Bioscara

Generated by Doxygen 1.9.8

1 Bioscara C++ API Documentation 1

2 Namespace Index 3

2.1 Namespace List . . . 3

3 Hierarchical Index 5

3.1 Class Hierarchy . . . 5

4 Class Index 7

4.1 Class List . . . 7

5 File Index 9

5.1 File List . . . 9

6 Namespace Documentation 11

6.1 bioscara_hardware_drivers Namespace Reference . . . 11

6.1.1 Enumeration Type Documentation . . . 11

6.1.1.1 err_type_t . . . 11

6.1.2 Function Documentation . . . 12

6.1.2.1 error_to_string() . . . 12

6.2 bioscara_hardware_interfaces Namespace Reference . . . 12

6.2.1 Variable Documentation . . . 12

6.2.1.1 HW_IF_HOME . . . 12

6.3 bioscara_joint_firmware Namespace Reference . . . 13

6.3.1 Detailed Description . . . 14

6.3.2 Enumeration Type Documentation . . . 14

6.3.2.1 stp_reg_t . . . 14

6.3.3 Function Documentation . . . 15

6.3.3.1 blocking_handler() . . . 15

6.3.3.2 loop() . . . 15

6.3.3.3 non_blocking_handler() . . . 16

6.3.3.4 readValue() . . . 16

6.3.3.5 receiveEvent() . . . 16

6.3.3.6 requestEvent() . . . 17

6.3.3.7 set_flags_for_blocking_handler() . . . 17

6.3.3.8 setup() . . . 17

6.3.3.9 stall_threshold() . . . 17

6.3.3.10 writeValue() . . . 18

6.3.4 Variable Documentation . . . 18

6.3.4.1 blk_reg . . . 18

6.3.4.2 reg . . . 18

6.3.4.3 rx_buf . . . 18

6.3.4.4 rx_data_ready . . . 18

6.3.4.5 rx_length . . . 18

6.3.4.6 stepper . . . 19
6.3.4.7 tx_buf . . . 19
6.3.4.8 tx_length . . . 19
6.4 bioscara_rviz_plugin Namespace Reference . . . 19
6.5 single_trigger_controller Namespace Reference . . . 19
6.5.1 Typedef Documentation . . . 19
6.5.1.1 CallbackReturn . . . 19
6.5.1.2 CmdType . . . 20
6.5.1.3 InterfacesNames . . . 20
6.5.1.4 MapOfReferencesToCommandInterfaces . . . 20
6.5.1.5 MapOfReferencesToStateInterfaces . . . 20
6.5.1.6 StateInterfaces . . . 20
6.5.1.7 StateType . . . 20

7 Class Documentation **21**

7.1 bioscara_hardware_drivers::BaseGripper Class Reference . . . 21
7.1.1 Detailed Description . . . 22
7.1.2 Constructor & Destructor Documentation . . . 22
7.1.2.1 BaseGripper() . . . 22
7.1.2.2 ~BaseGripper() . . . 23
7.1.3 Member Function Documentation . . . 23
7.1.3.1 deinit() . . . 23
7.1.3.2 disable() . . . 24
7.1.3.3 enable() . . . 24
7.1.3.4 getPosition() . . . 24
7.1.3.5 init() . . . 25
7.1.3.6 retrieve_last_position() . . . 25
7.1.3.7 save_last_position() . . . 25
7.1.3.8 setOffset() . . . 25
7.1.3.9 setPosition() . . . 26
7.1.3.10 setReduction() . . . 26
7.1.4 Member Data Documentation . . . 26
7.1.4.1 _backup_init_pos . . . 26
7.1.4.2 _max . . . 26
7.1.4.3 _min . . . 26
7.1.4.4 _new_cmd_time . . . 27
7.1.4.5 _offset . . . 27
7.1.4.6 _pos . . . 27
7.1.4.7 _pos_get . . . 27
7.1.4.8 _reduction . . . 27
7.2 bioscara_hardware_drivers::BaseJoint Class Reference . . . 27
7.2.1 Detailed Description . . . 30

7.2.2 Member Enumeration Documentation . . . 30
7.2.2.1 stp_reg_t . . . 30
7.2.3 Constructor & Destructor Documentation . . . 31
7.2.3.1 BaseJoint() . . . 31
7.2.3.2 ~BaseJoint() . . . 31
7.2.4 Member Function Documentation . . . 31
7.2.4.1 _home() . . . 31
7.2.4.2 checkOrientation() . . . 32
7.2.4.3 deinit() . . . 32
7.2.4.4 disable() . . . 32
7.2.4.5 disableCL() . . . 33
7.2.4.6 enable() . . . 33
7.2.4.7 enableStallguard() . . . 33
7.2.4.8 getCurrentBCmd() . . . 34
7.2.4.9 getFlags() [1/2] . . . 34
7.2.4.10 getFlags() [2/2] . . . 34
7.2.4.11 getPosition() . . . 35
7.2.4.12 getVelocity() . . . 35
7.2.4.13 home() . . . 35
7.2.4.14 init() . . . 36
7.2.4.15 isBusy() . . . 36
7.2.4.16 isEnabled() . . . 36
7.2.4.17 isHomed() . . . 37
7.2.4.18 isStalled() . . . 37
7.2.4.19 moveSteps() . . . 37
7.2.4.20 postHoming() . . . 38
7.2.4.21 setBrakeMode() . . . 38
7.2.4.22 setDriveCurrent() . . . 38
7.2.4.23 setHoldCurrent() . . . 39
7.2.4.24 setMaxAcceleration() . . . 39
7.2.4.25 setMaxVelocity() . . . 39
7.2.4.26 setPosition() . . . 40
7.2.4.27 setVelocity() . . . 40
7.2.4.28 startHoming() . . . 40
7.2.4.29 stop() . . . 41
7.2.4.30 wait_while_busy() . . . 41
7.2.5 Member Data Documentation . . . 41
7.2.5.1 current_b_cmd . . . 41
7.2.5.2 flags . . . 42
7.2.5.3 name . . . 42
7.3 bioscara_hardware_interfaces::BioscaraArmHardwareInterface Class Reference . . . 42
7.3.1 Detailed Description . . . 44

7.3.2 Member Function Documentation . . . 45
7.3.2.1 activate_joint() . . . 45
7.3.2.2 deactivate_joint() . . . 45
7.3.2.3 on_activate() . . . 45
7.3.2.4 on_cleanup() . . . 46
7.3.2.5 on_configure() . . . 46
7.3.2.6 on_deactivate() . . . 47
7.3.2.7 on_error() . . . 47
7.3.2.8 on_init() . . . 48
7.3.2.9 on_shutdown() . . . 49
7.3.2.10 perform_command_mode_switch() . . . 49
7.3.2.11 prepare_command_mode_switch() . . . 50
7.3.2.12 read() . . . 51
7.3.2.13 split_interface_string_to_joint_and_name() . . . 51
7.3.2.14 start_homing() . . . 52
7.3.2.15 stop_homing() . . . 52
7.3.2.16 write() . . . 52
7.3.3 Member Data Documentation . . . 53
7.3.3.1 _joint_cfg . . . 53
7.3.3.2 _joint_command_modes . . . 53
7.3.3.3 _joints . . . 54
7.3.3.4 _new_joint_command_modes . . . 54
7.3.3.5 _ordered_joint_state_interfaces_ptr . . . 54
7.3.3.6 mtx . . . 55
7.4 bioscara_hardware_interfaces::BioscaraGripperHardwareInterface Class Reference . . . 55
7.4.1 Detailed Description . . . 56
7.4.2 Member Function Documentation . . . 56
7.4.2.1 on_activate() . . . 56
7.4.2.2 on_cleanup() . . . 57
7.4.2.3 on_configure() . . . 57
7.4.2.4 on_deactivate() . . . 57
7.4.2.5 on_error() . . . 58
7.4.2.6 on_init() . . . 58
7.4.2.7 on_shutdown() . . . 59
7.4.2.8 read() . . . 59
7.4.2.9 write() . . . 59
7.4.3 Member Data Documentation . . . 60
7.4.3.1 _gripper . . . 60
7.4.3.2 _gripper_cfg . . . 60
7.4.3.3 _last_pos . . . 60
7.4.3.4 _vel . . . 60
7.5 bioscara_rviz_plugin::BioscaraPanel Class Reference . . . 61

7.5.1 Detailed Description . . . 63
7.5.2 Constructor & Destructor Documentation . . . 63
7.5.2.1 BioscaraPanel() . . . 63
7.5.2.2 ~BioscaraPanel() . . . 63
7.5.3 Member Function Documentation . . . 64
7.5.3.1 arm_en_btn_cb . . . 64
7.5.3.2 check_activation_conditions() . . . 64
7.5.3.3 cm_state_callback() . . . 64
7.5.3.4 configure_controller() . . . 64
7.5.3.5 ctrl_en_btn_cb() . . . 65
7.5.3.6 dynamic_joint_state_msg_to_map() . . . 65
7.5.3.7 ensure_jsb_is_active() . . . 65
7.5.3.8 gripper_en_btn_cb . . . 65
7.5.3.9 homing_cmd() . . . 66
7.5.3.10 joint_state_callback() . . . 66
7.5.3.11 named_lcs_msg_to_map() . . . 66
7.5.3.12 onInitialize() . . . 66
7.5.3.13 prepopulate_joint_state_map() . . . 67
7.5.3.14 prepopulate_state_map() . . . 67
7.5.3.15 print_cm_map() . . . 68
7.5.3.16 set_controller_state() . . . 68
7.5.3.17 set_en_btn() . . . 68
7.5.3.18 set_hardware_component_state() . . . 68
7.5.3.19 set_homing_state_label() . . . 69
7.5.3.20 set_state_label() . . . 69
7.5.3.21 switch_controllers() . . . 69
7.5.3.22 target_state_from_current() . . . 70
7.5.3.23 update_homing_grp_state() . . . 70
7.5.3.24 update_homing_state_labels() . . . 70
7.5.3.25 update_state_label_and_btn() . . . 71
7.5.3.26 update_state_labels_and_btns() . . . 71
7.5.4 Member Data Documentation . . . 71
7.5.4.1 cm_state_subsription_ . . . 71
7.5.4.2 configure_controller_client_ . . . 71
7.5.4.3 controller_states_ . . . 72
7.5.4.4 hardware_state_client_ . . . 72
7.5.4.5 hardware_states_ . . . 72
7.5.4.6 homing_publisher_ . . . 72
7.5.4.7 joint_state_subsription_ . . . 72
7.5.4.8 joint_states_ . . . 72
7.5.4.9 node_ . . . 73
7.5.4.10 prune_timer_ . . . 73

7.5.4.11 switch_controller_client_ . . . 73
7.5.4.12 ui_ . . . 73
7.6 bioscara_hardware_drivers::Gripper Class Reference . . . 73
7.6.1 Detailed Description . . . 75
7.6.2 Constructor & Destructor Documentation . . . 75
7.6.2.1 Gripper() . . . 75
7.6.3 Member Function Documentation . . . 75
7.6.3.1 disable() . . . 75
7.6.3.2 enable() . . . 76
7.6.3.3 setPosition() . . . 76
7.6.3.4 setServoPosition() . . . 76
7.6.4 Member Data Documentation . . . 77
7.6.4.1 _freq . . . 77
7.6.4.2 _pwm . . . 77
7.7 bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t Struct Reference 77
7.7.1 Detailed Description . . . 77
7.7.2 Member Data Documentation . . . 78
7.7.2.1 init_pos . . . 78
7.7.2.2 max . . . 78
7.7.2.3 min . . . 78
7.7.2.4 offset . . . 78
7.7.2.5 reduction . . . 78
7.8 bioscara_hardware_drivers::Joint Class Reference . . . 78
7.8.1 Detailed Description . . . 81
7.8.2 Constructor & Destructor Documentation . . . 82
7.8.2.1 Joint() . . . 82
7.8.2.2 ∼Joint() . . . 82
7.8.3 Member Function Documentation . . . 83
7.8.3.1 _home() . . . 83
7.8.3.2 checkCom() . . . 83
7.8.3.3 checkOrientation() . . . 83
7.8.3.4 deinit() . . . 84
7.8.3.5 disableCL() . . . 84
7.8.3.6 enable() . . . 84
7.8.3.7 enableStallguard() . . . 85
7.8.3.8 getFlags() . . . 85
7.8.3.9 getHomingOffset() . . . 85
7.8.3.10 getPosition() . . . 86
7.8.3.11 getVelocity() . . . 86
7.8.3.12 init() . . . 86
7.8.3.13 moveSteps() . . . 87
7.8.3.14 postHoming() . . . 87

7.8.3.15 read() . . . 87
7.8.3.16 setBrakeMode() . . . 88
7.8.3.17 setDriveCurrent() . . . 88
7.8.3.18 setHoldCurrent() . . . 88
7.8.3.19 setHomingOffset() . . . 90
7.8.3.20 setMaxAcceleration() . . . 90
7.8.3.21 setMaxVelocity() . . . 90
7.8.3.22 setPosition() . . . 91
7.8.3.23 setVelocity() . . . 91
7.8.3.24 stop() . . . 91
7.8.3.25 write() . . . 92
7.8.4 Member Data Documentation . . . 92
7.8.4.1 address . . . 92
7.8.4.2 handle . . . 92
7.8.4.3 max . . . 93
7.8.4.4 min . . . 93
7.8.4.5 offset . . . 93
7.8.4.6 reduction . . . 93
7.9 bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t Struct Reference . . . 93
7.9.1 Detailed Description . . . 94
7.9.2 Member Data Documentation . . . 94
7.9.2.1 drive_current . . . 94
7.9.2.2 hold_current . . . 94
7.9.2.3 homing . . . 95
7.9.2.4 i2c_address . . . 95
7.9.2.5 max . . . 95
7.9.2.6 max_acceleration . . . 95
7.9.2.7 max_velocity . . . 95
7.9.2.8 min . . . 96
7.9.2.9 reduction . . . 96
7.9.2.10 stall_threshold . . . 96
7.10 bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t Struct Reference . . . 96
7.10.1 Detailed Description . . . 97
7.10.2 Member Data Documentation . . . 97
7.10.2.1 acceleration . . . 97
7.10.2.2 current . . . 97
7.10.2.3 speed . . . 97
7.10.2.4 threshold . . . 98
7.11 bioscara_joint_firmware::Lowpass Class Reference . . . 98
7.11.1 Detailed Description . . . 98
7.11.2 Constructor & Destructor Documentation . . . 98

7.11.2.1 Lowpass() . . . 98
7.11.3 Member Function Documentation . . . 99
7.11.3.1 resetState() . . . 99
7.11.3.2 updateState() . . . 99
7.11.4 Member Data Documentation . . . 99
7.11.4.1 K . . . 99
7.11.4.2 tau . . . 99
7.11.4.3 Ts . . . 99
7.11.4.4 x . . . 100
7.12 bioscara_hardware_drivers::MockGripper Class Reference . . . 100
7.12.1 Detailed Description . . . 101
7.12.2 Constructor & Destructor Documentation . . . 101
7.12.2.1 MockGripper() . . . 101
7.13 bioscara_hardware_drivers::MockJoint Class Reference . . . 102
7.13.1 Detailed Description . . . 104
7.13.2 Constructor & Destructor Documentation . . . 105
7.13.2.1 MockJoint() . . . 105
7.13.3 Member Function Documentation . . . 105
7.13.3.1 _home() . . . 105
7.13.3.2 checkOrientation() . . . 105
7.13.3.3 disable() . . . 106
7.13.3.4 enable() . . . 106
7.13.3.5 getDeltaT() . . . 106
7.13.3.6 getFlags() . . . 108
7.13.3.7 getPosition() . . . 108
7.13.3.8 getVelocity() . . . 108
7.13.3.9 isHomed() . . . 109
7.13.3.10 setPosition() . . . 109
7.13.3.11 setVelocity() . . . 109
7.13.3.12 stop() . . . 110
7.13.4 Member Data Documentation . . . 110
7.13.4.1 async_start_time . . . 110
7.13.4.2 last_set_position . . . 110
7.13.4.3 last_set_velocity . . . 110
7.13.4.4 op_mode . . . 110
7.13.4.5 q . . . 111
7.13.4.6 qd . . . 111
7.14 bioscara_joint_firmware::MovMax Class Reference . . . 111
7.14.1 Detailed Description . . . 111
7.14.2 Constructor & Destructor Documentation . . . 111
7.14.2.1 MovMax() . . . 111
7.14.3 Member Function Documentation . . . 112

7.14.3.1 updateState() . . . 112
7.14.4 Member Data Documentation . . . 112
7.14.4.1 cb_data . . . 112
7.14.4.2 cb_index . . . 112
7.14.4.3 M . . . 112
7.15 RPI_PWM Class Reference . . . 112
7.15.1 Detailed Description . . . 113
7.15.2 Constructor & Destructor Documentation . . . 113
7.15.2.1 ~RPI_PWM() . . . 113
7.15.3 Member Function Documentation . . . 113
7.15.3.1 disable() . . . 113
7.15.3.2 enable() . . . 114
7.15.3.3 setDutyCycle() . . . 114
7.15.3.4 setDutyCycleNS() . . . 114
7.15.3.5 setPeriod() . . . 114
7.15.3.6 start() . . . 114
7.15.3.7 stop() . . . 115
7.15.3.8 writeSYS() . . . 115
7.15.4 Member Data Documentation . . . 115
7.15.4.1 chippath . . . 115
7.15.4.2 per . . . 115
7.15.4.3 pwmpath . . . 115
7.16 single_trigger_controller::SingleTriggerController Class Reference . . . 115
7.16.1 Detailed Description . . . 117
7.16.2 Constructor & Destructor Documentation . . . 117
7.16.2.1 SingleTriggerController() . . . 117
7.16.3 Member Function Documentation . . . 117
7.16.3.1 apply_command() . . . 117
7.16.3.2 apply_state_value() . . . 118
7.16.3.3 check_if_configured_interfaces_matches_received() . . . 118
7.16.3.4 command_interface_configuration() . . . 119
7.16.3.5 create_map_of_references_to_interfaces() . . . 119
7.16.3.6 get_state_interfaces_names() . . . 119
7.16.3.7 initialize_state_msg() . . . 120
7.16.3.8 on_activate() . . . 120
7.16.3.9 on_configure() . . . 120
7.16.3.10 on_deactivate() . . . 121
7.16.3.11 on_init() . . . 121
7.16.3.12 state_interface_configuration() . . . 121
7.16.3.13 store_command_interface_types() . . . 122
7.16.3.14 store_state_interface_types() . . . 122
7.16.3.15 update() . . . 122

7.16.3.16 update_commands() . . . 122
7.16.3.17 update_dynamic_map_parameters() . . . 123
7.16.3.18 update_states() . . . 123
7.16.4 Member Data Documentation . . . 123
7.16.4.1 command_interface_types_ . . . 123
7.16.4.2 command_interfaces_map_ . . . 123
7.16.4.3 command_subscriber_ . . . 124
7.16.4.4 param_listener_ . . . 124
7.16.4.5 params_ . . . 124
7.16.4.6 realtime_state_publisher_ . . . 124
7.16.4.7 state_interface_types_ . . . 124
7.16.4.8 state_interfaces_map_ . . . 124
7.16.4.9 state_msg_ . . . 125
7.16.4.10 state_publisher_ . . . 125

8 File Documentation **127**

8.1 docs/doxygen/index.md File Reference . . . 127
8.2 lib/joint_firmware/joint/configuration.h File Reference . . . 127
8.2.1 Detailed Description . . . 127
8.2.2 Macro Definition Documentation . . . 128
8.2.2.1 ADR . . . 128
8.2.2.2 MAXACCEL . . . 128
8.2.2.3 MAXVEL . . . 128
8.3 configuration.h . . . 128
8.4 lib/joint_firmware/joint/filters.h File Reference . . . 129
8.4.1 Detailed Description . . . 129
8.5 filters.h . . . 130
8.6 lib/joint_firmware/joint/joint.h File Reference . . . 130
8.6.1 Detailed Description . . . 132
8.6.2 Macro Definition Documentation . . . 132
8.6.2.1 ACK . . . 132
8.6.2.2 DUMP_BUFFER . . . 132
8.6.2.3 MAX_BUFFER . . . 133
8.6.2.4 NACK . . . 133
8.6.2.5 RFLAGS_SIZE . . . 133
8.7 joint.h . . . 133
8.8 lib/joint_firmware/joint/joint.ino File Reference . . . 134
8.8.1 Detailed Description . . . 135
8.8.2 Macro Definition Documentation . . . 136
8.8.2.1 J1 . . . 136
8.8.3 Function Documentation . . . 136
8.8.3.1 loop() . . . 136

8.8.3.2 setup() . . . 136
8.9 lib/joint_firmware/joint/stall.h File Reference . . . 136
8.9.1 Detailed Description . . . 137
8.10 stall.h . . . 137
8.11 lib/ros2_ws/src/bioscara_rviz_plugin/include/bioscara_rviz_plugin/bioscara_panel.hpp File Reference 137
8.12 bioscara_panel.hpp . . . 138
8.13 lib/ros2_ws/src/bioscara_rviz_plugin/src/bioscara_panel.cpp File Reference . . . 139
8.14 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/common.h File Reference . . . 140
8.14.1 Detailed Description . . . 140
8.14.2 Macro Definition Documentation . . . 140
8.14.2.1 DUMP_BUFFER . . . 140
8.15 common.h . . . 141
8.16 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/mBaseJoint.h File Reference . . . 141
8.16.1 Detailed Description . . . 142
8.17 mBaseJoint.h . . . 142
8.18 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/mJoint.h File Reference . . . 144
8.18.1 Detailed Description . . . 144
8.19 mJoint.h . . . 145
8.20 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/mJoint.hpp File Reference . . . 146
8.20.1 Detailed Description . . . 146
8.21 mJoint.hpp . . . 147
8.22 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/mMockJoint.h File Reference . . . 147
8.22.1 Detailed Description . . . 148
8.23 mMockJoint.h . . . 148
8.24 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/uErr.h File Reference . . . 149
8.24.1 Detailed Description . . . 149
8.24.2 Macro Definition Documentation . . . 150
8.24.2.1 RETURN_ON_ERROR . . . 150
8.24.2.2 RETURN_ON_FALSE . . . 150
8.24.2.3 RETURN_ON_NEGATIVE . . . 150
8.25 uErr.h . . . 151
8.26 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_↩ hardware_driver/uI2C.h File Reference . . . 152
8.26.1 Detailed Description . . . 152
8.26.2 Macro Definition Documentation . . . 153
8.26.2.1 ACK . . . 153
8.26.2.2 MAX_BUFFER . . . 153
8.26.2.3 NACK . . . 153

8.26.2.4 RFLAGS_SIZE . . . 153
8.26.3 Function Documentation . . . 153
8.26.3.1 closeI2CDevHandle() . . . 153
8.26.3.2 openI2CDevHandle() . . . 154
8.26.3.3 readFromI2CDev() . . . 154
8.26.3.4 writeToI2CDev() . . . 154
8.27 uI2C.h . . . 155
8.28 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/uTransmission.h File Reference . . . 155
8.28.1 Macro Definition Documentation . . . 156
8.28.1.1 ACTUATOR2JOINT . . . 156
8.28.1.2 DEG2RAD . . . 156
8.28.1.3 JOINT2ACTUATOR . . . 156
8.28.1.4 M_PI . . . 157
8.28.1.5 RAD2DEG . . . 157
8.29 uTransmission.h . . . 157
8.30 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/joint_comm_node.cpp File Reference . . . 157
8.30.1 Function Documentation . . . 158
8.30.1.1 INT_handler() . . . 158
8.30.1.2 main() . . . 158
8.30.2 Variable Documentation . . . 158
8.30.2.1 J1 . . . 158
8.30.2.2 J2 . . . 158
8.30.2.3 J3 . . . 158
8.30.2.4 J4 . . . 158
8.31 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mBaseJoint.cpp File Reference . . . 159
8.32 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mJoint.cpp File Reference . . . 159
8.33 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mMockJoint.cpp File Reference . . . 159
8.34 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uErr.cpp File Reference . . . 159
8.35 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uI2C.cpp File Reference . . . 160
8.35.1 Function Documentation . . . 160
8.35.1.1 closeI2CDevHandle() . . . 160
8.35.1.2 openI2CDevHandle() . . . 160
8.35.1.3 readFromI2CDev() . . . 161
8.35.1.4 writeToI2CDev() . . . 161
8.36 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_interface/arm_hardware.hpp File Reference . . . 162
8.37 arm_hardware.hpp . . . 163
8.38 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/src/arm_hardware.cpp File Reference . . . 164

8.39 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_↩ gripper_hardware_driver/mBaseGripper.h File Reference 165
8.39.1 Detailed Description 165
8.40 mBaseGripper.h 166
8.41 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_↩ gripper_hardware_driver/mGripper.h File Reference 166
8.41.1 Detailed Description 167
8.42 mGripper.h 167
8.43 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_↩ gripper_hardware_driver/mMockGripper.h File Reference 168
8.43.1 Detailed Description 168
8.44 mMockGripper.h 169
8.45 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_↩ gripper_hardware_driver/uPWM.h File Reference 169
8.45.1 Detailed Description 169
8.46 uPWM.h 170
8.47 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mBaseGripper.cpp File Reference 171
8.48 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mGripper.cpp File Reference 171
8.49 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mMockGripper.cpp File Reference 171
8.50 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/include/bioscara_↩ gripper_hardware_interface/gripper_hardware.hpp File Reference 172
8.51 gripper_hardware.hpp 172
8.52 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/src/gripper_↩ hardware.cpp File Reference 173
8.53 lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/include/single_trigger_controller/single↩ _trigger_controller.hpp File Reference 174
8.54 single_trigger_controller.hpp 175
8.55 lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/src/single_trigger_controller.cpp File Reference 176

Index **177**

Chapter 1

Bioscara C++ API Documentation

This is the API documentation for the Bioscara robotic control middleware based on the ROS2 framework.

Please find the key components below:

- bioscara_joint_firmware: contains the joint firmware and its associtated helper classes.
- bioscara_hardware_drivers: contains the BaseJoint, Joint and MockJoint drivers as well as the BaseGripper, Gripper and MockGripper drivers.
- bioscara_hardware_interfaces: contains the ros2_control hardware interfaces that implement the Bioscara arm and gripper hardware components in the BioscaraArmHardwareInterface and BioscaraGripper↩ HardwareInterface respectively.
- bioscara_rviz_plugin: contains the BioscaraPanel Rviz GUI panel.
- single_trigger_controller::SingleTriggerController: contains the documentation for the SingleTriggerController.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

bioscara_hardware_drivers 11
bioscara_hardware_interfaces 12
bioscara_joint_firmware
    Joint firmware 13
bioscara_rviz_plugin 19
single_trigger_controller 19

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

bioscara_hardware_drivers::BaseGripper 21
  bioscara_hardware_drivers::Gripper 73
  bioscara_hardware_drivers::MockGripper 100
bioscara_hardware_drivers::BaseJoint 27
  bioscara_hardware_drivers::Joint 78
  bioscara_hardware_drivers::MockJoint 102
controller_interface::ControllerInterface
  single_trigger_controller::SingleTriggerController 115
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t 77
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t 93
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t 96
bioscara_joint_firmware::Lowpass 98
bioscara_joint_firmware::MovMax 111
rviz_common::Panel
  bioscara_rviz_plugin::BioscaraPanel 61
RPI_PWM 112
hardware_interface::SystemInterface
  bioscara_hardware_interfaces::BioscaraArmHardwareInterface 42
  bioscara_hardware_interfaces::BioscaraGripperHardwareInterface 55

Chapter 4

Class Index

4.1 Class List

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

bioscara_hardware_drivers::BaseGripper
: Generic base class for gripper control implementations . . . 21

bioscara_hardware_drivers::BaseJoint
: Generic base class to control a single joint . . . 27

bioscara_hardware_interfaces::BioscaraArmHardwareInterface
: The bioscara arm hardware interface class . . . 42

bioscara_hardware_interfaces::BioscaraGripperHardwareInterface
: The bioscara gripper hardware interface class . . . 55

bioscara_rviz_plugin::BioscaraPanel
: RViz Panel to control the hardware specific functions of the Bioscara robot . . . 61

bioscara_hardware_drivers::Gripper
: Child class implementing control of the hardware gripper . . . 73

bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t
: Configuration structure holding the passed paramters from the ros2_control urdf . . . 77

bioscara_hardware_drivers::Joint
: Representing a single hardware joint connected via I2C . . . 78

bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t
: Configuration structure holding the passed paramters from the ros2_control urdf . . . 93

bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t
: Configuration structure holding the passed homing paramters from the ros2_control urdf . . . 96

bioscara_joint_firmware::Lowpass
: Simple discrete IIR lowpass filter . . . 98

bioscara_hardware_drivers::MockGripper
: Child class for to mock the gripper hardware . . . 100

bioscara_hardware_drivers::MockJoint
: Representing a single joint mocking the hardware joint . . . 102

bioscara_joint_firmware::MovMax
: Simple FIR moving maximum filter . . . 111

RPI_PWM
: Class to create a Pulse Width Modulated (PWM) signal on the Raspberry PI 4 and 5 . . . 112

single_trigger_controller::SingleTriggerController
: A general purpose controller which can be used to trigger command interfaces . . . 115

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

lib/joint_firmware/joint/configuration.h
Configuration definitions for Joint 1 to Joint 4 . . . 127
lib/joint_firmware/joint/filters.h
Helper classes for FIR and IIR filters . . . 129
lib/joint_firmware/joint/joint.h
Joint firmware header . . . 130
lib/joint_firmware/joint/joint.ino
Joint firmware . . . 134
lib/joint_firmware/joint/stall.h
Helper functions for improved stall detection . . . 136
lib/ros2_ws/src/bioscara_rviz_plugin/include/bioscara_rviz_plugin/bioscara_panel.hpp . . . 137
lib/ros2_ws/src/bioscara_rviz_plugin/src/bioscara_panel.cpp . . . 139
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/common.h
A file containing utility macros and functions . . . 140
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/mBaseJoint.h
File including the BaseJoint class . . . 141
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/mJoint.h
File including the Joint class . . . 144
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/mJoint.hpp
Templated functions for the Joint class . . . 146
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/mMockJoint.h
File including the MockJoint class . . . 147
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/uErr.h
Defining common return types . . . 149
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/uI2C.h
Low level utility for I2C communication on Raspberry Pi using lgpio library . . . 152
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩
driver/uTransmission.h . . . 155

lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/joint_comm_node.cpp 157
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mBaseJoint.cpp 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mJoint.cpp 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mMockJoint.cpp 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uErr.cpp 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uI2C.cpp 160
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_interface/arm_hardware.hpp 162
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/src/arm_hardware.cpp 164
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mBaseGripper.h
File containing the BaseGripper class 165
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mGripper.h
File containing the Gripper class 166
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mMockGripper.h
File containing the MockGripper class 168
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/uPWM.h
Includes source code for Hardware PWM generation on Raspberry Pi 4 169
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mBaseGripper.cpp . . . 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mGripper.cpp 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mMockGripper.cpp . . . 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/include/bioscara_gripper_hardware_interface/gripper_hardware.hpp 172
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/src/gripper_hardware.cpp 173
lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/include/single_trigger_controller/single_trigger_controller.hpp 174
lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/src/single_trigger_controller.cpp 176

Chapter 6

Namespace Documentation

6.1 bioscara_hardware_drivers Namespace Reference

Classes

- class BaseGripper

  Generic base class for gripper control implementations.
- class BaseJoint

  Generic base class to control a single joint.
- class Gripper

  Child class implementing control of the hardware gripper.
- class Joint

  Representing a single hardware joint connected via I2C.
- class MockGripper

  Child class for to mock the gripper hardware.
- class MockJoint

  Representing a single joint mocking the hardware joint.

Enumerations

- enum class err_type_t {
  OK = 0 , ERROR = -1 , NOT_HOMED = -2 , NOT_ENABLED = -3 ,
  STALLED = -4 , NOT_INIT = -5 , COMM_ERROR = -6 , INVALID_ARGUMENT = -101 ,
  INCORRECT_STATE = -109 }

  Enum defining common error types.

Functions

- std::string error_to_string (err_type_t err)

  Converts an error code to a string and returns it.

6.1.1 Enumeration Type Documentation

6.1.1.1 err_type_t

```
enum class bioscara_hardware_drivers::err_type_t  [strong]
```

Enum defining common error types.

Enumerator

OK	Success.
ERROR	Generic Error.
NOT_HOMED	Joint not homed or failed to home.
NOT_ENABLED	Joint not enabled or failed to enable.
STALLED	Joint stalled.
NOT_INIT	Joint not initialized.
COMM_ERROR	Communication error.
INVALID_ARGUMENT	Argument violates conditions.
INCORRECT_STATE	Joint is busy executing another blocking command.

6.1.2 Function Documentation

6.1.2.1 error_to_string()

```
std::string bioscara_hardware_drivers::error_to_string (
            err_type_t err )
```

Converts an error code to a string and returns it.

Parameters

err	

Returns

std::string

6.2 bioscara_hardware_interfaces Namespace Reference

Classes

- class BioscaraArmHardwareInterface

  The bioscara arm hardware interface class.
- class BioscaraGripperHardwareInterface

  The bioscara gripper hardware interface class.

Variables

- constexpr char HW_IF_HOME [ ] = "home"

6.2.1 Variable Documentation

6.2.1.1 HW_IF_HOME

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

6.3 bioscara_joint_firmware Namespace Reference

Joint firmware.

Classes

- class Lowpass

  Simple discrete IIR lowpass filter.
- class MovMax

  Simple FIR moving maximum filter.

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.
- 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 set_flags_for_blocking_handler (uint8_t reg)

  prepare flags to initiate the blocking handling of the received comman.
- void receiveEvent (int n)

  I2C receive event Handler.
- void requestEvent ()

  I2C request event Handler.
- void setup (void)

  Setup Peripherals.
- void loop (void)

  Main loop.
- float stall_threshold (float qd_rad, float offset)

  computes the speed adaptive threshold.

Variables

- UstepperS32 stepper

  The core UstepperS32 stepper object to control the motor.
- uint8_t reg = 0
- uint8_t blk_reg = 0
- uint8_t rx_buf [MAX_BUFFER] = {0}
- uint8_t tx_buf [MAX_BUFFER+RFLAGS_SIZE] = {0}
- bool rx_data_ready = 0
- size_t tx_length = 0
- size_t rx_length = 0

6.3.1 Detailed Description

Joint firmware.

The joint firmware is implemented in the Arduino framework. Initially the setup() function is called and subsequently the loop() function in an infinite loop. The receiveEvent() is executed when a I2C message is received, and the requestEvent() when data is requested. This always happens sequentially because at least always the #state flags are returned to the robot controller

6.3.2 Enumeration Type Documentation

6.3.2.1 stp_reg_t

```
enum bioscara_joint_firmware::stp_reg_t
```

register and command definitions

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

Enumerator

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

Enumerator

DISABLESTALLGUARD	
CLEARSTALL	
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) -].

6.3.3 Function Documentation

6.3.3.1 blocking_handler()

```
void bioscara_joint_firmware::blocking_handler (
            uint8_t reg )
```

Handles commands received via I2C.

Warning

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

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

Parameters

reg	command that should be executed.

6.3.3.2 loop()

```
void bioscara_joint_firmware::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

6.3.3.3 non_blocking_handler()

```
void bioscara_joint_firmware::non_blocking_handler (
            uint8_t reg )
```

Handles read request received via I2C.

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

Parameters

reg	command to execute/register to read.

6.3.3.4 readValue()

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

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

Parameters

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

6.3.3.5 receiveEvent()

```
void bioscara_joint_firmware::receiveEvent (
            int n )
```

I2C receive event Handler.

Reads the content of the received message. Saves the register so it can be used in the main loop. If the master invokes the read() function the message contains only the register byte and no payload. If the master invokes the write() the message has a payload of appropriate size for the command. Every I2C transaction starts with a receive event when the command is sent and is immediatly followed by a request since at minimum the flags need to be transmitted back. This means that the receive handler and request handler are always executed sequentially. The

main loop is not executed since both handlers are ISRs. For a read request the message looks like this:
< [REG]
> [TXBUFn]...[TXBUF2][TXBUF1][TXBUF0][FLAGS]
For a command the message looks like this:
< [REG][RXBUFn]...[RXBUF2][RXBUF1][RXBUF0]
> [FLAGS]
The payload is read into the rx_buf, rx_length is set to the payload length.

Parameters

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

6.3.3.6 requestEvent()

```
void bioscara_joint_firmware::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.

6.3.3.7 set_flags_for_blocking_handler()

```
void bioscara_joint_firmware::set_flags_for_blocking_handler (
            uint8_t reg )
```

prepare flags to initiate the blocking handling of the received comman.

Sets the blk_reg to reg, this makes sure that even if a new command is received the blocking hanlder access the latest blocking command. Sets the isBusy flag, to indicate immediatly that the blocking has not been finished. This is if a status request follows immidiatly the command, before the context back to the main loop has happened. Sets the rx_data_ready flag to indicate to the main loop that there is data to read in the blocking handler. Sets the tx_length to 0 because blocking commands can not return a payload.

6.3.3.8 setup()

```
void bioscara_joint_firmware::setup (
            void  )
```

Setup Peripherals.

Setup I2C with the address ADR, and begin Serial for debugging with baudrate 9600. Setup the stepper, perform orientation check to check wiring and disable the stepper again.

6.3.3.9 stall_threshold()

```
float bioscara_joint_firmware::stall_threshold (
            float qd_rad,
            float offset )
```

computes the speed adaptive threshold.

Parameters

qd_rad	speed in rad/s. Should be measured speed because set speed is only available in velocity mode.

6.3.3.10 writeValue()

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

Writes a value of the specified type to a buffer.

Parameters

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

Returns

0 On success

6.3.4 Variable Documentation

6.3.4.1 blk_reg

```
uint8_t bioscara_joint_firmware::blk_reg = 0
```

6.3.4.2 reg

```
uint8_t bioscara_joint_firmware::reg = 0
```

6.3.4.3 rx_buf

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

6.3.4.4 rx_data_ready

```
bool bioscara_joint_firmware::rx_data_ready = 0
```

6.3.4.5 rx_length

```
size_t bioscara_joint_firmware::rx_length = 0
```

6.3.4.6 stepper

```
UstepperS32 bioscara_joint_firmware::stepper
```

The core UstepperS32 stepper object to control the motor.

6.3.4.7 tx_buf

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

6.3.4.8 tx_length

```
size_t bioscara_joint_firmware::tx_length = 0
```

6.4 bioscara_rviz_plugin Namespace Reference

Classes

- class BioscaraPanel

  RViz Panel to control the hardware specific functions of the Bioscara robot.

6.5 single_trigger_controller Namespace Reference

Classes

- class SingleTriggerController

  A general purpose controller which can be used to trigger command interfaces.

Typedefs

- using CmdType = control_msgs::msg::DynamicInterfaceGroupValues
- using StateType = control_msgs::msg::DynamicInterfaceGroupValues
- using CallbackReturn = controller_interface::CallbackReturn
- using InterfacesNames = std::vector< std::string >
- using MapOfReferencesToCommandInterfaces = std::unordered_map< std::string, std::reference_wrapper< hardware_interface::LoanedCommandInterface > >
- using MapOfReferencesToStateInterfaces = std::unordered_map< std::string, std::reference_wrapper< hardware_interface::LoanedStateInterface > >
- using StateInterfaces = std::vector< std::reference_wrapper< hardware_interface::LoanedStateInterface > >

6.5.1 Typedef Documentation

6.5.1.1 CallbackReturn

```
using single_trigger_controller::CallbackReturn = typedef controller_interface::CallbackReturn
```

6.5.1.2 CmdType

```
using single_trigger_controller::CmdType = typedef control_msgs::msg::DynamicInterfaceGroup↩
Values
```

6.5.1.3 InterfacesNames

```
using single_trigger_controller::InterfacesNames = typedef std::vector<std::string>
```

6.5.1.4 MapOfReferencesToCommandInterfaces

```
using single_trigger_controller::MapOfReferencesToCommandInterfaces = typedef std::unordered↩
_map< std::string, std::reference_wrapper<hardware_interface::LoanedCommandInterface> >
```

6.5.1.5 MapOfReferencesToStateInterfaces

```
using single_trigger_controller::MapOfReferencesToStateInterfaces = typedef std::unordered_↩
map<std::string, std::reference_wrapper<hardware_interface::LoanedStateInterface> >
```

6.5.1.6 StateInterfaces

```
using single_trigger_controller::StateInterfaces = typedef std::vector<std::reference_wrapper<hardware↩
_interface::LoanedStateInterface> >
```

6.5.1.7 StateType

```
using single_trigger_controller::StateType = typedef control_msgs::msg::DynamicInterface↩
GroupValues
```

Chapter 7

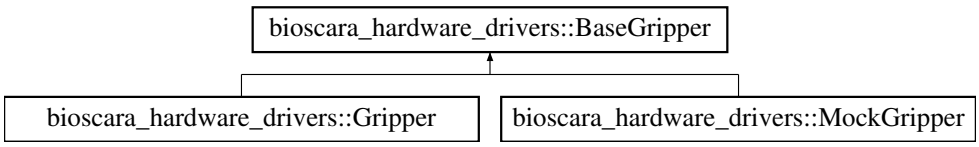
Class Documentation

7.1 bioscara_hardware_drivers::BaseGripper Class Reference

Generic base class for gripper control implementations.

`#include <mBaseGripper.h>`

Inheritance diagram for bioscara_hardware_drivers::BaseGripper:



Public Member Functions

- BaseGripper (float reduction, float offset, float min, float max, float backup_init_pos)
  Construct a new BaseGripper object.
- ~BaseGripper (void)
  Destroy the BaseGripper object.
- virtual err_type_t init (void)
  Initializes the gripper.
- virtual err_type_t deinit (void)
  Deinitializes the gripper.
- virtual err_type_t enable (void)
  Enables the gripper.
- virtual err_type_t disable (void)
  Disables the gripper.
- virtual err_type_t setPosition (float width)
  Sets the gripper width in m.
- virtual err_type_t getPosition (float &width)
  Gets the gripper width.
- virtual void setReduction (float reduction)
  Manually set reduction.
- virtual void setOffset (float offset)
  Manually set offset.

Protected Member Functions

- err_type_t save_last_position (float pos)
  Stores the latest position to the buffer file.
- err_type_t retrieve_last_position (float &pos)
  Retrieves the stored position from the buffer file.

Protected Attributes

- float _reduction = 1
  gripper width to actuator reduction ratio
- float _offset = 0
  gripper width position offset
- float _min = 0
  gripper width lower limit
- float _max = 0
  gripper width upper limit
- float _backup_init_pos = 0.0
  Initial position assumed if none can be retrieved from the buffer file.
- float _pos = _backup_init_pos
  stored position
- float _pos_get = _pos
  reported position

Private Attributes

- std::chrono::_V2::system_clock::time_point _new_cmd_time = std::chrono::_V2::system_clock::time_point()

7.1.1 Detailed Description

Generic base class for gripper control implementations.

This class is a wrapper function to interact with the robot gripper either through a MockGripper or hardware Gripper object.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BaseGripper()

```
bioscara_hardware_drivers::BaseGripper::BaseGripper (
            float reduction,
            float offset,
            float min,
            float max,
            float backup_init_pos )
```

Construct a new BaseGripper object.

The gripper has the reduction r and offset o parameters which are used to translate from a desired gripper width to the servo angle. The relationship between gripper width w and acutator angle α is as follows:

$$\alpha = r(w - o)$$

To determine these parameters execute the following steps:

1. Manually set the gripper to an open position by setting a actuator angle. Be carefull to not exceed the physical limits of the gripper since the actuator is strong enough to break PLA before stalling.
2. Measure the gripper width w_1 and note the set actuator angle α_1.
3. Move the gripper to a more closed position that still allows you to accurately measure the width
4. Measure the second width w_2 and note the corresponding angle α_2
5. Calculate the offset o:
$$o = \frac{\alpha_1 w_2 - \alpha_2 w_1}{\alpha_1 - \alpha_2}$$
6. Calculate the reduction r:
$$r = \frac{\alpha_1}{w_1 - o}$$

Parameters

reduction	the gripper width to actuator reduction ratio
offset	gripper width to actuator zero offset
min	lower limit in m
max	upper limit in m
backup_init_pos	initial position assumed if none can be retrieved from the buffer file

7.1.2.2 ~BaseGripper()

```
bioscara_hardware_drivers::BaseGripper::~BaseGripper (
            void  )
```

Destroy the BaseGripper object.

invokes the disable() and deinit() method to clean up.

7.1.3 Member Function Documentation

7.1.3.1 deinit()

```
err_type_t bioscara_hardware_drivers::BaseGripper::deinit (
            void  )  [virtual]
```

Deinitializes the gripper.

Returns

err_type_t::OK

7.1.3.2 disable()

```
err_type_t bioscara_hardware_drivers::BaseGripper::disable (
            void  )  [virtual]
```

Disables the gripper.

As described in enable()

Returns

err_type_t::OK

Reimplemented in bioscara_hardware_drivers::Gripper.

7.1.3.3 enable()

```
err_type_t bioscara_hardware_drivers::BaseGripper::enable (
            void  )  [virtual]
```

Enables the gripper.

Since the gripper has no position feedback its position is initially unknown. For this reason we try to retrieve the latest commanded position from a buffer file using retrieve_last_position(). If this fails the _backup_init_pos is used.

Returns

err_type_t::OK

Reimplemented in bioscara_hardware_drivers::Gripper.

7.1.3.4 getPosition()

```
err_type_t bioscara_hardware_drivers::BaseGripper::getPosition (
            float & width )  [virtual]
```

Gets the gripper width.

Note

Since the PWM servo has no position feedback, the latest position command is returned.

Parameters

width	width in m.

Returns

err_type_t::OK

7.1.3.5 init()

```
err_type_t bioscara_hardware_drivers::BaseGripper::init (
            void  )  [virtual]
```

Initializes the gripper.

Returns

err_type_t::OK

7.1.3.6 retrieve_last_position()

```
err_type_t bioscara_hardware_drivers::BaseGripper::retrieve_last_position (
            float & pos )  [protected]
```

Retrieves the stored position from the buffer file.

Parameters

pos	position value that is retrieved

Returns

err_type_t::OK on success, err_type_t::ERROR if parsing buffer file fails.

7.1.3.7 save_last_position()

```
err_type_t bioscara_hardware_drivers::BaseGripper::save_last_position (
            float pos )  [protected]
```

Stores the latest position to the buffer file.

Parameters

pos	position value to store

Returns

err_type_t::OK on success, err_type_t::ERROR if writing buffer file fails.

7.1.3.8 setOffset()

```
void bioscara_hardware_drivers::BaseGripper::setOffset (
            float offset )  [virtual]
```

Manually set offset.

7.1.3.9 setPosition()

```
err_type_t bioscara_hardware_drivers::BaseGripper::setPosition (
            float width ) [virtual]
```

Sets the gripper width in m.

Arguments outside the allowed range are bounded to _min and _max.

Parameters

width	width in m.

Returns

err_type_t::OK

Reimplemented in bioscara_hardware_drivers::Gripper.

7.1.3.10 setReduction()

```
void bioscara_hardware_drivers::BaseGripper::setReduction (
            float reduction ) [virtual]
```

Manually set reduction.

Parameters

reduction	

7.1.4 Member Data Documentation

7.1.4.1 _backup_init_pos

```
float bioscara_hardware_drivers::BaseGripper::_backup_init_pos = 0.0 [protected]
```

Initial position assumed if none can be retrieved from the buffer file.

7.1.4.2 _max

```
float bioscara_hardware_drivers::BaseGripper::_max = 0 [protected]
```

gripper width upper limit

7.1.4.3 _min

```
float bioscara_hardware_drivers::BaseGripper::_min = 0 [protected]
```

gripper width lower limit

7.1.4.4 _new_cmd_time

```
std::chrono::_V2::system_clock::time_point bioscara_hardware_drivers::BaseGripper::_new_cmd_↩
time = std::chrono::_V2::system_clock::time_point()  [private]
```

7.1.4.5 _offset

```
float bioscara_hardware_drivers::BaseGripper::_offset = 0  [protected]
```

gripper width position offset

7.1.4.6 _pos

```
float bioscara_hardware_drivers::BaseGripper::_pos = _backup_init_pos  [protected]
```

stored position

7.1.4.7 _pos_get

```
float bioscara_hardware_drivers::BaseGripper::_pos_get = _pos  [protected]
```

reported position

7.1.4.8 _reduction

```
float bioscara_hardware_drivers::BaseGripper::_reduction = 1  [protected]
```

gripper width to actuator reduction ratio

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

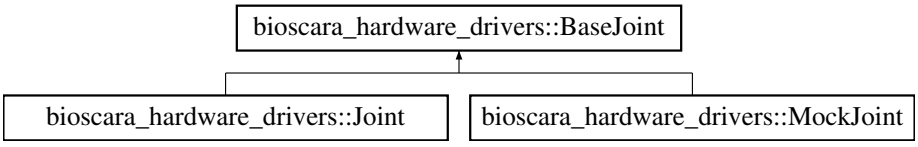
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_↩ hardware_driver/mBaseGripper.h
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mBaseGripper.cpp

7.2 bioscara_hardware_drivers::BaseJoint Class Reference

Generic base class to control a single joint.

```
#include <mBaseJoint.h>
```

Inheritance diagram for bioscara_hardware_drivers::BaseJoint:



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

- BaseJoint (const std::string name)

  Create a Joint object.
- ~BaseJoint (void)

  Destroy the BaseJoint object.
- virtual err_type_t init (void)

  Initialize the joint communication.
- virtual err_type_t deinit (void)

  Denitialize the joint communication.
- virtual err_type_t enable (u_int8_t driveCurrent, u_int8_t holdCurrent)

  Engages the joint.
- virtual err_type_t disable (void)

  disenganges the joint
- virtual err_type_t home (float velocity, u_int8_t sensitivity, u_int8_t current)

  Blocking implementation to home the joint.
- virtual err_type_t startHoming (float velocity, u_int8_t sensitivity, u_int8_t current)

  non-blocking implementation to home the joint
- virtual err_type_t postHoming (void)

  perform tasks after a non-blocking homing
- virtual err_type_t getPosition (float &pos)=0

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

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

  Move full steps.
- virtual err_type_t getVelocity (float &vel)=0

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

  Set the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.
- virtual err_type_t checkOrientation (float angle=2.0)

  Calls the checkOrientation method of the motor. Checks in which direction the motor is turning.
- virtual err_type_t stop (void)

  Stops the motor.
- virtual err_type_t disableCL (void)
- virtual err_type_t setDriveCurrent (u_int8_t current)

Set the Drive Current.

- virtual err_type_t setHoldCurrent (u_int8_t current)

  Set the Hold Current.
- virtual err_type_t setBrakeMode (u_int8_t mode)

  Set Brake Mode.
- virtual err_type_t setMaxAcceleration (float maxAccel)

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

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

  Enable encoder stall detection of the joint.
- virtual bool isHomed (void)

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

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

  Checks if the motor is stalled.
- virtual bool isBusy (void)

  Checks if the joint controller is busy processing a blocking command.
- virtual err_type_t getFlags (u_int8_t &flags)

  get the latest driver state flags from the joint
- virtual err_type_t getFlags (void)

  Overload of getFlags(u_int8_t &flags)
- virtual stp_reg_t getCurrentBCmd (void)

  get the currently active blocking command

Public Attributes

- std::string name

  Joint name for logging.

Protected Member Functions

- virtual void wait_while_busy (const float period_ms)

  Blocking loop waiting for BUSY flag to reset.
- virtual err_type_t _home (float velocity, u_int8_t sensitivity, u_int8_t current)=0

  Call to start the homing sequence of a joint.

Protected Attributes

- u_int8_t flags = 0b00001100

  State flags transmitted with every I2C transaction.
- stp_reg_t current_b_cmd = NONE

  Keeps track if a blocking command is being executed.

7.2.1 Detailed Description

Generic base class to control a single joint.

This class is a wrapper function to interact with a robot joint either through a MockJoint or hardware Joint object.

7.2.2 Member Enumeration Documentation

7.2.2.1 stp_reg_t

enum bioscara_hardware_drivers::BaseJoint::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) -].

7.2.3 Constructor & Destructor Documentation

7.2.3.1 BaseJoint()

```
bioscara_hardware_drivers::BaseJoint::BaseJoint (
            const std::string name )
```

Create a Joint object.

The Joint object represents a single joint.

Parameters

name	string device name for logging.

7.2.3.2 ~BaseJoint()

```
bioscara_hardware_drivers::BaseJoint::~BaseJoint (
            void  )
```

Destroy the BaseJoint object.

Invokes the disable() and deinit() method to clean up.

7.2.4 Member Function Documentation

7.2.4.1 _home()

```
virtual err_type_t bioscara_hardware_drivers::BaseJoint::_home (
            float velocity,
            u_int8_t sensitivity,
            u_int8_t current )  [protected], [pure virtual]
```

Call to start the homing sequence of a joint.

The joint will start rotating with 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

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

Returns

err_type_t

Implemented in bioscara_hardware_drivers::MockJoint, and bioscara_hardware_drivers::Joint.

7.2.4.2 checkOrientation()

```
err_type_t bioscara_hardware_drivers::BaseJoint::checkOrientation (
            float angle = 2.0 )  [virtual]
```

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

Note

This is a blocking function.

Parameters

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

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.3 deinit()

```
err_type_t bioscara_hardware_drivers::BaseJoint::deinit (
            void  )  [virtual]
```

Denitialize the joint communication.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.4 disable()

```
err_type_t bioscara_hardware_drivers::BaseJoint::disable (
            void  )  [virtual]
```

disenganges the joint

invokes stop(), sets hold and drive current to 0 and sets the the joint brake mode to freewheeling

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::MockJoint.

7.2.4.5 disableCL()

```
err_type_t bioscara_hardware_drivers::BaseJoint::disableCL (
            void  )  [virtual]
```

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.6 enable()

```
err_type_t bioscara_hardware_drivers::BaseJoint::enable (
            u_int8_t driveCurrent,
            u_int8_t holdCurrent )  [virtual]
```

Engages the joint.

This function prepares the motor for movement. After successfull execution the joint is ready to accept setPosition() and 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

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

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.7 enableStallguard()

```
err_type_t bioscara_hardware_drivers::BaseJoint::enableStallguard (
            u_int8_t sensitivity )  [virtual]
```

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 reenabling the joint using enable().

Note

If stall detection shall be enabled, invoke this method AFTER enabling the joint with enable().

Parameters

sensitivity	value of threshold. 0 - 255 where lower is more sensitive.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.8 getCurrentBCmd()

```
BaseJoint::stp_reg_t bioscara_hardware_drivers::BaseJoint::getCurrentBCmd (
            void  )  [virtual]
```

get the currently active blocking command

Returns

The the command of type stp_reg_t

7.2.4.9 getFlags() [1/2]

```
err_type_t bioscara_hardware_drivers::BaseJoint::getFlags (
            u_int8_t & flags )  [virtual]
```

get the latest driver state flags from the joint

Parameters

flags	if succesfull, populated with the latest flags

Returns

err_type_t

7.2.4.10 getFlags() [2/2]

```
err_type_t bioscara_hardware_drivers::BaseJoint::getFlags (
            void  )  [virtual]
```

Overload of getFlags(u_int8_t &flags)

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.11 getPosition()

```
virtual err_type_t bioscara_hardware_drivers::BaseJoint::getPosition (
            float & pos )  [pure virtual]
```

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

pos	the current joint position in rad or m.

Returns

err_type_t

Implemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.12 getVelocity()

```
virtual err_type_t bioscara_hardware_drivers::BaseJoint::getVelocity (
            float & vel )  [pure virtual]
```

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

Parameters

vel	the current joint velocity in rad/s or m/s.

Returns

err_type_t

Implemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.13 home()

```
err_type_t bioscara_hardware_drivers::BaseJoint::home (
            float velocity,
            u_int8_t sensitivity,
            u_int8_t current )  [virtual]
```

Blocking implementation to home the joint.

Homing the joint is neccessary after its controller has been powered off. The joint can be moved while the encoders are inactive and hence the position at startup is unknow. See _home() for information on implementation and description of the paramters.

This is a blocking implementation which only returns after the the joint is no longer busy. First startHoming() is called, and subsequently waits with wait_while_busy() to finish homing. Lastly postHoming() is called.

Returns

err_type_t

7.2.4.14 init()

```
err_type_t bioscara_hardware_drivers::BaseJoint::init (
            void  )  [virtual]
```

Initialize the joint communication.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.15 isBusy()

```
bool bioscara_hardware_drivers::BaseJoint::isBusy (
            void  )  [virtual]
```

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

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

Returns

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

7.2.4.16 isEnabled()

```
bool bioscara_hardware_drivers::BaseJoint::isEnabled (
            void  )  [virtual]
```

Checks the state if the motor is enabled.

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

Returns

true if the motor is enabled, false if not.

7.2.4.17 isHomed()

```
bool bioscara_hardware_drivers::BaseJoint::isHomed (
            void  )  [virtual]
```

Checks the state if the motor is homed.

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

Returns

true if the motor is homed, false if not.

Reimplemented in bioscara_hardware_drivers::MockJoint.

7.2.4.18 isStalled()

```
bool bioscara_hardware_drivers::BaseJoint::isStalled (
            void  )  [virtual]
```

Checks if the motor is stalled.

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

Returns

true if the motor is stalled, false if not.

7.2.4.19 moveSteps()

```
err_type_t bioscara_hardware_drivers::BaseJoint::moveSteps (
            int32_t steps )  [virtual]
```

Move full steps.

This function can be called even when not homed.

Parameters

steps	number of full steps

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.20 postHoming()

```
err_type_t bioscara_hardware_drivers::BaseJoint::postHoming (
            void  )  [virtual]
```

perform tasks after a non-blocking homing

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

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.21 setBrakeMode()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setBrakeMode (
            u_int8_t mode )  [virtual]
```

Set Brake Mode.

Parameters

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

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.22 setDriveCurrent()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setDriveCurrent (
            u_int8_t current )  [virtual]
```

Set the Drive Current.

Parameters

current	0% - 100% of driver current

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.23 setHoldCurrent()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setHoldCurrent (
            u_int8_t current ) [virtual]
```

Set the Hold Current.

Parameters

current	0% - 100% of driver current

Returns

err_type_t -1 on communication error,

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.24 setMaxAcceleration()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setMaxAcceleration (
            float maxAccel ) [virtual]
```

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

Parameters

maxAccel	maximum joint acceleration.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.25 setMaxVelocity()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setMaxVelocity (
            float maxVel ) [virtual]
```

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

Parameters

maxVel	maximum joint velocity.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint.

7.2.4.26 setPosition()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setPosition (
            float pos )  [virtual]
```

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

Parameters

pos	the commanded joint position in rad or m.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.27 setVelocity()

```
err_type_t bioscara_hardware_drivers::BaseJoint::setVelocity (
            float vel )  [virtual]
```

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

Parameters

vel	the commanded joint velocity in rad/s or m/s.

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.28 startHoming()

```
err_type_t bioscara_hardware_drivers::BaseJoint::startHoming (
            float velocity,
            u_int8_t sensitivity,
            u_int8_t current )  [virtual]
```

non-blocking implementation to home the joint

Homing the joint is neccessary after its controller has been powered off. The joint can be moved while the encoders are inactive and hence the position at startup is unknow. See _home() for information on implementation and description of the paramters.

This method returns immediatly after starting the homing sequence and should be used when the blocking implementation is not acceptable, for example in a realtime loop.

This method sets the current_b_cmd flag to stp_reg_t::HOME

Returns

err_type_t

7.2.4.29 stop()

```
err_type_t bioscara_hardware_drivers::BaseJoint::stop (
            void  )  [virtual]
```

Stops the motor.

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

Returns

err_type_t

Reimplemented in bioscara_hardware_drivers::Joint, and bioscara_hardware_drivers::MockJoint.

7.2.4.30 wait_while_busy()

```
void bioscara_hardware_drivers::BaseJoint::wait_while_busy (
            const float period_ms )  [protected], [virtual]
```

Blocking loop waiting for BUSY flag to reset.

Parameters

period_ms	time in ms between polls.

7.2.5 Member Data Documentation

7.2.5.1 current_b_cmd

```
stp_reg_t bioscara_hardware_drivers::BaseJoint::current_b_cmd = NONE  [protected]
```

Keeps track if a blocking command is being executed.

7.2.5.2 flags

```
u_int8_t bioscara_hardware_drivers::BaseJoint::flags = 0b00001100  [protected]
```

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 enable()

7.2.5.3 name

```
std::string bioscara_hardware_drivers::BaseJoint::name
```

Joint name for logging.

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

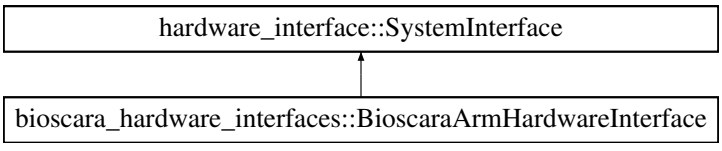
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mBaseJoint.h
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mBaseJoint.cpp

7.3 bioscara_hardware_interfaces::BioscaraArmHardwareInterface Class Reference

The bioscara arm hardware interface class.

```
#include <arm_hardware.hpp>
```

Inheritance diagram for bioscara_hardware_interfaces::BioscaraArmHardwareInterface:



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

  Called on initialization to the `unconfigured` *state.*
- hardware_interface::CallbackReturn on_shutdown (const rclcpp_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 rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `unconfigured` *to the* `inactive` *state.*
- hardware_interface::CallbackReturn on_cleanup (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `inactive` *to the* `unconfigured` *state.*
- hardware_interface::CallbackReturn on_activate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `inactive` *to the* `active` *state.*
- hardware_interface::CallbackReturn on_deactivate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `active` *to the* `inactive` *state.*
- hardware_interface::return_type read (const rclcpp::Time &time, const rclcpp::Duration &period) override

  Reads from the hardware and populates the state interfaces.
- hardware_interface::return_type write (const rclcpp::Time &time, const rclcpp::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 in non-realtime context.
- hardware_interface::return_type perform_command_mode_switch (const std::vector< std::string > &start↩ _interfaces, const std::vector< std::string > &stop_interfaces) override

  Perform the mode-switching for the new command interface combination in realtime context.
- hardware_interface::CallbackReturn on_error (const rclcpp_lifecycle::State &previous_state) override

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

Private Member Functions

- bioscara_hardware_drivers::err_type_t start_homing (const std::string name, float velocity)

  wrapper method to start homing.
- bioscara_hardware_drivers::err_type_t stop_homing (const std::string name)

  wrapper method to stop homing.
- void split_interface_string_to_joint_and_name (std::string interface, std::string &joint_name, std::string &interface_name)

  Split a interface string like "<joint_name>/<interface_name>" to "<joint_name>" and "<interface_name>".
- bioscara_hardware_drivers::err_type_t activate_joint (const std::string name)

  Enables each joint, enables the stall detection and sets the maximmum acceleration.
- bioscara_hardware_drivers::err_type_t deactivate_joint (const std::string name)

  Disables each joint.

Private Attributes

- std::unordered_map< std::string, std::unique_ptr< bioscara_hardware_drivers::BaseJoint > > _joints

  unordered map storing the pointers to BaseJoint objects. This will either be a MockJoint or Joint.
- 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.
- std::unordered_map< std::string, std::set< std::string > > _new_joint_command_modes

  Temporary cache of new joint_command_modes when switching controllers.
- std::vector< std::pair< std::string, hardware_interface::InterfaceDescription * > > _ordered_joint_state_interfaces_ptr

  A vector of a pair of interface name and pointer of the hardwares state interface which is ordered by joint and state interface type.
- std::mutex mtx

  A mutex that is used to prevent concurrent access to hardware and _joint_command_modes.

7.3.1 Detailed Description

The bioscara arm hardware interface class.

The hardware interface serves to wrap custom hardware interaction with the arm joints in the standardized ros2_control architecture.

Hardware Lifecycle
The hardware follows the ros2_control hardware interface lifecyle which intern is following the ROS2 managed node lifecycle.

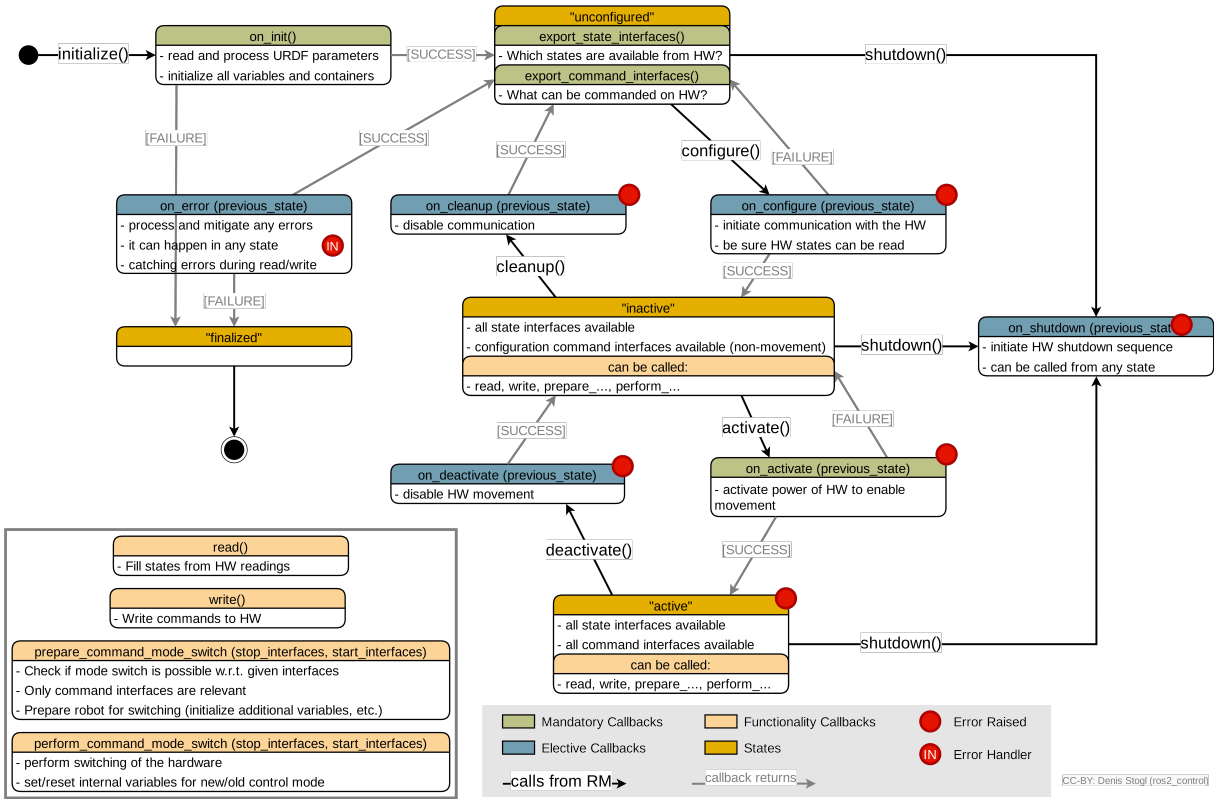


Figure 7.1 Hardware interface lifecycle

7.3.2 Member Function Documentation

7.3.2.1 activate_joint()

```
bioscara_hardware_drivers::err_type_t bioscara_hardware_interfaces::BioscaraArmHardware↩
Interface::activate_joint (
            const std::string name )  [private]
```

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

Parameters

name	joint name to enable

Returns

bioscara_hardware_drivers::err_type_t

7.3.2.2 deactivate_joint()

```
bioscara_hardware_drivers::err_type_t bioscara_hardware_interfaces::BioscaraArmHardware↩
Interface::deactivate_joint (
            const std::string name )  [private]
```

Disables each joint.

Parameters

name	joint name to disable

Returns

bioscara_hardware_drivers::err_type_t

7.3.2.3 on_activate()

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

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

Calls activate_joint() to enable the joints.
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

previous_state	

Returns

hardware_interface::CallbackReturn

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

7.3.2.4 on_cleanup()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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

previous_state	

Returns

hardware_interface::CallbackReturn

Disconnect from the joints and throw error if it fails

7.3.2.5 on_configure()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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

previous_state	

Returns

hardware_interface::CallbackReturn

7.3.2.6 on_deactivate()

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

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

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

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

disable the joints and throw error if it fails

7.3.2.7 on_error()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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 bioscara_hardware_drivers::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, disableing the joints to allow backdriving.

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.3.2.8 on_init()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::on_init (
            const hardware_interface::HardwareComponentInterfaceParams & params )  [override]
```

Called on initialization to the `unconfigured` state.

Performs the following checks on the configures joints parsed form the URDF description:

- Each joint must have the 3 command interfaces (in this order): 'position', 'velocity', 'home'
- Each joint must have the 3 state interfaces (in this order): 'position', 'velocity', 'home'

Stores the configuration parameters for each joint in the _joint_cfg map. Each joint must have these parameters:

- i2c_address (int, HEX)
- reduction (float)
- min (float)
- max (float)
- stall_threshold (int, DEC)
- hold_current (int, DEC)
- drive_current (int, DEC)
- max_acceleration (float)
- max_velocity (float)
- homing
  - speed (float)
  - threshold (int, DEC)
  - current (int, DEC)
  - acceleration (float)

Adds each joint to the internal _joints map. Creates a MockJoint object if the use_mock_hardware parameter is 'True' or 'true', or else a hardware Joint.

Parameters

params	

Returns

hardware_interface::CallbackReturn

Loop over all joints decribed 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

7.3.2.9 on_shutdown()

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

Called on the transistion 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

previous_state	

Returns

hardware_interface::CallbackReturn

7.3.2.10 perform_command_mode_switch()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::perform_command_mode_switch (
            const std::vector< std::string > & start_interfaces,
            const std::vector< std::string > & stop_interfaces )  [override]
```

Perform the mode-switching for the new command interface combination in realtime context.

Performs the following actions:

- acquires the mtx lock to protect the critical members.
- Copies the cached _new_joint_command_modes to the _joint_command_modes
- **On activation**:
  - **home** interface:
    - Reset command to 0.0. This clears any remaining commands that have been written to the command interface while the hardware was unable to act on it. For example if it was inactive or the homing command was not the active command mode.

Parameters

start_interfaces	vector of string identifiers for the command interfaces starting.
stop_interfaces	vector of string identifiers for the command interfaces stopping.

Returns

return_type::OK if the new command interface combination can be switched to (or) if the interface key is not relevant to this system. Returns return_type::ERROR otherwise.

7.3.2.11 prepare_command_mode_switch()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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 in non-realtime context.

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 furture 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 (bioscara_hardware_drivers::Joint::getCurrentBCmd() != bioscara_hardware_drivers::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.
  - [ERROR] Activating 'position' or 'velocity' command interface if the joint is not homed (bioscara_hardware_drivers::Joint::isH == false).

Since this method operates in non-realtime context it must not access critical members (_joint_command_modes and _joints) to avoid priority inversion. Therefore the new command modes are first saved to a cache _new_joint_command_modes. This will then be applied to _joint_command_modes in the perform_command_mode_switch() which is executed in RT context.

Parameters

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

Returns

hardware_interface::return_type

7.3.2.12 read()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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" -> bioscara_hardware_drivers::Joint::getPosition()
- State interface "velocity" -> bioscara_hardware_drivers::Joint::getVelocity()
- State interface "home" -> bioscara_hardware_drivers::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 immediatly before the return flags are assumed to be valid.
  - If the the homing of a joint has been activated through the command interface (bioscara_hardware_drivers::Joint::getCurrer == bioscara_hardware_drivers::Joint::HOME) the device signals BUSY (bioscara_hardware_drivers::Joint::isBusy()) as long as it is still homing.
    If the BUSY flag is reset while the current command is still bioscara_hardware_drivers::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

time	
period	

Returns

hardware_interface::return_type

7.3.2.13 split_interface_string_to_joint_and_name()

```
void bioscara_hardware_interfaces::BioscaraArmHardwareInterface::split_interface_string_to_↩
joint_and_name (
            std::string interface,
            std::string & joint_name,
            std::string & interface_name )  [private]
```

Split a interface string like "<joint_name>/<interface_name>" to "<joint_name>" and "<interface_name>".

Parameters

interface	
joint_name	
interface_name	

7.3.2.14 start_homing()

```
bioscara_hardware_drivers::err_type_t bioscara_hardware_interfaces::BioscaraArmHardware↩
Interface::start_homing (
            const std::string name,
            float velocity )  [private]
```

wrapper method to start homing.

Activate the joint, set homing acceleration and start homing.

Parameters

name	
velocity	

Returns

bioscara_hardware_drivers::err_type_t

7.3.2.15 stop_homing()

```
bioscara_hardware_drivers::err_type_t bioscara_hardware_interfaces::BioscaraArmHardware↩
Interface::stop_homing (
            const std::string name )  [private]
```

wrapper method to stop homing.

Stop the homing. Reset acceleration and velocity and perform the postHoming cleanup, then deactivate the joint.

Parameters

name	

Returns

bioscara_hardware_drivers::err_type_t

7.3.2.16 write()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩
::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 _joint_command_modes map. See prepare_command_mode_switch() for a detailed reasoning why this approach has been chosen.

- Command interface "position" -> bioscara_hardware_drivers::Joint::setPosition()
- Command interface "velocity" -> bioscara_hardware_drivers::Joint::setVelocity()
- Command interface "home" -> bioscara_hardware_drivers::Joint::startHoming()
  - If the commanded value in "home" is != 0.0 the and the joint is currently executing a blocking function, for example homing (bioscara_hardware_drivers::Joint::getCurrentBCmd() == bioscara_hardware_drivers::Joint::NONE), the homing sequence is started with the speed, sensitivity, current and acceleration defined in the _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

time	
period	

Returns

hardware_interface::return_type

7.3.3 Member Data Documentation

7.3.3.1 _joint_cfg

```
std::unordered_map<std::string, joint_config_t> bioscara_hardware_interfaces::BioscaraArm↩
HardwareInterface::_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.

7.3.3.2 _joint_command_modes

```
std::unordered_map<std::string, std::set<std::string> > bioscara_hardware_interfaces::↩
BioscaraArmHardwareInterface::_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.

7.3.3.3 _joints

```
std::unordered_map<std::string, std::unique_ptr<bioscara_hardware_drivers::BaseJoint> > bioscara↩
_hardware_interfaces::BioscaraArmHardwareInterface::_joints  [private]
```

unordered map storing the pointers to BaseJoint objects. This will either be a MockJoint or Joint.

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.

Since the BaseJoint methods are implemented as virtual, dynamic method dispatch can be utilized to call the correct implementation of a method. So either BaseJoint::foo() or Joint::foo()/MockJoint::foo() if foo() is overwritten in Joint or MockJoint. a smart pointer is used to guarantee destruction when the pointer is destructed. A unique pointer is used to prevent copying of the object.

7.3.3.4 _new_joint_command_modes

```
std::unordered_map<std::string, std::set<std::string> > bioscara_hardware_interfaces::↩
BioscaraArmHardwareInterface::_new_joint_command_modes  [private]
```

Temporary cache of new joint_command_modes when switching controllers.

Since the prepare_command_mode_switch() is executed in a non-RT context we save the new joint command modes to this cache first to avoid needing to lock the _joint_command_modes.

7.3.3.5 _ordered_joint_state_interfaces_ptr

```
std::vector<std::pair<std::string, hardware_interface::InterfaceDescription *> > bioscara_↩
hardware_interfaces::BioscaraArmHardwareInterface::_ordered_joint_state_interfaces_ptr  [private]
```

A vector of a pair of interface name and pointer of the hardwares state interface which is ordered by joint and state interface type.

A vector is chosen since it guarantees the correct order by insertion. The vector holds pointers to the actual interface structs stored in the parents HardwareComponentInterface::joint_state_interfaces_ but in the desired order. The order is:

1. position
2. velocity
3. home

This order guarantees that when reading the state interfaces in read() that the 'home' interface is read last and has the latest state flags from the joint.

7.3.3.6 mtx

```
std::mutex bioscara_hardware_interfaces::BioscaraArmHardwareInterface::mtx  [private]
```

A mutex that is used to prevent concurrent access to hardware and _joint_command_modes.

The mutex prevents two things:

- Modifying the _joint_command_modes concurrently.
- Concurrent access to the hardware via the _joints map. The Joint harwdare is not thread safe. In particular the read() and write() methods are executed in one RT thread while the perform_command_mode_switch() is called from another RT thread. The latter also tries to modify the Joint object via activate_joint() and deactivate_joint() which must not happen concurrently with a read() or write() call.

All methods that need to acquire this lock need to be performed in RT context to avoid being preempted by other lower priority threads potentially blocking the other RT threads from continuing.

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

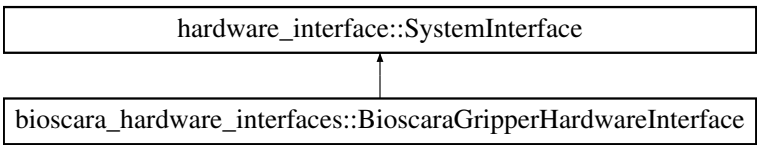
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_↩ interface/arm_hardware.hpp
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/src/arm_hardware.cpp

7.4 bioscara_hardware_interfaces::BioscaraGripperHardwareInterface Class Reference

The bioscara gripper hardware interface class.

```
#include <gripper_hardware.hpp>
```

Inheritance diagram for bioscara_hardware_interfaces::BioscaraGripperHardwareInterface:



Classes

- struct gripper_config_t

  configuration structure holding the passed 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 rclcpp_lifecycle::State &previous_state) override

  Called on the transition from the `inactive`*,* `unconfigured` *and* `active` *to the* `finalized` *state.*
- hardware_interface::CallbackReturn on_configure (const rclcpp_lifecycle::State &previous_state) override

  Called on the transition from the `unconfigured` *to the* `inactive` *state.*
- hardware_interface::CallbackReturn on_cleanup (const rclcpp_lifecycle::State &previous_state) override

  Called on the transition from the `inactive` *to the* `unconfigured` *state.*
- hardware_interface::CallbackReturn on_activate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transition from the `inactive` *to the* `active` *state.*
- hardware_interface::CallbackReturn on_deactivate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transition from the `active` *to the* `inactive` *state.*
- hardware_interface::return_type read (const rclcpp::Time &time, const rclcpp::Duration &period) override

  Reads from the hardware and populates the state interfaces.
- hardware_interface::return_type write (const rclcpp::Time &time, const rclcpp::Duration &period) override

  Writes commands to the hardware from the command interfaces.
- hardware_interface::CallbackReturn on_error (const rclcpp_lifecycle::State &previous_state) override

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

Private Attributes

- std::unique_ptr< bioscara_hardware_drivers::BaseGripper > _gripper

  Smart pointer to the local BaseGripper.
- gripper_config_t _gripper_cfg

  configuration struct of the gripper.
- float _last_pos = std::numeric_limits<double>::quiet_NaN()
- float _vel = std::numeric_limits<double>::quiet_NaN()

7.4.1 Detailed Description

The bioscara gripper hardware interface class.

System interface has been chosen to allow future modifications for grippers that provide feedback. refer to the BioscaraArmHardwareInterface class for a more detailed description of the hardware life-cycle.

7.4.2 Member Function Documentation

7.4.2.1 on_activate()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_activate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Enables PWM generation.

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.2 on_cleanup()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_cleanup (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.3 on_configure()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_configure (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.4 on_deactivate()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_deactivate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Disables PWM generation.

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.5 on_error()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::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.
  - Clean-Up hardware (on_cleanup()) -> unconfigured

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.6 on_init()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_init (
            const hardware_interface::HardwareComponentInterfaceParams & params )  [override]
```

check that only one joint is defined, more mimic joints may be defined.

7.4.2.7 on_shutdown()

```
hardware_interface::CallbackReturn bioscara_hardware_interfaces::BioscaraGripperHardware↩
Interface::on_shutdown (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

Called on the transistion 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.

Parameters

previous_state	

Returns

hardware_interface::CallbackReturn

7.4.2.8 read()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraGripperHardwareInterface↩
::read (
            const rclcpp::Time & time,
            const rclcpp::Duration & period )  [override]
```

Reads from the hardware and populates the state interfaces.

Invokes the bioscara_hardware_drivers::Gripper::getPosition() method to read the gripper position.

Parameters

time	
period	

Returns

hardware_interface::return_type

7.4.2.9 write()

```
hardware_interface::return_type bioscara_hardware_interfaces::BioscaraGripperHardwareInterface↩
::write (
            const rclcpp::Time & time,
            const rclcpp::Duration & period )  [override]
```

Writes commands to the hardware from the command interfaces.

Invokes the bioscara_hardware_drivers::Gripper::setPosition() method to write the command to the gripper.

Parameters

time	
period	

Returns

hardware_interface::return_type

7.4.3 Member Data Documentation

7.4.3.1 _gripper

```
std::unique_ptr<bioscara_hardware_drivers::BaseGripper> bioscara_hardware_interfaces::Bioscara↩
GripperHardwareInterface::_gripper  [private]
```

Smart pointer to the local BaseGripper.

This will be used to either interact with the hardware or mock hardware. A smart pointer is used to guarantee destruction when the pointer is destructed. A unique pointer is used to prevent copying of the object.

7.4.3.2 _gripper_cfg

```
gripper_config_t bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::_gripper_cfg
[private]
```

configuration struct of the gripper.

7.4.3.3 _last_pos

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::_last_pos = std::numeric↩
_limits<double>::quiet_NaN()  [private]
```

7.4.3.4 _vel

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::_vel = std::numeric_↩
limits<double>::quiet_NaN()  [private]
```

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

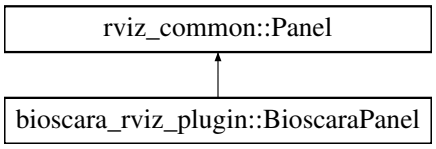
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/include/bioscara_gripper_↩ hardware_interface/gripper_hardware.hpp
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/src/gripper_hardware.cpp

7.5 bioscara_rviz_plugin::BioscaraPanel Class Reference

RViz Panel to control the hardware specific functions of the Bioscara robot.

`#include <bioscara_panel.hpp>`

Inheritance diagram for bioscara_rviz_plugin::BioscaraPanel:



Public Member Functions

- BioscaraPanel (QWidget *parent=0)

  Construct a new Bioscara Panel object.
- ~BioscaraPanel () override

  Destroy the Bioscara Panel object.
- void onInitialize () override

  Automatically invoked by the RViz parent class. Initializes the panel.

Protected Member Functions

- void cm_state_callback (const ControllerManagerActivity &msg)

  callback on reported controller or hardware state changes.
- void joint_state_callback (const DynamicJointState &msg)

  callback on joint state update.
- void ensure_jsb_is_active (void)

  Ensure the joint state broadcaster is active.
- void set_hardware_component_state (const std::string component, const lifecycle_msgs::msg::State target_state)

  Makes a service request to change a hardware component state.
- void configure_controller (const std::string controller)

  Makes a service request to configure a controller state.
- void switch_controllers (const std::vector< std::string > &activate_controllers, const std::vector< std::string > &deactivate_controllers, const int32_t=SwitchController::Request::BEST_EFFORT, const bool activate_asap=false, const builtin_interfaces::msg::Duration timeout=builtin_interfaces::msg::Duration())

  Makes a service request to change controller states.
- void set_controller_state (const std::string controller, const lifecycle_msgs::msg::State target_state)

  calls the correct functions to bring a controller to the correct state.
- void dynamic_joint_state_msg_to_map (const DynamicJointState &dynamic_joint_state_in, std::unordered_map< std::string, InterfaceValue > &map_out)

  simple conversion message type conversion function
- void named_lcs_msg_to_map (const std::vector< NamedLifecycleState > &named_lcs_in, std::unordered_map< std::string, NamedLifecycleState > &map_out)

  simple conversion message type conversion function
- void print_cm_map (const std::string prefix, const std::unordered_map< std::string, NamedLifecycleState > &map_in)

  Print life-cycle state message to console.

- void prepopulate_joint_state_map (std::unordered_map< std::string, InterfaceValue > &state_map, std::vector< std::string > augment_vec)

  Prepopulates a state map with a default NamedLifecycleState for every specified key.
- void prepopulate_state_map (std::unordered_map< std::string, NamedLifecycleState > &state_map, std::vector< std::string > augment_vec)

  Prepopulates a state map with a default NamedLifecycleState for every specified key.
- void update_homing_grp_state (void)

  Enabl/Disables the homing group depending on the homing controller state.
- void update_homing_state_labels (void)

  updates all homing state labels.
- void set_homing_state_label (QLabel *label, const InterfaceValue &state)

  Set state label text and color of the homing state label.
- void update_state_labels_and_btns (void)

  Update all hardware and controller state lables.
- void update_state_label_and_btn (const std::unordered_map< std::string, NamedLifecycleState > &state_map, QLabel *state_label, QPushButton *en_button, const std::string &state_key)

  updates component state label and control button.
- void set_state_label (QLabel *label, const NamedLifecycleState &state)

  Set the state label.
- void set_en_btn (QPushButton *button, const NamedLifecycleState &state)

  Set the control button state.
- bool check_activation_conditions (const std::string component_key)

  Check under which conditions a component is allowed to be activated.
- lifecycle_msgs::msg::State target_state_from_current (lifecycle_msgs::msg::State current_state)

  returns the next logical state from the current one.
- void homing_cmd (const std::string joint, const int cmd)

  callback function called when a homing button is pressed
- void ctrl_en_btn_cb (const std::string controller)

  callback on controller enable button

Protected Attributes

- rclcpp::Node::SharedPtr node_

  Pointer to the parent node. Used for access to the executor.
- rclcpp::Subscription< ControllerManagerActivity >::SharedPtr cm_state_subsription_

  Subscription to the the controller_manager state publisher.
- rclcpp::Subscription< DynamicJointState >::SharedPtr joint_state_subsription_

  Subscription to the the joint state publisher state publisher.
- rclcpp::Publisher< DynamicInterfaceGroupValues >::SharedPtr homing_publisher_

  Publisher to send commands to the homing_controller.
- rclcpp::Client< SwitchController >::SharedPtr switch_controller_client_

  Service client to the "/controller_manager/switch_controller" service.
- rclcpp::Client< ConfigureController >::SharedPtr configure_controller_client_

  Service client to the "/controller_manager/configure_controller" service.
- rclcpp::Client< SetHardwareComponentState >::SharedPtr hardware_state_client_

  Service client to the "/controller_manager/set_hardware_component_state" service.
- rclcpp::TimerBase::SharedPtr prune_timer_

  Timer triggered every 5 s to prune unanswered service requests.
- std::unordered_map< std::string, InterfaceValue > joint_states_

  Map storing all joint state interface values by each joint.

- std::unordered_map< std::string, NamedLifecycleState > hardware_states_

  Map correlating hardware component name to its lifecycle_msgs/State.
- std::unordered_map< std::string, NamedLifecycleState > controller_states_

  Map correlating controller name to its lifecycle_msgs/State.
- Ui::BioscaraUI * ui_

  The QT object containing all UI elements.

Private Slots

- void arm_en_btn_cb (void)

  Tries to set the bioscara_arm hardware component to the `active` state and tries to enable the velocity_joint_↩ trajectory_controller.
- void gripper_en_btn_cb (void)

  Tries to set the bioscara_gripper_128 hardware component to the `active` state and tries to enable the gripper_↩ controller.

7.5.1 Detailed Description

RViz Panel to control the hardware specific functions of the Bioscara robot.

Through the panel it is possible to modify the robot's hardware's and controller states and to execute the homing procedures.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BioscaraPanel()

```
bioscara_rviz_plugin::BioscaraPanel::BioscaraPanel (
            QWidget * parent = 0 )  [explicit]
```

Construct a new Bioscara Panel object.

Creates a new ui_ object from the ui object autogenerated from the "bioscara_rviz_plugin_frame.ui". This object contains all UI elements. Each button gets assigned a callback function.

Parameters

parent	

7.5.2.2 ~BioscaraPanel()

```
bioscara_rviz_plugin::BioscaraPanel::~BioscaraPanel ( )  [override]
```

Destroy the Bioscara Panel object.

7.5.3 Member Function Documentation

7.5.3.1 arm_en_btn_cb

```
void bioscara_rviz_plugin::BioscaraPanel::arm_en_btn_cb (
            void  )  [private], [slot]
```

Tries to set the bioscara_arm hardware component to the `active` state and tries to enable the velocity_joint_↩ trajectory_controller.

7.5.3.2 check_activation_conditions()

```
bool bioscara_rviz_plugin::BioscaraPanel::check_activation_conditions (
            const std::string component_key )  [protected]
```

Check under which conditions a component is allowed to be activated.

Some controllers will only activate under certain conditions. This functions disables the activation button if the coditions are met.

Parameters

component_key	

Returns

true

false

7.5.3.3 cm_state_callback()

```
void bioscara_rviz_plugin::BioscaraPanel::cm_state_callback (
            const ControllerManagerActivity & msg )  [protected]
```

callback on reported controller or hardware state changes.

Updates the controller_states_ and hardware_states_ state maps. Reactivates the Joint State Broadcaster if it has become inactive. Updates controller and hardware state labels and buttons by invoking update_state_labels_and_btns() Updates homing group state by invoking update_homing_grp_state().

Parameters

msg	received controller states

7.5.3.4 configure_controller()

```
void bioscara_rviz_plugin::BioscaraPanel::configure_controller (
            const std::string controller )  [protected]
```

Makes a service request to configure a controller state.

Request issued through the configure_controller_client_. Asynchrounous call, method exits after having issued the request.

Parameters

controller	name of controller to activate

7.5.3.5 ctrl_en_btn_cb()

```
void bioscara_rviz_plugin::BioscaraPanel::ctrl_en_btn_cb (
            const std::string controller )  [protected]
```

callback on controller enable button

Parameters

controller	string name of controller to change state

7.5.3.6 dynamic_joint_state_msg_to_map()

```
void bioscara_rviz_plugin::BioscaraPanel::dynamic_joint_state_msg_to_map (
            const DynamicJointState & dynamic_joint_state_in,
            std::unordered_map< std::string, InterfaceValue > & map_out )  [protected]
```

simple conversion message type conversion function

Parameters

dynamic_joint_state_in	message in
map_out	joint states map out

7.5.3.7 ensure_jsb_is_active()

```
void bioscara_rviz_plugin::BioscaraPanel::ensure_jsb_is_active (
            void  )  [protected]
```

Ensure the joint state broadcaster is active.

The JSB can become inactive, ensure that it comes active again.

Calls the set_controller_state() method

7.5.3.8 gripper_en_btn_cb

```
void bioscara_rviz_plugin::BioscaraPanel::gripper_en_btn_cb (
            void  )  [private], [slot]
```

Tries to set the bioscara_gripper_128 hardware component to the `active` state and tries to enable the gripper_controller.

7.5.3.9 homing_cmd()

```
void bioscara_rviz_plugin::BioscaraPanel::homing_cmd (
            const std::string joint,
            const int cmd ) [protected]
```

callback function called when a homing button is pressed

publishes on homing_publisher_ to initiate/stop homing

Parameters

joint	string name of joint
cmd	-1, 0, +1 to start homing to negative direction, stop homing, start homing to positve direction.

7.5.3.10 joint_state_callback()

```
void bioscara_rviz_plugin::BioscaraPanel::joint_state_callback (
            const DynamicJointState & msg ) [protected]
```

callback on joint state update.

Updates joint_states_ with the latest values using dynamic_joint_state_msg_to_map(). Updates homing group state by invoking update_homing_grp_state().

Parameters

msg	received joint states

7.5.3.11 named_lcs_msg_to_map()

```
void bioscara_rviz_plugin::BioscaraPanel::named_lcs_msg_to_map (
            const std::vector< NamedLifecycleState > & named_lcs_in,
            std::unordered_map< std::string, NamedLifecycleState > & map_out ) [protected]
```

simple conversion message type conversion function

Parameters

named_lcs_in	NamedLifecycleState message in
map_out	state map out

7.5.3.12 onInitialize()

```
void bioscara_rviz_plugin::BioscaraPanel::onInitialize ( ) [override]
```

Automatically invoked by the RViz parent class. Initializes the panel.

Retrieves the pointer to the parent node and saves it in node_. This node and its executor is used for all clients and publishers.

Additionally a the prune_timer_ is created which periodically cleans all dangling service requests. All subscribers, publishers and clients are created and saved in their corresponding handler. Also the #controller_states_state and hardware_states_ state maps are prepopulated by invoking prepopulate_state_map()

7.5.3.13 prepopulate_joint_state_map()

```
void bioscara_rviz_plugin::BioscaraPanel::prepopulate_joint_state_map (
            std::unordered_map< std::string, InterfaceValue > & state_map,
            std::vector< std::string > augment_vec )  [protected]
```

Prepopulates a state map with a default NamedLifecycleState for every specified key.

This is done to avoid having to check for the existence of key when accessing the map. Keys could be missing if hardware interfaces are not specified or named differently.

For every joint the state interfaces are initialized as follows:

- *position*: 0.0
- *velocity*: 0.0
- *home*: 0.0

Some joints (the gripper) dont have all state interfaces, but they are initialized regardless.

Parameters

state_map	The state map to prepopulate
augment_vec	keys that are added/overwritten

7.5.3.14 prepopulate_state_map()

```
void bioscara_rviz_plugin::BioscaraPanel::prepopulate_state_map (
            std::unordered_map< std::string, NamedLifecycleState > & state_map,
            std::vector< std::string > augment_vec )  [protected]
```

Prepopulates a state map with a default NamedLifecycleState for every specified key.

This is done to avoid having to check for the existence of key when accessing the map. Keys could be missing if an expected controller or hardware component has not been loaded or has a different name.

Parameters

state_map	The state map to prepopulate
augment_vec	keys that are added/overwritten

7.5.3.15 print_cm_map()

```
void bioscara_rviz_plugin::BioscaraPanel::print_cm_map (
            const std::string prefix,
            const std::unordered_map< std::string, NamedLifecycleState > & map_in )  [protected]
```

Print life-cycle state message to console.

Parameters

prefix	string prefix
map_in	map to be printed

7.5.3.16 set_controller_state()

```
void bioscara_rviz_plugin::BioscaraPanel::set_controller_state (
            const std::string controller,
            const lifecycle_msgs::msg::State target_state )  [protected]
```

calls the correct functions to bring a controller to the correct state.

Checks if configure_controller() needs to be called before switch_controllers() can be called to bring the controller to the desired state.

Parameters

controller	controller name
target_state	target state

7.5.3.17 set_en_btn()

```
void bioscara_rviz_plugin::BioscaraPanel::set_en_btn (
            QPushButton * button,
            const NamedLifecycleState & state )  [protected]
```

Set the control button state.

Sets control buttons color, text and active status based on the component state.

Parameters

button	pointer to button to modify
state	state of the component

7.5.3.18 set_hardware_component_state()

```
void bioscara_rviz_plugin::BioscaraPanel::set_hardware_component_state (
```

```
            const std::string component,
            const lifecycle_msgs::msg::State target_state )  [protected]
```

Makes a service request to change a hardware component state.

Request issued through the hardware_state_client_. Asynchrounous call, method exits after having issued the request.

Parameters

component	name of hardware component to modify
target_state	target state

7.5.3.19 set_homing_state_label()

```
void bioscara_rviz_plugin::BioscaraPanel::set_homing_state_label (
            QLabel * label,
            const InterfaceValue & state )  [protected]
```

Set state label text and color of the homing state label.

Sets color and text of label.

Parameters

label	pointer to label
state	joint state

7.5.3.20 set_state_label()

```
void bioscara_rviz_plugin::BioscaraPanel::set_state_label (
            QLabel * label,
            const NamedLifecycleState & state )  [protected]
```

Set the state label.

sets color and text based on the component state.

Parameters

label	pointer to label to modify
state	state of the component

7.5.3.21 switch_controllers()

```
void bioscara_rviz_plugin::BioscaraPanel::switch_controllers (
            const std::vector< std::string > & activate_controllers,
            const std::vector< std::string > & deactivate_controllers,
```

```
            const int32_t strictness = SwitchController::Request::BEST_EFFORT,
            const bool activate_asap = false,
            const builtin_interfaces::msg::Duration timeout = builtin_interfaces::msg:↩
:Duration() )  [protected]
```

Makes a service request to change controller states.

Activates and deactivates a combination of controllers.

Parameters

activate_controllers	list of controllers to activate
deactivate_controllers	list of controllers to deactivate
activate_asap	if true controller switch happens in the realtime loop
timeout	switch timeout

7.5.3.22 target_state_from_current()

```
lifecycle_msgs::msg::State bioscara_rviz_plugin::BioscaraPanel::target_state_from_current (
            lifecycle_msgs::msg::State current_state )  [protected]
```

returns the next logical state from the current one.

Parameters

current_state	current state

Returns

lifecycle_msgs::msg::State

7.5.3.23 update_homing_grp_state()

```
void bioscara_rviz_plugin::BioscaraPanel::update_homing_grp_state (
            void  )  [protected]
```

Enabl/Disables the homing group depending on the homing controller state.

7.5.3.24 update_homing_state_labels()

```
void bioscara_rviz_plugin::BioscaraPanel::update_homing_state_labels (
            void  )  [protected]
```

updates all homing state labels.

Calls set_homing_state_label() for each joint state label.

7.5.3.25 update_state_label_and_btn()

```
void bioscara_rviz_plugin::BioscaraPanel::update_state_label_and_btn (
            const std::unordered_map< std::string, NamedLifecycleState > & state_map,
            QLabel * state_label,
            QPushButton * en_button,
            const std::string & state_key )  [protected]
```

updates component state label and control button.

Calls set_state_label() for state_label and set_en_btn() for en_button.

Parameters

state_map	component state map
state_label	pointer to state label
en_button	pointer to enable button
state_key	string key to access in state_map

7.5.3.26 update_state_labels_and_btns()

```
void bioscara_rviz_plugin::BioscaraPanel::update_state_labels_and_btns (
            void  )  [protected]
```

Update all hardware and controller state lables.

calls update_state_label_and_btn() for each controller and hardware component.

7.5.4 Member Data Documentation

7.5.4.1 cm_state_subsription_

```
rclcpp::Subscription<ControllerManagerActivity>::SharedPtr bioscara_rviz_plugin::Bioscara↩
Panel::cm_state_subsription_  [protected]
```

Subscription to the the controller_manager state publisher.

Used to retrieve information about hardware and controller state changes.

7.5.4.2 configure_controller_client_

```
rclcpp::Client<ConfigureController>::SharedPtr bioscara_rviz_plugin::BioscaraPanel::configure↩
_controller_client_  [protected]
```

Service client to the "/controller_manager/configure_controller" service.

Used to request configuration and cleanup of controllers.

7.5.4.3 controller_states_

```
std::unordered_map<std::string, NamedLifecycleState> bioscara_rviz_plugin::BioscaraPanel↩
::controller_states_ [protected]
```

Map correlating controller name to its lifecycle_msgs/State.

Stored as NamedLifecycleState to also simply access controller name from map value.

7.5.4.4 hardware_state_client_

```
rclcpp::Client<SetHardwareComponentState>::SharedPtr bioscara_rviz_plugin::BioscaraPanel↩
::hardware_state_client_ [protected]
```

Service client to the "/controller_manager/set_hardware_component_state" service.

Used to request activation and deactivation of harddware components.

7.5.4.5 hardware_states_

```
std::unordered_map<std::string, NamedLifecycleState> bioscara_rviz_plugin::BioscaraPanel↩
::hardware_states_ [protected]
```

Map correlating hardware component name to its lifecycle_msgs/State.

Stored as NamedLifecycleState to also simply access component name from map value.

7.5.4.6 homing_publisher_

```
rclcpp::Publisher<DynamicInterfaceGroupValues>::SharedPtr bioscara_rviz_plugin::Bioscara↩
Panel::homing_publisher_ [protected]
```

Publisher to send commands to the homing_controller.

Used to start and stop homing of joints.

7.5.4.7 joint_state_subsription_

```
rclcpp::Subscription<DynamicJointState>::SharedPtr bioscara_rviz_plugin::BioscaraPanel::joint↩
_state_subsription_ [protected]
```

Subscription to the the joint state publisher state publisher.

Uses the "/dynamic_joint_states" to receive updates of all joints states.

7.5.4.8 joint_states_

```
std::unordered_map<std::string, InterfaceValue> bioscara_rviz_plugin::BioscaraPanel::joint_↩
states_ [protected]
```

Map storing all joint state interface values by each joint.

7.5.4.9 node_

```
rclcpp::Node::SharedPtr bioscara_rviz_plugin::BioscaraPanel::node_  [protected]
```

Pointer to the parent node. Used for access to the executor.

7.5.4.10 prune_timer_

```
rclcpp::TimerBase::SharedPtr bioscara_rviz_plugin::BioscaraPanel::prune_timer_  [protected]
```

Timer triggered every 5 s to prune unanswered service requests.

7.5.4.11 switch_controller_client_

```
rclcpp::Client<SwitchController>::SharedPtr bioscara_rviz_plugin::BioscaraPanel::switch_↩
controller_client_  [protected]
```

Service client to the "/controller_manager/switch_controller" service.

Used to request activation and deactivation of controllers.

7.5.4.12 ui_

```
Ui::BioscaraUI* bioscara_rviz_plugin::BioscaraPanel::ui_  [protected]
```

The QT object containing all UI elements.

Autogenerated from the "bioscara_rviz_plugin_frame.ui" file.

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

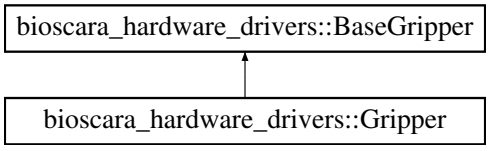
- lib/ros2_ws/src/bioscara_rviz_plugin/include/bioscara_rviz_plugin/bioscara_panel.hpp
- lib/ros2_ws/src/bioscara_rviz_plugin/src/bioscara_panel.cpp

7.6 bioscara_hardware_drivers::Gripper Class Reference

Child class implementing control of the hardware gripper.

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

Inheritance diagram for bioscara_hardware_drivers::Gripper:



Public Member Functions

- Gripper (float reduction, float offset, float min, float max, float backup_init_pos)
  Construct a new Gripper object.
- err_type_t enable (void) override
  Activates the the hardware.
- err_type_t disable (void) override
  Deactivates the hardware.
- err_type_t setPosition (float width) override
  Sets the gripper width in m.
- err_type_t setServoPosition (float angle)
  Sets the servo position of the gripper actuator in degrees.

Public Member Functions inherited from bioscara_hardware_drivers::BaseGripper

- BaseGripper (float reduction, float offset, float min, float max, float backup_init_pos)
  Construct a new BaseGripper object.
- ~BaseGripper (void)
  Destroy the BaseGripper object.
- virtual err_type_t init (void)
  Initializes the gripper.
- virtual err_type_t deinit (void)
  Deinitializes the gripper.
- virtual err_type_t getPosition (float &width)
  Gets the gripper width.
- virtual void setReduction (float reduction)
  Manually set reduction.
- virtual void setOffset (float offset)
  Manually set offset.

Protected Attributes

- RPI_PWM _pwm
  RPI_PWM object to generate PWM voltage for servo control.
- int _freq = 50
  PWM frequency in Hz.

Protected Attributes inherited from bioscara_hardware_drivers::BaseGripper

- float _reduction = 1
  gripper width to actuator reduction ratio
- float _offset = 0
  gripper width position offset
- float _min = 0
  gripper width lower limit
- float _max = 0
  gripper width upper limit
- float _backup_init_pos = 0.0
  Initial position assumed if none can be retrieved from the buffer file.
- float _pos = _backup_init_pos
  stored position
- float _pos_get = _pos
  reported position

Additional Inherited Members

Protected Member Functions inherited from bioscara_hardware_drivers::BaseGripper

- err_type_t save_last_position (float pos)

  Stores the latest position to the buffer file.
- err_type_t retrieve_last_position (float &pos)

  Retrieves the stored position from the buffer file.

7.6.1 Detailed Description

Child class implementing control of the hardware gripper.

Use this class if you wish to control the actual hardware by generating a PWM voltage on GPIO18 of the Raspberry Pi.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Gripper()

```
bioscara_hardware_drivers::Gripper::Gripper (
            float reduction,
            float offset,
            float min,
            float max,
            float backup_init_pos )
```

Construct a new Gripper object.

see the BaseGripper constructor for a description of the parameters.

7.6.3 Member Function Documentation

7.6.3.1 disable()

```
err_type_t bioscara_hardware_drivers::Gripper::disable (
            void  )  [override], [virtual]
```

Deactivates the hardware.

Invokes BaseGripper::disable() and stops the servo by disableing the PWM generation.

Returns

err_type_t::OK

Reimplemented from bioscara_hardware_drivers::BaseGripper.

7.6.3.2 enable()

```
err_type_t bioscara_hardware_drivers::Gripper::enable (
            void  )  [override], [virtual]
```

Activates the the hardware.

Invokes BaseGripper::enable() and starts the PWM generation but does not set a position. Must be called before a position is set. The PWM pin is GPIO18 on the Raspberry Pi 4. PWM chip is 0, channel 0.

Returns

err_type_t::OK on success, err_type_t::COMM_ERROR if PWM can not be enabled

Reimplemented from bioscara_hardware_drivers::BaseGripper.

7.6.3.3 setPosition()

```
err_type_t bioscara_hardware_drivers::Gripper::setPosition (
            float width )  [override], [virtual]
```

Sets the gripper width in m.

Invokes BaseGripper::setPosition(), transforms the desired gripper width to a servo angle using the JOINT2ACTUATOR() macro and finally sets the servo position using setServoPosition()

Parameters

width	width in m.

Returns

see setServoPosition()

Reimplemented from bioscara_hardware_drivers::BaseGripper.

7.6.3.4 setServoPosition()

```
err_type_t bioscara_hardware_drivers::Gripper::setServoPosition (
            float angle )
```

Sets the servo position of the gripper actuator in degrees.

Calculates a PWM dutycycle for the desired servo angle and invokes RPI_PWM::setDutyCycle() to set it.

Parameters

angle	in degrees.

Returns

err_type_t::OK on success, err_type_t::COMM_ERROR if the dutycycle can not be set.

7.6.4 Member Data Documentation

7.6.4.1 _freq

```
int bioscara_hardware_drivers::Gripper::_freq = 50  [protected]
```

PWM frequency in Hz.

7.6.4.2 _pwm

```
RPI_PWM bioscara_hardware_drivers::Gripper::_pwm  [protected]
```

RPI_PWM object to generate PWM voltage for servo control.

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

- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_↩ hardware_driver/mGripper.h
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mGripper.cpp

7.7 bioscara_hardware_interfaces::BioscaraGripperHardwareInterface↩ ::gripper_config_t Struct Reference

configuration structure holding the passed paramters from the ros2_control urdf

Public Attributes

- float reduction = 1
- float offset = 0
- float min
- float max
- float init_pos

7.7.1 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.

7.7.2 Member Data Documentation

7.7.2.1 init_pos

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t::init↩
_pos
```

7.7.2.2 max

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t::max
```

7.7.2.3 min

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t::min
```

7.7.2.4 offset

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t::offset
= 0
```

7.7.2.5 reduction

```
float bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t::reduction
= 1
```

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

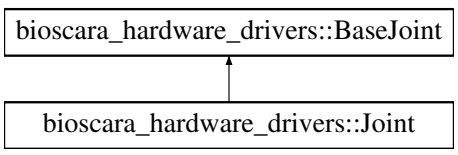
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/include/bioscara_gripper_↩ hardware_interface/gripper_hardware.hpp

7.8 bioscara_hardware_drivers::Joint Class Reference

Representing a single hardware joint connected via I2C.

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

Inheritance diagram for bioscara_hardware_drivers::Joint:



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)

  Destroy the Joint object.
- err_type_t init (void) override

  Initialize connection to a joint via I2C.
- err_type_t deinit (void) override

  Disconnects from a joint.
- err_type_t enable (u_int8_t driveCurrent, u_int8_t holdCurrent) override

  Engages the joint.
- err_type_t postHoming (void) override

  perform tasks after a non-blocking homing
- err_type_t getPosition (float &pos) override

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

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

  Move full steps.
- err_type_t getVelocity (float &vel) override

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

  Set the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.
- err_type_t checkOrientation (float angle=2.0) override

  Calls the checkOrientation method of the motor. Checks in which direction the motor is turning.
- err_type_t stop (void) override

  Stops the motor.
- err_type_t disableCL (void) override
- err_type_t setDriveCurrent (u_int8_t current) override

  Set the Drive Current.
- err_type_t setHoldCurrent (u_int8_t current) override

  Set the Hold Current.
- err_type_t setBrakeMode (u_int8_t mode) override

  Set Brake Mode.
- err_type_t setMaxAcceleration (float maxAccel) override

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

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

  Enable encoder stall detection of the joint.
- err_type_t getFlags (void) override

  Overload of getFlags(u_int8_t &flags)

Public Member Functions inherited from bioscara_hardware_drivers::BaseJoint

- BaseJoint (const std::string name)

  Create a Joint object.
- ~BaseJoint (void)

  Destroy the BaseJoint object.
- virtual err_type_t disable (void)

  disenganges the joint
- virtual err_type_t home (float velocity, u_int8_t sensitivity, u_int8_t current)

  Blocking implementation to home the joint.
- virtual err_type_t startHoming (float velocity, u_int8_t sensitivity, u_int8_t current)

  non-blocking implementation to home the joint
- virtual bool isHomed (void)

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

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

  Checks if the motor is stalled.
- virtual bool isBusy (void)

  Checks if the joint controller is busy processing a blocking command.
- virtual err_type_t getFlags (u_int8_t &flags)

  get the latest driver state flags from the joint
- virtual stp_reg_t getCurrentBCmd (void)

  get the currently active blocking command

Protected Member Functions

- err_type_t getHomingOffset (float &offset)

  Retrieves the homing position from the last homing.
- err_type_t setHomingOffset (const float offset)

  Stores the homing position on the joint.
- err_type_t _home (float velocity, u_int8_t sensitivity, u_int8_t current) override

  Call to start the homing sequence of a joint.
- err_type_t checkCom (void)

  Check if communication to the joint is established.

Protected Member Functions inherited from bioscara_hardware_drivers::BaseJoint

- virtual void wait_while_busy (const float period_ms)

  Blocking loop waiting for BUSY flag to reset.

Protected Attributes

- 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.

Protected Attributes inherited from bioscara_hardware_drivers::BaseJoint

- u_int8_t flags = 0b00001100

  State flags transmitted with every I2C transaction.
- stp_reg_t current_b_cmd = NONE

  Keeps track if a blocking command is being executed.

Private Member Functions

- template<typename T >
  int read (const stp_reg_t reg, T &data, u_int8_t &flags)

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

  Wrapper function to send command to the I2C slave.

Private Attributes

- int address

  I2C adress.
- int handle = -1

  I2C bus handle.

Additional Inherited Members

Public Types inherited from bioscara_hardware_drivers::BaseJoint

- 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 Attributes inherited from bioscara_hardware_drivers::BaseJoint

- std::string name

  Joint name for logging.

7.8.1 Detailed Description

Representing a single hardware joint connected via I2C.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Joint()

```
bioscara_hardware_drivers::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

name	string device name for identification
address	1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4
reduction	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
min	lower joint limit in joint units. J1: -3.04647 J2: -0.0016 J3: -2.62672 J4: -3.01069
max	upper joint limit in joint units. J1: 3.04647 J2: 0.3380 J3: 2.62672 J4: 3.01069

7.8.2.2 ∼Joint()

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

Destroy the Joint object.

Create a Joint object.

7.8.3 Member Function Documentation

7.8.3.1 _home()

```
err_type_t bioscara_hardware_drivers::Joint::_home (
            float velocity,
            u_int8_t sensitivity,
            u_int8_t current )  [override], [protected], [virtual]
```

Call to start the homing sequence of a joint.

The joint will start rotating with 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

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

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.8.3.2 checkCom()

```
err_type_t bioscara_hardware_drivers::Joint::checkCom (
            void  )  [protected]
```

Check if communication to the joint is established.

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

Returns

err_type_t

7.8.3.3 checkOrientation()

```
err_type_t bioscara_hardware_drivers::Joint::checkOrientation (
            float angle = 2.0 )  [override], [virtual]
```

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

Note

This is a blocking function.

Parameters

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.4 deinit()

```
err_type_t bioscara_hardware_drivers::Joint::deinit (
            void  )  [override], [virtual]
```

Disconnects from a joint.

Removes the joint from the I2C bus by invoking closeI2CDevHandle().

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.5 disableCL()

```
err_type_t bioscara_hardware_drivers::Joint::disableCL (
            void  )  [override], [virtual]
```

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.6 enable()

```
err_type_t bioscara_hardware_drivers::Joint::enable (
            u_int8_t driveCurrent,
            u_int8_t holdCurrent )  [override], [virtual]
```

Engages the joint.

This function prepares the motor for movement. After successfull execution the joint is ready to accept setPosition() and 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

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.7 enableStallguard()

```
err_type_t bioscara_hardware_drivers::Joint::enableStallguard (
            u_int8_t sensitivity )  [override], [virtual]
```

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 reenabling the joint using enable().

Note

If stall detection shall be enabled, invoke this method AFTER enabling the joint with enable().

Parameters

sensitivity	value of threshold. 0 - 255 where lower is more sensitive.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.8 getFlags()

```
err_type_t bioscara_hardware_drivers::Joint::getFlags (
            void  )  [override], [virtual]
```

Overload of getFlags(u_int8_t &flags)

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.9 getHomingOffset()

```
err_type_t bioscara_hardware_drivers::Joint::getHomingOffset (
            float & offset )  [protected]
```

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

err_type_t

7.8.3.10 getPosition()

```
err_type_t bioscara_hardware_drivers::Joint::getPosition (
            float & pos )  [override], [virtual]
```

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

pos	the current joint position in rad or m.

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.8.3.11 getVelocity()

```
err_type_t bioscara_hardware_drivers::Joint::getVelocity (
            float & vel )  [override], [virtual]
```

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

Parameters

vel	the current joint velocity in rad/s or m/s.

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.8.3.12 init()

```
err_type_t bioscara_hardware_drivers::Joint::init (
            void  )  [override], [virtual]
```

Initialize connection to a joint via I2C.

Adds the joint to the I2C bus and tests if is responsive by invoking checkCom(). If the joint is homed, retrieve the homing position stored on the joint by invoking getHomingOffset().

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.13 moveSteps()

```
err_type_t bioscara_hardware_drivers::Joint::moveSteps (
            int32_t steps )  [override], [virtual]
```

Move full steps.

This function can be called even when not homed.

Parameters

steps	number of full steps

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.14 postHoming()

```
err_type_t bioscara_hardware_drivers::Joint::postHoming (
            void  )  [override], [virtual]
```

perform tasks after a non-blocking homing

Invokes BaseJoint::postHoming() and additionally saves the homing offset to the joint controller setHomingOffset()

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.15 read()

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

Wrapper function to request data from the I2C slave.

Allocates a buffer of size sizeof(T) + RFLAGS_SIZE. Invokes readFromI2CDev(), and copies the received payload to *data* and the transmisison flags to *flags*. See BaseJoint::flags for details.

Template Parameters

T	Datatype of value to be transmitted

Parameters

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

Returns

0 on OK, negative on error

7.8.3.16 setBrakeMode()

```
err_type_t bioscara_hardware_drivers::Joint::setBrakeMode (
            u_int8_t mode )  [override], [virtual]
```

Set Brake Mode.

Parameters

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.17 setDriveCurrent()

```
err_type_t bioscara_hardware_drivers::Joint::setDriveCurrent (
            u_int8_t current )  [override], [virtual]
```

Set the Drive Current.

Parameters

current	0% - 100% of driver current

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.18 setHoldCurrent()

```
err_type_t bioscara_hardware_drivers::Joint::setHoldCurrent (
            u_int8_t current )  [override], [virtual]
```

Set the Hold Current.

Parameters

current	0% - 100% of driver current

Returns

err_type_t -1 on communication error,

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.19 setHomingOffset()

```
err_type_t bioscara_hardware_drivers::Joint::setHomingOffset (
            const float offset )  [protected]
```

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

err_type_t

7.8.3.20 setMaxAcceleration()

```
err_type_t bioscara_hardware_drivers::Joint::setMaxAcceleration (
            float maxAccel )  [override], [virtual]
```

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

Parameters

maxAccel	maximum joint acceleration.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.21 setMaxVelocity()

```
err_type_t bioscara_hardware_drivers::Joint::setMaxVelocity (
            float maxVel )  [override], [virtual]
```

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

Parameters

maxVel	maximum joint velocity.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.22 setPosition()

```
err_type_t bioscara_hardware_drivers::Joint::setPosition (
            float pos )  [override], [virtual]
```

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

Parameters

pos	the commanded joint position in rad or m.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.23 setVelocity()

```
err_type_t bioscara_hardware_drivers::Joint::setVelocity (
            float vel )  [override], [virtual]
```

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

Parameters

vel	the commanded joint velocity in rad/s or m/s.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.24 stop()

```
err_type_t bioscara_hardware_drivers::Joint::stop (
            void  )  [override], [virtual]
```

Stops the motor.

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.8.3.25 write()

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

Wrapper function to send command to the I2C slave.

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

Template Parameters

T	Datatype of value to be transmitted

Parameters

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

Returns

0 on OK, negative on error

7.8.4 Member Data Documentation

7.8.4.1 address

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

I2C adress.

7.8.4.2 handle

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

I2C bus handle.

7.8.4.3 max

```
float bioscara_hardware_drivers::Joint::max = 0  [protected]
```

Joint upper limit.

7.8.4.4 min

```
float bioscara_hardware_drivers::Joint::min = 0  [protected]
```

Joint lower limit.

7.8.4.5 offset

```
float bioscara_hardware_drivers::Joint::offset = 0  [protected]
```

Joint position offset.

7.8.4.6 reduction

```
float bioscara_hardware_drivers::Joint::reduction = 1  [protected]
```

Joint to actuator reduction ratio.

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

- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩ driver/mJoint.h
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩ driver/mJoint.hpp
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mJoint.cpp

7.9 bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩ ::joint_config_t Struct Reference

configuration structure holding the passed paramters from the ros2_control urdf

Public Attributes

- int i2c_address

  [int, HEX] I2C device adress
- float reduction = 1

  [float] Gear reduction of the joint
- float min

  [float] Lower joint limit in joint units
- float max

  [float] Upper joint limit in joint units
- u_int8_t drive_current

  [int, DEC] Drive current of stepper driver in 0-100 % of 2.5A output (check uStepper doc.)
- u_int8_t hold_current

  [int, DEC] Hold current of stepper driver in 0-100 % of 2.5A output (check uStepper doc.)
- u_int8_t stall_threshold

  [int, DEC] Stall protection threshold in 0-255, where lower is more sensitive.
- float max_velocity

  [float] Maximum permitted joint velocity.
- float max_acceleration

  [float] Maximum permitted joint acceleration (and deceleration).
- joint_homing_config_t homing

  Holding the joint_homing_config_t configruation values.

7.9.1 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.

7.9.2 Member Data Documentation

7.9.2.1 drive_current

```
u_int8_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::drive_↩
current
```

[int, DEC] Drive current of stepper driver in 0-100 % of 2.5A output (check uStepper doc.)

Set when joint is enabled, see bioscara_hardware_drivers::BaseJoint::enable().

7.9.2.2 hold_current

```
u_int8_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::hold_↩
current
```

[int, DEC] Hold current of stepper driver in 0-100 % of 2.5A output (check uStepper doc.)

Set when joint is enabled, see bioscara_hardware_drivers::BaseJoint::enable().

7.9.2.3 homing

```
joint_homing_config_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_↩
config_t::homing
```

Holding the joint_homing_config_t configruation values.

7.9.2.4 i2c_address

```
int bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::i2c_address
```

[int, HEX] I2C device adress

1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4

7.9.2.5 max

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::max
```

[float] Upper joint limit in joint units

Initial values are (subject to tuning)
J1: 3.04647
J2: 0.3380
J3: 2.62672
J4: 3.01069

7.9.2.6 max_acceleration

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::max_acceleration
```

[float] Maximum permitted joint acceleration (and deceleration).

In rad/s^2 or m/s^2 for cylindrical and prismatic joints respectively. Set when joint is enabled, see bioscara_hardware_drivers::BaseJoint::setMaxAcceleration().

Note

This is the "last" limit and shall ALWAYS be greater than all limits set in the controller and application (MoveIt) configuration.

7.9.2.7 max_velocity

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::max_velocity
```

[float] Maximum permitted joint velocity.

In rad/s or m/s for cylindrical and prismatic joints respectively. Set when joint is enabled, see bioscara_hardware_drivers::BaseJoint::se

Note

This is the "last" limit and shall ALWAYS be greater than all limits set in the controller and application (MoveIt) configuration.

7.9.2.8 min

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::min
```

[float] Lower joint limit in joint units

Initial values are (subject to tuning)
J1: -3.04647
J2: -0.0016
J3: -2.62672
J4: -3.01069

7.9.2.9 reduction

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::reduction =
1
```

[float] 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, positive rotation makes the joint go down (negative), hence the minus sign for correction)
J3: 24
J4: 12

7.9.2.10 stall_threshold

```
u_int8_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t::stall_↩
threshold
```

[int, DEC] Stall protection threshold in 0-255, where lower is more sensitive.

If the PID error exceeds the set threshold a stall is triggered and the motor disabled. Set when joint is enabled, see bioscara_hardware_drivers::BaseJoint::enableStallguard().

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

- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_↩ interface/arm_hardware.hpp

7.10 bioscara_hardware_interfaces::BioscaraArmHardwareInterface↩ ::joint_homing_config_t Struct Reference

configuration structure holding the passed homing paramters from the ros2_control urdf

Public Attributes

- float speed

  [float] Signed homing velocity.
- u_int8_t threshold

  [int, DEC] Encoder error threshold 0 to 255.
- u_int8_t current

  [int, DEC] Homing current between 0 and 100.
- float acceleration = 0.01

  [float] Joint acceleration during homing.

7.10.1 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. See bioscara_hardware_drivers::BaseJoint for the description of the homing procedure.

7.10.2 Member Data Documentation

7.10.2.1 acceleration

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t↩
::acceleration = 0.01
```

[float] Joint acceleration during homing.

In rad/s^2 or m/s^2 for cylindrical and prismatic joints respectively. Accelerating too fast can trigger a stall and hence false positve homing. See joint_config_t::max_acceleration.

7.10.2.2 current

```
u_int8_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t↩
::current
```

[int, DEC] Homing current between 0 and 100.

Determines how easy it is to stop the motor and thereby provoke a stall. See joint_config_t::drive_current.

7.10.2.3 speed

```
float bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t::speed
```

[float] Signed homing velocity.

In rad/s or m/s for cylindrical and prismatic joints respectively. Must satisfy: `1.0 < RAD2DEG(JOINT2↩ ACTUATOR(<speed>, reduction, 0)) / 6 < 250.0`

7.10.2.4 threshold

```
u_int8_t bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t↩
::threshold
```

[int, DEC] Encoder error threshold 0 to 255.

Lower values make the homing more sensitve but also more susceptible to false positives.

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

- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_↩ interface/arm_hardware.hpp

7.11 bioscara_joint_firmware::Lowpass Class Reference

Simple discrete IIR lowpass filter.

```
#include <filters.h>
```

Public Member Functions

- Lowpass (float gain=1, float sampleTime=0.1, float timeconstant=1.0)
  Constucts and initializes the Lowpass filter.
- float updateState (float u)
  Update the filter state. Must be called with the constant period Ts.
- void resetState (void)
  Resets the filter state x to 0.0.

Protected Attributes

- float K
  Filter gain.
- float Ts
  Filter sampling time.
- float tau
  Filter timeconstant.
- float x
  Filter state.

7.11.1 Detailed Description

Simple discrete IIR lowpass filter.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Lowpass()

```
bioscara_joint_firmware::Lowpass::Lowpass (
            float gain = 1,
            float sampleTime = 0.1,
            float timeconstant = 1.0 )  [inline]
```

Constucts and initializes the Lowpass filter.

Parameters

gain	
sampleTime	
timeconstant	

7.11.3 Member Function Documentation

7.11.3.1 resetState()

```
void bioscara_joint_firmware::Lowpass::resetState (
            void  )  [inline]
```

Resets the filter state x to 0.0.

7.11.3.2 updateState()

```
float bioscara_joint_firmware::Lowpass::updateState (
            float u )  [inline]
```

Update the filter state. Must be called with the constant period Ts.

Returns

The updated filter state

7.11.4 Member Data Documentation

7.11.4.1 K

```
float bioscara_joint_firmware::Lowpass::K  [protected]
```

Filter gain.

7.11.4.2 tau

```
float bioscara_joint_firmware::Lowpass::tau  [protected]
```

Filter timeconstant.

7.11.4.3 Ts

```
float bioscara_joint_firmware::Lowpass::Ts  [protected]
```

Filter sampling time.

7.11.4.4 x

```
float bioscara_joint_firmware::Lowpass::x  [protected]
```

Filter state.

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

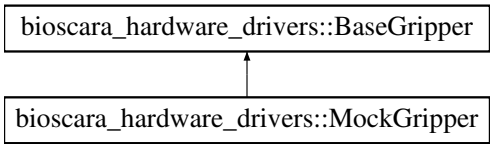
- lib/joint_firmware/joint/filters.h

7.12 bioscara_hardware_drivers::MockGripper Class Reference

Child class for to mock the gripper hardware.

`#include <mMockGripper.h>`

Inheritance diagram for bioscara_hardware_drivers::MockGripper:



Public Member Functions

- MockGripper (float reduction, float offset, float min, float max, float backup_init_pos)
  Construct a new MockGripper object.

Public Member Functions inherited from bioscara_hardware_drivers::BaseGripper

- BaseGripper (float reduction, float offset, float min, float max, float backup_init_pos)
  Construct a new BaseGripper object.
- ~BaseGripper (void)
  Destroy the BaseGripper object.
- virtual err_type_t init (void)
  Initializes the gripper.
- virtual err_type_t deinit (void)
  Deinitializes the gripper.
- virtual err_type_t enable (void)
  Enables the gripper.
- virtual err_type_t disable (void)
  Disables the gripper.
- virtual err_type_t setPosition (float width)
  Sets the gripper width in m.
- virtual err_type_t getPosition (float &width)
  Gets the gripper width.
- virtual void setReduction (float reduction)
  Manually set reduction.
- virtual void setOffset (float offset)
  Manually set offset.

Additional Inherited Members

Protected Member Functions inherited from bioscara_hardware_drivers::BaseGripper

- err_type_t save_last_position (float pos)
  Stores the latest position to the buffer file.
- err_type_t retrieve_last_position (float &pos)
  Retrieves the stored position from the buffer file.

Protected Attributes inherited from bioscara_hardware_drivers::BaseGripper

- float _reduction = 1
  gripper width to actuator reduction ratio
- float _offset = 0
  gripper width position offset
- float _min = 0
  gripper width lower limit
- float _max = 0
  gripper width upper limit
- float _backup_init_pos = 0.0
  Initial position assumed if none can be retrieved from the buffer file.
- float _pos = _backup_init_pos
  stored position
- float _pos_get = _pos
  reported position

7.12.1 Detailed Description

Child class for to mock the gripper hardware.

Use this class if you dont wish to use the gripper hardware. Commands are simply mirrored.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 MockGripper()

```
bioscara_hardware_drivers::MockGripper::MockGripper (
            float reduction,
            float offset,
            float min,
            float max,
            float backup_init_pos )
```

Construct a new MockGripper object.

see the BaseGripper constructor for a description of the parameters.

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

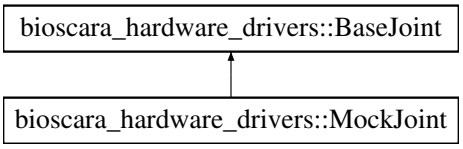
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_↩ hardware_driver/mMockGripper.h
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mMockGripper.cpp

7.13 bioscara_hardware_drivers::MockJoint Class Reference

Representing a single joint mocking the hardware joint.

`#include <mMockJoint.h>`

Inheritance diagram for bioscara_hardware_drivers::MockJoint:



Public Member Functions

- MockJoint (const std::string name)
- err_type_t enable (u_int8_t driveCurrent, u_int8_t holdCurrent) override

  Engages the joint.
- err_type_t disable (void) override

  disenganges the joint
- err_type_t getPosition (float &pos) override

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

  get the current joint position in radians or m for cylindrical and prismatic joints respectively.
- err_type_t getVelocity (float &vel) override

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

  Set the current joint velocity in radians/s or m/s for cylindrical and prismatic joints respectively.
- err_type_t checkOrientation (float angle=2.0) override

  Calls the checkOrientation method of the motor. Checks in which direction the motor is turning.
- err_type_t stop (void) override

  Stops the motor.
- err_type_t getFlags (void) override

  Overload of getFlags(u_int8_t &flags)
- bool isHomed (void) override

  Checks the state if the motor is homed.

Public Member Functions inherited from bioscara_hardware_drivers::BaseJoint

- BaseJoint (const std::string name)

  Create a Joint object.
- ~BaseJoint (void)

  Destroy the BaseJoint object.
- virtual err_type_t init (void)

  Initialize the joint communication.
- virtual err_type_t deinit (void)

  Denitialize the joint communication.
- virtual err_type_t home (float velocity, u_int8_t sensitivity, u_int8_t current)

  Blocking implementation to home the joint.

- virtual err_type_t startHoming (float velocity, u_int8_t sensitivity, u_int8_t current)

  non-blocking implementation to home the joint
- virtual err_type_t postHoming (void)

  perform tasks after a non-blocking homing
- virtual err_type_t moveSteps (int32_t steps)

  Move full steps.
- virtual err_type_t disableCL (void)
- virtual err_type_t setDriveCurrent (u_int8_t current)

  Set the Drive Current.
- virtual err_type_t setHoldCurrent (u_int8_t current)

  Set the Hold Current.
- virtual err_type_t setBrakeMode (u_int8_t mode)

  Set Brake Mode.
- virtual err_type_t setMaxAcceleration (float maxAccel)

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

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

  Enable encoder stall detection of the joint.
- virtual bool isEnabled (void)

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

  Checks if the motor is stalled.
- virtual bool isBusy (void)

  Checks if the joint controller is busy processing a blocking command.
- virtual err_type_t getFlags (u_int8_t &flags)

  get the latest driver state flags from the joint
- virtual stp_reg_t getCurrentBCmd (void)

  get the currently active blocking command

Protected Member Functions

- err_type_t _home (float velocity, u_int8_t sensitivity, u_int8_t current)

  Call to start the homing sequence of a joint.

Protected Member Functions inherited from bioscara_hardware_drivers::BaseJoint

- virtual void wait_while_busy (const float period_ms)

  Blocking loop waiting for BUSY flag to reset.

Private Member Functions

- float getDeltaT (std::chrono::_V2::system_clock::time_point &last_call, bool update=true)

  Compute the time since the given time point.

Private Attributes

- float q = 0.0

  position
- float qd = 0.0

  velocity
- std::chrono::_V2::system_clock::time_point last_set_position = std::chrono::high_resolution_clock::now()
- std::chrono::_V2::system_clock::time_point last_set_velocity = last_set_position
- std::chrono::_V2::system_clock::time_point async_start_time = last_set_position
- stp_reg_t op_mode = NONE

Additional Inherited Members

Public Types inherited from bioscara_hardware_drivers::BaseJoint

- 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 Attributes inherited from bioscara_hardware_drivers::BaseJoint

- std::string name

  Joint name for logging.

Protected Attributes inherited from bioscara_hardware_drivers::BaseJoint

- u_int8_t flags = 0b00001100

  State flags transmitted with every I2C transaction.
- stp_reg_t current_b_cmd = NONE

  Keeps track if a blocking command is being executed.

7.13.1 Detailed Description

Representing a single joint mocking the hardware joint.

This mock hardware implementation simply mirrors (potentially delayed) commands.

Note

Much of this implementation is very basic not well written.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 MockJoint()

```
bioscara_hardware_drivers::MockJoint::MockJoint (
            const std::string name )
```

7.13.3 Member Function Documentation

7.13.3.1 _home()

```
err_type_t bioscara_hardware_drivers::MockJoint::_home (
            float velocity,
            u_int8_t sensitivity,
            u_int8_t current )  [protected], [virtual]
```

Call to start the homing sequence of a joint.

The joint will start rotating with 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

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

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.13.3.2 checkOrientation()

```
err_type_t bioscara_hardware_drivers::MockJoint::checkOrientation (
            float angle = 2.0 )  [override], [virtual]
```

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

Note

This is a blocking function.

Parameters

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.3 disable()

```
err_type_t bioscara_hardware_drivers::MockJoint::disable (
            void  )  [override], [virtual]
```

disenganges the joint

invokes stop(), sets hold and drive current to 0 and sets the the joint brake mode to freewheeling

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.4 enable()

```
err_type_t bioscara_hardware_drivers::MockJoint::enable (
            u_int8_t driveCurrent,
            u_int8_t holdCurrent )  [override], [virtual]
```

Engages the joint.

This function prepares the motor for movement. After successfull execution the joint is ready to accept setPosition() and 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

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.5 getDeltaT()

```
float bioscara_hardware_drivers::MockJoint::getDeltaT (
            std::chrono::_V2::system_clock::time_point & last_call,
            bool update = true )  [private]
```

Compute the time since the given time point.

Parameters

last_call	the time point to calculate the duration
update	If true compute the duration and also reset *last_call* to current time

Returns

elapsed time in seconds since *last_call*

7.13.3.6 getFlags()

```
err_type_t bioscara_hardware_drivers::MockJoint::getFlags (
            void  )  [override], [virtual]
```

Overload of getFlags(u_int8_t &flags)

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.7 getPosition()

```
err_type_t bioscara_hardware_drivers::MockJoint::getPosition (
            float & pos )  [override], [virtual]
```

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

pos	the current joint position in rad or m.

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.13.3.8 getVelocity()

```
err_type_t bioscara_hardware_drivers::MockJoint::getVelocity (
            float & vel )  [override], [virtual]
```

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

Parameters

vel	the current joint velocity in rad/s or m/s.

Returns

err_type_t

Implements bioscara_hardware_drivers::BaseJoint.

7.13.3.9 isHomed()

```
bool bioscara_hardware_drivers::MockJoint::isHomed (
            void  )  [override], [virtual]
```

Checks the state if the motor is homed.

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

Returns

true if the motor is homed, false if not.

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.10 setPosition()

```
err_type_t bioscara_hardware_drivers::MockJoint::setPosition (
            float pos )  [override], [virtual]
```

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

Parameters

pos	the commanded joint position in rad or m.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.11 setVelocity()

```
err_type_t bioscara_hardware_drivers::MockJoint::setVelocity (
            float vel )  [override], [virtual]
```

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

Parameters

vel	the commanded joint velocity in rad/s or m/s.

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.3.12 stop()

```
err_type_t bioscara_hardware_drivers::MockJoint::stop (
            void  )  [override], [virtual]
```

Stops the motor.

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

Returns

err_type_t

Reimplemented from bioscara_hardware_drivers::BaseJoint.

7.13.4 Member Data Documentation

7.13.4.1 async_start_time

```
std::chrono::_V2::system_clock::time_point bioscara_hardware_drivers::MockJoint::async_start↩
_time = last_set_position  [private]
```

7.13.4.2 last_set_position

```
std::chrono::_V2::system_clock::time_point bioscara_hardware_drivers::MockJoint::last_set_↩
position = std::chrono::high_resolution_clock::now()  [private]
```

7.13.4.3 last_set_velocity

```
std::chrono::_V2::system_clock::time_point bioscara_hardware_drivers::MockJoint::last_set_↩
velocity = last_set_position  [private]
```

7.13.4.4 op_mode

```
stp_reg_t bioscara_hardware_drivers::MockJoint::op_mode = NONE  [private]
```

7.13.4.5 q

```
float bioscara_hardware_drivers::MockJoint::q = 0.0  [private]
```

position

7.13.4.6 qd

```
float bioscara_hardware_drivers::MockJoint::qd = 0.0  [private]
```

velocity

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

- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_↩ driver/mMockJoint.h
- lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mMockJoint.cpp

7.14 bioscara_joint_firmware::MovMax Class Reference

Simple FIR moving maximum filter.

`#include <filters.h>`

Public Member Functions

- MovMax (float windowSize)

  Constucts and initializes the MovMax filter.
- float updateState (float u)

  Update and return the filter state.

Protected Attributes

- unsigned int M = 200

  Window Size.
- float * cb_data

  Pointer to ring buffer holding the data.
- unsigned int cb_index

  Current ringbuffer index.

7.14.1 Detailed Description

Simple FIR moving maximum filter.

Implements a circular ringbuffer holding the sampled data over the window size M

7.14.2 Constructor & Destructor Documentation

7.14.2.1 MovMax()

```
bioscara_joint_firmware::MovMax::MovMax (
            float windowSize )  [inline]
```

Constucts and initializes the MovMax filter.

Allocates a ring buffer cb_data of size M.

Parameters

windowSize	the window size

7.14.3 Member Function Documentation

7.14.3.1 updateState()

```
float bioscara_joint_firmware::MovMax::updateState (
            float u )  [inline]
```

Update and return the filter state.

Returns

The updated filter state

7.14.4 Member Data Documentation

7.14.4.1 cb_data

```
float* bioscara_joint_firmware::MovMax::cb_data  [protected]
```

Pointer to ring buffer holding the data.

7.14.4.2 cb_index

```
unsigned int bioscara_joint_firmware::MovMax::cb_index  [protected]
```

Current ringbuffer index.

7.14.4.3 M

```
unsigned int bioscara_joint_firmware::MovMax::M = 200  [protected]
```

Window Size.

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

- lib/joint_firmware/joint/filters.h

7.15 RPI_PWM Class Reference

Class to create a Pulse Width Modulated (PWM) signal on 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)

  Starts the PWM.
- void stop ()

  Stops the PWM.
- ~RPI_PWM ()

  Destroy the RPI_PWM object.
- int setDutyCycle (float v) const

  Sets the duty cycle in percent 0 - 100.

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

7.15.1 Detailed Description

Class to create a Pulse Width Modulated (PWM) signal on the Raspberry PI 4 and 5.

Based on https://github.com/berndporr/rpi_pwm/blob/main/rpi_pwm.h and slightly modified.

For servo control it is important to use a hardware generated PWM, since software PWM, like it is created by the lgpio library, is subject to timing jitter which can cause the servo to figet which might lead to overheating and wear.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 ~RPI_PWM()

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

Destroy the RPI_PWM object.

Invokes disable()

7.15.3 Member Function Documentation

7.15.3.1 disable()

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

7.15.3.2 enable()

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

7.15.3.3 setDutyCycle()

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

Sets the duty cycle in percent 0 - 100.

Parameters

v	The duty cycle in percent.

Returns

>0 on success and -1 after an error.

7.15.3.4 setDutyCycleNS()

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

7.15.3.5 setPeriod()

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

7.15.3.6 start()

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

Starts the PWM.

Parameters

channel	The GPIO channel
frequency	The PWM frequency
duty_cycle	The initial duty cycle of the PWM (default 0)
chip	The chip number

Returns

>0 on success and -1 if an error has happened.

7.15.3.7 stop()

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

Stops the PWM.

7.15.3.8 writeSYS()

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

7.15.4 Member Data Documentation

7.15.4.1 chippath

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

7.15.4.2 per

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

7.15.4.3 pwmpath

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

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

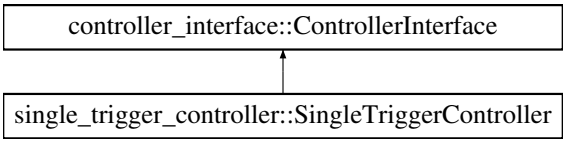
- lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_↩ hardware_driver/uPWM.h

7.16 single_trigger_controller::SingleTriggerController Class Reference

A general purpose controller which can be used to trigger command interfaces.

`#include <single_trigger_controller.hpp>`

Inheritance diagram for single_trigger_controller::SingleTriggerController:



Public Member Functions

- SingleTriggerController ()
- controller_interface::InterfaceConfiguration command_interface_configuration () const override

  Sets the command interface configuration to INDIVIDUAL.
- controller_interface::InterfaceConfiguration state_interface_configuration () const override

  Sets the state interface configuration to INDIVIDUAL.
- CallbackReturn on_init () override

  Called on initialization to the `unconfigured` state.
- CallbackReturn on_configure (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `unconfigured` to the `inactive` state.
- CallbackReturn on_activate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `inactive` to the `active` state.
- CallbackReturn on_deactivate (const rclcpp_lifecycle::State &previous_state) override

  Called on the transistion from the `active` to the `inactive` state.
- controller_interface::return_type update (const rclcpp::Time &time, const rclcpp::Duration &period) override

  Realtime update method called when in `active` state.

Protected Attributes

- InterfacesNames command_interface_types_

  vector of string holding all command interface names as configured from the parameters.
- InterfacesNames state_interface_types_

  vector of string holding all state interface names as configured from the parameters.
- MapOfReferencesToCommandInterfaces command_interfaces_map_

  map containing a reference to the actual hardwares command interfaces
- MapOfReferencesToStateInterfaces state_interfaces_map_

  map containing a reference to the actual hardwares state interfaces
- rclcpp::Subscription< CmdType >::SharedPtr command_subscriber_ {}

  Subscriber object for the '~/commands' topic.
- std::shared_ptr< rclcpp::Publisher< StateType > > state_publisher_ {}

  Publisher object to the '~/states' topic.
- std::shared_ptr< realtime_tools::RealtimePublisher< StateType > > realtime_state_publisher_ {}

  Realtime publisher wrapper of the state_publisher_.
- StateType state_msg_

  Message object holding the states to be published.
- std::shared_ptr< single_trigger_controller_parameters::ParamListener > param_listener_ {}

  parameter listerner object.
- single_trigger_controller_parameters::Params params_

  Parameter structure holding controller configuration parameters.

Private Member Functions

- void store_command_interface_types ()

  Stores all configured command interfaces in command_interface_types_.
- void store_state_interface_types ()

  Stores all configured state interfaces in state_interface_types_.
- void initialize_state_msg ()

  Initializes the state message state_msg_ with NaN values for each configured state interface.
- void update_states ()

Publishes current state values.

- controller_interface::return_type update_commands (const CmdType &commands)

  Callback method for received command msg. Calls apply_command() for each command interface.

- template<typename T >
  std::unordered_map< std::string, std::reference_wrapper< T > > create_map_of_references_to_interfaces (const InterfacesNames &interfaces_from_params, std::vector< T > &configured_interfaces)

  Create a map of references to the hardware components interfaces.

- template<typename T >
  bool check_if_configured_interfaces_matches_received (const InterfacesNames &interfaces_from_params, const T &configured_interfaces)

  performs checks on the configured interfaces vs received interfaces.

- void apply_state_value (StateType &state_msg, std::size_t index, std::size_t interface_index) const

  retrieves the state values from the hardware and populates the state_msg with the result.

- void apply_command (const CmdType &commands, std::size_t index, std::size_t command_interface_index) const

  *calls the set_value() method for the hardwares command interface references from the command_interfaces_map_*

- InterfacesNames get_state_interfaces_names (const std::string &name) const

  Gets the all qualified state interface names from the state_interface_types_ for the joint matching `name`.

- bool update_dynamic_map_parameters ()

  Update the controller parameters.

7.16.1 Detailed Description

A general purpose controller which can be used to trigger command interfaces.

This controller is a derivative of the ros2_control `GpioCommandController`. The key differences between this controller and the GpioCommandController is two-fold:

1. Besides GPIO interfaces, this controller allows Joint interfaces to be used, enabling the user to utilize the controller in broader applications and simplifying the URDF description.
2. The GpioCommandController wirtes the latest received command to the configured state interfaces on every update cycle. On the contrary, the SingleTriggerController only writes to the command interfaces once in the subscription callback when a new command has been received. The user can for example leverage this by clearing the command interface (writing NaN to it) in the hardware interface once the latest trigger has been handled.

The controller subscribes and publishes to the same topics as the GpioCommandController.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 SingleTriggerController()

```
single_trigger_controller::SingleTriggerController::SingleTriggerController ( )
```

7.16.3 Member Function Documentation

7.16.3.1 apply_command()

```
void single_trigger_controller::SingleTriggerController::apply_command (
            const CmdType & commands,
            std::size_t index,
            std::size_t command_interface_index ) const  [private]
```

calls the set_value() method for the hardwares command interface references from the command_interfaces_map_

Parameters

commands	
index	
command_interface_index	

7.16.3.2 apply_state_value()

```
void single_trigger_controller::SingleTriggerController::apply_state_value (
            StateType & state_msg,
            std::size_t index,
            std::size_t interface_index ) const  [private]
```

retrieves the state values from the hardware and populates the state_msg with the result.

calls the get_optional() method of each state interface from the state_interfaces_map_ for the state referenced by the index and interface_index.

Parameters

state_msg	
index	
interface_index	

7.16.3.3 check_if_configured_interfaces_matches_received()

```
template<typename T >
bool single_trigger_controller::SingleTriggerController::check_if_configured_interfaces_↩
matches_received (
            const InterfacesNames & interfaces_from_params,
            const T & configured_interfaces )  [private]
```

performs checks on the configured interfaces vs received interfaces.

Template Parameters

T	

Parameters

interfaces_from_params	
configured_interfaces	

Returns

std::unordered_map<std::string, std::reference_wrapper<T>>

7.16.3.4 command_interface_configuration()

```
controller_interface::InterfaceConfiguration single_trigger_controller::SingleTriggerController↩
::command_interface_configuration ( ) const  [override]
```

Sets the command interface configuration to INDIVIDUAL.

See ControllerInterfaceBase::command_interface_configuration()

Returns

controller_interface::InterfaceConfiguration

7.16.3.5 create_map_of_references_to_interfaces()

```
template<typename T >
std::unordered_map< std::string, std::reference_wrapper< T > > single_trigger_controller::↩
SingleTriggerController::create_map_of_references_to_interfaces (
            const InterfacesNames & interfaces_from_params,
            std::vector< T > & configured_interfaces )  [private]
```

Create a map of references to the hardware components interfaces.

Template Parameters

T	

Parameters

interfaces_from_params	
configured_interfaces	

Returns

std::unordered_map<std::string, std::reference_wrapper<T>>

7.16.3.6 get_state_interfaces_names()

```
InterfacesNames single_trigger_controller::SingleTriggerController::get_state_interfaces_names
(
            const std::string & name ) const  [private]
```

Gets the all qualified state interface names from the state_interface_types_ for the joint matching `name`.

Parameters

name	

Returns

InterfacesNames

7.16.3.7 initialize_state_msg()

```
void single_trigger_controller::SingleTriggerController::initialize_state_msg ( )  [private]
```

Initializes the state message state_msg_ with NaN values for each configured state interface.

For example the state message could look like this:

```
{
 header:
 {
   stamp: 12345678
 },
 interface_groups: ['j1', 'gpio1'],
 interface_values:
 [
   {
     interface_names: ['home'],
     values: [NaN]
   },
   {
     interface_names: ['ai0', 'ai1'],
     values: [NaN, NaN]
   }
 ]
}
```

7.16.3.8 on_activate()

```
CallbackReturn single_trigger_controller::SingleTriggerController::on_activate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Populates command_interfaces_map_ and state_interfaces_map_ by invoking create_map_of_references_to_interfaces() for both command and state interfaces.

Initializes an empty state message by calling initialize_state_msg().

Parameters

previous_state	

Returns

CallbackReturn

7.16.3.9 on_configure()

```
CallbackReturn single_trigger_controller::SingleTriggerController::on_configure (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Stores all configured command and state interfaces using the store_command_interface_types() and store_state_interface_types() methods.

Creates a subscription to the '~/commands' topic if command interfaces are configured. Creates a realtime publisher to the '~/states' topic is state interfaces are configured.

7.16.3.10 on_deactivate()

```
CallbackReturn single_trigger_controller::SingleTriggerController::on_deactivate (
            const rclcpp_lifecycle::State & previous_state )  [override]
```

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

Parameters

previous_state	

Returns

CallbackReturn

7.16.3.11 on_init()

```
CallbackReturn single_trigger_controller::SingleTriggerController::on_init ( )  [override]
```

Called on initialization to the `unconfigured` state.

Retrieves and stores the parameters supplied to the controller in params_

Returns

hardware_interface::CallbackReturn

7.16.3.12 state_interface_configuration()

```
controller_interface::InterfaceConfiguration single_trigger_controller::SingleTriggerController↩
::state_interface_configuration ( ) const  [override]
```

Sets the state interface configuration to INDIVIDUAL.

See ControllerInterfaceBase::state_interface_configuration()

Returns

controller_interface::InterfaceConfiguration

7.16.3.13 store_command_interface_types()

```
void single_trigger_controller::SingleTriggerController::store_command_interface_types ( )
[private]
```

Stores all configured command interfaces in command_interface_types_.

7.16.3.14 store_state_interface_types()

```
void single_trigger_controller::SingleTriggerController::store_state_interface_types ( )  [private]
```

Stores all configured state interfaces in state_interface_types_.

7.16.3.15 update()

```
controller_interface::return_type single_trigger_controller::SingleTriggerController::update (
            const rclcpp::Time & time,
            const rclcpp::Duration & period )  [override]
```

Realtime update method called when in `active` state.

Only publishes the states by invoking update_states().

Parameters

time	
period	

Returns

controller_interface::return_type

7.16.3.16 update_commands()

```
controller_interface::return_type single_trigger_controller::SingleTriggerController::update↩
_commands (
            const CmdType & commands )  [private]
```

Callback method for received command msg. Calls apply_command() for each command interface.

Sets all command interaces to 0.0 if an empty command message is received.

Parameters

commands	

Returns

controller_interface::return_type

7.16.3.17 update_dynamic_map_parameters()

```
bool single_trigger_controller::SingleTriggerController::update_dynamic_map_parameters ( )
[private]
```

Update the controller parameters.

Only called in on_configure() to update the parameters in case they have been changed.

Returns

true

7.16.3.18 update_states()

```
void single_trigger_controller::SingleTriggerController::update_states ( )  [private]
```

Publishes current state values.

Loops over all interface groups and command interfaces and calls the apply_state_value(). Then publishes the result using the realtime_state_publisher_.

7.16.4 Member Data Documentation

7.16.4.1 command_interface_types_

```
InterfacesNames single_trigger_controller::SingleTriggerController::command_interface_types_
[protected]
```

vector of string holding all command interface names as configured from the parameters.

Looks like this for example:

```
['j1/home', 'gpio1/ao0', 'gpio1/ao1']
```

7.16.4.2 command_interfaces_map_

```
MapOfReferencesToCommandInterfaces single_trigger_controller::SingleTriggerController::command↩
_interfaces_map_  [protected]
```

map containing a reference to the actual hardwares command interfaces

7.16.4.3 command_subscriber_

```
rclcpp::Subscription<CmdType>::SharedPtr single_trigger_controller::SingleTriggerController↩
::command_subscriber_ {}  [protected]
```

Subscriber object for the '~/commands' topic.

7.16.4.4 param_listener_

```
std::shared_ptr<single_trigger_controller_parameters::ParamListener> single_trigger_controller↩
::SingleTriggerController::param_listener_ {}  [protected]
```

parameter listerner object.

7.16.4.5 params_

```
single_trigger_controller_parameters::Params single_trigger_controller::SingleTriggerController↩
::params_  [protected]
```

Parameter structure holding controller configuration parameters.

7.16.4.6 realtime_state_publisher_

```
std::shared_ptr<realtime_tools::RealtimePublisher<StateType> > single_trigger_controller::↩
SingleTriggerController::realtime_state_publisher_ {}  [protected]
```

Realtime publisher wrapper of the state_publisher_.

7.16.4.7 state_interface_types_

```
InterfacesNames single_trigger_controller::SingleTriggerController::state_interface_types_↩
[protected]
```

vector of string holding all state interface names as configured from the parameters.

Looks like this for example:

```
['j1/home', 'gpio1/ai0', 'gpio1/ai1']
```

7.16.4.8 state_interfaces_map_

```
MapOfReferencesToStateInterfaces single_trigger_controller::SingleTriggerController::state_↩
interfaces_map_  [protected]
```

map containing a reference to the actual hardwares state interfaces

7.16.4.9 state_msg_

```
StateType single_trigger_controller::SingleTriggerController::state_msg_  [protected]
```

Message object holding the states to be published.

7.16.4.10 state_publisher_

```
std::shared_ptr<rclcpp::Publisher<StateType> > single_trigger_controller::SingleTrigger↩
Controller::state_publisher_ {}  [protected]
```

Publisher object to the '~/states' topic.

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

- lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/include/single_trigger_controller/single_trigger_controller.hpp
- lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/src/single_trigger_controller.cpp

Chapter 8

File Documentation

8.1 docs/doxygen/index.md File Reference

8.2 lib/joint_firmware/joint/configuration.h File Reference

Configuration definitions for Joint 1 to Joint 4.

Macros

- #define ADR 0x11

  I2C adress of joint n is 0x1n.
- #define MAXACCEL 100

  Maximum acceleration in steps/s^2. Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.
- #define MAXVEL 10

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

8.2.1 Detailed Description

Configuration definitions for Joint 1 to Joint 4.

Author

sbstorz

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.

8.2.2 Macro Definition Documentation

8.2.2.1 ADR

```
#define ADR 0x11
```

I2C adress of joint n is 0x1n.

8.2.2.2 MAXACCEL

```
#define MAXACCEL 100
```

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

8.2.2.3 MAXVEL

```
#define MAXVEL 10
```

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

8.3 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 100
00021 #define MAXVEL 10
00022 #define STALL_WINDOW_B1 12
00023 #define STALL_WINDOW_B2 450
00024 #define STALL_WINDOW_OFFSET 90.0
00025 #define STALL_SLOPE 0.0
00026
00027 #elif defined(J2)
00028 #define ADR 0x12
00029 #define MAXACCEL 100
00030 #define MAXVEL 10
00031 #define STALL_WINDOW_B1 12
00032 #define STALL_WINDOW_B2 450
00033 #define STALL_WINDOW_OFFSET 90.0
00034 #define STALL_SLOPE 0.0
00035
00036 #elif defined(J3)
00037 #define ADR 0x13
00038 #define MAXACCEL 100
00039 #define MAXVEL 10
00040 #define STALL_WINDOW_B1 12
00041 #define STALL_WINDOW_B2 450
00042 #define STALL_WINDOW_OFFSET 90.0
00043 #define STALL_SLOPE 0.0
00044
00045 #elif defined(J4)
00046 #define ADR 0x14
00047 #define MAXACCEL 100
00048 #define MAXVEL 10
00049 #define STALL_WINDOW_B1 12
00050 #define STALL_WINDOW_B2 450
00051 #define STALL_WINDOW_OFFSET 90.0
00052 #define STALL_SLOPE 0.0
```

```
00053 #else
00054
00055 /* Below only defined for documentation */
00059 #define ADR 0x11
00060
00065 #define MAXACCEL 100
00066
00071 #define MAXVEL 10
00072 #error "No Joint has been defined. Define one of 'JX' where X 1,2,3,4"
00073 #endif
00074
00075 #endif
```

8.4 lib/joint_firmware/joint/filters.h File Reference

Helper classes for FIR and IIR filters.

```
#include <stdlib.h>
```

Classes

- class bioscara_joint_firmware::Lowpass
  Simple discrete IIR lowpass filter.
- class bioscara_joint_firmware::MovMax
  Simple FIR moving maximum filter.

Namespaces

- namespace bioscara_joint_firmware
  Joint firmware.

8.4.1 Detailed Description

Helper classes for FIR and IIR filters.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains time series filters that can be used to filter data.

8.5 filters.h

Go to the documentation of this file.

```
00001
00013 #include <stdlib.h>
00014
00015 namespace bioscara_joint_firmware {
00016
00021 class Lowpass {
00022
00023 protected:
00024   float K;
00025   float Ts;
00026   float tau;
00027   float x;
00028
00029 public:
00030
00038   Lowpass(float gain = 1, float sampleTime = 0.1, float timeconstant = 1.0) {
00039     this->K = gain;
00040     this->Ts = sampleTime;
00041     this->tau = timeconstant;
00042     x = 0.0;
00043   }
00044
00049   float updateState(float u) {
00050     x = (1 - (Ts / tau)) * x + K * (Ts / tau) * u;
00051     return x;
00052   }
00053
00058   void resetState(void) {
00059     x = 0.0;
00060   }
00061 };
00062
00068 class MovMax {
00069
00070 protected:
00071   unsigned int M = 200;
00072
00073   float *cb_data;
00074   unsigned int cb_index;
00075
00076
00077 public:
00085   MovMax(float windowSize)
00086     : M(windowSize), cb_index(0), cb_data(0) {
00087
00088     cb_data = (float *)malloc(windowSize * sizeof(float));  // allocate memory for buffer
00089   }
00090
00095   float updateState(float u) {
00096
00097     cb_data[cb_index] = u;
00098     cb_index = (cb_index + 1) % M;
00099
00100
00101     float max = 0;
00102     for (size_t i = 0; i < M; i++) {
00103       if (cb_data[i] > max) {
00104         max = cb_data[i];
00105       }
00106     }
00107
00108     return max;
00109   }
00110 };
00111 }
```

8.6 lib/joint_firmware/joint/joint.h File Reference

joint firmware header

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

Namespaces

- namespace bioscara_joint_firmware

  Joint firmware.

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

  register and command definitions

Functions

- template<typename T >
  void bioscara_joint_firmware::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 bioscara_joint_firmware::writeValue (const T val, uint8_t *txBuf, size_t &tx_length)

  Writes a value of the specified type to a buffer.

8.6.1 Detailed Description

joint firmware header

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains definitions and macros for the joint firmware.

8.6.2 Macro Definition Documentation

8.6.2.1 ACK

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

8.6.2.2 DUMP_BUFFER

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

Value:

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

Macro to dump a buffer to the serial console.

Parameters

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

8.6.2.3 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

8.6.2.4 NACK

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

8.6.2.5 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

8.7 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
00052 namespace bioscara_joint_firmware {
00061 enum stp_reg_t
00062 {
00063   PING = 0x0f,
00064   SETUP = 0x10,
00065   SETRPM = 0x11,
00066   GETDRIVERRPM = 0x12,
00067   MOVESTEPS = 0x13,
00068   MOVEANGLE = 0x14,
00069   MOVETOANGLE = 0x15,
00070   GETMOTORSTATE = 0x16,
00071   RUNCOTINOUS = 0x17,
00072   ANGLEMOVED = 0x18,
00073   SETCURRENT = 0x19,
00074   SETHOLDCURRENT = 0x1A,
00075   SETMAXACCELERATION = 0x1B,
00076   SETMAXDECELERATION = 0x1C,
00077   SETMAXVELOCITY = 0x1D,
00078   ENABLESTALLGUARD = 0x1E,
00079   DISABLESTALLGUARD = 0x1F,
```

```
00080   CLEARSTALL = 0x20,
00081   SETBRAKEMODE = 0x22,
00082   ENABLEPID = 0x23,
00083   DISABLEPID = 0x24,
00084   ENABLECLOSEDLOOP = 0x25,
00085   DISABLECLOSEDLOOP = 0x26,
00086   SETCONTROLTHRESHOLD = 0x27,
00087   MOVETOEND = 0x28,
00088   STOP = 0x29,
00089   GETPIDERROR = 0x2A,
00090   CHECKORIENTATION = 0x2B,
00091   GETENCODERRPM = 0x2C,
00092   HOME = 0x2D,
00093   HOMEOFFSET = 0x2E,
00094 };
00095
00102 template <typename T>
00103 void readValue(T &val, uint8_t *rxBuf, size_t rx_length)
00104 {
00105   memcpy(&val, rxBuf, rx_length);
00106 }
00107
00115 template <typename T>
00116 int writeValue(const T val, uint8_t *txBuf, size_t &tx_length)
00117 {
00118   tx_length = sizeof(T);
00119   memcpy(txBuf, &val, tx_length);
00120   return 0;
00121 }
00122 }
00123 #endif
```

8.8 lib/joint_firmware/joint/joint.ino File Reference

joint firmware

```
#include "configuration.h"
#include <UstepperS32.h>
#include <Wire.h>
#include "joint.h"
#include "filters.h"
#include "stall.h"
```

Namespaces

- namespace bioscara_joint_firmware

  Joint firmware.

Macros

- #define J1

  Define either joint that is to be flashed.

Functions

- void bioscara_joint_firmware::blocking_handler (uint8_t reg)

  Handles commands received via I2C.
- void bioscara_joint_firmware::non_blocking_handler (uint8_t reg)

  Handles read request received via I2C.
- void bioscara_joint_firmware::set_flags_for_blocking_handler (uint8_t reg)

prepare flags to initiate the blocking handling of the received comman.

- void bioscara_joint_firmware::receiveEvent (int n)

  I2C receive event Handler.
- void bioscara_joint_firmware::requestEvent ()

  I2C request event Handler.
- void bioscara_joint_firmware::setup (void)

  Setup Peripherals.
- void bioscara_joint_firmware::loop (void)

  Main loop.
- void setup (void)
- void loop (void)

Variables

- UstepperS32 bioscara_joint_firmware::stepper

  The core UstepperS32 stepper object to control the motor.
- uint8_t bioscara_joint_firmware::reg = 0
- uint8_t bioscara_joint_firmware::blk_reg = 0
- uint8_t bioscara_joint_firmware::rx_buf [MAX_BUFFER] = {0}
- uint8_t bioscara_joint_firmware::tx_buf [MAX_BUFFER+RFLAGS_SIZE] = {0}
- bool bioscara_joint_firmware::rx_data_ready = 0
- size_t bioscara_joint_firmware::tx_length = 0
- size_t bioscara_joint_firmware::rx_length = 0

8.8.1 Detailed Description

joint firmware

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

This file contains the joint firmware.

8.8.2 Macro Definition Documentation

8.8.2.1 J1

```
#define J1
```

Define either joint that is to be flashed.

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

8.8.3 Function Documentation

8.8.3.1 loop()

```
void loop (
            void  )
```

8.8.3.2 setup()

```
void setup (
            void  )
```

8.9 lib/joint_firmware/joint/stall.h File Reference

Helper functions for improved stall detection.

```
#include <stdlib.h>
```

Namespaces

- namespace bioscara_joint_firmware
  Joint firmware.

Functions

- float bioscara_joint_firmware::stall_threshold (float qd_rad, float offset)
  computes the speed adaptive threshold.

8.9.1 Detailed Description

Helper functions for improved stall detection.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

8.10 stall.h

Go to the documentation of this file.

```
00001
00012 #ifndef STALL_H
00013 #define STALL_H
00014 #include <stdlib.h>
00015
00016 namespace bioscara_joint_firmware {
00023 float stall_threshold(float qd_rad, float offset) {
00024   /* y = ax + b */
00025   float a = STALL_SLOPE;
00026   float b = offset;
00027   if ((abs(qd_rad) >= STALL_WINDOW_B1) && (abs(qd_rad) <= STALL_WINDOW_B2)) {
00028     b += STALL_WINDOW_OFFSET;
00029   }
00030   return a * qd_rad + b;
00031 }
00032 }
00033 #endif
```

8.11 lib/ros2_ws/src/bioscara_rviz_plugin/include/bioscara_rviz_↩ plugin/bioscara_panel.hpp File Reference

```
#include <QLabel>
#include <QPushButton>
#include <unordered_map>
#include <vector>
#include <rviz_common/panel.hpp>
#include <rviz_common/ros_integration/ros_node_abstraction_iface.hpp>
#include "controller_manager_msgs/msg/controller_manager_activity.hpp"
#include "controller_manager_msgs/msg/named_lifecycle_state.hpp"
#include "controller_manager_msgs/srv/list_controllers.hpp"
#include "controller_manager_msgs/srv/set_hardware_component_state.hpp"
#include "controller_manager_msgs/srv/switch_controller.hpp"
#include "controller_manager_msgs/srv/configure_controller.hpp"
#include "control_msgs/msg/dynamic_joint_state.hpp"
#include "control_msgs/msg/dynamic_interface_group_values.hpp"
```

Classes

- class bioscara_rviz_plugin::BioscaraPanel

  RViz Panel to control the hardware specific functions of the Bioscara robot.

Namespaces

- namespace bioscara_rviz_plugin

8.12 bioscara_panel.hpp

Go to the documentation of this file.

```
00001 /* Author: sbstorz */
00002
00003 #ifndef BIOSCARA_RVIZ_PANEL_HPP_
00004 #define BIOSCARA_RVIZ_PANEL_HPP_
00005
00006 #include <QLabel>
00007 #include <QPushButton>
00008 #include <unordered_map>
00009 #include <vector>
00010
00011 #include <rviz_common/panel.hpp>
00012 #include <rviz_common/ros_integration/ros_node_abstraction_iface.hpp>
00013 #include "controller_manager_msgs/msg/controller_manager_activity.hpp"
00014 #include "controller_manager_msgs/msg/named_lifecycle_state.hpp"
00015 #include "controller_manager_msgs/srv/list_controllers.hpp"
00016 #include "controller_manager_msgs/srv/set_hardware_component_state.hpp"
00017 #include "controller_manager_msgs/srv/switch_controller.hpp"
00018 #include "controller_manager_msgs/srv/configure_controller.hpp"
00019 #include "control_msgs/msg/dynamic_joint_state.hpp"
00020 #include "control_msgs/msg/dynamic_interface_group_values.hpp"
00021
00022 namespace Ui
00023 {
00024   class BioscaraUI;
00025 }
00026
00027 namespace bioscara_rviz_plugin
00028 {
00029   using namespace controller_manager_msgs::msg;
00030   using namespace controller_manager_msgs::srv;
00031   using namespace control_msgs::msg;
00032
00038   class BioscaraPanel : public rviz_common::Panel
00039   {
00040     Q_OBJECT
00041   public:
00051     explicit BioscaraPanel(QWidget *parent = 0);
00052
00057     ~BioscaraPanel() override;
00058
00069     void onInitialize() override;
00070
00071   protected:
00075     rclcpp::Node::SharedPtr node_;
00076
00083     rclcpp::Subscription<ControllerManagerActivity>::SharedPtr cm_state_subsription_;
00084
00091     rclcpp::Subscription<DynamicJointState>::SharedPtr joint_state_subsription_;
00092
00099     rclcpp::Publisher<DynamicInterfaceGroupValues>::SharedPtr homing_publisher_;
00100
00107     rclcpp::Client<SwitchController>::SharedPtr switch_controller_client_;
00108
00115     rclcpp::Client<ConfigureController>::SharedPtr configure_controller_client_;
00116
00123     rclcpp::Client<SetHardwareComponentState>::SharedPtr hardware_state_client_;
00124
00129     rclcpp::TimerBase::SharedPtr prune_timer_;
00130
00135     std::unordered_map<std::string, InterfaceValue> joint_states_;
00136
00143     std::unordered_map<std::string, NamedLifecycleState> hardware_states_;
00144
00150     std::unordered_map<std::string, NamedLifecycleState> controller_states_;
```

```
00151
00158     Ui::BioscaraUI *ui_;
00159
00170     void cm_state_callback(const ControllerManagerActivity &msg);
00171
00180     void joint_state_callback(const DynamicJointState &msg);
00181
00190     void ensure_jsb_is_active(void);
00191
00201     void set_hardware_component_state(const std::string component, const lifecycle_msgs::msg::State
      target_state);
00202
00211     void configure_controller(const std::string controller);
00212
00223     void switch_controllers(const std::vector<std::string> &activate_controllers,
00224                             const std::vector<std::string> &deactivate_controllers,
00225                             const int32_t = SwitchController::Request::BEST_EFFORT,
00226                             const bool activate_asap = false,
00227                             const builtin_interfaces::msg::Duration timeout =
      builtin_interfaces::msg::Duration());
00228
00238     void set_controller_state(const std::string controller,
00239                               const lifecycle_msgs::msg::State target_state);
00240
00247     void dynamic_joint_state_msg_to_map(const DynamicJointState &dynamic_joint_state_in,
00248                                         std::unordered_map<std::string, InterfaceValue> &map_out);
00249
00256     void named_lcs_msg_to_map(const std::vector<NamedLifecycleState> &named_lcs_in,
00257                               std::unordered_map<std::string, NamedLifecycleState> &map_out);
00258
00265     void print_cm_map(const std::string prefix,
00266                       const std::unordered_map<std::string, NamedLifecycleState> &map_in);
00267
00288     void prepopulate_joint_state_map(std::unordered_map<std::string, InterfaceValue> &state_map,
00289                                      std::vector<std::string> augment_vec);
00290
00303     void prepopulate_state_map(std::unordered_map<std::string, NamedLifecycleState> &state_map,
00304                                std::vector<std::string> augment_vec);
00305
00310     void update_homing_grp_state(void);
00311
00318     void update_homing_state_labels(void);
00319
00327     void set_homing_state_label(QLabel *label, const InterfaceValue &state);
00328
00334     void update_state_labels_and_btns(void);
00335
00346     void update_state_label_and_btn(
00347         const std::unordered_map<std::string, NamedLifecycleState> &state_map,
00348         QLabel *state_label,
00349         QPushButton *en_button,
00350         const std::string &state_key);
00351
00360     void set_state_label(QLabel *label, const NamedLifecycleState &state);
00361
00370     void set_en_btn(QPushButton *button, const NamedLifecycleState &state);
00371
00382     bool check_activation_conditions(const std::string component_key);
00383
00391     lifecycle_msgs::msg::State target_state_from_current(lifecycle_msgs::msg::State current_state);
00392
00401     void homing_cmd(const std::string joint, const int cmd);
00402
00408     void ctrl_en_btn_cb(const std::string controller);
00409
00410   private Q_SLOTS:
00411
00417     void arm_en_btn_cb(void);
00418
00424     void gripper_en_btn_cb(void);
00425   };
00426
00427 } // namespace bioscara_rviz_plugin
00428
00429 #endif // BIOSCARA_RVIZ_PANEL_HPP_
```

8.13 lib/ros2_ws/src/bioscara_rviz_plugin/src/bioscara_panel.cpp File Reference

```
#include <bioscara_rviz_plugin/bioscara_panel.hpp>
#include <rviz_common/display_context.hpp>
```

```
#include <chrono>
#include "ui_bioscara_rviz_plugin_frame.h"
#include <pluginlib/class_list_macros.hpp>
```

Namespaces

- namespace bioscara_rviz_plugin

8.14 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/common.h File Reference

A file containing utility macros and functions.

Macros

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

8.14.1 Detailed Description

A file containing utility macros and functions.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

8.14.2 Macro Definition Documentation

8.14.2.1 DUMP_BUFFER

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

Value:

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

Macro to dump a buffer to cout.

Parameters

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

8.15 common.h

Go to the documentation of this file.

```
00001
00011 #ifndef COMMON_H
00012 #define COMMON_H
00013
00020 #define DUMP_BUFFER(buffer, size)      \
00021   {                                     \
00022     std::cout « "Buffer dump: ";       \
00023     for (size_t i = 0; i < size; i++) \
00024     {                                   \
00025       printf("%#x ", buffer[i]);        \
00026     }                                   \
00027     std::cout « std::endl;             \
00028   }
00029
00030 #endif // COMMON_H
```

8.16 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mBaseJoint.h File Reference

File including the BaseJoint class.

```
#include <iostream>
#include "bioscara_arm_hardware_driver/uErr.h"
```

Classes

- class bioscara_hardware_drivers::BaseJoint

  Generic base class to control a single joint.

Namespaces

- namespace bioscara_hardware_drivers

8.16.1 Detailed Description

File including the BaseJoint class.

Author

sbstorz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

8.17 mBaseJoint.h

Go to the documentation of this file.

```
00001
00012 #ifndef MBASEJOINT_H
00013 #define MBASEJOINT_H
00014
00015 #include <iostream>
00016 #include "bioscara_arm_hardware_driver/uErr.h"
00017
00018 namespace bioscara_hardware_drivers
00019 {
00026   class BaseJoint
00027   {
00028   public:
00038     enum stp_reg_t
00039     {
00040       NONE = 0x00,
00041       PING = 0x0f,
00042       SETUP = 0x10,
00043       SETRPM = 0x11,
00044       GETDRIVERRPM = 0x12,
00045       MOVESTEPS = 0x13,
00046       MOVEANGLE = 0x14,
00047       MOVETOANGLE = 0x15,
00048       GETMOTORSTATE = 0x16,
00049       RUNCOTINOUS = 0x17,
00050       ANGLEMOVED = 0x18,
00051       SETCURRENT = 0x19,
00052       SETHOLDCURRENT = 0x1A,
00053       SETMAXACCELERATION = 0x1B,
00054       SETMAXDECELERATION = 0x1C,
00055       SETMAXVELOCITY = 0x1D,
00056       ENABLESTALLGUARD = 0x1E,
00057       DISABLESTALLGUARD = 0x1F,
00058       CLEARSTALL = 0x20,
00059       SETBRAKEMODE = 0x22,
00060       ENABLEPID = 0x23,
00061       DISABLEPID = 0x24,
00062       ENABLECLOSEDLOOP = 0x25,
00063       DISABLECLOSEDLOOP = 0x26,
00064       SETCONTROLTHRESHOLD = 0x27,
00065       MOVETOEND = 0x28,
00066       STOP = 0x29,
00067       GETPIDERROR = 0x2A,
00068       CHECKORIENTATION = 0x2B,
00069       GETENCODERRPM = 0x2C,
00070       HOME = 0x2D,
```

```
00071       HOMEOFFSET = 0x2E,
00072     };
00073
00081     BaseJoint(const std::string name);
00082
00088     ~BaseJoint(void);
00089
00096     virtual err_type_t init(void);
00097
00104     virtual err_type_t deinit(void);
00105
00118     virtual err_type_t enable(u_int8_t driveCurrent, u_int8_t holdCurrent);
00119
00127     virtual err_type_t disable(void);
00128
00142     virtual err_type_t home(float velocity, u_int8_t sensitivity, u_int8_t current);
00143
00160     virtual err_type_t startHoming(float velocity, u_int8_t sensitivity, u_int8_t current);
00161
00170     virtual err_type_t postHoming(void);
00171
00182     virtual err_type_t getPosition(float &pos) = 0;
00183
00191     virtual err_type_t setPosition(float pos);
00192
00201     virtual err_type_t moveSteps(int32_t steps);
00202
00210     virtual err_type_t getVelocity(float &vel) = 0;
00211
00219     virtual err_type_t setVelocity(float vel);
00220
00228     virtual err_type_t checkOrientation(float angle = 2.0);
00229
00238     virtual err_type_t stop(void);
00239
00240     virtual err_type_t disableCL(void);
00241
00247     virtual err_type_t setDriveCurrent(u_int8_t current);
00248
00255     virtual err_type_t setHoldCurrent(u_int8_t current);
00256
00262     virtual err_type_t setBrakeMode(u_int8_t mode);
00263
00271     virtual err_type_t setMaxAcceleration(float maxAccel);
00272
00280     virtual err_type_t setMaxVelocity(float maxVel);
00281
00291     virtual err_type_t enableStallguard(u_int8_t sensitivity);
00292
00301     virtual bool isHomed(void);
00302
00312     virtual bool isEnabled(void);
00313
00321     virtual bool isStalled(void);
00322
00330     virtual bool isBusy(void);
00331
00337     virtual err_type_t getFlags(u_int8_t &flags);
00338
00343     virtual err_type_t getFlags(void);
00344
00350     virtual stp_reg_t getCurrentBCmd(void);
00351
00352     std::string name;
00353
00354   protected:
00360     virtual void wait_while_busy(const float period_ms);
00361
00374     virtual err_type_t _home(float velocity, u_int8_t sensitivity, u_int8_t current) = 0;
00375
00394     u_int8_t flags = 0b00001100;
00395
00396     stp_reg_t current_b_cmd = NONE;
00397
00398   private:
00399   };
00400 }
00401 #endif
```

8.18 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mJoint.h File Reference

File including the Joint class.

```
#include <iostream>
#include "bioscara_arm_hardware_driver/mBaseJoint.h"
#include "bioscara_arm_hardware_driver/uTransmission.h"
#include "bioscara_arm_hardware_driver/mJoint.hpp"
```

Classes

- class bioscara_hardware_drivers::Joint

  Representing a single hardware joint connected via I2C.

Namespaces

- namespace bioscara_hardware_drivers

8.18.1 Detailed Description

File including the Joint class.

Author

sbstorz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

8.19 mJoint.h

Go to the documentation of this file.

```
00001
00012 #ifndef MJOINT_H
00013 #define MJOINT_H
00014
00015 #include <iostream>
00016 #include "bioscara_arm_hardware_driver/mBaseJoint.h"
00017 #include "bioscara_arm_hardware_driver/uTransmission.h"
00018
00019 namespace bioscara_hardware_drivers
00020 {
00025   class Joint : public BaseJoint
00026   {
00027   public:
00060     Joint(const std::string name, const int address, const float reduction, const float min, const
      float max);
00061
00067     ~Joint(void);
00068
00078     err_type_t init(void) override;
00079
00087     err_type_t deinit(void) override;
00088
00089     err_type_t enable(u_int8_t driveCurrent, u_int8_t holdCurrent) override;
00090
00099     err_type_t postHoming(void) override;
00100
00101     err_type_t getPosition(float &pos) override;
00102
00103     err_type_t setPosition(float pos) override;
00104
00105     err_type_t moveSteps(int32_t steps) override;
00106
00107     err_type_t getVelocity(float &vel) override;
00108
00109     err_type_t setVelocity(float vel) override;
00110
00111     err_type_t checkOrientation(float angle = 2.0) override;
00112
00113     err_type_t stop(void) override;
00114
00115     err_type_t disableCL(void) override;
00116
00117     err_type_t setDriveCurrent(u_int8_t current) override;
00118
00119     err_type_t setHoldCurrent(u_int8_t current) override;
00120
00121     err_type_t setBrakeMode(u_int8_t mode) override;
00122
00123     err_type_t setMaxAcceleration(float maxAccel) override;
00124
00125     err_type_t setMaxVelocity(float maxVel) override;
00126
00127     err_type_t enableStallguard(u_int8_t sensitivity) override;
00128
00129     err_type_t getFlags(void) override;
00130
00131   protected:
00132
00140     err_type_t getHomingOffset(float &offset);
00141
00149     err_type_t setHomingOffset(const float offset);
00150
00151     err_type_t _home(float velocity, u_int8_t sensitivity, u_int8_t current) override;
00152
00160     err_type_t checkCom(void);
00161
00162     float reduction = 1;
00163     float offset = 0;
00164     float min = 0;
00165     float max = 0;
00166
00167   private:
00168     template <typename T>
00169     int read(const stp_reg_t reg, T &data, u_int8_t &flags);
00170
00171     template <typename T>
00172     int write(const stp_reg_t reg, T data, u_int8_t &flags);
00173
00174     int address;
00175     int handle = -1;
00176   };
00177 }
```

```
00178 #include "bioscara_arm_hardware_driver/mJoint.hpp"
00179
00180 #endif
```

8.20 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mJoint.hpp File Reference

Templated functions for the Joint class.

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

Namespaces

- namespace bioscara_hardware_drivers

8.20.1 Detailed Description

Templated functions for the Joint class.

Author

sbstorz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

This header must be included at the END of the mJoint.h file.

8.21 mJoint.hpp

Go to the documentation of this file.

```
00001
00012 #include "bioscara_arm_hardware_driver/mJoint.h"
00013 #include "bioscara_arm_hardware_driver/uI2C.h"
00014 #include "bioscara_arm_hardware_driver/common.h"
00015
00016 namespace bioscara_hardware_drivers
00017 {
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     }
00072 }
```

8.22 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mMockJoint.h File Reference

File including the MockJoint class.

```
#include <iostream>
#include "bioscara_arm_hardware_driver/mBaseJoint.h"
#include <chrono>
```

Classes

- class bioscara_hardware_drivers::MockJoint

  Representing a single joint mocking the hardware joint.

Namespaces

- namespace bioscara_hardware_drivers

8.22.1 Detailed Description

File including the MockJoint class.

Author

sbstorz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

8.23 mMockJoint.h

Go to the documentation of this file.

```
00001
00012 #ifndef MMOCKJOINT_H
00013 #define MMOCKJOINT_H
00014
00015 #include <iostream>
00016 #include "bioscara_arm_hardware_driver/mBaseJoint.h"
00017 #include <chrono>
00018
00019 namespace bioscara_hardware_drivers
00020 {
00028   class MockJoint : public BaseJoint
00029   {
00030   public:
00031     MockJoint(const std::string name);
00032
00033     err_type_t enable(u_int8_t driveCurrent, u_int8_t holdCurrent) override;
00034
00035     err_type_t disable(void) override;
00036
00037     err_type_t getPosition(float &pos) override;
00038
00039     err_type_t setPosition(float pos) override;
00040
00041     err_type_t getVelocity(float &vel) override;
00042
00043     err_type_t setVelocity(float vel) override;
00044
00045     err_type_t checkOrientation(float angle = 2.0) override;
00046
00047     err_type_t stop(void) override;
00048
00049     err_type_t getFlags(void) override;
00050
00051     bool isHomed(void) override;
00052
00053   protected:
00054     err_type_t _home(float velocity, u_int8_t sensitivity, u_int8_t current);
00055
00056   private:
00057     float q = 0.0;
00058     float qd = 0.0;
00059
00060     std::chrono::_V2::system_clock::time_point last_set_position =
      std::chrono::high_resolution_clock::now();
00061     std::chrono::_V2::system_clock::time_point last_set_velocity = last_set_position;
00062     std::chrono::_V2::system_clock::time_point async_start_time = last_set_position;
00063
00071     float getDeltaT(std::chrono::_V2::system_clock::time_point &last_call, bool update = true);
00072
00073     stp_reg_t op_mode = NONE;
00074   };
00075 }
00076 #endif
```

8.24 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_↩ driver/include/bioscara_arm_hardware_driver/uErr.h File Reference

Defining common return types.

```
#include <string>
```

Namespaces

- namespace bioscara_hardware_drivers

Macros

- #define RETURN_ON_ERROR(x)

  Macro which executes a function and returns from the calling function with the error code if the called function fails.
- #define RETURN_ON_FALSE(a, err_code)

  Macro which returns the calling function with specified error_code if the given condition is false.
- #define RETURN_ON_NEGATIVE(a, err_code)

  Macro which returns the calling function with specified error_code if the given condition is negative.

Enumerations

- enum class bioscara_hardware_drivers::err_type_t { bioscara_hardware_drivers::OK = 0 , bioscara_hardware_drivers::ERROR = -1 , bioscara_hardware_drivers::NOT_HOMED = -2 , bioscara_hardware_drivers::NOT_ENABLED = -3 , bioscara_hardware_drivers::STALLED = -4 , bioscara_hardware_drivers::NOT_INIT = -5 , bioscara_hardware_drivers::COMM_ = -6 , bioscara_hardware_drivers::INVALID_ARGUMENT = -101 , bioscara_hardware_drivers::INCORRECT_STATE = -109 }

  Enum defining common error types.

Functions

- std::string bioscara_hardware_drivers::error_to_string (err_type_t err)

  Converts an error code to a string and returns it.

8.24.1 Detailed Description

Defining common return types.

Author

sbstorz

Version

0.1

Date

2025-11-05

Copyright

Copyright (c) 2025

8.24.2 Macro Definition Documentation

8.24.2.1 RETURN_ON_ERROR

```
#define RETURN_ON_ERROR(
                x )
```

Value:

```
    do                                                                  \
    {                                                                   \
        bioscara_hardware_drivers::err_type_t err_rc_ = (x);            \
        if (err_rc_ != bioscara_hardware_drivers::err_type_t::OK) \
        {                                                               \
            return err_rc_;                                             \
        }                                                               \
    } while (0);
```

Macro which executes a function and returns from the calling function with the error code if the called function fails.

Adapted from the ESP-IDF

Parameters

x	function to call

8.24.2.2 RETURN_ON_FALSE

```
#define RETURN_ON_FALSE(
                a,
                err_code )
```

Value:

```
    do                                  \
    {                                   \
        if (!(a))                       \
        {                               \
            return err_code;            \
        }                               \
    } while (0);
```

Macro which returns the calling function with specified error_code if the given condition is false.

Adapted from the ESP-IDF

Parameters

a	expression that evaluates to true or false
err_code	return code to return on false

8.24.2.3 RETURN_ON_NEGATIVE

```
#define RETURN_ON_NEGATIVE(
                a,
                err_code )
```

Value:

```
    do                                  \
    {                                   \
        if ((a) < 0)                    \
        {                               \
            return err_code;            \
        }                               \
    } while (0);
```

Macro which returns the calling function with specified error_code if the given condition is negative.

Adapted from the ESP-IDF

Parameters

a	expression that evaluates to a signed number
err_code	return code to return on false

8.25 uErr.h

Go to the documentation of this file.

```
00001
00011 #ifndef UERR_H
00012 #define UERR_H
00013
00014 #include <string>
00015
00016 namespace bioscara_hardware_drivers
00017 {
00022     enum class err_type_t
00023     {
00024         OK = 0,
00025         ERROR = -1,
00026         NOT_HOMED = -2,
00027         NOT_ENABLED = -3,
00028         STALLED = -4,
00029         NOT_INIT = -5,
00030         COMM_ERROR = -6,
00031         INVALID_ARGUMENT = -101,
00032         INCORRECT_STATE = -109,
00033
00034     };
00035
00042     std::string error_to_string(err_type_t err);
00043 }
00044
00052 #define RETURN_ON_ERROR(x)                                              \
00053     do                                                                  \
00054     {                                                                   \
00055         bioscara_hardware_drivers::err_type_t err_rc_ = (x);            \
00056         if (err_rc_ != bioscara_hardware_drivers::err_type_t::OK) \
00057         {                                                               \
00058             return err_rc_;                                             \
00059         }                                                               \
00060     } while (0);
00061
00069 #define RETURN_ON_FALSE(a, err_code) \
00070     do                               \
00071     {                                \
00072         if (!(a))                    \
00073         {                            \
00074             return err_code;         \
00075         }                            \
00076     } while (0);
00077
00085 #define RETURN_ON_NEGATIVE(a, err_code) \
00086     do                                  \
00087     {                                   \
00088         if ((a) < 0)                    \
00089         {                               \
00090             return err_code;            \
00091         }                               \
00092     } while (0);
00093
00094 #endif // UERR_H
```

8.26 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_↩ driver/include/bioscara_arm_hardware_driver/ul2C.h File Reference

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

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

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

8.26.1 Detailed Description

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

Author

sbstorz

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

8.26.2 Macro Definition Documentation

8.26.2.1 ACK

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

8.26.2.2 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

8.26.2.3 NACK

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

8.26.2.4 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

8.26.3 Function Documentation

8.26.3.1 closeI2CDevHandle()

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

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

8.26.3.2 openI2CDevHandle()

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

Initiates an I2C device on the bus.

Parameters

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

Returns

the device handle, negative on error.

8.26.3.3 readFromI2CDev()

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

reads block of bytes from device to buffer

Parameters

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

Returns

number of bytes read, negative on error.

8.26.3.4 writeToI2CDev()

```
int writeToI2CDev (
            const int dev_handle,
```

```
            const int reg,
            char * tx_buffer,
            const int data_length,
            char * RFLAGS_buffer )
```

writes block of bytes from buffer to device

Parameters

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

Returns

0 on OK, negative on error.

8.27 uI2C.h

Go to the documentation of this file.

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

8.28 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_↩ driver/include/bioscara_arm_hardware_driver/uTransmission.h File Reference

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.

8.28.1 Macro Definition Documentation

8.28.1.1 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

8.28.1.2 DEG2RAD

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

Macro to convert degree to radians.

8.28.1.3 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

8.28.1.4 M_PI

```
#define M_PI 3.14159265358979323846
```

pi

8.28.1.5 RAD2DEG

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

Macro to convert radians to degree.

8.29 uTransmission.h

Go to the documentation of this file.

```
00001 #ifndef UTRANSMISSION_H
00002 #define UTRANSMISSION_H
00003
00012 #define JOINT2ACTUATOR(in, reduction, offset) (reduction * (in - offset))
00013
00022 #define ACTUATOR2JOINT(in, reduction, offset) (in / reduction + offset)
00023
00028 #define M_PI 3.14159265358979323846
00029
00034 #define RAD2DEG(rad) (rad / M_PI * 180)
00035
00040 #define DEG2RAD(deg) (deg * M_PI / 180)
00041 #endif //UTRANSMISSION_H
```

8.30 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/joint_comm_node.cpp File Reference

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

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)

8.30.1 Function Documentation

8.30.1.1 INT_handler()

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

8.30.1.2 main()

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

8.30.2 Variable Documentation

8.30.2.1 J1

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

8.30.2.2 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 )
```

8.30.2.3 J3

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

8.30.2.4 J4

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

8.31 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mBaseJoint.cpp File Reference

```
#include "bioscara_arm_hardware_driver/mBaseJoint.h"
#include <unistd.h>
```

Namespaces

- namespace bioscara_hardware_drivers

8.32 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mJoint.cpp File Reference

```
#include "bioscara_arm_hardware_driver/uI2C.h"
#include "bioscara_arm_hardware_driver/mJoint.h"
#include <cmath>
```

Namespaces

- namespace bioscara_hardware_drivers

8.33 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mMockJoint.cpp File Reference

```
#include "bioscara_arm_hardware_driver/mMockJoint.h"
#include <unistd.h>
```

Namespaces

- namespace bioscara_hardware_drivers

8.34 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uErr.cpp File Reference

```
#include "bioscara_arm_hardware_driver/uErr.h"
```

Namespaces

- namespace bioscara_hardware_drivers

Functions

- std::string bioscara_hardware_drivers::error_to_string (err_type_t err)

  Converts an error code to a string and returns it.

8.35 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uI2C.cpp File Reference

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

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

8.35.1 Function Documentation

8.35.1.1 closeI2CDevHandle()

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

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

8.35.1.2 openI2CDevHandle()

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

Initiates an I2C device on the bus.

Parameters

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

Returns

the device handle, negative on error.

8.35.1.3 readFromI2CDev()

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

reads block of bytes from device to buffer

Parameters

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

Returns

number of bytes read, negative on error.

8.35.1.4 writeToI2CDev()

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

writes block of bytes from buffer to device

Parameters

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

Returns

0 on OK, negative on error.

8.36 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_interface/arm_hardware.hpp File Reference

```
#include <memory>
#include <string>
#include <vector>
#include <set>
#include <unordered_map>
#include <mutex>
#include "bioscara_arm_hardware_driver/mJoint.h"
#include "bioscara_arm_hardware_driver/mMockJoint.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"
```

Classes

- class bioscara_hardware_interfaces::BioscaraArmHardwareInterface

  The bioscara arm hardware interface class.
- struct bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_homing_config_t

  configuration structure holding the passed homing paramters from the ros2_control urdf
- struct bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t

  configuration structure holding the passed paramters from the ros2_control urdf

Namespaces

- namespace bioscara_hardware_interfaces

Variables

- constexpr char bioscara_hardware_interfaces::HW_IF_HOME [ ] = "home"

8.37 arm_hardware.hpp

Go to the documentation of this file.

```
00001 // Copyright 2023 ros2_control Development Team
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00015 #ifndef ARM_HARDWARE_HPP_
00016 #define ARM_HARDWARE_HPP_
00017
00018 #include <memory>
00019 #include <string>
00020 #include <vector>
00021 #include <set>
00022 #include <unordered_map>
00023 #include <memory>
00024 #include <mutex>
00025
00026 #include "bioscara_arm_hardware_driver/mJoint.h"
00027 #include "bioscara_arm_hardware_driver/mMockJoint.h"
00028
00029 #include "hardware_interface/handle.hpp"
00030 #include "hardware_interface/hardware_info.hpp"
00031 #include "hardware_interface/system_interface.hpp"
00032 #include "hardware_interface/types/hardware_interface_return_values.hpp"
00033 #include "rclcpp/macros.hpp"
00034 #include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
00035 #include "rclcpp_lifecycle/state.hpp"
00036
00037 namespace bioscara_hardware_interfaces
00038 {
00039     constexpr char HW_IF_HOME[] = "home";
00040
00054     class BioscaraArmHardwareInterface : public hardware_interface::SystemInterface
00055     {
00056     public:
00057         RCLCPP_SHARED_PTR_DEFINITIONS(BioscaraArmHardwareInterface)
00058
00059
00088         hardware_interface::CallbackReturn on_init(
00089             const hardware_interface::HardwareComponentInterfaceParams &params) override;
00090
00100         hardware_interface::CallbackReturn on_shutdown(
00101             const rclcpp_lifecycle::State &previous_state) override;
00102
00110         hardware_interface::CallbackReturn on_configure(
00111             const rclcpp_lifecycle::State &previous_state) override;
00112
00121         hardware_interface::CallbackReturn on_cleanup(
00122             const rclcpp_lifecycle::State &previous_state) override;
00123
00138         hardware_interface::CallbackReturn on_activate(
00139             const rclcpp_lifecycle::State &previous_state) override;
00140
00149         hardware_interface::CallbackReturn on_deactivate(
00150             const rclcpp_lifecycle::State &previous_state) override;
00151
00172         hardware_interface::return_type read(
00173             const rclcpp::Time &time,
00174             const rclcpp::Duration &period) override;
00175
00197         hardware_interface::return_type write(
00198             const rclcpp::Time &time,
00199             const rclcpp::Duration &period) override;
00200
00234         hardware_interface::return_type prepare_command_mode_switch(
00235             const std::vector<std::string> &start_interfaces,
00236             const std::vector<std::string> &stop_interfaces) override;
00237
00255         hardware_interface::return_type perform_command_mode_switch(
00256             const std::vector<std::string> &start_interfaces,
00257             const std::vector<std::string> &stop_interfaces) override;
00258
00286         hardware_interface::CallbackReturn on_error(
00287             const rclcpp_lifecycle::State &previous_state) override;
```

```
00288
00289     private:
00296         struct joint_homing_config_t
00297         {
00304             float speed;
00305
00311             u_int8_t threshold;
00312
00319             u_int8_t current;
00320
00328             float acceleration = 0.01;
00329         };
00330
00337         struct joint_config_t
00338         {
00344             int i2c_address;
00345
00360             float reduction = 1;
00361
00371             float min;
00372
00382             float max;
00383
00389             u_int8_t drive_current;
00390
00396             u_int8_t hold_current;
00397
00404             u_int8_t stall_threshold;
00405
00414             float max_velocity;
00415
00424             float max_acceleration;
00425
00430             joint_homing_config_t homing;
00431         };
00432
00443         std::unordered_map<std::string, std::unique_ptr<bioscara_hardware_drivers::BaseJoint» _joints;
00444
00452         std::unordered_map<std::string, joint_config_t> _joint_cfg;
00453
00467         std::unordered_map<std::string, std::set<std::string» _joint_command_modes;
00468
00476         std::unordered_map<std::string, std::set<std::string» _new_joint_command_modes;
00477
00493         std::vector<std::pair<std::string, hardware_interface::InterfaceDescription *»
      _ordered_joint_state_interfaces_ptr;
00494
00508         std::mutex mtx;
00509
00518         bioscara_hardware_drivers::err_type_t start_homing(const std::string name, float velocity);
00519
00527         bioscara_hardware_drivers::err_type_t stop_homing(const std::string name);
00528
00536         void split_interface_string_to_joint_and_name(std::string interface, std::string &joint_name,
     std::string &interface_name);
00537
00544         bioscara_hardware_drivers::err_type_t activate_joint(const std::string name);
00545
00552         bioscara_hardware_drivers::err_type_t deactivate_joint(const std::string name);
00553     };
00554
00555 } // namespace bioscara_hardware_interfaces
00556
00557 #endif // ARM_HARDWARE_HPP_
```

8.38 lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_↩ interface/src/arm_hardware.cpp File Reference

```
#include "bioscara_arm_hardware_interface/arm_hardware.hpp"
#include <chrono>
#include <cmath>
#include <limits>
#include <memory>
#include <vector>
#include "hardware_interface/types/hardware_interface_type_values.hpp"
#include "rclcpp/rclcpp.hpp"
```

```
#include "pluginlib/class_list_macros.hpp"
```

Namespaces

- namespace bioscara_hardware_interfaces

8.39 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_driver/include/bioscara_gripper_hardware_driver/m↩ BaseGripper.h File Reference

File containing the BaseGripper class.

```
#include "bioscara_arm_hardware_driver/uErr.h"
#include <chrono>
```

Classes

- class bioscara_hardware_drivers::BaseGripper

  Generic base class for gripper control implementations.

Namespaces

- namespace bioscara_hardware_drivers

8.39.1 Detailed Description

File containing the BaseGripper class.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

Dont include this file directly, instead use one of the derived classes.

8.40 mBaseGripper.h

Go to the documentation of this file.

```
00001
00012 #ifndef MBASEGRIPPER_H
00013 #define MBASEGRIPPER_H
00014 #include "bioscara_arm_hardware_driver/uErr.h"
00015 #include <chrono>
00016
00017 namespace bioscara_hardware_drivers
00018 {
00019
00026     class BaseGripper
00027     {
00028     public:
00058         BaseGripper(float reduction, float offset, float min, float max, float backup_init_pos);
00059
00065         ~BaseGripper(void);
00066
00072         virtual err_type_t init(void);
00073
00079         virtual err_type_t deinit(void);
00080
00090         virtual err_type_t enable(void);
00091
00098         virtual err_type_t disable(void);
00099
00107         virtual err_type_t setPosition(float width);
00108
00117         virtual err_type_t getPosition(float &width);
00118
00124         virtual void setReduction(float reduction);
00125
00129         virtual void setOffset(float offset);
00130
00131     protected:
00138         err_type_t save_last_position(float pos);
00139
00146         err_type_t retrieve_last_position(float &pos);
00147
00148         float _reduction = 1;
00149         float _offset = 0;
00150         float _min = 0;
00151         float _max = 0;
00152         float _backup_init_pos = 0.0;
00153         float _pos = _backup_init_pos;
00154         float _pos_get = _pos;
00155     private:
00156         std::chrono::_V2::system_clock::time_point _new_cmd_time =
      std::chrono::_V2::system_clock::time_point();
00157     };
00158 }
00159 #endif // MBASEGRIPPER_H
```

8.41 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_driver/include/bioscara_gripper_hardware_driver/m↩ Gripper.h File Reference

File containing the Gripper class.

```
#include "bioscara_gripper_hardware_driver/mBaseGripper.h"
#include "bioscara_gripper_hardware_driver/uPWM.h"
#include "bioscara_arm_hardware_driver/uErr.h"
```

Classes

- class bioscara_hardware_drivers::Gripper

  Child class implementing control of the hardware gripper.

Namespaces

- namespace bioscara_hardware_drivers

8.41.1 Detailed Description

File containing the Gripper class.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

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

8.42 mGripper.h

Go to the documentation of this file.

```
00001
00013 #ifndef MGRIPPER_H
00014 #define MGRIPPER_H
00015 #include "bioscara_gripper_hardware_driver/mBaseGripper.h"
00016 #include "bioscara_gripper_hardware_driver/uPWM.h"
00017 #include "bioscara_arm_hardware_driver/uErr.h"
00018
00019 namespace bioscara_hardware_drivers
00020 {
00027     class Gripper : public BaseGripper
00028     {
00029     public:
00035         Gripper(float reduction, float offset, float min, float max, float backup_init_pos);
00036
00046         err_type_t enable(void) override;
00047
00056         err_type_t disable(void) override;
00057
00068         err_type_t setPosition(float width) override;
00069
00078         err_type_t setServoPosition(float angle);
00079
00080     protected:
00081
00082         RPI_PWM _pwm;
00083         int _freq = 50;
00084
00085     private:
00086
00087     };
00088 }
00089 #endif // MGRIPPER_H
```

8.43 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_driver/include/bioscara_gripper_hardware_driver/m↩ MockGripper.h File Reference

File containing the MockGripper class.

```
#include "bioscara_gripper_hardware_driver/mBaseGripper.h"
#include "bioscara_arm_hardware_driver/uErr.h"
```

Classes

- class bioscara_hardware_drivers::MockGripper

  Child class for to mock the gripper hardware.

Namespaces

- namespace bioscara_hardware_drivers

8.43.1 Detailed Description

File containing the MockGripper class.

Author

sbstorz

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

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

8.44 mMockGripper.h

Go to the documentation of this file.

```
00001
00013 #ifndef MMOCKGRIPPER_H
00014 #define MMOCKGRIPPER_H
00015 #include "bioscara_gripper_hardware_driver/mBaseGripper.h"
00016 #include "bioscara_arm_hardware_driver/uErr.h"
00017
00018 namespace bioscara_hardware_drivers
00019 {
00026     class MockGripper : public BaseGripper
00027     {
00028     public:
00034         MockGripper(float reduction, float offset, float min, float max, float backup_init_pos);
00035
00036     protected:
00037     private:
00038     };
00039 }
00040 #endif // MMOCKGRIPPER_H
```

8.45 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_driver/include/bioscara_gripper_hardware_driver/u↩ PWM.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>
```

Classes

- class RPI_PWM

  Class to create a Pulse Width Modulated (PWM) signal on the Raspberry PI 4 and 5.

8.45.1 Detailed Description

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

Author

sbstorz and Bernd Porr, bernd.porr@glasgow.ac.uk

Version

0.1

Date

2025-05-27

Copyright

Copyright (c) 2025

8.46 uPWM.h

Go to the documentation of this file.

```
00001
00012 #ifndef __RPIPWM
00013 #define __RPIPWM
00014
00015 #include <stdio.h>
00016 #include <stdlib.h>
00017 #include <string.h>
00018 #include <unistd.h>
00019 #include <string>
00020 #include <iostream>
00021 #include <math.h>
00022
00033 class RPI_PWM
00034 {
00035 public:
00045     int start(int channel, int frequency, float duty_cycle = 0, int chip = 2)
00046     {
00047         chippath = "/sys/class/pwm/pwmchip" + std::to_string(chip);
00048         pwmpath = chippath + "/pwm" + std::to_string(channel);
00049         std::string p = chippath + "/export";
00050         FILE *const fp = fopen(p.c_str(), "w");
00051         if (NULL == fp)
00052         {
00053             std::cerr « "PWM device does not exist. Make sure to add 'dtoverlay=pwm-2chan' to
      /boot/firmware/config.txt.\n";
00054             return -1;
00055         }
00056         const int r = fprintf(fp, "%d", channel);
00057         fclose(fp);
00058         if (r < 0)
00059             return r;
00060         usleep(100000); // it takes a while till the PWM subdir is created
00061         per = (int)1E9 / frequency;
00062         setPeriod(per);
00063         setDutyCycle(duty_cycle);
00064         enable();
00065         return r;
00066     }
00067
00071     void stop()
00072     {
00073         disable();
00074     }
00075
00082     ~RPI_PWM()
00083     {
00084         disable();
00085     }
00086
00092     inline int setDutyCycle(float v) const
00093     {
00094         const int dc = (int)round((float)per * (v / 100.0));
00095         const int r = setDutyCycleNS(dc);
00096         return r;
00097     }
00098
00099 private:
00100
00101     void setPeriod(int ns) const
00102     {
00103         writeSYS(pwmpath + "/" + "period", ns);
00104     }
00105
00106     inline int setDutyCycleNS(int ns) const
00107     {
00108         const int r = writeSYS(pwmpath + "/" + "duty_cycle", ns);
00109         return r;
00110     }
00111
00112     void enable() const
00113     {
00114         writeSYS(pwmpath + "/" + "enable", 1);
00115     }
00116
00117     void disable() const
00118     {
00119         writeSYS(pwmpath + "/" + "enable", 0);
00120     }
00121
00122     int per = 0;
00123
00124     std::string chippath;
```

```
00125     std::string pwmpath;
00126
00127     inline int writeSYS(std::string filename, int value) const
00128     {
00129         FILE *const fp = fopen(filename.c_str(), "w");
00130         if (NULL == fp)
00131         {
00132             return -1;
00133         }
00134         const int r = fprintf(fp, "%d", value);
00135         fclose(fp);
00136         return r;
00137     }
00138 };
00139
00140 #endif
```

8.47 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mBaseGripper.cpp File Reference

```
#include "bioscara_gripper_hardware_driver/mBaseGripper.h"
#include <fstream>
#include <iostream>
#include <cmath>
```

Namespaces

- namespace bioscara_hardware_drivers

8.48 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mGripper.cpp File Reference

```
#include "bioscara_gripper_hardware_driver/mGripper.h"
#include "bioscara_arm_hardware_driver/uTransmission.h"
```

Namespaces

- namespace bioscara_hardware_drivers

8.49 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/src/mMockGripper.cpp File Reference

```
#include "bioscara_gripper_hardware_driver/mMockGripper.h"
```

Namespaces

- namespace bioscara_hardware_drivers

8.50 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_interface/include/bioscara_gripper_hardware_↩ interface/gripper_hardware.hpp File Reference

```
#include <memory>
#include <string>
#include <vector>
#include <set>
#include <unordered_map>
#include <limits>
#include "bioscara_gripper_hardware_driver/mGripper.h"
#include "bioscara_gripper_hardware_driver/mMockGripper.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"
```

Classes

- class bioscara_hardware_interfaces::BioscaraGripperHardwareInterface

  The bioscara gripper hardware interface class.
- struct bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t

  configuration structure holding the passed paramters from the ros2_control urdf

Namespaces

- namespace bioscara_hardware_interfaces

8.51 gripper_hardware.hpp

Go to the documentation of this file.

```
00001 // Copyright 2023 ros2_control Development Team
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License");
00004 // you may not use this file except in compliance with the License.
00005 // You may obtain a copy of the License at
00006 //
00007 //     http://www.apache.org/licenses/LICENSE-2.0
00008 //
00009 // Unless required by applicable law or agreed to in writing, software
00010 // distributed under the License is distributed on an "AS IS" BASIS,
00011 // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 // See the License for the specific language governing permissions and
00013 // limitations under the License.
00014
00015 #ifndef GRIPPER_HARDWARE_HPP_
00016 #define GRIPPER_HARDWARE_HPP_
00017
00018 #include <memory>
00019 #include <string>
00020 #include <vector>
00021 #include <set>
00022 #include <unordered_map>
00023 #include <memory>
00024 #include <limits>
```

```
00025
00026 #include "bioscara_gripper_hardware_driver/mGripper.h"
00027 #include "bioscara_gripper_hardware_driver/mMockGripper.h"
00028
00029 #include "hardware_interface/handle.hpp"
00030 #include "hardware_interface/hardware_info.hpp"
00031 #include "hardware_interface/system_interface.hpp"
00032 #include "hardware_interface/types/hardware_interface_return_values.hpp"
00033 #include "rclcpp/macros.hpp"
00034 #include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
00035 #include "rclcpp_lifecycle/state.hpp"
00036
00037 namespace bioscara_hardware_interfaces
00038 {
00039
00046     class BioscaraGripperHardwareInterface : public hardware_interface::SystemInterface
00047     {
00048     public:
00049         RCLCPP_SHARED_PTR_DEFINITIONS(BioscaraGripperHardwareInterface)
00050
00051         hardware_interface::CallbackReturn on_init(
00052             const hardware_interface::HardwareComponentInterfaceParams &params) override;
00053
00062         hardware_interface::CallbackReturn on_shutdown(
00063             const rclcpp_lifecycle::State &previous_state) override;
00064
00071         hardware_interface::CallbackReturn on_configure(
00072             const rclcpp_lifecycle::State &previous_state) override;
00073
00080         hardware_interface::CallbackReturn on_cleanup(
00081             const rclcpp_lifecycle::State &previous_state) override;
00082
00091         hardware_interface::CallbackReturn on_activate(
00092             const rclcpp_lifecycle::State &previous_state) override;
00093
00102         hardware_interface::CallbackReturn on_deactivate(
00103             const rclcpp_lifecycle::State &previous_state) override;
00104
00114         hardware_interface::return_type read(
00115             const rclcpp::Time &time,
00116             const rclcpp::Duration &period) override;
00117
00127         hardware_interface::return_type write(
00128             const rclcpp::Time &time,
00129             const rclcpp::Duration &period) override;
00130
00153         hardware_interface::CallbackReturn on_error(
00154             const rclcpp_lifecycle::State &previous_state) override;
00155
00156     private:
00163         struct gripper_config_t
00164         {
00165             float reduction = 1;
00166             float offset = 0;
00167             float min;
00168             float max;
00169             float init_pos;
00170         };
00171
00178         std::unique_ptr<bioscara_hardware_drivers::BaseGripper> _gripper;
00179
00184         gripper_config_t _gripper_cfg;
00185
00186         float _last_pos = std::numeric_limits<double>::quiet_NaN();
00187         float _vel = std::numeric_limits<double>::quiet_NaN();
00188     };
00189
00190 } // namespace bioscara_hardware_interfaces
00191
00192 #endif // GRIPPER_HARDWARE_HPP_
```

8.52 lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_↩ hardware_interface/src/gripper_hardware.cpp File Reference

```
#include "bioscara_gripper_hardware_interface/gripper_hardware.hpp"
#include <chrono>
#include <cmath>
#include <memory>
```

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

Namespaces

- namespace bioscara_hardware_interfaces

8.53 lib/ros2_ws/src/dalsa_controllers/single_trigger_↩ controller/include/single_trigger_controller/single_trigger_↩ controller.hpp File Reference

```
#include <memory>
#include <string>
#include <unordered_map>
#include <vector>
#include "control_msgs/msg/dynamic_interface_group_values.hpp"
#include "controller_interface/controller_interface.hpp"
#include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
#include "rclcpp_lifecycle/state.hpp"
#include "realtime_tools/realtime_publisher.hpp"
#include "realtime_tools/realtime_thread_safe_box.hpp"
#include "single_trigger_controller/single_trigger_controller_parameters.↩
hpp"
```

Classes

- class single_trigger_controller::SingleTriggerController

  A general purpose controller which can be used to trigger command interfaces.

Namespaces

- namespace single_trigger_controller

Typedefs

- using single_trigger_controller::CmdType = control_msgs::msg::DynamicInterfaceGroupValues
- using single_trigger_controller::StateType = control_msgs::msg::DynamicInterfaceGroupValues
- using single_trigger_controller::CallbackReturn = controller_interface::CallbackReturn
- using single_trigger_controller::InterfacesNames = std::vector< std::string >
- using single_trigger_controller::MapOfReferencesToCommandInterfaces = std::unordered_map< std::string, std::reference_wrapper< hardware_interface::LoanedCommandInterface > >
- using single_trigger_controller::MapOfReferencesToStateInterfaces = std::unordered_map< std::string, std::reference_wrapper< hardware_interface::LoanedStateInterface > >
- using single_trigger_controller::StateInterfaces = std::vector< std::reference_wrapper< hardware_↩ interface::LoanedStateInterface > >

8.54 single_trigger_controller.hpp

Go to the documentation of this file.

```
00001 // Copyright 2022 ICUBE Laboratory, University of Strasbourg
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 SINGLE_TRIGGER_CONTROLLER__GPIO_COMMAND_CONTROLLER_HPP_
00016 #define SINGLE_TRIGGER_CONTROLLER__GPIO_COMMAND_CONTROLLER_HPP_
00017
00018 #include <memory>
00019 #include <string>
00020 #include <unordered_map>
00021 #include <vector>
00022
00023 #include "control_msgs/msg/dynamic_interface_group_values.hpp"
00024 #include "controller_interface/controller_interface.hpp"
00025 #include "rclcpp_lifecycle/node_interfaces/lifecycle_node_interface.hpp"
00026 #include "rclcpp_lifecycle/state.hpp"
00027 #include "realtime_tools/realtime_publisher.hpp"
00028 #include "realtime_tools/realtime_thread_safe_box.hpp"
00029
00030 #include "single_trigger_controller/single_trigger_controller_parameters.hpp"
00031
00032 namespace single_trigger_controller
00033 {
00034   using CmdType = control_msgs::msg::DynamicInterfaceGroupValues;
00035   using StateType = control_msgs::msg::DynamicInterfaceGroupValues;
00036   using CallbackReturn = controller_interface::CallbackReturn;
00037   using InterfacesNames = std::vector<std::string>;
00038   using MapOfReferencesToCommandInterfaces = std::unordered_map<
00039       std::string, std::reference_wrapper<hardware_interface::LoanedCommandInterface>>;
00040   using MapOfReferencesToStateInterfaces =
00041       std::unordered_map<std::string,
      std::reference_wrapper<hardware_interface::LoanedStateInterface>>;
00042   using StateInterfaces =
00043       std::vector<std::reference_wrapper<hardware_interface::LoanedStateInterface>>;
00044
00059   class SingleTriggerController : public controller_interface::ControllerInterface
00060   {
00061   public:
00062     SingleTriggerController();
00063
00071     controller_interface::InterfaceConfiguration command_interface_configuration() const override;
00072
00080     controller_interface::InterfaceConfiguration state_interface_configuration() const override;
00081
00089     CallbackReturn on_init() override;
00090
00100     CallbackReturn on_configure(const rclcpp_lifecycle::State &previous_state) override;
00101
00102
00114     CallbackReturn on_activate(const rclcpp_lifecycle::State &previous_state) override;
00115
00121     CallbackReturn on_deactivate(const rclcpp_lifecycle::State &previous_state) override;
00122
00132     controller_interface::return_type update(
00133         const rclcpp::Time &time, const rclcpp::Duration &period) override;
00134
00135   private:
00136
00140     void store_command_interface_types();
00141
00145     void store_state_interface_types();
00146
00172     void initialize_state_msg();
00173
00181     void update_states();
00182
00190     controller_interface::return_type update_commands(const CmdType &commands);
00191
00200     template <typename T>
00201     std::unordered_map<std::string, std::reference_wrapper<T>> create_map_of_references_to_interfaces(
00202         const InterfacesNames &interfaces_from_params, std::vector<T> &configured_interfaces);
00203
```

```
00212     template <typename T>
00213     bool check_if_configured_interfaces_matches_received(
00214         const InterfacesNames &interfaces_from_params, const T &configured_interfaces);
00215
00226     void apply_state_value(
00227         StateType &state_msg, std::size_t index, std::size_t interface_index) const;
00228
00237     void apply_command(
00238         const CmdType &commands, std::size_t index,
00239         std::size_t command_interface_index) const;
00240
00241
00248     InterfacesNames get_state_interfaces_names(const std::string &name) const;
00249
00257     bool update_dynamic_map_parameters();
00258
00259   protected:
00268     InterfacesNames command_interface_types_;
00269
00278     InterfacesNames state_interface_types_;
00279
00284     MapOfReferencesToCommandInterfaces command_interfaces_map_;
00285
00290     MapOfReferencesToStateInterfaces state_interfaces_map_;
00291
00296     rclcpp::Subscription<CmdType>::SharedPtr command_subscriber_{};
00297
00302     std::shared_ptr<rclcpp::Publisher<StateType>> state_publisher_{};
00303
00307     std::shared_ptr<realtime_tools::RealtimePublisher<StateType>> realtime_state_publisher_{};
00308
00313     StateType state_msg_;
00314
00319     std::shared_ptr<single_trigger_controller_parameters::ParamListener> param_listener_{};
00320
00325     single_trigger_controller_parameters::Params params_;
00326
00327   };
00328
00329 } // namespace single_trigger_controller
00330
00331 #endif // SINGLE_TRIGGER_CONTROLLER__GPIO_COMMAND_CONTROLLER_HPP_
```

8.55 lib/ros2_ws/src/dalsa_controllers/single_trigger_↩ controller/src/single_trigger_controller.cpp File Reference

```
#include "single_trigger_controller/single_trigger_controller.hpp"
#include <algorithm>
#include "controller_interface/helpers.hpp"
#include "hardware_interface/component_parser.hpp"
#include "rclcpp/logging.hpp"
#include "rclcpp/qos.hpp"
#include "rclcpp/subscription.hpp"
#include "pluginlib/class_list_macros.hpp"
```

Namespaces

- namespace single_trigger_controller

Index

_backup_init_pos
bioscara_hardware_drivers::BaseGripper, 26
_freq
bioscara_hardware_drivers::Gripper, 77
_gripper
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 60
_gripper_cfg
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 60
_home
bioscara_hardware_drivers::BaseJoint, 31
bioscara_hardware_drivers::Joint, 83
bioscara_hardware_drivers::MockJoint, 105
_joint_cfg
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 53
_joint_command_modes
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 53
_joints
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 53
_last_pos
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 60
_max
bioscara_hardware_drivers::BaseGripper, 26
_min
bioscara_hardware_drivers::BaseGripper, 26
_new_cmd_time
bioscara_hardware_drivers::BaseGripper, 26
_new_joint_command_modes
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 54
_offset
bioscara_hardware_drivers::BaseGripper, 27
_ordered_joint_state_interfaces_ptr
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 54
_pos
bioscara_hardware_drivers::BaseGripper, 27
_pos_get
bioscara_hardware_drivers::BaseGripper, 27
_pwm
bioscara_hardware_drivers::Gripper, 77
_reduction
bioscara_hardware_drivers::BaseGripper, 27
_vel
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 60
~BaseGripper
bioscara_hardware_drivers::BaseGripper, 23
~BaseJoint
bioscara_hardware_drivers::BaseJoint, 31
~BioscaraPanel
bioscara_rviz_plugin::BioscaraPanel, 63
~Joint
bioscara_hardware_drivers::Joint, 82
~RPI_PWM
RPI_PWM, 113

acceleration
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_, 97
ACK
joint.h, 132
rtI2C.h, 153
activate_joint
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 45
ACTUATOR2JOINT
uTransmission.h, 156
address
bioscara_hardware_drivers::Joint, 92
ADR
configuration.h, 128
ANGLEMOVED
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
apply_command
single_trigger_controller::SingleTriggerController, 117
apply_state_value
single_trigger_controller::SingleTriggerController, 118
arm_en_btn_cb
bioscara_rviz_plugin::BioscaraPanel, 64
async_start_time
bioscara_hardware_drivers::MockJoint, 110

BaseGripper
bioscara_hardware_drivers::BaseGripper, 22
BaseJoint
bioscara_hardware_drivers::BaseJoint, 31
Bioscara C++ API Documentation, 1
bioscara_hardware_drivers, 11
COMM_ERROR, 12
err_type_t, 11

ERROR, 12
error_to_string, 12
INCORRECT_STATE, 12
INVALID_ARGUMENT, 12
NOT_ENABLED, 12
NOT_HOMED, 12
NOT_INIT, 12
OK, 12
STALLED, 12
bioscara_hardware_drivers::BaseGripper, 21
_backup_init_pos, 26
_max, 26
_min, 26
_new_cmd_time, 26
_offset, 27
_pos, 27
_pos_get, 27
_reduction, 27
~BaseGripper, 23
BaseGripper, 22
deinit, 23
disable, 23
enable, 24
getPosition, 24
init, 24
retrieve_last_position, 25
save_last_position, 25
setOffset, 25
setPosition, 25
setReduction, 26
bioscara_hardware_drivers::BaseJoint, 27
_home, 31
~BaseJoint, 31
ANGLEMOVED, 30
BaseJoint, 31
CHECKORIENTATION, 30
checkOrientation, 31
CLEARSTALL, 30
current_b_cmd, 41
deinit, 32
disable, 32
disableCL, 32
DISABLECLOSEDLOOP, 30
DISABLEPID, 30
DISABLESTALLGUARD, 30
enable, 33
ENABLECLOSEDLOOP, 30
ENABLEPID, 30
ENABLESTALLGUARD, 30
enableStallguard, 33
flags, 41
getCurrentBCmd, 34
GETDRIVERRPM, 30
GETENCODERRPM, 30
getFlags, 34
GETMOTORSTATE, 30
GETPIDERROR, 30
getPosition, 34
getVelocity, 35
HOME, 30
home, 35
HOMEOFFSET, 30
init, 36
isBusy, 36
isEnabled, 36
isHomed, 36
isStalled, 37
MOVEANGLE, 30
MOVESTEPS, 30
moveSteps, 37
MOVETOANGLE, 30
MOVETOEND, 30
name, 42
NONE, 30
PING, 30
postHoming, 37
RUNCOTINOUS, 30
SETBRAKEMODE, 30
setBrakeMode, 38
SETCONTROLTHRESHOLD, 30
SETCURRENT, 30
setDriveCurrent, 38
SETHOLDCURRENT, 30
setHoldCurrent, 38
SETMAXACCELERATION, 30
setMaxAcceleration, 39
SETMAXDECELERATION, 30
SETMAXVELOCITY, 30
setMaxVelocity, 39
setPosition, 40
SETRPM, 30
SETUP, 30
setVelocity, 40
startHoming, 40
STOP, 30
stop, 41
stp_reg_t, 30
wait_while_busy, 41
bioscara_hardware_drivers::Gripper, 73
_freq, 77
_pwm, 77
disable, 75
enable, 75
Gripper, 75
setPosition, 76
setServoPosition, 76
bioscara_hardware_drivers::Joint, 78
_home, 83
~Joint, 82
address, 92
checkCom, 83
checkOrientation, 83
deinit, 84
disableCL, 84
enable, 84
enableStallguard, 85

getFlags, 85
getHomingOffset, 85
getPosition, 85
getVelocity, 86
handle, 92
init, 86
Joint, 82
max, 92
min, 93
moveSteps, 86
offset, 93
postHoming, 87
read, 87
reduction, 93
setBrakeMode, 88
setDriveCurrent, 88
setHoldCurrent, 88
setHomingOffset, 90
setMaxAcceleration, 90
setMaxVelocity, 90
setPosition, 91
setVelocity, 91
stop, 91
write, 92
bioscara_hardware_drivers::MockGripper, 100
MockGripper, 101
bioscara_hardware_drivers::MockJoint, 102
_home, 105
async_start_time, 110
checkOrientation, 105
disable, 106
enable, 106
getDeltaT, 106
getFlags, 108
getPosition, 108
getVelocity, 108
isHomed, 109
last_set_position, 110
last_set_velocity, 110
MockJoint, 105
op_mode, 110
q, 110
qd, 111
setPosition, 109
setVelocity, 109
stop, 110
bioscara_hardware_interfaces, 12
HW_IF_HOME, 12
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 42
_joint_cfg, 53
_joint_command_modes, 53
_joints, 53
_new_joint_command_modes, 54
_ordered_joint_state_interfaces_ptr, 54
activate_joint, 45
deactivate_joint, 45
mtx, 54
on_activate, 45
on_cleanup, 46
on_configure, 46
on_deactivate, 46
on_error, 47
on_init, 48
on_shutdown, 49
perform_command_mode_switch, 49
prepare_command_mode_switch, 50
read, 50
split_interface_string_to_joint_and_name, 51
start_homing, 51
stop_homing, 52
write, 52
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_conf 93
drive_current, 94
hold_current, 94
homing, 94
i2c_address, 95
max, 95
max_acceleration, 95
max_velocity, 95
min, 95
reduction, 96
stall_threshold, 96
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_hom 96
acceleration, 97
current, 97
speed, 97
threshold, 97
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 55
_gripper, 60
_gripper_cfg, 60
_last_pos, 60
_vel, 60
on_activate, 56
on_cleanup, 57
on_configure, 57
on_deactivate, 57
on_error, 58
on_init, 58
on_shutdown, 59
read, 59
write, 59
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::grippe 77
init_pos, 78
max, 78
min, 78
offset, 78
reduction, 78
bioscara_joint_firmware, 13
ANGLEMOVED, 14
blk_reg, 18
blocking_handler, 15

CHECKORIENTATION, 15
CLEARSTALL, 15
DISABLECLOSEDLOOP, 15
DISABLEPID, 15
DISABLESTALLGUARD, 15
ENABLECLOSEDLOOP, 15
ENABLEPID, 15
ENABLESTALLGUARD, 14
GETDRIVERRPM, 14
GETENCODERRPM, 15
GETMOTORSTATE, 14
GETPIDERROR, 15
HOME, 15
HOMEOFFSET, 15
loop, 15
MOVEANGLE, 14
MOVESTEPS, 14
MOVETOANGLE, 14
MOVETOEND, 15
non_blocking_handler, 16
PING, 14
readValue, 16
receiveEvent, 16
reg, 18
requestEvent, 17
RUNCOTINOUS, 14
rx_buf, 18
rx_data_ready, 18
rx_length, 18
set_flags_for_blocking_handler, 17
SETBRAKEMODE, 15
SETCONTROLTHRESHOLD, 15
SETCURRENT, 14
SETHOLDCURRENT, 14
SETMAXACCELERATION, 14
SETMAXDECELERATION, 14
SETMAXVELOCITY, 14
SETRPM, 14
SETUP, 14
setup, 17
stall_threshold, 17
stepper, 18
STOP, 15
stp_reg_t, 14
tx_buf, 19
tx_length, 19
writeValue, 18
bioscara_joint_firmware::Lowpass, 98
K, 99
Lowpass, 98
resetState, 99
tau, 99
Ts, 99
updateState, 99
x, 99
bioscara_joint_firmware::MovMax, 111
cb_data, 112
cb_index, 112
M, 112
MovMax, 111
updateState, 112
bioscara_rviz_plugin, 19
bioscara_rviz_plugin::BioscaraPanel, 61
~BioscaraPanel, 63
arm_en_btn_cb, 64
BioscaraPanel, 63
check_activation_conditions, 64
cm_state_callback, 64
cm_state_subsription_, 71
configure_controller, 64
configure_controller_client_, 71
controller_states_, 71
ctrl_en_btn_cb, 65
dynamic_joint_state_msg_to_map, 65
ensure_jsb_is_active, 65
gripper_en_btn_cb, 65
hardware_state_client_, 72
hardware_states_, 72
homing_cmd, 65
homing_publisher_, 72
joint_state_callback, 66
joint_state_subsription_, 72
joint_states_, 72
named_lcs_msg_to_map, 66
node_, 72
onInitialize, 66
prepopulate_joint_state_map, 67
prepopulate_state_map, 67
print_cm_map, 67
prune_timer_, 73
set_controller_state, 68
set_en_btn, 68
set_hardware_component_state, 68
set_homing_state_label, 69
set_state_label, 69
switch_controller_client_, 73
switch_controllers, 69
target_state_from_current, 70
ui_, 73
update_homing_grp_state, 70
update_homing_state_labels, 70
update_state_label_and_btn, 70
update_state_labels_and_btns, 71
BioscaraPanel
bioscara_rviz_plugin::BioscaraPanel, 63
blk_reg
bioscara_joint_firmware, 18
blocking_handler
bioscara_joint_firmware, 15

CallbackReturn
single_trigger_controller, 19
cb_data
bioscara_joint_firmware::MovMax, 112
cb_index
bioscara_joint_firmware::MovMax, 112
check_activation_conditions

bioscara_rviz_plugin::BioscaraPanel, 64
check_if_configured_interfaces_matches_received
single_trigger_controller::SingleTriggerController, 118
checkCom
bioscara_hardware_drivers::Joint, 83
CHECKORIENTATION
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
checkOrientation
bioscara_hardware_drivers::BaseJoint, 31
bioscara_hardware_drivers::Joint, 83
bioscara_hardware_drivers::MockJoint, 105
chippath
RPI_PWM, 115
CLEARSTALL
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
closeI2CDevHandle
uI2C.cpp, 160
uI2C.h, 153
cm_state_callback
bioscara_rviz_plugin::BioscaraPanel, 64
cm_state_subsription_
bioscara_rviz_plugin::BioscaraPanel, 71
CmdType
single_trigger_controller, 19
COMM_ERROR
bioscara_hardware_drivers, 12
command_interface_configuration
single_trigger_controller::SingleTriggerController, 118
command_interface_types_
single_trigger_controller::SingleTriggerController, 123
command_interfaces_map_
single_trigger_controller::SingleTriggerController, 123
command_subscriber_
single_trigger_controller::SingleTriggerController, 123
common.h
DUMP_BUFFER, 140
configuration.h
ADR, 128
MAXACCEL, 128
MAXVEL, 128
configure_controller
bioscara_rviz_plugin::BioscaraPanel, 64
configure_controller_client_
bioscara_rviz_plugin::BioscaraPanel, 71
controller_states_
bioscara_rviz_plugin::BioscaraPanel, 71
create_map_of_references_to_interfaces
single_trigger_controller::SingleTriggerController, 119
ctrl_en_btn_cb
bioscara_rviz_plugin::BioscaraPanel, 65
current
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 97
current_b_cmd
bioscara_hardware_drivers::BaseJoint, 41

deactivate_joint
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 45
DEG2RAD
uTransmission.h, 156
deinit
bioscara_hardware_drivers::BaseGripper, 23
bioscara_hardware_drivers::BaseJoint, 32
bioscara_hardware_drivers::Joint, 84
disable
bioscara_hardware_drivers::BaseGripper, 23
bioscara_hardware_drivers::BaseJoint, 32
bioscara_hardware_drivers::Gripper, 75
bioscara_hardware_drivers::MockJoint, 106
RPI_PWM, 113
disableCL
bioscara_hardware_drivers::BaseJoint, 32
bioscara_hardware_drivers::Joint, 84
DISABLECLOSEDLOOP
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
DISABLEPID
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
DISABLESTALLGUARD
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
docs/doxygen/index.md, 127
drive_current
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 94
DUMP_BUFFER
common.h, 140
joint.h, 132
dynamic_joint_state_msg_to_map
bioscara_rviz_plugin::BioscaraPanel, 65

enable
bioscara_hardware_drivers::BaseGripper, 24
bioscara_hardware_drivers::BaseJoint, 33
bioscara_hardware_drivers::Gripper, 75
bioscara_hardware_drivers::Joint, 84
bioscara_hardware_drivers::MockJoint, 106
RPI_PWM, 113
ENABLECLOSEDLOOP
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
ENABLEPID
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
ENABLESTALLGUARD
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14

enableStallguard
bioscara_hardware_drivers::BaseJoint, 33
bioscara_hardware_drivers::Joint, 85
ensure_jsb_is_active
bioscara_rviz_plugin::BioscaraPanel, 65
err_type_t
bioscara_hardware_drivers, 11
ERROR
bioscara_hardware_drivers, 12
error_to_string
bioscara_hardware_drivers, 12

flags
bioscara_hardware_drivers::BaseJoint, 41

get_state_interfaces_names
single_trigger_controller::SingleTriggerController, 119
getCurrentBCmd
bioscara_hardware_drivers::BaseJoint, 34
getDeltaT
bioscara_hardware_drivers::MockJoint, 106
GETDRIVERRPM
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
GETENCODERRPM
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
getFlags
bioscara_hardware_drivers::BaseJoint, 34
bioscara_hardware_drivers::Joint, 85
bioscara_hardware_drivers::MockJoint, 108
getHomingOffset
bioscara_hardware_drivers::Joint, 85
GETMOTORSTATE
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
GETPIDERROR
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
getPosition
bioscara_hardware_drivers::BaseGripper, 24
bioscara_hardware_drivers::BaseJoint, 34
bioscara_hardware_drivers::Joint, 85
bioscara_hardware_drivers::MockJoint, 108
getVelocity
bioscara_hardware_drivers::BaseJoint, 35
bioscara_hardware_drivers::Joint, 86
bioscara_hardware_drivers::MockJoint, 108
Gripper
bioscara_hardware_drivers::Gripper, 75
gripper_en_btn_cb
bioscara_rviz_plugin::BioscaraPanel, 65

handle
bioscara_hardware_drivers::Joint, 92
hardware_state_client_
bioscara_rviz_plugin::BioscaraPanel, 72
hardware_states_
bioscara_rviz_plugin::BioscaraPanel, 72
hold_current
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 94
HOME
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
home
bioscara_hardware_drivers::BaseJoint, 35
HOMEOFFSET
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
homing
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 94
homing_cmd
bioscara_rviz_plugin::BioscaraPanel, 65
homing_publisher_
bioscara_rviz_plugin::BioscaraPanel, 72
HW_IF_HOME
bioscara_hardware_interfaces, 12

i2c_address
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 95
INCORRECT_STATE
bioscara_hardware_drivers, 12
init
bioscara_hardware_drivers::BaseGripper, 24
bioscara_hardware_drivers::BaseJoint, 36
bioscara_hardware_drivers::Joint, 86
init_pos
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::g 78
initialize_state_msg
single_trigger_controller::SingleTriggerController, 120
INT_handler
joint_comm_node.cpp, 158
InterfacesNames
single_trigger_controller, 20
INVALID_ARGUMENT
bioscara_hardware_drivers, 12
isBusy
bioscara_hardware_drivers::BaseJoint, 36
isEnabled
bioscara_hardware_drivers::BaseJoint, 36
isHomed
bioscara_hardware_drivers::BaseJoint, 36
bioscara_hardware_drivers::MockJoint, 109
isStalled
bioscara_hardware_drivers::BaseJoint, 37

J1
joint.ino, 136
joint_comm_node.cpp, 158
J2
joint_comm_node.cpp, 158
J3

joint_comm_node.cpp, 158
J4
joint_comm_node.cpp, 158
Joint
bioscara_hardware_drivers::Joint, 82
joint.h
ACK, 132
DUMP_BUFFER, 132
MAX_BUFFER, 133
NACK, 133
RFLAGS_SIZE, 133
joint.ino
J1, 136
loop, 136
setup, 136
JOINT2ACTUATOR
uTransmission.h, 156
joint_comm_node.cpp
INT_handler, 158
J1, 158
J2, 158
J3, 158
J4, 158
main, 158
joint_state_callback
bioscara_rviz_plugin::BioscaraPanel, 66
joint_state_subsription_
bioscara_rviz_plugin::BioscaraPanel, 72
joint_states_
bioscara_rviz_plugin::BioscaraPanel, 72

K
bioscara_joint_firmware::Lowpass, 99

last_set_position
bioscara_hardware_drivers::MockJoint, 110
last_set_velocity
bioscara_hardware_drivers::MockJoint, 110
lib/joint_firmware/joint/configuration.h, 127, 128
lib/joint_firmware/joint/filters.h, 129, 130
lib/joint_firmware/joint/joint.h, 130, 133
lib/joint_firmware/joint/joint.ino, 134
lib/joint_firmware/joint/stall.h, 136, 137
lib/ros2_ws/src/bioscara_rviz_plugin/include/bioscara_rviz_plugin/bioscara_panel.hpp, 137, 138
lib/ros2_ws/src/bioscara_rviz_plugin/src/bioscara_panel.cpp, 139
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/common.h, 140, 141
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mBaseJoint.h, 141, 142
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mJoint.h, 144, 145
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mJoint.hpp, 146, 147
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/mMockJoint.h, 147, 148
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/uErr.h, 149, 151
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/uTransmission.h, 152, 155
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/include/bioscara_arm_hardware_driver/uTypes.h, 155, 157
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/joint_comm_node.cpp, 157
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mBaseJoint.cpp, 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mJoint.cpp, 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/mMockJoint.cpp, 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uErr.cpp, 159
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_driver/src/uTransmission.cpp, 160
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/include/bioscara_arm_hardware_interface/bioscara_arm_hardware_interface.hpp, 162, 163
lib/ros2_ws/src/dalsa_bioscara_arm/bioscara_arm_hardware_interface/src/bioscara_arm_hardware_interface.cpp, 164
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/common.h, 165, 166
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mBaseJoint.h, 166, 167
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mJoint.h, 168, 169
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mJoint.hpp, 169, 170
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/mMockJoint.h, 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/uErr.h, 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_driver/include/bioscara_gripper_hardware_driver/uTransmission.h, 171
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/include/bioscara_gripper_hardware_interface/bioscara_gripper_hardware_interface.hpp, 172
lib/ros2_ws/src/dalsa_bioscara_grippers/bioscara_gripper_hardware_interface/src/bioscara_gripper_hardware_interface.cpp, 173
lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/include/single_trigger_controller/single_trigger_controller.hpp, 174, 175
lib/ros2_ws/src/dalsa_controllers/single_trigger_controller/src/single_trigger_controller.cpp, 176
loop
bioscara_joint_firmware, 15
joint.ino, 136
Lowpass
bioscara_joint_firmware::Lowpass, 98

M
bioscara_joint_firmware::MovMax, 112
M_PI
uTransmission.h, 156
main
joint_comm_node.cpp, 158
MapOfReferencesToCommandInterfaces
single_trigger_controller, 20
MapOfReferencesToStateInterfaces
single_trigger_controller, 20
max

bioscara_hardware_drivers::Joint, 92
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 95
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 78
max_acceleration
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 95
MAX_BUFFER
joint.h, 133
uI2C.h, 153
max_velocity
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 95
MAXACCEL
configuration.h, 128
MAXVEL
configuration.h, 128
min
bioscara_hardware_drivers::Joint, 93
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 95
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::gripper_config_t, 78
MockGripper
bioscara_hardware_drivers::MockGripper, 101
MockJoint
bioscara_hardware_drivers::MockJoint, 105
MOVEANGLE
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
MOVESTEPS
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
moveSteps
bioscara_hardware_drivers::BaseJoint, 37
bioscara_hardware_drivers::Joint, 86
MOVETOANGLE
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
MOVETOEND
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
MovMax
bioscara_joint_firmware::MovMax, 111
mtx
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 54

NACK
joint.h, 133
uI2C.h, 153
name
bioscara_hardware_drivers::BaseJoint, 42
named_lcs_msg_to_map
bioscara_rviz_plugin::BioscaraPanel, 66
node_
bioscara_rviz_plugin::BioscaraPanel, 72
non_blocking_handler
bioscara_joint_firmware, 16
NONE
bioscara_hardware_drivers::BaseJoint, 30
NOT_ENABLED
bioscara_hardware_drivers, 12
NOT_HOMED
bioscara_hardware_drivers, 12
NOT_INIT
bioscara_hardware_drivers, 12

offset
bioscara_hardware_drivers::Joint, 93
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::g
78
OK
bioscara_hardware_drivers, 12
on_activate
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 45
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 56
single_trigger_controller::SingleTriggerController, 120
on_cleanup
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 46
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 57
on_configure
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 46
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 57
single_trigger_controller::SingleTriggerController, 120
on_deactivate
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 46
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 57
single_trigger_controller::SingleTriggerController, 121
on_error
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 47
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 58
on_init
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 48
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 58
single_trigger_controller::SingleTriggerController, 121
on_shutdown
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 49
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 59

onInitialize
bioscara_rviz_plugin::BioscaraPanel, 66
op_mode
bioscara_hardware_drivers::MockJoint, 110
openI2CDevHandle
uI2C.cpp, 160
uI2C.h, 154

param_listener_
single_trigger_controller::SingleTriggerController, 124
params_
single_trigger_controller::SingleTriggerController, 124
per
RPI_PWM, 115
perform_command_mode_switch
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 49
PING
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
postHoming
bioscara_hardware_drivers::BaseJoint, 37
bioscara_hardware_drivers::Joint, 87
prepare_command_mode_switch
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 50
prepopulate_joint_state_map
bioscara_rviz_plugin::BioscaraPanel, 67
prepopulate_state_map
bioscara_rviz_plugin::BioscaraPanel, 67
print_cm_map
bioscara_rviz_plugin::BioscaraPanel, 67
prune_timer_
bioscara_rviz_plugin::BioscaraPanel, 73
pwmpath
RPI_PWM, 115

q
bioscara_hardware_drivers::MockJoint, 110
qd
bioscara_hardware_drivers::MockJoint, 111

RAD2DEG
uTransmission.h, 157
read
bioscara_hardware_drivers::Joint, 87
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 50
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 59
readFromI2CDev
uI2C.cpp, 161
uI2C.h, 154
readValue
bioscara_joint_firmware, 16
realtime_state_publisher_
single_trigger_controller::SingleTriggerController, 124
receiveEvent
bioscara_joint_firmware, 16
reduction
bioscara_hardware_drivers::Joint, 93
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_ 96
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface::g 78
reg
bioscara_joint_firmware, 18
requestEvent
bioscara_joint_firmware, 17
resetState
bioscara_joint_firmware::Lowpass, 99
retrieve_last_position
bioscara_hardware_drivers::BaseGripper, 25
RETURN_ON_ERROR
uErr.h, 150
RETURN_ON_FALSE
uErr.h, 150
RETURN_ON_NEGATIVE
uErr.h, 150
RFLAGS_SIZE
joint.h, 133
uI2C.h, 153
RPI_PWM, 112
~RPI_PWM, 113
chippath, 115
disable, 113
enable, 113
per, 115
pwmpath, 115
setDutyCycle, 114
setDutyCycleNS, 114
setPeriod, 114
start, 114
stop, 115
writeSYS, 115
RUNCOTINOUS
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
rx_buf
bioscara_joint_firmware, 18
rx_data_ready
bioscara_joint_firmware, 18
rx_length
bioscara_joint_firmware, 18

save_last_position
bioscara_hardware_drivers::BaseGripper, 25
set_controller_state
bioscara_rviz_plugin::BioscaraPanel, 68
set_en_btn
bioscara_rviz_plugin::BioscaraPanel, 68
set_flags_for_blocking_handler
bioscara_joint_firmware, 17
set_hardware_component_state

bioscara_rviz_plugin::BioscaraPanel, 68
set_homing_state_label
bioscara_rviz_plugin::BioscaraPanel, 69
set_state_label
bioscara_rviz_plugin::BioscaraPanel, 69
SETBRAKEMODE
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
setBrakeMode
bioscara_hardware_drivers::BaseJoint, 38
bioscara_hardware_drivers::Joint, 88
SETCONTROLTHRESHOLD
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
SETCURRENT
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setDriveCurrent
bioscara_hardware_drivers::BaseJoint, 38
bioscara_hardware_drivers::Joint, 88
setDutyCycle
RPI_PWM, 114
setDutyCycleNS
RPI_PWM, 114
SETHOLDCURRENT
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setHoldCurrent
bioscara_hardware_drivers::BaseJoint, 38
bioscara_hardware_drivers::Joint, 88
setHomingOffset
bioscara_hardware_drivers::Joint, 90
SETMAXACCELERATION
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setMaxAcceleration
bioscara_hardware_drivers::BaseJoint, 39
bioscara_hardware_drivers::Joint, 90
SETMAXDECELERATION
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
SETMAXVELOCITY
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setMaxVelocity
bioscara_hardware_drivers::BaseJoint, 39
bioscara_hardware_drivers::Joint, 90
setOffset
bioscara_hardware_drivers::BaseGripper, 25
setPeriod
RPI_PWM, 114
setPosition
bioscara_hardware_drivers::BaseGripper, 25
bioscara_hardware_drivers::BaseJoint, 40
bioscara_hardware_drivers::Gripper, 76
bioscara_hardware_drivers::Joint, 91
bioscara_hardware_drivers::MockJoint, 109
setReduction
bioscara_hardware_drivers::BaseGripper, 26
SETRPM
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setServoPosition
bioscara_hardware_drivers::Gripper, 76
SETUP
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
setup
bioscara_joint_firmware, 17
joint.ino, 136
setVelocity
bioscara_hardware_drivers::BaseJoint, 40
bioscara_hardware_drivers::Joint, 91
bioscara_hardware_drivers::MockJoint, 109
single_trigger_controller, 19
CallbackReturn, 19
CmdType, 19
InterfacesNames, 20
MapOfReferencesToCommandInterfaces, 20
MapOfReferencesToStateInterfaces, 20
StateInterfaces, 20
StateType, 20
single_trigger_controller::SingleTriggerController, 115
apply_command, 117
apply_state_value, 118
check_if_configured_interfaces_matches_received, 118
command_interface_configuration, 118
command_interface_types_, 123
command_interfaces_map_, 123
command_subscriber_, 123
create_map_of_references_to_interfaces, 119
get_state_interfaces_names, 119
initialize_state_msg, 120
on_activate, 120
on_configure, 120
on_deactivate, 121
on_init, 121
param_listener_, 124
params_, 124
realtime_state_publisher_, 124
SingleTriggerController, 117
state_interface_configuration, 121
state_interface_types_, 124
state_interfaces_map_, 124
state_msg_, 124
state_publisher_, 125
store_command_interface_types, 121
store_state_interface_types, 122
update, 122
update_commands, 122
update_dynamic_map_parameters, 123
update_states, 123
SingleTriggerController
single_trigger_controller::SingleTriggerController, 117

speed
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 97
split_interface_string_to_joint_and_name
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 51
stall_threshold
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 96
bioscara_joint_firmware, 17
STALLED
bioscara_hardware_drivers, 12
start
RPI_PWM, 114
start_homing
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 51
startHoming
bioscara_hardware_drivers::BaseJoint, 40
state_interface_configuration
single_trigger_controller::SingleTriggerController, 121
state_interface_types_
single_trigger_controller::SingleTriggerController, 124
state_interfaces_map_
single_trigger_controller::SingleTriggerController, 124
state_msg_
single_trigger_controller::SingleTriggerController, 124
state_publisher_
single_trigger_controller::SingleTriggerController, 125
StateInterfaces
single_trigger_controller, 20
StateType
single_trigger_controller, 20
stepper
bioscara_joint_firmware, 18
STOP
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 15
stop
bioscara_hardware_drivers::BaseJoint, 41
bioscara_hardware_drivers::Joint, 91
bioscara_hardware_drivers::MockJoint, 110
RPI_PWM, 115
stop_homing
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 52
store_command_interface_types
single_trigger_controller::SingleTriggerController, 121
store_state_interface_types
single_trigger_controller::SingleTriggerController, 122
stp_reg_t
bioscara_hardware_drivers::BaseJoint, 30
bioscara_joint_firmware, 14
switch_controller_client_
bioscara_rviz_plugin::BioscaraPanel, 73
switch_controllers
bioscara_rviz_plugin::BioscaraPanel, 69

target_state_from_current
bioscara_rviz_plugin::BioscaraPanel, 70
tau
bioscara_joint_firmware::Lowpass, 99
threshold
bioscara_hardware_interfaces::BioscaraArmHardwareInterface::joint_config_t, 97
Ts
bioscara_joint_firmware::Lowpass, 99
tx_buf
bioscara_joint_firmware, 19
tx_length
bioscara_joint_firmware, 19

uErr.h
RETURN_ON_ERROR, 150
RETURN_ON_FALSE, 150
RETURN_ON_NEGATIVE, 150
uI2C.cpp
closeI2CDevHandle, 160
openI2CDevHandle, 160
readFromI2CDev, 161
writeToI2CDev, 161
uI2C.h
ACK, 153
closeI2CDevHandle, 153
MAX_BUFFER, 153
NACK, 153
openI2CDevHandle, 154
readFromI2CDev, 154
RFLAGS_SIZE, 153
writeToI2CDev, 154
ui_
bioscara_rviz_plugin::BioscaraPanel, 73
update
single_trigger_controller::SingleTriggerController, 122
update_commands
single_trigger_controller::SingleTriggerController, 122
update_dynamic_map_parameters
single_trigger_controller::SingleTriggerController, 123
update_homing_grp_state
bioscara_rviz_plugin::BioscaraPanel, 70
update_homing_state_labels
bioscara_rviz_plugin::BioscaraPanel, 70
update_state_label_and_btn
bioscara_rviz_plugin::BioscaraPanel, 70
update_state_labels_and_btns
bioscara_rviz_plugin::BioscaraPanel, 71
update_states

single_trigger_controller::SingleTriggerController, 123
updateState
bioscara_joint_firmware::Lowpass, 99
bioscara_joint_firmware::MovMax, 112
uTransmission.h
ACTUATOR2JOINT, 156
DEG2RAD, 156
JOINT2ACTUATOR, 156
M_PI, 156
RAD2DEG, 157

wait_while_busy
bioscara_hardware_drivers::BaseJoint, 41
write
bioscara_hardware_drivers::Joint, 92
bioscara_hardware_interfaces::BioscaraArmHardwareInterface, 52
bioscara_hardware_interfaces::BioscaraGripperHardwareInterface, 59
writeSYS
RPI_PWM, 115
writeToI2CDev
uI2C.cpp, 161
uI2C.h, 154
writeValue
bioscara_joint_firmware, 18

x
bioscara_joint_firmware::Lowpass, 99