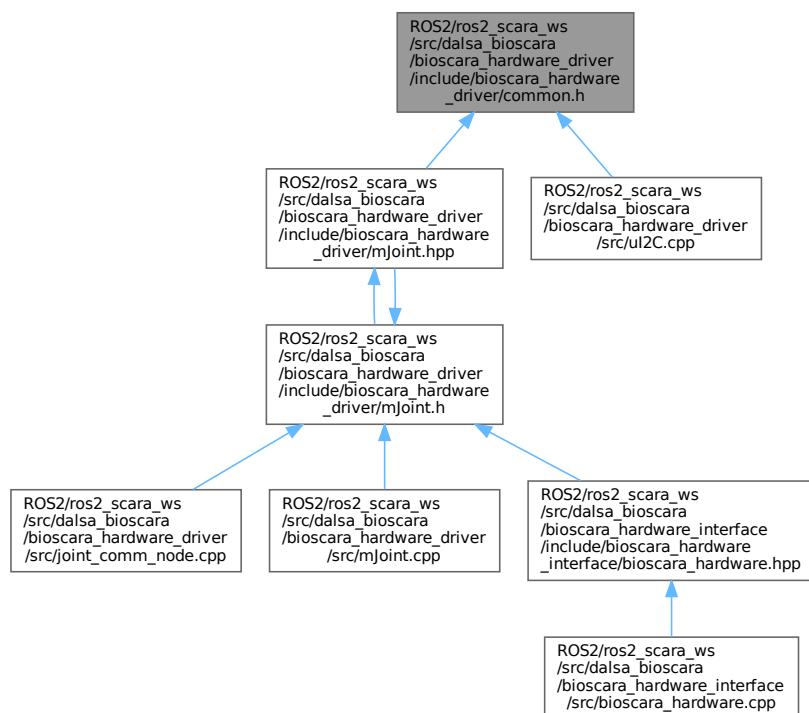
## Functions

- [test\\_pep257.test\\_pep257](#) ()

## 12.19 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_driver/include/bioscara\_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.

### 12.19.1 Detailed Description

A file containing utility macros and functions.

#### Author

Sebastian Storz

**Version**

0.1

**Date**

2025-05-27

**Copyright**

Copyright (c) 2025

**12.19.2 Macro Definition Documentation****12.19.2.1 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**

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

**12.20 common.h**

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

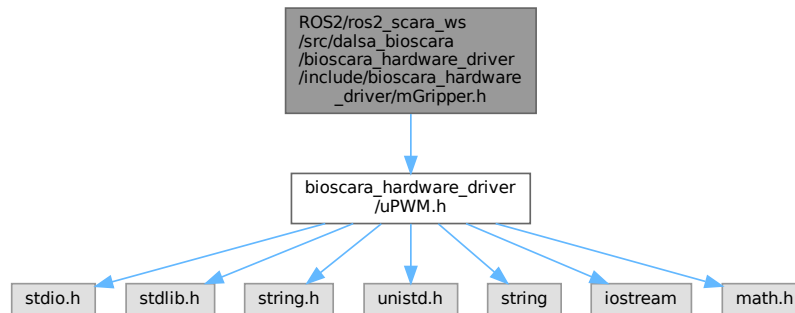
```
00001
00011 #ifndef COMMON_H
00012 #define COMMON_H
00013
00014
00021 #define DUMP_BUFFER(buffer, size)
00022 {
00023     std::cout << "Buffer dump: ";
00024     for (size_t i = 0; i < size; i++)
00025     {
00026         printf("%#x ", buffer[i]);
00027     }
00028     std::cout << std::endl;
00029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

## 12.21 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/include/bioscara\_hardware\_driver/mGripper.h File Reference

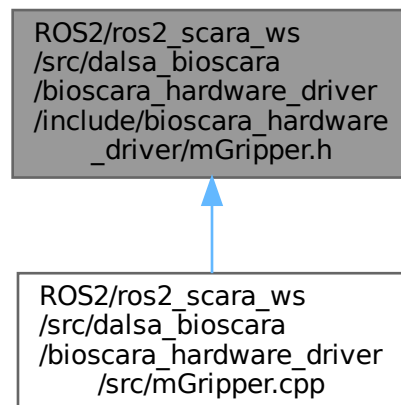
File containing the [Gripper](#) class.

```
#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.*

### 12.21.1 Detailed Description

File containing the [Gripper](#) class.

#### Author

Sebastian Storz

#### Version

0.1

#### Date

2025-05-27

#### Copyright

Copyright (c) 2025

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

## 12.22 mGripper.h

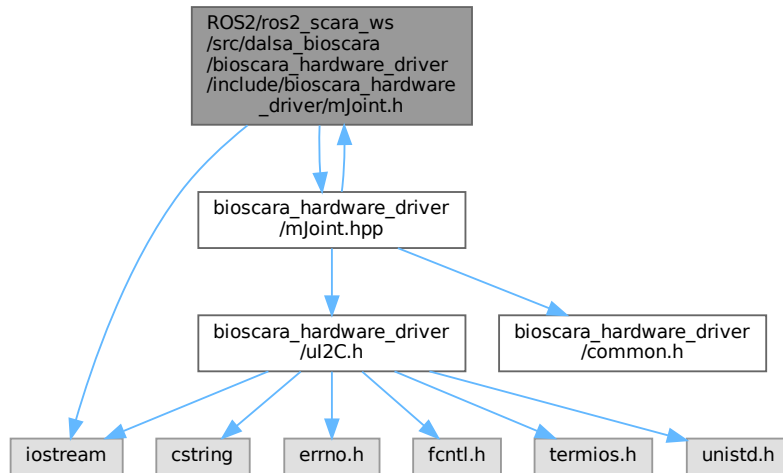
[Go to the documentation of this file.](#)

```
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
```

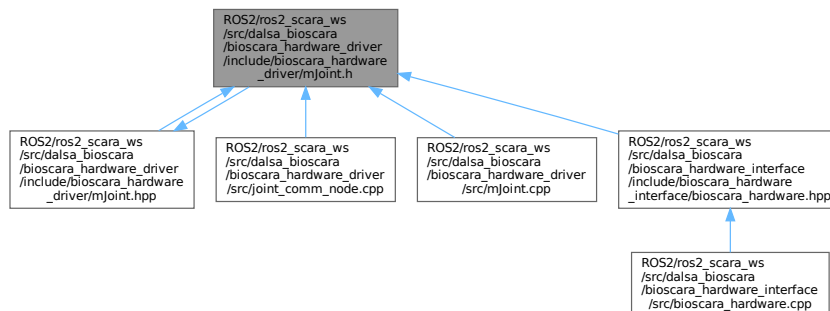
## 12.23 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/include/bioscara\_hardware\_driver/mJoint.h File Reference

File including the [Joint](#) class.

```
#include <iostream>
#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.*

### 12.23.1 Detailed Description

File including the `Joint` class.

#### Author

Sebastian Storz

#### Version

0.1

#### Date

2025-05-29

#### Copyright

Copyright (c) 2025

### 12.23.2 Macro Definition Documentation

#### 12.23.2.1 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

### 12.23.2.2 DEG2RAD

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

Macro to convert degree to radians.

### 12.23.2.3 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

### 12.23.2.4 M\_PI

```
#define M_PI 3.14159265358979323846
```

pi

### 12.23.2.5 RAD2DEG

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

Macro to convert radians to degree.

## 12.24 mJoint.h

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

```
00001  
00012 #ifndef MJOINT_H  
00013 #define MJOINT_H  
00014  
00015 #include <iostream>  
00016  
00025 #define JOINT2ACTUATOR(in, reduction, offset) (reduction * (in - offset))  
00026  
00035 #define ACTUATOR2JOINT(in, reduction, offset) (in / reduction + offset)  
00036  
00041 #define M_PI 3.14159265358979323846  
00042  
00047 #define RAD2DEG(rad) (rad / M_PI * 180)  
00048  
00053 #define DEG2RAD(deg) (deg * M_PI / 180)  
00054  
00059 class Joint  
00060 {
```

```

00061 public:
00071     enum stp_reg_t
00072     {
00073         NONE = 0x00,
00074         PING = 0x0f,
00075         SETUP = 0x10,
00076         SETRPM = 0x11,
00077         GETDRIVERRPM = 0x12,
00078         MOVESTEPS = 0x13,
00079         MOVEANGLE = 0x14,
00080         MOVETOANGLE = 0x15,
00081         GETMOTORSTATE = 0x16,
00082         RUNCOTINOUS = 0x17,
00083         ANGLEMOVED = 0x18,
00084         SETCURRENT = 0x19,
00085         SETHOLDCURRENT = 0x1A,
00086         SETMAXACCELERATION = 0x1B,
00087         SETMAXDECELERATION = 0x1C,
00088         SETMAXVELOCITY = 0x1D,
00089         ENABLESTALLGUARD = 0x1E,
00090         DISABLESTALLGUARD = 0x1F,
00091         CLEARSTALL = 0x20,
00092         SETBRAKEMODE = 0x22,
00093         ENABLEPID = 0x23,
00094         DISABLEPID = 0x24,
00095         ENABLECLOSEDLOOP = 0x25,
00096         DISABLECLOSEDLOOP = 0x26,
00097         SETCONTROLTHRESHOLD = 0x27,
00098         MOVETOEND = 0x28,
00099         STOP = 0x29,
00100         GETPIDERROR = 0x2A,
00101         CHECKORIENTATION = 0x2B,
00102         GETENCODERRPM = 0x2C,
00103         HOME = 0x2D,
00104         HOMEOFFSET = 0x2E,
00105     };
00106
00139     Joint(const std::string name, const int address, const float reduction, const float min, const float
max);
00140     ~Joint(void);
00141
00151     int init(void);
00152
00162     int deinit(void);
00163
00178     int enable(u_int8_t driveCurrent, u_int8_t holdCurrent);
00179
00186     int disable(void);
00187
00197     int home(float velocity, u_int8_t sensitivity, u_int8_t current);
00198
00209     int startHoming(float velocity, u_int8_t sensitivity, u_int8_t current);
00210
00224     int postHoming(void);
00225
00226     int printInfo(void);
00227
00240     int getPosition(float &pos);
00241
00254     int setPosition(float pos);
00255
00268     int moveSteps(int32_t steps);
00269
00279     int getVelocity(float &vel);
00280
00293     int setVelocity(float vel);
00294
00307     int checkOrientation(float angle = 10.0);
00308
00319     int stop(void);
00325     int disableCL(void);
00326
00334     int setDriveCurrent(u_int8_t current);
00335
00344     int setHoldCurrent(u_int8_t current);
00345
00352     int setBrakeMode(u_int8_t mode);
00353
00362     int setMaxAcceleration(float maxAccel);
00363
00372     int setMaxVelocity(float maxVel);
00373
00383     int enableStallguard(u_int8_t sensitivity);
00384
00393     bool isHomed(void);
00394
00404     bool isEnabled(void);

```



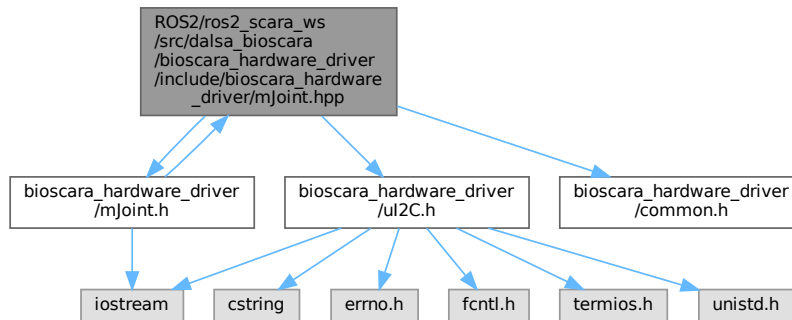
```
00405
00413     bool isStalled(void);
00414
00422     bool isBusy(void);
00423
00432     int checkCom(void);
00433
00439     u_int8_t getFlags(void);
00440
00451     int getHomingOffset(float &offset);
00452
00463     int setHomingOffset(const float offset);
00464
00470     stp_reg_t getCurrentBCmd(void);
00471
00472     std::string name;
00473
00474 protected:
00475 private:
00476     template <typename T>
00477     int read(const stp_reg_t reg, T &data, u_int8_t &flags);
00478
00479     template <typename T>
00480     int write(const stp_reg_t reg, T data, u_int8_t &flags);
00481
00487     void wait_while_busy(const float period_ms);
00488
00506     int _home(float velocity, u_int8_t sensitivity, u_int8_t current);
00507
00526     u_int8_t flags = 0x00;
00527
00528     int address;
00529     float reduction = 1;
00530     float offset = 0;
00531     float min = 0;
00532     float max = 0;
00533
00534     stp_reg_t current_b_cmd = NONE;
00535
00536     int handle = -1;
00537 };
00538
00539 #include "bioscara_hardware_driver/mJoint.hpp"
00540
00541 #endif
```

## 12.25 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/include/bioscara\_hardware\_driver/mJoint.hpp File Reference

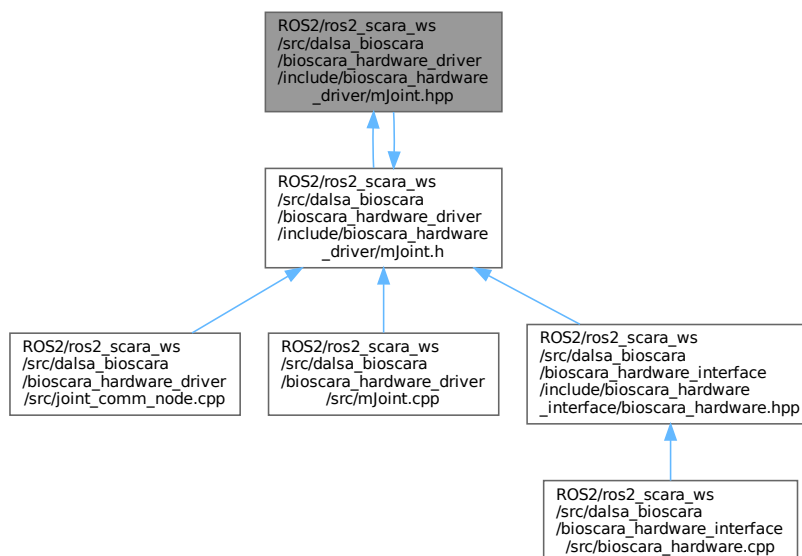
Templated functions for the [Joint](#) class.

```
#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:



### 12.25.1 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.

## 12.26 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[MAX_BUFFER+RFLAGS_SIZE];
00036     int n = readFromI2CDev(this->handle, reg, buf, size);
00037     if (n != static_cast<int>(size))
00038     {
00039         return -1;
00040     }
00041     memcpy(&data, buf, size - RFLAGS_SIZE);
00042     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00043     return 0;
00044 }
00045
00061 template <typename T>
00062 int Joint::write(const stp_reg_t reg, T data, u_int8_t &flags)
00063 {
00064     size_t size = sizeof(T) + RFLAGS_SIZE;
00065     char buf[MAX_BUFFER+RFLAGS_SIZE];
00066     memcpy(buf, &data, size - RFLAGS_SIZE);
00067     int rc = writeToI2CDev(this->handle, reg, buf, size - RFLAGS_SIZE, buf + size - RFLAGS_SIZE);
00068     rc = rc > 0 ? 0 : rc;
00069     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00070     return rc;
00071 }

```

## 12.27 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_↵ driver/include/bioscara\_hardware\_driver/uI2C.h File Reference

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

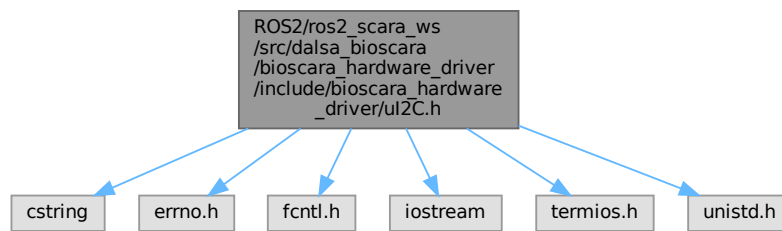
```

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

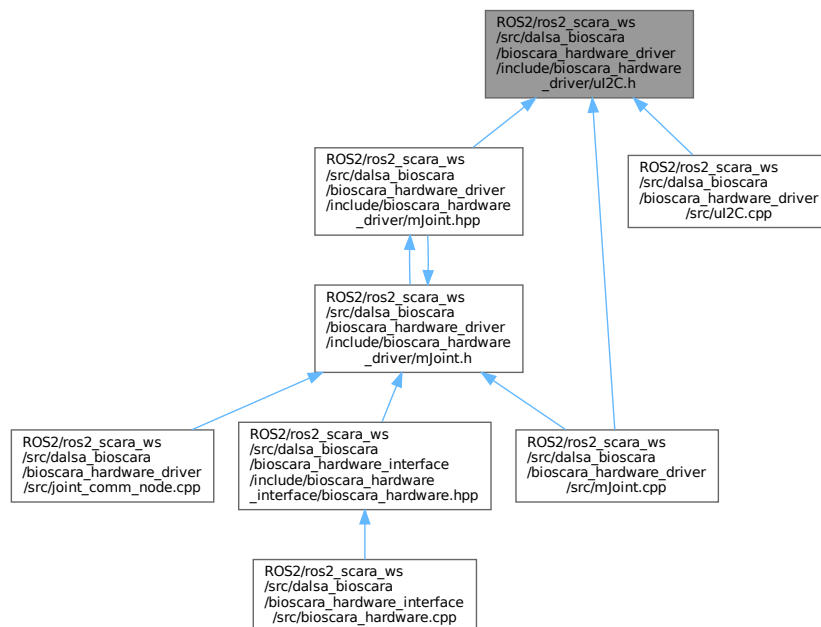
```

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

Include dependency graph for ul2C.h:



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



## Macros

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

## Functions

- int [openI2CDevHandle](#) (const int dev\_addr)  
*Initiates an I2C device on the bus.*
- int [readFromI2CDev](#) (const int dev\_handle, const int [reg](#), char \*buffer, const int data\_length)  
*reads block of bytes from device to buffer*
- int [writeToI2CDev](#) (const int dev\_handle, const int [reg](#), char \*tx\_buffer, const int data\_length, char \*RFLAGS\_buffer)  
*writes block of bytes from buffer to device*
- int [closeI2CDevHandle](#) (int &dev\_handle)  
*close an I2C device on the bus*

## 12.27.1 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

## 12.27.2 Macro Definition Documentation

### 12.27.2.1 ACK

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

### 12.27.2.2 MAX\_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32\_t

### 12.27.2.3 NACK

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

### 12.27.2.4 RFLAGS\_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

## 12.27.3 Function Documentation

### 12.27.3.1 closeI2CDevHandle()

```
int closeI2CDevHandle (
    int & dev_handle )
```

close an I2C device on the bus

#### Parameters

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

#### Returns

0 on OK, negative on error.

### 12.27.3.2 openI2CDevHandle()

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

Initiates an I2C device on the bus.

#### Parameters

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

#### Returns

the device handle, negative on error.

### 12.27.3.3 readFromI2CDev()

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

reads block of bytes from device to buffer

#### Parameters

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

#### Returns

number of bytes read, negative on error.

### 12.27.3.4 writeToI2CDev()

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

writes block of bytes from buffer to device

#### Parameters

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

**Returns**

0 on OK, negative on error.

**12.28 ul2C.h**

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

```

00001
00028 #ifndef SERIAL_H
00029 #define SERIAL_H
00030 #include <cstring>
00031 #include <errno.h>
00032 #include <fcntl.h>
00033 #include <iostream>
00034 #include <termios.h>
00035 #include <unistd.h>
00036
00037 #define ACK 'O'
00038 #define NACK 'N'
00039
00043 #define RFLAGS_SIZE 1
00044
00048 #define MAX_BUFFER 4 // Bytes
00049
00055 int openI2CDevHandle(const int dev_addr);
00056
00065 int readFromI2CDev(const int dev_handle, const int reg, char *buffer, const int data_length);
00066
00076 int writeToI2CDev(const int dev_handle, const int reg, char *tx_buffer, const int data_length, char
    *RFLAGS_buffer);
00077
00083 int closeI2CDevHandle(int &dev_handle);
00084
00085
00086 #endif

```

**12.29 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/include/bioscara\_hardware\_driver/uPWM.h File Reference**

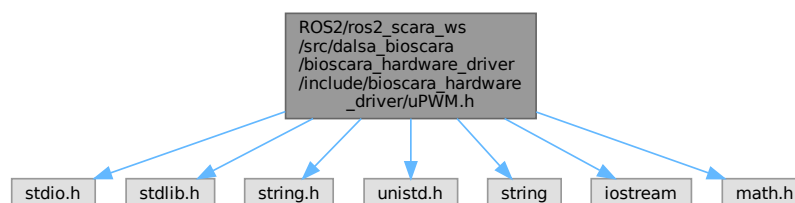
Includes source code for Hardware PWM generation on Raspberry Pi 4.

```

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

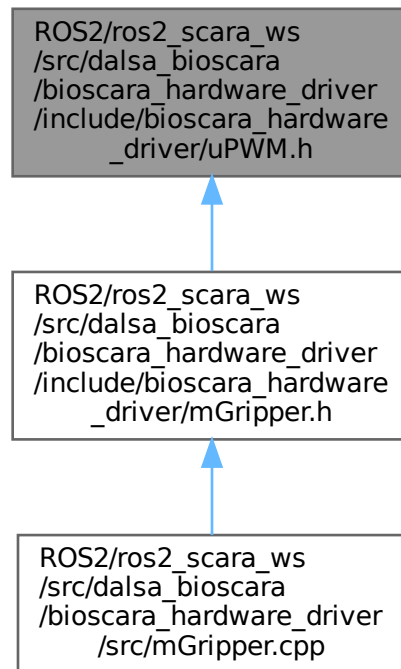
```

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.*

### 12.29.1 Detailed Description

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

#### Author

Sebastian Storz and Bernd Porr, [bernd.porr@glasgow.ac.uk](mailto: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](https://github.com/berndporr/rpi_pwm/blob/main/rpi_pwm.h) and slightly modified it.

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

**Copyright**

Copyright (c) 2025

## 12.30 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

```

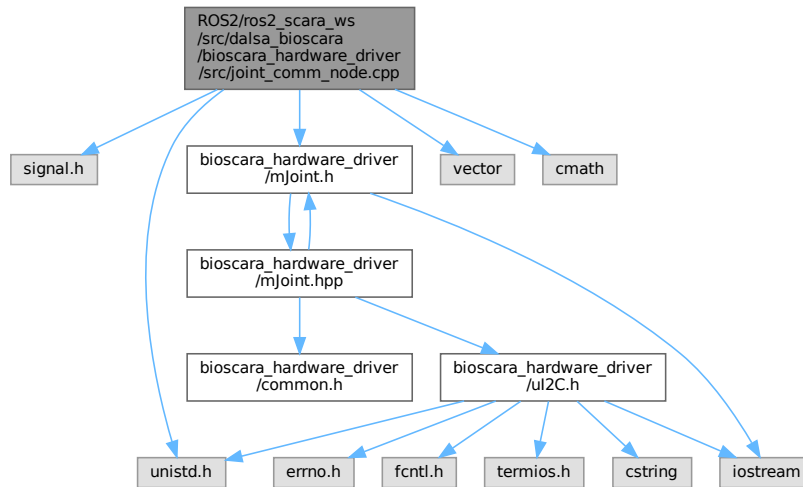
## 12.31 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/joint\_comm\_node.cpp File Reference ↩

```

#include <signal.h>
#include <unistd.h>
#include "bioscara_hardware_driver/mJoint.h"
#include <vector>
#include <cmath>

```

Include dependency graph for joint\_comm\_node.cpp:



## Functions

- void [INT\\_handler](#) (int s)
- int [main](#) (int argc, char \*\*argv)

## Variables

- [Joint J1](#) ("j1", 0x11, 35, -3.04647, 3.04647)
- [Joint J2](#) ("j2", 0x12, -2 \*[M\\_PI](#)/0.004, 0.338, 0.0)
- [Joint J3](#) ("j3", 0x13, 24, -2.62672, 2.62672)
- [Joint J4](#) ("j4", 0x14, 12, -3.01069, 3.01069)

## 12.31.1 Function Documentation

### 12.31.1.1 INT\_handler()

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

### 12.31.1.2 main()

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

## 12.31.2 Variable Documentation

### 12.31.2.1 J1

```
Joint J1("j1", 0x11, 35, -3.04647, 3.04647) (
    "j1" ,
    0x11 ,
    35 ,
    -3. 04647,
    3. 04647 )
```

### 12.31.2.2 J2

```
Joint J2("j2", 0x12, -2 *M_PI/0.004, 0.338, 0.0) (
    "j2" ,
    0x12 ,
    -2 *M_PI/0. 004,
    0. 338,
    0. 0 )
```

### 12.31.2.3 J3

```
Joint J3("j3", 0x13, 24, -2.62672, 2.62672) (
    "j3" ,
    0x13 ,
    24 ,
    -2. 62672,
    2. 62672 )
```

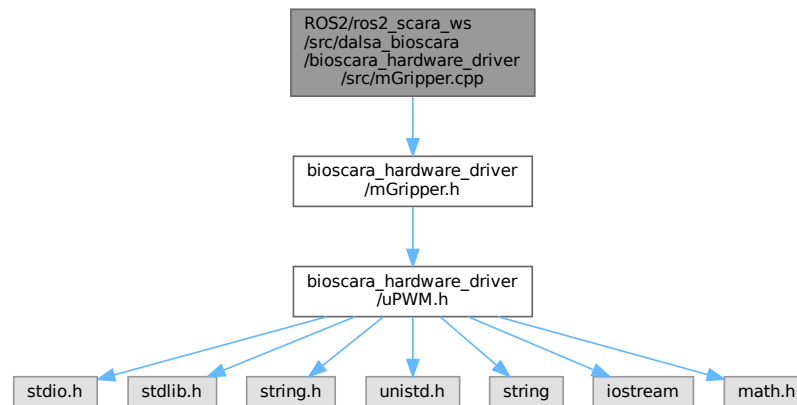
### 12.31.2.4 J4

```
Joint J4("j4", 0x14, 12, -3.01069, 3.01069) (
    "j4" ,
    0x14 ,
    12 ,
    -3. 01069,
    3. 01069 )
```

## 12.32 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/mGripper.cpp File Reference

```
#include "bioscara_hardware_driver/mGripper.h"
```

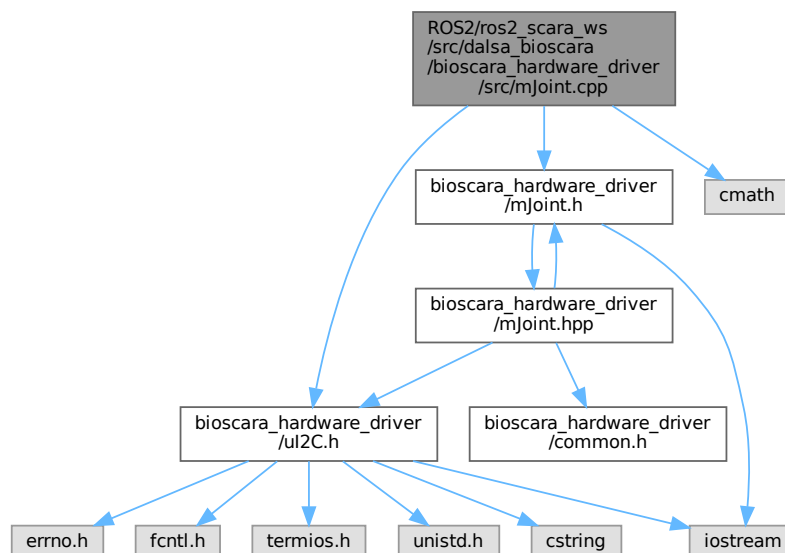
Include dependency graph for mGripper.cpp:



### 12.33 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 <cmath>
```

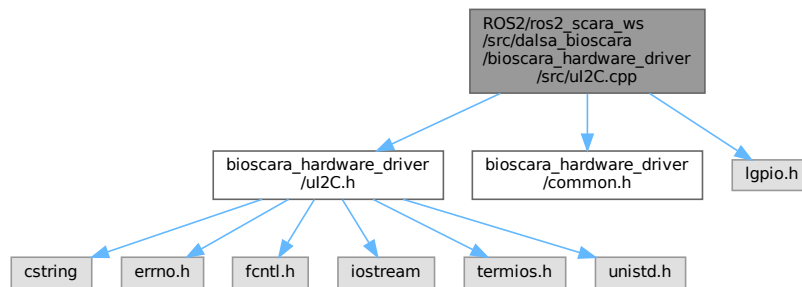
Include dependency graph for mJoint.cpp:



## 12.34 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara硬件\_driver/src/ul2C.cpp File Reference

```
#include "bioscara硬件_driver/ul2C.h"
#include "bioscara硬件_driver/common.h"
#include <lgpio.h>
```

Include dependency graph for ul2C.cpp:



### Functions

- int [openI2CDevHandle](#) (const int dev\_addr)  
*Initiates an I2C device on the bus.*
- int [readFromI2CDev](#) (const int dev\_handle, const int [reg](#), char \*buffer, const int data\_length)  
*reads block of bytes from device to buffer*
- int [writeToI2CDev](#) (const int dev\_handle, const int [reg](#), char \*tx\_buffer, const int data\_length, char \*RFLAGS\_buffer)  
*writes block of bytes from buffer to device*
- int [closeI2CDevHandle](#) (int &dev\_handle)  
*close an I2C device on the bus*

### 12.34.1 Function Documentation

#### 12.34.1.1 closeI2CDevHandle()

```
int closeI2CDevHandle (
    int & dev_handle )
```

close an I2C device on the bus

#### Parameters

<i>dev_handle</i>	device handle obtained from <a href="#">openI2CDevHandle</a>
-------------------	--

**Returns**

0 on OK, negative on error.

**12.34.1.2 openI2CDevHandle()**

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

Initiates an I2C device on the bus.

**Parameters**

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

**Returns**

the device handle, negative on error.

**12.34.1.3 readFromI2CDev()**

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

reads block of bytes from device to buffer

**Parameters**

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

**Returns**

number of bytes read, negative on error.

**12.34.1.4 writeToI2CDev()**

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

writes block of bytes from buffer to device



## Parameters

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

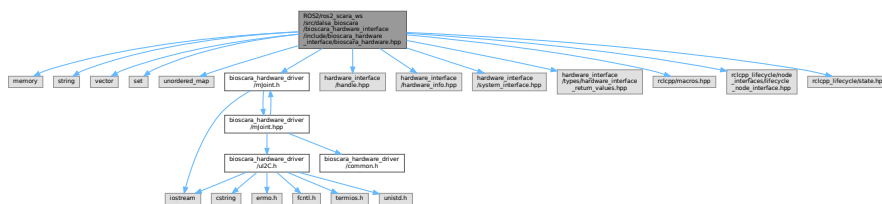
## Returns

0 on OK, negative on error.

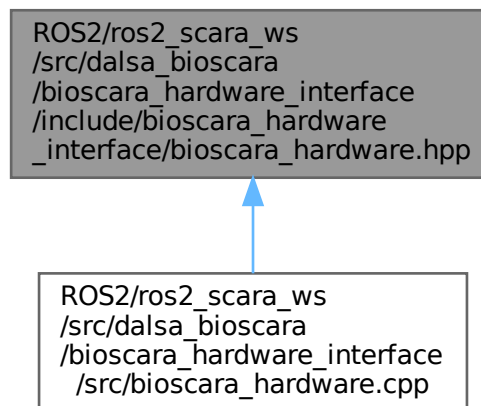
## 12.35 ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_↵ interface/include/bioscara\_hardware\_interface/bioscara\_↵ hardware.hpp File Reference

```
#include <memory>
#include <string>
#include <vector>
#include <set>
#include <unordered_map>
#include "bioscara_hardware_driver/mJoint.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 [bioscaraHardwareInterface::BioscaraHardwareInterface](#)  
*The bioscara hardware interface class.*
- struct [bioscaraHardwareInterface::BioscaraHardwareInterface::joint\\_homing\\_config\\_t](#)  
*configuration structure holding the passed homing parameters from the ros2\_control urdf*
- struct [bioscaraHardwareInterface::BioscaraHardwareInterface::joint\\_config\\_t](#)  
*configuration structure holding the passed parameters from the ros2\_control urdf*

## Namespaces

- namespace [bioscaraHardwareInterface](#)

## Variables

- constexpr char [bioscaraHardwareInterface::HW\\_IF\\_HOME](#) [] = "home"

## 12.36 bioscaraHardware.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 #include <set>
00022 #include <unordered_map>
00023
00024 #include "bioscara_hardware_driver/mJoint.h"
00025
00026 #include "hardware_interface/handle.hpp"
00027 #include "hardware_interface/hardware_info.hpp"
00028 #include "hardware_interface/system_interface.hpp"
00029 #include "hardware_interface/types/hardware_interface_return_values.hpp"
00030 #include "rclcpp/macros.hpp"
00031 #include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
00032 #include "rclcpp_lifecycle/state.hpp"
00033
00034 namespace bioscara_hardware_interface
00035 {
00036     constexpr char HW_IF_HOME[] = "home";
00037
00051     class BioscaraHardwareInterface : public hardware_interface::SystemInterface
00052     {
00053     public:
00054         RCLCPP_SHARED_PTR_DEFINITIONS(BioscaraHardwareInterface)
00055
00056         hardware_interface::CallbackReturn on_init(
00057             const hardware_interface::HardwareComponentInterfaceParams &params) override;
00058
00068         hardware_interface::CallbackReturn on_shutdown(
00069             const rclcpp_lifecycle::State &previous_state) override;
00070
00078         hardware_interface::CallbackReturn on_configure(
00079             const rclcpp_lifecycle::State &previous_state) override;
00080
00089         hardware_interface::CallbackReturn on_cleanup(
00090             const rclcpp_lifecycle::State &previous_state) override;
00091
00106         hardware_interface::CallbackReturn on_activate(
00107             const rclcpp_lifecycle::State &previous_state) override;
00108
00117         hardware_interface::CallbackReturn on_deactivate(
00118             const rclcpp_lifecycle::State &previous_state) override;
00119
00140         hardware_interface::return_type read(
00141             const rclcpp::Time &time,
00142             const rclcpp::Duration &period) override;
00143
00165         hardware_interface::return_type write(
00166             const rclcpp::Time &time,
00167             const rclcpp::Duration &period) override;
00168
00196         hardware_interface::return_type prepare_command_mode_switch(
00197             const std::vector<std::string> &start_interfaces,
00198             const std::vector<std::string> &stop_interfaces) override;
00199
00227         hardware_interface::CallbackReturn on_error(
00228             const rclcpp_lifecycle::State &previous_state) override;
00229
00230     private:
00237         struct joint_homing_config_t
00238         {
00239             float speed = 0;
00240             u_int8_t threshold = 10;
00241             u_int8_t current = 10;
00242             float acceleration = 0.01;
00243         };
00244
00251         struct joint_config_t
00252         {
00253             int i2c_address;
00254             float reduction = 1;
00255             float min;
00256             float max;
00257             u_int8_t drive_current;
00258             u_int8_t hold_current;
00259             u_int8_t stall_threshold;
00260             float max_velocity;
00261             float max_acceleration;
00262             joint_homing_config_t homing;
00263         };
00264
00272         std::unordered_map<std::string, Joint> _joints;

```



# Index

- [\\_home](#)
  - [Joint](#), [42](#)
- [\\_joint\\_cfg](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface](#), [write](#), [32](#)
  - [33](#)
- [\\_joint\\_command\\_modes](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface](#), [drive\\_current](#), [56](#)
  - [33](#)
- [\\_joints](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface](#), [i2c\\_address](#), [57](#)
  - [33](#)
- [~Joint](#)
  - [Joint](#), [42](#)
- [~RPI\\_PWM](#)
  - [RPI\\_PWM](#), [59](#)
- [acceleration](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface::joint\\_homing\\_config\\_t](#), [58](#)
- [ACK](#)
  - [joint.h](#), [67](#)
  - [ul2C.h](#), [91](#)
- [ACTUATOR2JOINT](#)
  - [mJoint.h](#), [84](#)
- [address](#)
  - [Joint](#), [54](#)
- [ADR](#)
  - [configuration.h](#), [64](#)
- [ANGLEMOVED](#)
  - [Joint](#), [41](#)
  - [joint.h](#), [68](#)
- [Arduino/joint/configuration.h](#), [63](#), [65](#)
- [Arduino/joint/joint.h](#), [65](#), [70](#)
- [Arduino/joint/joint.ino](#), [71](#)
- [bioscara](#), [19](#)
  - [generate\\_launch\\_description](#), [19](#)
- [bioscara\\_hardware\\_interface](#), [19](#)
  - [HW\\_IF\\_HOME](#), [19](#)
- [bioscara\\_hardware\\_interface::BioscaraHardwareInterface](#), [25](#)
  - [\\_joint\\_cfg](#), [33](#)
  - [\\_joint\\_command\\_modes](#), [33](#)
  - [\\_joints](#), [33](#)
  - [on\\_activate](#), [27](#)
  - [on\\_cleanup](#), [28](#)
  - [on\\_configure](#), [28](#)
  - [on\\_deactivate](#), [29](#)
  - [on\\_error](#), [29](#)
  - [on\\_init](#), [30](#)
  - [on\\_shutdown](#), [30](#)
  - [prepare\\_command\\_mode\\_switch](#), [31](#)
  - [read](#), [31](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface::joint\\_config\\_t](#), [56](#)
  - [hold\\_current](#), [56](#)
  - [homing](#), [57](#)
  - [max](#), [57](#)
  - [max\\_acceleration](#), [57](#)
  - [max\\_velocity](#), [57](#)
  - [min](#), [57](#)
  - [reduction](#), [57](#)
  - [stall\\_threshold](#), [57](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface::joint\\_homing\\_config\\_t](#), [58](#)
  - [acceleration](#), [58](#)
  - [current](#), [58](#)
  - [speed](#), [58](#)
  - [threshold](#), [58](#)
- [blocking\\_handler](#)
  - [joint.ino](#), [72](#)
- [checkCom](#)
  - [Joint](#), [43](#)
- [CHECKORIENTATION](#)
  - [Joint](#), [41](#)
  - [joint.h](#), [69](#)
- [checkOrientation](#)
  - [Joint](#), [43](#)
- [chippath](#)
  - [RPI\\_PWM](#), [61](#)
- [CLEARSTALL](#)
  - [Joint](#), [41](#)
  - [joint.h](#), [69](#)
- [closeI2CDevHandle](#)
  - [ul2C.cpp](#), [101](#)
  - [ul2C.h](#), [92](#)
- [common.h](#)
  - [DUMP\\_BUFFER](#), [80](#)
- [configuration.h](#)
  - [ADR](#), [64](#)
  - [MAXACCEL](#), [64](#)
  - [MAXVEL](#), [64](#)
- [current](#)
  - [bioscara\\_hardware\\_interface::BioscaraHardwareInterface::joint\\_homing\\_config\\_t](#), [58](#)
- [current\\_b\\_cmd](#)

- Joint, [54](#)
- data\_files
  - setup, [21](#)
- DEG2RAD
  - mJoint.h, [84](#)
- deinit
  - Gripper, [35](#)
  - Joint, [43](#)
- description
  - setup, [21](#)
- disable
  - Gripper, [35](#)
  - Joint, [43](#)
  - RPI\_PWM, [59](#)
- disableCL
  - Joint, [44](#)
- DISABLECLOSEDLOOP
  - Joint, [41](#)
  - joint.h, [69](#)
- DISABLEPID
  - Joint, [41](#)
  - joint.h, [69](#)
- DISABLESTALLGUARD
  - Joint, [41](#)
  - joint.h, [69](#)
- display, [20](#)
  - generate\_launch\_description, [20](#)
- docs/DOCS\_README.md, [76](#)
- Documentation, [1](#)
- drive\_current
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t, [56](#)
- DUMP\_BUFFER
  - common.h, [80](#)
  - joint.h, [67](#)
- enable
  - Gripper, [36](#)
  - Joint, [44](#)
  - RPI\_PWM, [59](#)
- ENABLECLOSEDLOOP
  - Joint, [41](#)
  - joint.h, [69](#)
- ENABLEPID
  - Joint, [41](#)
  - joint.h, [69](#)
- ENABLESTALLGUARD
  - Joint, [41](#)
  - joint.h, [68](#)
- enableStallguard
  - Joint, [44](#)
- entry\_points
  - setup, [21](#)
- flags
  - Joint, [54](#)
- gazebo, [20](#)
  - generate\_launch\_description, [20](#)
- generate\_launch\_description
  - bioscara, [19](#)
  - display, [20](#)
  - gazebo, [20](#)
  - test\_joint\_trajectory\_controller, [23](#)
- getCurrentBCmd
  - Joint, [45](#)
- GETDRIVERRPM
  - Joint, [41](#)
  - joint.h, [68](#)
- GETENCODERRPM
  - Joint, [41](#)
  - joint.h, [69](#)
- getFlags
  - Joint, [45](#)
- getHomingOffset
  - Joint, [45](#)
- GETMOTORSTATE
  - Joint, [41](#)
  - joint.h, [68](#)
- GETPIDERROR
  - Joint, [41](#)
  - joint.h, [69](#)
- getPosition
  - Joint, [45](#)
- getVelocity
  - Joint, [46](#)
- Gripper, [34](#)
  - deinit, [35](#)
  - disable, [35](#)
  - enable, [35](#)
  - Gripper, [35](#)
  - init, [36](#)
  - pwm, [37](#)
  - setPosition, [36](#)
- handle
  - Joint, [55](#)
- hold\_current
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t, [56](#)
- HOME
  - Joint, [41](#)
  - joint.h, [69](#)
- home
  - Joint, [46](#)
- HOMEOFFSET
  - Joint, [41](#)
  - joint.h, [69](#)
- homing
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t, [57](#)
- HW\_IF\_HOME
  - bioscara\_hardware\_interface, [19](#)
- i2c\_address
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_config\_t, [57](#)

- init
  - Gripper, 36
  - Joint, 46
- install\_requires
  - setup, 21
- INT\_handler
  - joint\_comm\_node.cpp, 98
- isBusy
  - Joint, 46
- isEnabled
  - Joint, 47
- isHomed
  - Joint, 47
- isStalled
  - Joint, 47
- J1
  - joint\_comm\_node.cpp, 99
- J2
  - joint\_comm\_node.cpp, 99
- J3
  - joint\_comm\_node.cpp, 99
- J4
  - joint.ino, 72
  - joint\_comm\_node.cpp, 99
- Joint, 37
  - \_home, 42
  - ~Joint, 42
  - address, 54
  - ANGLEMOVED, 41
  - checkCom, 43
  - CHECKORIENTATION, 41
  - checkOrientation, 43
  - CLEARSTALL, 41
  - current\_b\_cmd, 54
  - deinit, 43
  - disable, 43
  - disableCL, 44
  - DISABLECLOSEDLOOP, 41
  - DISABLEPID, 41
  - DISABLESTALLGUARD, 41
  - enable, 44
  - ENABLECLOSEDLOOP, 41
  - ENABLEPID, 41
  - ENABLESTALLGUARD, 41
  - enableStallguard, 44
  - flags, 54
  - getCurrentBCmd, 45
  - GETDRIVERRPM, 41
  - GETENCODERRPM, 41
  - getFlags, 45
  - getHomingOffset, 45
  - GETMOTORSTATE, 41
  - GETPIDERROR, 41
  - getPosition, 45
  - getVelocity, 46
  - handle, 55
  - HOME, 41
  - home, 46
  - HOMEOFFSET, 41
  - init, 46
  - isBusy, 46
  - isEnabled, 47
  - isHomed, 47
  - isStalled, 47
  - Joint, 41
  - max, 55
  - min, 55
  - MOVEANGLE, 41
  - MOVESTEPS, 41
  - moveSteps, 47
  - MOVETOANGLE, 41
  - MOVETOEND, 41
  - name, 55
  - NONE, 41
  - offset, 55
  - PING, 41
  - postHoming, 49
  - printInfo, 49
  - read, 49
  - reduction, 55
  - RUNCOTINOUS, 41
  - SETBRAKEMODE, 41
  - setBrakeMode, 50
  - SETCONTROLTHRESHOLD, 41
  - SETCURRENT, 41
  - setDriveCurrent, 50
  - SETHOLDCURRENT, 41
  - setHoldCurrent, 50
  - setHomingOffset, 51
  - SETMAXACCELERATION, 41
  - setMaxAcceleration, 51
  - SETMAXDECELERATION, 41
  - SETMAXVELOCITY, 41
  - setMaxVelocity, 51
  - setPosition, 52
  - SETRPM, 41
  - SETUP, 41
  - setVelocity, 52
  - startHoming, 52
  - STOP, 41
  - stop, 53
  - stp\_reg\_t, 40
  - wait\_while\_busy, 53
  - write, 53
- joint.h
  - ACK, 67
  - ANGLEMOVED, 68
  - CHECKORIENTATION, 69
  - CLEARSTALL, 69
  - DISABLECLOSEDLOOP, 69
  - DISABLEPID, 69
  - DISABLESTALLGUARD, 69
  - DUMP\_BUFFER, 67
  - ENABLECLOSEDLOOP, 69
  - ENABLEPID, 69
  - ENABLESTALLGUARD, 68

- GETDRIVERRPM, [68](#)
- GETENCODERRPM, [69](#)
- GETMOTORSTATE, [68](#)
- GETPIDERROR, [69](#)
- HOME, [69](#)
- HOMEOFFSET, [69](#)
- MAX\_BUFFER, [68](#)
- MOVEANGLE, [68](#)
- MOVESTEPS, [68](#)
- MOVETOANGLE, [68](#)
- MOVETOEND, [69](#)
- NACK, [68](#)
- PING, [68](#)
- readValue, [69](#)
- RFLAGS\_SIZE, [68](#)
- RUNCOTINOUS, [68](#)
- SETBRAKEMODE, [69](#)
- SETCONTROLTHRESHOLD, [69](#)
- SETCURRENT, [68](#)
- SETHOLDCURRENT, [68](#)
- SETMAXACCELERATION, [68](#)
- SETMAXDECELERATION, [68](#)
- SETMAXVELOCITY, [68](#)
- SETRPM, [68](#)
- SETUP, [68](#)
- STOP, [69](#)
- stp\_reg\_t, [68](#)
- writeValue, [69](#)
- joint.ino
  - blocking\_handler, [72](#)
  - J4, [72](#)
  - loop, [73](#)
  - non\_blocking\_handler, [73](#)
  - receiveEvent, [73](#)
  - reg, [75](#)
  - requestEvent, [75](#)
  - rx\_buf, [75](#)
  - rx\_data\_ready, [75](#)
  - rx\_length, [75](#)
  - setup, [75](#)
  - stepper, [75](#)
  - tx\_buf, [76](#)
  - tx\_data\_ready, [76](#)
  - tx\_length, [76](#)
- JOINT2ACTUATOR
  - mJoint.h, [85](#)
- joint\_comm\_node.cpp
  - INT\_handler, [98](#)
  - J1, [99](#)
  - J2, [99](#)
  - J3, [99](#)
  - J4, [99](#)
  - main, [98](#)
- license
  - setup, [21](#)
- loop
  - joint.ino, [73](#)
- M\_PI
  - mJoint.h, [85](#)
- main
  - joint\_comm\_node.cpp, [98](#)
- maintainer
  - setup, [21](#)
- maintainer\_email
  - setup, [21](#)
- max
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_conf, [57](#)
  - Joint, [55](#)
- max\_acceleration
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_conf, [57](#)
- MAX\_BUFFER
  - joint.h, [68](#)
  - ul2C.h, [91](#)
- max\_velocity
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_conf, [57](#)
- MAXACCEL
  - configuration.h, [64](#)
- MAXVEL
  - configuration.h, [64](#)
- min
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_conf, [57](#)
  - Joint, [55](#)
- mJoint.h
  - ACTUATOR2JOINT, [84](#)
  - DEG2RAD, [84](#)
  - JOINT2ACTUATOR, [85](#)
  - M\_PI, [85](#)
  - RAD2DEG, [85](#)
- MOVEANGLE
  - Joint, [41](#)
  - joint.h, [68](#)
- MOVESTEPS
  - Joint, [41](#)
  - joint.h, [68](#)
- moveSteps
  - Joint, [47](#)
- MOVETOANGLE
  - Joint, [41](#)
  - joint.h, [68](#)
- MOVETOEND
  - Joint, [41](#)
  - joint.h, [69](#)
- NACK
  - joint.h, [68](#)
  - ul2C.h, [92](#)
- name
  - Joint, [55](#)
  - setup, [21](#)
- non\_blocking\_handler
  - joint.ino, [73](#)
- NONE



- Joint, 41
- offset
  - Joint, 55
- on\_activate
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 27
- on\_cleanup
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, Joint, 28
- on\_configure
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 28
- on\_deactivate
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, joint.h, 29
- on\_error
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 29
- on\_init
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 30
- on\_shutdown
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 30
- openI2CDevHandle
  - ul2C.cpp, 102
  - ul2C.h, 92
- package\_name
  - setup, 21
- packages
  - setup, 21
- per
  - RPI\_PWM, 61
- PING
  - Joint, 41
  - joint.h, 68
- postHoming
  - Joint, 49
- prepare\_command\_mode\_switch
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 31
- printInfo
  - Joint, 49
- pwm
  - Gripper, 37
- pwmpath
  - RPI\_PWM, 61
- RAD2DEG
  - mJoint.h, 85
- read
  - bioscara\_hardware\_interface::BioscaraHardwareInterface, 31
  - Joint, 49
- readFromI2CDev
  - ul2C.cpp, 102
  - ul2C.h, 93
- README, 3, 5, 7
- readValue
  - joint.h, 69
- receiveEvent
  - joint.ino, 73
- reduction
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_confir
- reg
  - joint.ino, 75
- requestEvent
  - joint.ino, 75
- RFLAGS\_SIZE
- ul2C.h, 92
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_bringup/launch/bioscara
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_bringup/launch/test\_jo
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_bringup/README.md,
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/bioscara\_c
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/launch/disp
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/launch/gaz
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/README.
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/setup.py,
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/test/test\_c
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/test/test\_fl
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_description/test/test\_p
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/includ
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/includ
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/includ
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/includ
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/includ
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/jo
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/m
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/m
- ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_driver/src/u

ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/
 

- include/bioscara\_hardware\_interface/bioscara\_hardware.hpp, 103, 104
- joint.h, 68

ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/bioscara\_hardware\_interface/
 

- SETMAXVELOCITY.cpp, 106
- Joint, 41
- joint.h, 68
- setMaxVelocity
  - Joint, 51
- setPeriod
  - RPI\_PWM, 60
- setPosition
  - Gripper, 36
  - Joint, 52
- SETRPM
  - Joint, 41
  - joint.h, 68
- SETUP
  - Joint, 41
  - joint.h, 68
- setup, 20
  - data\_files, 21
  - description, 21
  - entry\_points, 21
  - install\_requires, 21
  - joint.ino, 75
  - license, 21
  - maintainer, 21
  - maintainer\_email, 21
  - name, 21
  - package\_name, 21
  - packages, 21
  - tests\_require, 21
  - version, 22
  - zip\_safe, 22
- setVelocity
  - Joint, 52
- speed
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_homi, 58
- stall\_threshold
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_conf, 57
- start
  - RPI\_PWM, 60
- startHoming
  - Joint, 52
- stepper
  - joint.ino, 75
- STOP
  - Joint, 41
  - joint.h, 69
- stop
  - Joint, 53
  - RPI\_PWM, 60
- stp\_reg\_t
  - Joint, 40
  - joint.h, 68
- test\_copyright, 22
  - test\_copyright, 22

ROS2/ros2\_scara\_ws/src/dalsa\_bioscara/README.md, 77

RPI\_PWM, 59
 

- ~RPI\_PWM, 59
- chippath, 61
- disable, 59
- enable, 59
- per, 61
- pwmpath, 61
- setDutyCycle, 59
- setDutyCycleNS, 60
- setPeriod, 60
- start, 60
- stop, 60
- writeSYS, 60

RUNCOTINOUS
 

- Joint, 41
- joint.h, 68

rx\_buf
 

- joint.ino, 75

rx\_data\_ready
 

- joint.ino, 75

rx\_length
 

- joint.ino, 75

SETBRAKEMODE
 

- Joint, 41
- joint.h, 69

setBrakeMode
 

- Joint, 50

SETCONTROLTHRESHOLD
 

- Joint, 41
- joint.h, 69

SETCURRENT
 

- Joint, 41
- joint.h, 68

setDriveCurrent
 

- Joint, 50

setDutyCycle
 

- RPI\_PWM, 59

setDutyCycleNS
 

- RPI\_PWM, 60

SETHOLDCURRENT
 

- Joint, 41
- joint.h, 68

setHoldCurrent
 

- Joint, 50

setHomingOffset
 

- Joint, 51

SETMAXACCELERATION
 

- Joint, 41
- joint.h, 68

setMaxAcceleration
 

- Joint, 51

SETMAXDECELERATION

- test\_flake8, [22](#)
  - test\_flake8, [22](#)
- test\_joint\_trajectory\_controller, [22](#)
  - generate\_launch\_description, [23](#)
- test\_pep257, [23](#)
  - test\_pep257, [23](#)
- tests\_require
  - setup, [21](#)
- threshold
  - bioscara\_hardware\_interface::BioscaraHardwareInterface::joint\_homing\_config\_t,  
[58](#)
- Todo List, [9](#)
- tx\_buf
  - joint.ino, [76](#)
- tx\_data\_ready
  - joint.ino, [76](#)
- tx\_length
  - joint.ino, [76](#)
- ul2C.cpp
  - closeI2CDevHandle, [101](#)
  - openI2CDevHandle, [102](#)
  - readFromI2CDev, [102](#)
  - writeToI2CDev, [102](#)
- ul2C.h
  - ACK, [91](#)
  - closeI2CDevHandle, [92](#)
  - MAX\_BUFFER, [91](#)
  - NACK, [92](#)
  - openI2CDevHandle, [92](#)
  - readFromI2CDev, [93](#)
  - RFLAGS\_SIZE, [92](#)
  - writeToI2CDev, [93](#)
- version
  - setup, [22](#)
- wait\_while\_busy
  - Joint, [53](#)
- write
  - bioscara\_hardware\_interface::BioscaraHardwareInterface,  
[32](#)
  - Joint, [53](#)
- writeSYS
  - RPI\_PWM, [60](#)
- writeToI2CDev
  - ul2C.cpp, [102](#)
  - ul2C.h, [93](#)
- writeValue
  - joint.h, [69](#)
- zip\_safe
  - setup, [22](#)