

WISCO\_OverUnder

Generated by Doxygen 1.10.0



---

<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List . . . . .	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy . . . . .	3
<b>3 Class Index</b>	<b>7</b>
3.1 Class List . . . . .	7
<b>4 Namespace Documentation</b>	<b>13</b>
4.1 pros_adapters Namespace Reference . . . . .	13
4.1.1 Detailed Description . . . . .	13
4.2 wisco Namespace Reference . . . . .	14
4.2.1 Detailed Description . . . . .	15
4.3 wisco::alliances Namespace Reference . . . . .	15
4.3.1 Detailed Description . . . . .	15
4.4 wisco::autons Namespace Reference . . . . .	15
4.4.1 Detailed Description . . . . .	16
4.5 wisco::configs Namespace Reference . . . . .	16
4.5.1 Detailed Description . . . . .	16
4.6 wisco::control Namespace Reference . . . . .	16
4.6.1 Detailed Description . . . . .	17
4.7 wisco::control::boomerang Namespace Reference . . . . .	17
4.7.1 Detailed Description . . . . .	17
4.8 wisco::control::motion Namespace Reference . . . . .	17
4.8.1 Detailed Description . . . . .	18
4.8.2 Enumeration Type Documentation . . . . .	18
4.8.2.1 ETurnDirection . . . . .	18
4.9 wisco::control::path Namespace Reference . . . . .	18
4.9.1 Detailed Description . . . . .	19
4.9.2 Function Documentation . . . . .	19
4.9.2.1 operator*() . . . . .	19
4.10 wisco::hal Namespace Reference . . . . .	19
4.10.1 Detailed Description . . . . .	20
4.10.2 Enumeration Type Documentation . . . . .	20
4.10.2.1 DistanceBooleanMode . . . . .	20
4.11 wisco::io Namespace Reference . . . . .	20
4.11.1 Detailed Description . . . . .	21
4.12 wisco::menu Namespace Reference . . . . .	21
4.12.1 Detailed Description . . . . .	21
4.12.2 Function Documentation . . . . .	21
4.12.2.1 startButtonEventHandler() . . . . .	21
4.12.2.2 settingsButtonEventHandler() . . . . .	22

---

4.12.2.3 settingsBackButtonEventHandler() . . . . .	22
4.12.2.4 settingsButtonMatrixEventHandler() . . . . .	23
4.13 wisco::profiles Namespace Reference . . . . .	23
4.13.1 Detailed Description . . . . .	23
4.14 wisco::robot Namespace Reference . . . . .	23
4.14.1 Detailed Description . . . . .	24
4.15 wisco::robot::subsystems Namespace Reference . . . . .	24
4.15.1 Detailed Description . . . . .	25
4.16 wisco::robot::subsystems::drive Namespace Reference . . . . .	25
4.16.1 Detailed Description . . . . .	25
4.17 wisco::robot::subsystems::elevator Namespace Reference . . . . .	25
4.17.1 Detailed Description . . . . .	26
4.18 wisco::robot::subsystems::hang Namespace Reference . . . . .	26
4.18.1 Detailed Description . . . . .	27
4.19 wisco::robot::subsystems::intake Namespace Reference . . . . .	27
4.19.1 Detailed Description . . . . .	27
4.20 wisco::robot::subsystems::loader Namespace Reference . . . . .	27
4.20.1 Detailed Description . . . . .	28
4.21 wisco::robot::subsystems::position Namespace Reference . . . . .	28
4.21.1 Detailed Description . . . . .	29
4.22 wisco::robot::subsystems::umbrella Namespace Reference . . . . .	29
4.22.1 Detailed Description . . . . .	29
4.23 wisco::robot::subsystems::wings Namespace Reference . . . . .	29
4.23.1 Detailed Description . . . . .	30
4.24 wisco::rtos Namespace Reference . . . . .	30
4.24.1 Detailed Description . . . . .	30
4.25 wisco::testing Namespace Reference . . . . .	30
4.25.1 Detailed Description . . . . .	31
4.26 wisco::testing::pros_testing Namespace Reference . . . . .	31
4.26.1 Detailed Description . . . . .	31
4.26.2 Variable Documentation . . . . .	31
4.26.2.1 FILE_PATH . . . . .	31
4.27 wisco::user Namespace Reference . . . . .	31
4.27.1 Detailed Description . . . . .	33
4.27.2 Enumeration Type Documentation . . . . .	33
4.27.2.1 EControl . . . . .	33
4.27.2.2 EControllerAnalog . . . . .	34
4.27.2.3 EControllerDigital . . . . .	34
4.27.2.4 EControlType . . . . .	34
4.28 wisco::user::drive Namespace Reference . . . . .	35
4.28.1 Detailed Description . . . . .	35
4.28.2 Enumeration Type Documentation . . . . .	35

4.28.2.1 EChassisControlMode . . . . .	35
4.29 wisco::user::elevator Namespace Reference . . . . .	35
4.29.1 Detailed Description . . . . .	36
4.29.2 Enumeration Type Documentation . . . . .	36
4.29.2.1 EElevatorControlMode . . . . .	36
4.30 wisco::user::hang Namespace Reference . . . . .	36
4.30.1 Detailed Description . . . . .	37
4.30.2 Enumeration Type Documentation . . . . .	37
4.30.2.1 EHangControlMode . . . . .	37
4.31 wisco::user::intake Namespace Reference . . . . .	37
4.31.1 Detailed Description . . . . .	38
4.31.2 Enumeration Type Documentation . . . . .	38
4.31.2.1 EIIntakeControlMode . . . . .	38
4.32 wisco::user::loader Namespace Reference . . . . .	38
4.32.1 Detailed Description . . . . .	38
4.32.2 Enumeration Type Documentation . . . . .	39
4.32.2.1 ELoaderControlMode . . . . .	39
4.33 wisco::user::umbrella Namespace Reference . . . . .	39
4.33.1 Detailed Description . . . . .	39
4.33.2 Enumeration Type Documentation . . . . .	40
4.33.2.1 EUmbrellaControlMode . . . . .	40
4.34 wisco::user::wings Namespace Reference . . . . .	40
4.34.1 Detailed Description . . . . .	40
4.34.2 Enumeration Type Documentation . . . . .	40
4.34.2.1 EWingsControlMode . . . . .	40
<b>5 Class Documentation</b>	<b>41</b>
5.1 MatchControllerFactory Class Reference . . . . .	41
5.1.1 Detailed Description . . . . .	41
5.1.2 Member Function Documentation . . . . .	41
5.1.2.1 createMatchController() . . . . .	41
5.2 pros_adapters::ProsClock Class Reference . . . . .	42
5.2.1 Detailed Description . . . . .	43
5.2.2 Member Function Documentation . . . . .	43
5.2.2.1 clone() . . . . .	43
5.2.2.2 getTime() . . . . .	43
5.3 pros_adapters::ProsController Class Reference . . . . .	44
5.3.1 Detailed Description . . . . .	45
5.3.2 Constructor & Destructor Documentation . . . . .	45
5.3.2.1 ProsController() . . . . .	45
5.3.3 Member Function Documentation . . . . .	46
5.3.3.1 taskLoop() . . . . .	46

5.3.3.2 updateRumble()	46
5.3.3.3 taskUpdate()	47
5.3.3.4 initialize()	47
5.3.3.5 run()	47
5.3.3.6 getAnalog()	47
5.3.3.7 getDigital()	48
5.3.3.8 getNewDigital()	48
5.3.3.9 rumble()	49
5.3.4 Member Data Documentation	49
5.3.4.1 TASK_DELAY	49
5.3.4.2 RUMBLE_REFRESH_RATE	49
5.3.4.3 ANALOG_CONVERSION	50
5.3.4.4 MAX_RUMBLE_LENGTH	50
5.3.4.5 ANALOG_MAP	50
5.3.4.6 DIGITAL_MAP	50
5.3.4.7 m_controller	51
5.3.4.8 mutex	51
5.3.4.9 rumble_pattern	51
5.3.4.10 new_rumble_pattern	51
5.3.4.11 last_rumble_refresh	52
5.4 pros_adapters::ProsDelay Class Reference	52
5.4.1 Detailed Description	52
5.4.2 Member Function Documentation	53
5.4.2.1 clone()	53
5.4.2.2 delay()	53
5.4.2.3 delayUntil()	53
5.5 pros_adapters::ProsDistance Class Reference	54
5.5.1 Detailed Description	55
5.5.2 Constructor & Destructor Documentation	55
5.5.2.1 ProsDistance()	55
5.5.3 Member Function Documentation	55
5.5.3.1 initialize()	55
5.5.3.2 reset()	56
5.5.3.3 getDistance()	56
5.5.4 Member Data Documentation	56
5.5.4.1 UNIT_CONVERTER	56
5.5.4.2 m_sensor	56
5.5.4.3 m_tuning_constant	57
5.5.4.4 m_tuning_offset	57
5.6 pros_adapters::ProsEXPMotor Class Reference	57
5.6.1 Detailed Description	58
5.6.2 Constructor & Destructor Documentation	58

---

5.6.2.1 ProsEXPMotor() . . . . .	58
5.6.3 Member Function Documentation . . . . .	59
5.6.3.1 initialize() . . . . .	59
5.6.3.2 getTorqueConstant() . . . . .	59
5.6.3.3 getResistance() . . . . .	59
5.6.3.4 getAngularVelocityConstant() . . . . .	60
5.6.3.5 getGearRatio() . . . . .	60
5.6.3.6 getAngularVelocity() . . . . .	60
5.6.3.7 setVoltage() . . . . .	60
5.6.4 Member Data Documentation . . . . .	61
5.6.4.1 TORQUE_CONSTANT . . . . .	61
5.6.4.2 RESISTANCE . . . . .	61
5.6.4.3 ANGULAR_VELOCITY_CONSTANT . . . . .	61
5.6.4.4 GEAR_RATIO . . . . .	61
5.6.4.5 VELOCITY_CONVERSION . . . . .	62
5.6.4.6 VOLTAGE_CONVERSION . . . . .	62
5.6.4.7 m_motor . . . . .	62
5.7 pros_adapters::ProsHeading Class Reference . . . . .	62
5.7.1 Detailed Description . . . . .	63
5.7.2 Constructor & Destructor Documentation . . . . .	63
5.7.2.1 ProsHeading() . . . . .	63
5.7.3 Member Function Documentation . . . . .	64
5.7.3.1 initialize() . . . . .	64
5.7.3.2 reset() . . . . .	64
5.7.3.3 getHeading() . . . . .	64
5.7.3.4 setHeading() . . . . .	64
5.7.3.5 getRotation() . . . . .	65
5.7.3.6 setRotation() . . . . .	65
5.7.4 Member Data Documentation . . . . .	66
5.7.4.1 UNIT_CONVERTER . . . . .	66
5.7.4.2 m_sensor . . . . .	66
5.7.4.3 m_tuning_constant . . . . .	66
5.8 pros_adapters::ProsMutex Class Reference . . . . .	66
5.8.1 Detailed Description . . . . .	67
5.8.2 Member Function Documentation . . . . .	67
5.8.2.1 take() . . . . .	67
5.8.2.2 give() . . . . .	67
5.8.3 Member Data Documentation . . . . .	68
5.8.3.1 mutex . . . . .	68
5.9 pros_adapters::ProsPiston Class Reference . . . . .	68
5.9.1 Detailed Description . . . . .	69
5.9.2 Constructor & Destructor Documentation . . . . .	69

5.9.2.1 ProsPiston()	69
5.9.3 Member Function Documentation	69
5.9.3.1 extend()	69
5.9.3.2 retract()	70
5.9.3.3 toggle()	70
5.9.3.4 isExtended()	70
5.9.3.5 isRetracted()	71
5.9.4 Member Data Documentation	71
5.9.4.1 m_digital_out	71
5.9.4.2 m_extended_value	71
5.9.4.3 extended	71
5.10 pros_adapters::ProsRotation Class Reference	72
5.10.1 Detailed Description	72
5.10.2 Constructor & Destructor Documentation	73
5.10.2.1 ProsRotation()	73
5.10.3 Member Function Documentation	73
5.10.3.1 initialize()	73
5.10.3.2 reset()	73
5.10.3.3 getRotation()	74
5.10.3.4 setRotation()	74
5.10.3.5 getAngle()	74
5.10.4 Member Data Documentation	75
5.10.4.1 UNIT_CONVERSION	75
5.10.4.2 m_sensor	75
5.11 pros_adapters::ProsTask Class Reference	75
5.11.1 Detailed Description	76
5.11.2 Member Function Documentation	76
5.11.2.1 start()	76
5.11.2.2 remove()	76
5.11.2.3 suspend()	77
5.11.2.4 resume()	77
5.11.2.5 join()	77
5.11.3 Member Data Documentation	77
5.11.3.1 task	77
5.12 pros_adapters::ProsV5Motor Class Reference	78
5.12.1 Detailed Description	79
5.12.2 Constructor & Destructor Documentation	79
5.12.2.1 ProsV5Motor()	79
5.12.3 Member Function Documentation	80
5.12.3.1 initialize()	80
5.12.3.2 getTorqueConstant()	80
5.12.3.3 getResistance()	80

---

5.12.3.4 getAngularVelocityConstant()	81
5.12.3.5 getGearRatio()	81
5.12.3.6 getAngularVelocity()	81
5.12.3.7 getPosition()	82
5.12.3.8 setVoltage()	82
5.12.4 Member Data Documentation	82
5.12.4.1 cartridge_map	82
5.12.4.2 NO_CARTRIDGE	83
5.12.4.3 TORQUE_CONSTANT	83
5.12.4.4 RESISTANCE	83
5.12.4.5 ANGULAR_VELOCITY_CONSTANT	83
5.12.4.6 VELOCITY_CONVERSION	84
5.12.4.7 POSITION_CONVERSION	84
5.12.4.8 VOLTAGE_CONVERSION	84
5.12.4.9 MAX_MILLIVOLTS	84
5.12.4.10 m_motor	84
5.13 wisco::alliances::BlueAlliance Class Reference	85
5.13.1 Detailed Description	85
5.13.2 Member Function Documentation	85
5.13.2.1 getName()	85
5.13.3 Member Data Documentation	86
5.13.3.1 ALLIANCE_NAME	86
5.14 wisco::alliances::RedAlliance Class Reference	86
5.14.1 Detailed Description	86
5.14.2 Member Function Documentation	87
5.14.2.1 getName()	87
5.14.3 Member Data Documentation	87
5.14.3.1 ALLIANCE_NAME	87
5.15 wisco::alliances::SkillsAlliance Class Reference	87
5.15.1 Detailed Description	88
5.15.2 Member Function Documentation	88
5.15.2.1 getName()	88
5.15.3 Member Data Documentation	88
5.15.3.1 ALLIANCE_NAME	88
5.16 wisco::AutonomousManager Class Reference	89
5.16.1 Detailed Description	89
5.16.2 Constructor & Destructor Documentation	89
5.16.2.1 AutonomousManager()	89
5.16.3 Member Function Documentation	90
5.16.3.1 setAutonomous()	90
5.16.3.2 initializeAutonomous()	90
5.16.3.3 runAutonomous()	90

5.16.4 Member Data Documentation . . . . .	91
5.16.4.1 m_autonomous . . . . .	91
5.16.4.2 m_clock . . . . .	91
5.16.4.3 m_delayer . . . . .	91
5.17 wisco::autons::BlueMatchAuton Class Reference . . . . .	91
5.17.1 Detailed Description . . . . .	93
5.17.2 Member Function Documentation . . . . .	93
5.17.2.1 boomerangGoToPoint() . . . . .	93
5.17.2.2 boomerangPause() . . . . .	94
5.17.2.3 boomerangResume() . . . . .	94
5.17.2.4 boomerangTargetReached() . . . . .	94
5.17.2.5 odometrySetPosition() . . . . .	95
5.17.2.6 odometryGetPosition() . . . . .	95
5.17.2.7 motionTurnToAngle() . . . . .	96
5.17.2.8 motionTurnToPoint() . . . . .	96
5.17.2.9 motionPauseTurn() . . . . .	97
5.17.2.10 motionResumeTurn() . . . . .	97
5.17.2.11 motionTurnTargetReached() . . . . .	98
5.17.2.12 getName() . . . . .	98
5.17.2.13 initialize() . . . . .	98
5.17.2.14 run() . . . . .	99
5.17.3 Member Data Documentation . . . . .	99
5.17.3.1 AUTONOMOUS_NAME . . . . .	99
5.17.3.2 BOOMERANG_CONTROL_NAME . . . . .	100
5.17.3.3 ODOMETRY_SUBSYSTEM_NAME . . . . .	100
5.17.3.4 BOOMERANG_GO_TO_POSITION_COMMAND_NAME . . . . .	100
5.17.3.5 BOOMERANG_PAUSE_COMMAND_NAME . . . . .	100
5.17.3.6 BOOMERANG_RESUME_COMMAND_NAME . . . . .	100
5.17.3.7 ODOMETRY_SET_POSITION_COMMAND_NAME . . . . .	101
5.17.3.8 BOOMERANG_TARGET_REACHED_STATE_NAME . . . . .	101
5.17.3.9 ODOMETRY_GET_POSITION_STATE_NAME . . . . .	101
5.17.3.10 MOTION_CONTROL_NAME . . . . .	101
5.17.3.11 MOTION_TURN_TO_ANGLE_COMMAND_NAME . . . . .	101
5.17.3.12 MOTION_TURN_TO_POINT_COMMAND_NAME . . . . .	102
5.17.3.13 MOTION_PAUSE_TURN_COMMAND_NAME . . . . .	102
5.17.3.14 MOTION_RESUME_TURN_COMMAND_NAME . . . . .	102
5.17.3.15 MOTION_TURN_TARGET_REACHED_STATE_NAME . . . . .	102
5.18 wisco::autons::BlueSkillsAuton Class Reference . . . . .	102
5.18.1 Detailed Description . . . . .	103
5.18.2 Member Function Documentation . . . . .	103
5.18.2.1 getName() . . . . .	103
5.18.2.2 initialize() . . . . .	103

---

5.18.2.3 run()	104
5.18.3 Member Data Documentation	104
5.18.3.1 AUTONOMOUS_NAME	104
5.19 wisco::autons::OrangeMatchAuton Class Reference	105
5.19.1 Detailed Description	105
5.19.2 Member Function Documentation	106
5.19.2.1 getName()	106
5.19.2.2 initialize()	106
5.19.2.3 run()	106
5.19.3 Member Data Documentation	107
5.19.3.1 AUTONOMOUS_NAME	107
5.20 wisco::autons::OrangeSkillsAuton Class Reference	107
5.20.1 Detailed Description	108
5.20.2 Member Function Documentation	108
5.20.2.1 getName()	108
5.20.2.2 initialize()	108
5.20.2.3 run()	109
5.20.3 Member Data Documentation	109
5.20.3.1 AUTONOMOUS_NAME	109
5.21 wisco::configs::BlueConfiguration Class Reference	109
5.21.1 Detailed Description	114
5.21.2 Member Function Documentation	114
5.21.2.1 getName()	114
5.21.2.2 buildControlSystem()	115
5.21.2.3 buildController()	115
5.21.2.4 buildRobot()	116
5.21.3 Member Data Documentation	121
5.21.3.1 CONFIGURATION_NAME	121
5.21.3.2 ODOMETRY_HEADING_PORT	121
5.21.3.3 ODOMETRY_HEADING_TUNING_CONSTANT	122
5.21.3.4 ODOMETRY_LINEAR_PORT	122
5.21.3.5 ODOMETRY_LINEAR_RADIUS	122
5.21.3.6 ODOMETRY_LINEAR_OFFSET	122
5.21.3.7 ODOMETRY_STRAFE_PORT	122
5.21.3.8 ODOMETRY_STRAFE_RADIUS	123
5.21.3.9 ODOMETRY_STRAFE_OFFSET	123
5.21.3.10 RESETTER_DISTANCE_PORT	123
5.21.3.11 RESETTER_DISTANCE_CONSTANT	123
5.21.3.12 RESETTER_DISTANCE_OFFSET	123
5.21.3.13 RESETTER_OFFSET_X	124
5.21.3.14 RESETTER_OFFSET_Y	124
5.21.3.15 RESETTER_OFFSET_THETA	124

---

5.21.3.16 DRIVE_KINEMATIC . . . . .	124
5.21.3.17 DRIVE_VELOCITY_PROFILE_JERK_RATE . . . . .	124
5.21.3.18 DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION . . . . .	125
5.21.3.19 DRIVE_LEFT_MOTOR_1_PORT . . . . .	125
5.21.3.20 DRIVE_LEFT_MOTOR_1_GEARSET . . . . .	125
5.21.3.21 DRIVE_LEFT_MOTOR_2_PORT . . . . .	125
5.21.3.22 DRIVE_LEFT_MOTOR_2_GEARSET . . . . .	125
5.21.3.23 DRIVE_LEFT_MOTOR_3_PORT . . . . .	126
5.21.3.24 DRIVE_LEFT_MOTOR_3_GEARSET . . . . .	126
5.21.3.25 DRIVE_LEFT_MOTOR_4_PORT . . . . .	126
5.21.3.26 DRIVE_LEFT_MOTOR_4_GEARSET . . . . .	126
5.21.3.27 DRIVE_RIGHT_MOTOR_1_PORT . . . . .	126
5.21.3.28 DRIVE_RIGHT_MOTOR_1_GEARSET . . . . .	127
5.21.3.29 DRIVE_RIGHT_MOTOR_2_PORT . . . . .	127
5.21.3.30 DRIVE_RIGHT_MOTOR_2_GEARSET . . . . .	127
5.21.3.31 DRIVE_RIGHT_MOTOR_3_PORT . . . . .	127
5.21.3.32 DRIVE_RIGHT_MOTOR_3_GEARSET . . . . .	127
5.21.3.33 DRIVE_RIGHT_MOTOR_4_PORT . . . . .	128
5.21.3.34 DRIVE_RIGHT_MOTOR_4_GEARSET . . . . .	128
5.21.3.35 DRIVE_VELOCITY_TO_VOLTAGE . . . . .	128
5.21.3.36 DRIVE_MASS . . . . .	128
5.21.3.37 DRIVE_RADIUS . . . . .	128
5.21.3.38 DRIVE_MOMENT_OF_INERTIA . . . . .	129
5.21.3.39 DRIVE_GEAR_RATIO . . . . .	129
5.21.3.40 DRIVE_WHEEL_RADIUS . . . . .	129
5.21.3.41 INTAKE_KP . . . . .	129
5.21.3.42 INTAKE_KI . . . . .	129
5.21.3.43 INTAKE_KD . . . . .	130
5.21.3.44 INTAKE_MOTOR_1_PORT . . . . .	130
5.21.3.45 INTAKE_MOTOR_1_GEARSET . . . . .	130
5.21.3.46 INTAKE_MOTOR_2_PORT . . . . .	130
5.21.3.47 INTAKE_MOTOR_2_GEARSET . . . . .	130
5.21.3.48 INTAKE_ROLLER_RADIUS . . . . .	131
5.21.3.49 BALL_DETECTOR_DISTANCE_PORT . . . . .	131
5.21.3.50 BALL_DETECTOR_DISTANCE_CONSTANT . . . . .	131
5.21.3.51 BALL_DETECTOR_DISTANCE_OFFSET . . . . .	131
5.21.3.52 BOOMERANG_LINEAR_KP . . . . .	131
5.21.3.53 BOOMERANG_LINEAR_KI . . . . .	132
5.21.3.54 BOOMERANG_LINEAR_KD . . . . .	132
5.21.3.55 BOOMERANG_ROTATIONAL_KP . . . . .	132
5.21.3.56 BOOMERANG_ROTATIONAL_KI . . . . .	132
5.21.3.57 BOOMERANG_ROTATIONAL_KD . . . . .	132

---

5.21.3.58 BOOMERANG_LEAD . . . . .	133
5.21.3.59 BOOMERANG_TARGET_TOLERANCE . . . . .	133
5.21.3.60 ELEVATOR_KP . . . . .	133
5.21.3.61 ELEVATOR_KI . . . . .	133
5.21.3.62 ELEVATOR_KD . . . . .	133
5.21.3.63 ELEVATOR_MOTOR_1_PORT . . . . .	134
5.21.3.64 ELEVATOR_MOTOR_1_GEARSET . . . . .	134
5.21.3.65 ELEVATOR_MOTOR_2_PORT . . . . .	134
5.21.3.66 ELEVATOR_MOTOR_2_GEARSET . . . . .	134
5.21.3.67 ELEVATOR_ROTATION_SENSOR_PORT . . . . .	134
5.21.3.68 ELEVATOR_INCHES_PER_RADIAN . . . . .	135
5.21.3.69 ELEVATOR_DISTANCE_PORT . . . . .	135
5.21.3.70 ELEVATOR_DISTANCE_CONSTANT . . . . .	135
5.21.3.71 ELEVATOR_DISTANCE_OFFSET . . . . .	135
5.21.3.72 HANG_CLAW_PISTON_1_PORT . . . . .	135
5.21.3.73 HANG_CLAW_PISTON_1_EXTENDED_STATE . . . . .	136
5.21.3.74 HANG_CLAW_CLOSED_STATE . . . . .	136
5.21.3.75 HANG_ARM_PISTON_1_PORT . . . . .	136
5.21.3.76 HANG_ARM_PISTON_1_EXTENDED_STATE . . . . .	136
5.21.3.77 HANG_ARM_UP_STATE . . . . .	136
5.21.3.78 HANG_WINCH_PISTON_1_PORT . . . . .	137
5.21.3.79 HANG_WINCH_PISTON_1_EXTENDED_STATE . . . . .	137
5.21.3.80 HANG_WINCH_ENGAGED_STATE . . . . .	137
5.21.3.81 LOADER_KP . . . . .	137
5.21.3.82 LOADER_KI . . . . .	137
5.21.3.83 LOADER_KD . . . . .	138
5.21.3.84 LOADER_MOTOR_1_PORT . . . . .	138
5.21.3.85 LOADER_MOTOR_1_GEARSET . . . . .	138
5.21.3.86 LOADER_LOADED_POSITION . . . . .	138
5.21.3.87 LOADER_READY_POSITION . . . . .	138
5.21.3.88 LOADER_POSITION_TOLERANCE . . . . .	139
5.21.3.89 TURN_KP . . . . .	139
5.21.3.90 TURN_KI . . . . .	139
5.21.3.91 TURN_KD . . . . .	139
5.21.3.92 TURN_TARGET_TOLERANCE . . . . .	139
5.21.3.93 TURN_TARGET_VELOCITY . . . . .	140
5.21.3.94 UMBRELLA_PISTON_1_PORT . . . . .	140
5.21.3.95 UMBRELLA_PISTON_1_EXTENDED_STATE . . . . .	140
5.21.3.96 UMBRELLA_OUT_STATE . . . . .	140
5.21.3.97 LEFT_WING_PISTON_1_PORT . . . . .	140
5.21.3.98 LEFT_WING_PISTON_1_EXTENDED_STATE . . . . .	141
5.21.3.99 RIGHT_WING_PISTON_1_PORT . . . . .	141

5.21.3.100 RIGHT_WING_PISTON_1_EXTENDED_STATE . . . . .	141
5.21.3.101 WINGS_OUT_STATE . . . . .	141
5.22 wisco::configs::OrangeConfiguration Class Reference . . . . .	141
5.22.1 Detailed Description . . . . .	142
5.22.2 Member Function Documentation . . . . .	142
5.22.2.1 getName() . . . . .	142
5.22.2.2 buildControlSystem() . . . . .	143
5.22.2.3 buildController() . . . . .	143
5.22.2.4 buildRobot() . . . . .	143
5.22.3 Member Data Documentation . . . . .	144
5.22.3.1 CONFIGURATION_NAME . . . . .	144
5.23 wisco::control::AControl Class Reference . . . . .	144
5.23.1 Detailed Description . . . . .	145
5.23.2 Constructor & Destructor Documentation . . . . .	145
5.23.2.1 AControl() [1/3] . . . . .	145
5.23.2.2 AControl() [2/3] . . . . .	145
5.23.2.3 AControl() [3/3] . . . . .	145
5.23.3 Member Function Documentation . . . . .	146
5.23.3.1 getName() . . . . .	146
5.23.3.2 initialize() . . . . .	146
5.23.3.3 run() . . . . .	146
5.23.3.4 command() . . . . .	146
5.23.3.5 state() . . . . .	147
5.23.3.6 operator=() [1/2] . . . . .	147
5.23.3.7 operator=() [2/2] . . . . .	147
5.23.4 Member Data Documentation . . . . .	148
5.23.4.1 m_name . . . . .	148
5.24 wisco::control::boomerang::BoomerangControl Class Reference . . . . .	148
5.24.1 Detailed Description . . . . .	149
5.24.2 Constructor & Destructor Documentation . . . . .	149
5.24.2.1 BoomerangControl() . . . . .	149
5.24.3 Member Function Documentation . . . . .	150
5.24.3.1 initialize() . . . . .	150
5.24.3.2 run() . . . . .	150
5.24.3.3 command() . . . . .	150
5.24.3.4 state() . . . . .	151
5.24.4 Member Data Documentation . . . . .	151
5.24.4.1 CONTROL_NAME . . . . .	151
5.24.4.2 GO_TO_POSITION_COMMAND_NAME . . . . .	152
5.24.4.3 PAUSE_COMMAND_NAME . . . . .	152
5.24.4.4 RESUME_COMMAND_NAME . . . . .	152
5.24.4.5 TARGET_REACHED_STATE_NAME . . . . .	152

---

5.24.4.6 m_boomerang . . . . .	152
5.25 wisco::control::boomerang::IBoomerang Class Reference . . . . .	153
5.25.1 Detailed Description . . . . .	153
5.25.2 Member Function Documentation . . . . .	153
5.25.2.1 initialize() . . . . .	153
5.25.2.2 run() . . . . .	154
5.25.2.3 goToPosition() . . . . .	154
5.25.2.4 pause() . . . . .	154
5.25.2.5 resume() . . . . .	154
5.25.2.6 targetReached() . . . . .	155
5.26 wisco::control::boomerang::PIDBoomerang Class Reference . . . . .	155
5.26.1 Detailed Description . . . . .	157
5.26.2 Member Function Documentation . . . . .	157
5.26.2.1 taskLoop() . . . . .	157
5.26.2.2 taskUpdate() . . . . .	157
5.26.2.3 setDriveVelocity() . . . . .	158
5.26.2.4 getOdometryPosition() . . . . .	158
5.26.2.5 calculateDistanceToTarget() . . . . .	159
5.26.2.6 calculateCarrotPoint() . . . . .	159
5.26.2.7 updateVelocity() . . . . .	160
5.26.2.8 initialize() . . . . .	160
5.26.2.9 run() . . . . .	160
5.26.2.10 goToPosition() . . . . .	161
5.26.2.11 pause() . . . . .	161
5.26.2.12 resume() . . . . .	162
5.26.2.13 targetReached() . . . . .	162
5.26.2.14 setDelay() . . . . .	162
5.26.2.15 setMutex() . . . . .	162
5.26.2.16 setTask() . . . . .	163
5.26.2.17 setLinearPID() . . . . .	163
5.26.2.18 setRotationalPID() . . . . .	163
5.26.2.19 setLead() . . . . .	164
5.26.2.20 setTargetTolerance() . . . . .	164
5.26.3 Member Data Documentation . . . . .	164
5.26.3.1 TASK_DELAY . . . . .	164
5.26.3.2 DRIVE_SUBSYSTEM_NAME . . . . .	165
5.26.3.3 ODOMETRY_SUBSYSTEM_NAME . . . . .	165
5.26.3.4 DRIVE_SET_VELOCITY_COMMAND_NAME . . . . .	165
5.26.3.5 ODOMETRY_GET_POSITION_STATE_NAME . . . . .	165
5.26.3.6 m_delayer . . . . .	165
5.26.3.7 m_mutex . . . . .	166
5.26.3.8 m_task . . . . .	166

5.26.3.9 m_linear_pid . . . . .	166
5.26.3.10 m_rotational_pid . . . . .	166
5.26.3.11 m_lead . . . . .	166
5.26.3.12 m_target_tolerance . . . . .	166
5.26.3.13 control_robot . . . . .	167
5.26.3.14 motion_velocity . . . . .	167
5.26.3.15 target_x . . . . .	167
5.26.3.16 target_y . . . . .	167
5.26.3.17 target_theta . . . . .	167
5.26.3.18 paused . . . . .	167
5.26.3.19 target_reached . . . . .	168
5.27 wisco::control::boomerang::PIDBoomerangBuilder Class Reference . . . . .	168
5.27.1 Detailed Description . . . . .	169
5.27.2 Member Function Documentation . . . . .	169
5.27.2.1 withDelaye() . . . . .	169
5.27.2.2 withMutex() . . . . .	169
5.27.2.3 withTask() . . . . .	170
5.27.2.4 withLinearPID() . . . . .	170
5.27.2.5 withRotationalPID() . . . . .	170
5.27.2.6 withLead() . . . . .	171
5.27.2.7 withTargetTolerance() . . . . .	171
5.27.2.8 build() . . . . .	172
5.27.3 Member Data Documentation . . . . .	172
5.27.3.1 m_delaye . . . . .	172
5.27.3.2 m_mutex . . . . .	172
5.27.3.3 m_task . . . . .	172
5.27.3.4 m_linear_pid . . . . .	173
5.27.3.5 m_rotational_pid . . . . .	173
5.27.3.6 m_lead . . . . .	173
5.27.3.7 m_target_tolerance . . . . .	173
5.28 wisco::control::ControlSystem Class Reference . . . . .	173
5.28.1 Detailed Description . . . . .	174
5.28.2 Member Function Documentation . . . . .	174
5.28.2.1 addControl() . . . . .	174
5.28.2.2 removeControl() . . . . .	174
5.28.2.3 initialize() . . . . .	175
5.28.2.4 sendCommand() . . . . .	175
5.28.2.5 getState() . . . . .	175
5.28.3 Member Data Documentation . . . . .	176
5.28.3.1 controls . . . . .	176
5.29 wisco::control::motion::ITurn Class Reference . . . . .	176
5.29.1 Detailed Description . . . . .	177

---

5.29.2 Member Function Documentation . . . . .	177
5.29.2.1 initialize() . . . . .	177
5.29.2.2 run() . . . . .	177
5.29.2.3 turnToAngle() . . . . .	177
5.29.2.4 turnToPoint() . . . . .	178
5.29.2.5 pause() . . . . .	178
5.29.2.6 resume() . . . . .	178
5.29.2.7 targetReached() . . . . .	179
5.30 wisco::control::motion::MotionControl Class Reference . . . . .	179
5.30.1 Detailed Description . . . . .	180
5.30.2 Constructor & Destructor Documentation . . . . .	180
5.30.2.1 MotionControl() . . . . .	180
5.30.3 Member Function Documentation . . . . .	181
5.30.3.1 initialize() . . . . .	181
5.30.3.2 run() . . . . .	181
5.30.3.3 command() . . . . .	181
5.30.3.4 state() . . . . .	182
5.30.4 Member Data Documentation . . . . .	182
5.30.4.1 CONTROL_NAME . . . . .	182
5.30.4.2 TURN_TO_ANGLE_COMMAND_NAME . . . . .	183
5.30.4.3 TURN_TO_POINT_COMMAND_NAME . . . . .	183
5.30.4.4 PAUSE_TURN_COMMAND_NAME . . . . .	183
5.30.4.5 RESUME_TURN_COMMAND_NAME . . . . .	183
5.30.4.6 TURN_TARGET_REACHED_STATE_NAME . . . . .	183
5.30.4.7 m_turn . . . . .	184
5.31 wisco::control::motion::PIDTurn Class Reference . . . . .	184
5.31.1 Detailed Description . . . . .	186
5.31.2 Member Function Documentation . . . . .	186
5.31.2.1 taskLoop() . . . . .	186
5.31.2.2 taskUpdate() . . . . .	187
5.31.2.3 setDriveVelocity() . . . . .	187
5.31.2.4 getDriveRadius() . . . . .	187
5.31.2.5 getOdometryPosition() . . . . .	188
5.31.2.6 calculateAngleToTarget() . . . . .	188
5.31.2.7 calculateDriveVelocity() . . . . .	189
5.31.2.8 initialize() . . . . .	189
5.31.2.9 run() . . . . .	190
5.31.2.10 turnToAngle() . . . . .	190
5.31.2.11 turnToPoint() . . . . .	190
5.31.2.12 pause() . . . . .	191
5.31.2.13 resume() . . . . .	192
5.31.2.14 targetReached() . . . . .	192

---

5.31.2.15 setDelay()	192
5.31.2.16 setMutex()	192
5.31.2.17 setTask()	193
5.31.2.18 setPID()	193
5.31.2.19 setTargetTolerance()	193
5.31.2.20 setTargetVelocity()	194
5.31.3 Member Data Documentation	194
5.31.3.1 TASK_DELAY	194
5.31.3.2 DRIVE_SUBSYSTEM_NAME	194
5.31.3.3 ODOMETRY_SUBSYSTEM_NAME	195
5.31.3.4 DRIVE_SET_VELOCITY_COMMAND_NAME	195
5.31.3.5 DRIVE_GET_RADIUS_STATE_NAME	195
5.31.3.6 ODOMETRY_GET_POSITION_STATE_NAME	195
5.31.3.7 TURN_TO_ANGLE_DISTANCE	195
5.31.3.8 m_delayer	196
5.31.3.9 m_mutex	196
5.31.3.10 m_task	196
5.31.3.11 m_pid	196
5.31.3.12 m_target_tolerance	196
5.31.3.13 m_target_velocity	196
5.31.3.14 control_robot	197
5.31.3.15 turn_direction	197
5.31.3.16 turn_velocity	197
5.31.3.17 target_x	197
5.31.3.18 target_y	197
5.31.3.19 paused	197
5.31.3.20 target_reached	198
5.31.3.21 forced_direction_reached	198
5.32 wisco::control::motion::PITurnBuilder Class Reference	198
5.32.1 Detailed Description	199
5.32.2 Member Function Documentation	199
5.32.2.1 withDelay()	199
5.32.2.2 withMutex()	199
5.32.2.3 withTask()	200
5.32.2.4 withPID()	200
5.32.2.5 withTargetTolerance()	200
5.32.2.6 withTargetVelocity()	201
5.32.2.7 build()	201
5.32.3 Member Data Documentation	202
5.32.3.1 m_delayer	202
5.32.3.2 m_mutex	202
5.32.3.3 m_task	202

---

5.32.3.4 m_pid . . . . .	202
5.32.3.5 m_target_tolerance . . . . .	202
5.32.3.6 m_target_velocity . . . . .	203
5.33 wisco::control::path::Point Class Reference . . . . .	203
5.33.1 Detailed Description . . . . .	204
5.33.2 Constructor & Destructor Documentation . . . . .	204
5.33.2.1 Point() [1/3] . . . . .	204
5.33.2.2 Point() [2/3] . . . . .	204
5.33.2.3 Point() [3/3] . . . . .	204
5.33.3 Member Function Documentation . . . . .	205
5.33.3.1 setX() . . . . .	205
5.33.3.2 setY() . . . . .	205
5.33.3.3 getX() . . . . .	205
5.33.3.4 getY() . . . . .	206
5.33.3.5 operator+() . . . . .	206
5.33.3.6 operator-() . . . . .	206
5.33.3.7 operator*() . . . . .	207
5.33.3.8 operator/() . . . . .	207
5.33.3.9 operator+=() . . . . .	208
5.33.3.10 operator-=() . . . . .	208
5.33.3.11 operator*=(()) . . . . .	208
5.33.3.12 operator/=(()) . . . . .	209
5.33.3.13 operator() [1/2] . . . . .	209
5.33.3.14 operator() [2/2] . . . . .	210
5.33.4 Member Data Documentation . . . . .	210
5.33.4.1 m_x . . . . .	210
5.33.4.2 m_y . . . . .	210
5.34 wisco::control::path::QuinticBezier Class Reference . . . . .	210
5.34.1 Detailed Description . . . . .	211
5.34.2 Constructor & Destructor Documentation . . . . .	211
5.34.2.1 QuinticBezier() [1/3] . . . . .	211
5.34.2.2 QuinticBezier() [2/3] . . . . .	211
5.34.2.3 QuinticBezier() [3/3] . . . . .	212
5.34.3 Member Function Documentation . . . . .	212
5.34.3.1 calculatePoint() . . . . .	212
5.34.3.2 operator=() [1/2] . . . . .	213
5.34.3.3 operator=() [2/2] . . . . .	213
5.34.4 Member Data Documentation . . . . .	213
5.34.4.1 k0 . . . . .	213
5.34.4.2 c0 . . . . .	213
5.34.4.3 c1 . . . . .	214
5.34.4.4 c2 . . . . .	214

---

5.34.4.5 c3 . . . . .	214
5.34.4.6 k1 . . . . .	214
5.35 wisco::control::path::QuinticBezierSpline Class Reference . . . . .	214
5.35.1 Detailed Description . . . . .	215
5.35.2 Member Function Documentation . . . . .	215
5.35.2.1 calculate() . . . . .	215
5.36 wisco::control::PID Class Reference . . . . .	216
5.36.1 Detailed Description . . . . .	216
5.36.2 Constructor & Destructor Documentation . . . . .	217
5.36.2.1 PID() [1/3] . . . . .	217
5.36.2.2 PID() [2/3] . . . . .	217
5.36.2.3 PID() [3/3] . . . . .	217
5.36.3 Member Function Documentation . . . . .	218
5.36.3.1 getControlValue() . . . . .	218
5.36.3.2 reset() . . . . .	218
5.36.3.3 operator=() [1/2] . . . . .	218
5.36.3.4 operator=() [2/2] . . . . .	219
5.36.4 Member Data Documentation . . . . .	219
5.36.4.1 m_clock . . . . .	219
5.36.4.2 m_kp . . . . .	219
5.36.4.3 m_ki . . . . .	220
5.36.4.4 m_kd . . . . .	220
5.36.4.5 accumulated_error . . . . .	220
5.36.4.6 last_error . . . . .	220
5.36.4.7 last_time . . . . .	220
5.37 wisco::hal::DistanceBooleanSensor Class Reference . . . . .	221
5.37.1 Detailed Description . . . . .	222
5.37.2 Constructor & Destructor Documentation . . . . .	222
5.37.2.1 DistanceBooleanSensor() [1/2] . . . . .	222
5.37.2.2 DistanceBooleanSensor() [2/2] . . . . .	222
5.37.3 Member Function Documentation . . . . .	223
5.37.3.1 initialize() . . . . .	223
5.37.3.2 reset() . . . . .	223
5.37.3.3 getValue() . . . . .	223
5.37.4 Member Data Documentation . . . . .	224
5.37.4.1 m_distance_sensor . . . . .	224
5.37.4.2 m_mode . . . . .	224
5.37.4.3 m_lower_threshold . . . . .	224
5.37.4.4 m_upper_threshold . . . . .	225
5.37.4.5 value . . . . .	225
5.38 wisco::hal::MotorGroup Class Reference . . . . .	225
5.38.1 Detailed Description . . . . .	226

---

5.38.2 Member Function Documentation . . . . .	226
5.38.2.1 addMotor() . . . . .	226
5.38.2.2 initialize() . . . . .	226
5.38.2.3 getTorqueConstant() . . . . .	226
5.38.2.4 getResistance() . . . . .	227
5.38.2.5 getAngularVelocityConstant() . . . . .	227
5.38.2.6 getGearRatio() . . . . .	227
5.38.2.7 getAngularVelocity() . . . . .	228
5.38.2.8 getPosition() . . . . .	228
5.38.2.9 setVoltage() . . . . .	228
5.38.2.10 operator=(()) . . . . .	229
5.38.3 Member Data Documentation . . . . .	229
5.38.3.1 motors . . . . .	229
5.39 wisco::hal::PistonGroup Class Reference . . . . .	229
5.39.1 Detailed Description . . . . .	230
5.39.2 Member Function Documentation . . . . .	230
5.39.2.1 addPiston() . . . . .	230
5.39.2.2 setState() . . . . .	230
5.39.2.3 getState() . . . . .	231
5.39.2.4 operator=(()) . . . . .	231
5.39.3 Member Data Documentation . . . . .	231
5.39.3.1 pistons . . . . .	231
5.39.3.2 m_state . . . . .	232
5.40 wisco::hal::TrackingWheel Class Reference . . . . .	232
5.40.1 Detailed Description . . . . .	233
5.40.2 Constructor & Destructor Documentation . . . . .	233
5.40.2.1 TrackingWheel() . . . . .	233
5.40.3 Member Function Documentation . . . . .	233
5.40.3.1 initialize() . . . . .	233
5.40.3.2 reset() . . . . .	234
5.40.3.3 getDistance() . . . . .	234
5.40.3.4 setDistance() . . . . .	234
5.40.4 Member Data Documentation . . . . .	234
5.40.4.1 m_sensor . . . . .	234
5.40.4.2 m_wheel_radius . . . . .	235
5.41 wisco::IAlliance Class Reference . . . . .	235
5.41.1 Detailed Description . . . . .	235
5.41.2 Member Function Documentation . . . . .	236
5.41.2.1 getName() . . . . .	236
5.42 wisco::IAutonomous Class Reference . . . . .	236
5.42.1 Detailed Description . . . . .	236
5.42.2 Member Function Documentation . . . . .	237

---

5.42.2.1 getName() . . . . .	237
5.42.2.2 initialize() . . . . .	237
5.42.2.3 run() . . . . .	237
5.43 wisco::IConfiguration Class Reference . . . . .	238
5.43.1 Detailed Description . . . . .	238
5.43.2 Member Function Documentation . . . . .	238
5.43.2.1 getName() . . . . .	238
5.43.2.2 buildControlSystem() . . . . .	239
5.43.2.3 buildController() . . . . .	239
5.43.2.4 buildRobot() . . . . .	239
5.44 wisco::IMenu Class Reference . . . . .	239
5.44.1 Detailed Description . . . . .	240
5.44.2 Member Function Documentation . . . . .	240
5.44.2.1 addAlliance() . . . . .	240
5.44.2.2 addAutonomous() . . . . .	240
5.44.2.3 addConfiguration() . . . . .	241
5.44.2.4 addProfile() . . . . .	241
5.44.2.5 display() . . . . .	241
5.44.2.6 isStarted() . . . . .	241
5.44.2.7 getSystemConfiguration() . . . . .	242
5.45 wisco::io::IBooleanSensor Class Reference . . . . .	242
5.45.1 Detailed Description . . . . .	242
5.45.2 Member Function Documentation . . . . .	243
5.45.2.1 initialize() . . . . .	243
5.45.2.2 reset() . . . . .	243
5.45.2.3 getValue() . . . . .	243
5.46 wisco::io::IDistanceSensor Class Reference . . . . .	243
5.46.1 Detailed Description . . . . .	244
5.46.2 Member Function Documentation . . . . .	244
5.46.2.1 initialize() . . . . .	244
5.46.2.2 reset() . . . . .	244
5.46.2.3 getDistance() . . . . .	244
5.47 wisco::io::IDistanceTrackingSensor Class Reference . . . . .	244
5.47.1 Detailed Description . . . . .	245
5.47.2 Member Function Documentation . . . . .	245
5.47.2.1 initialize() . . . . .	245
5.47.2.2 reset() . . . . .	245
5.47.2.3 getDistance() . . . . .	245
5.47.2.4 setDistance() . . . . .	245
5.48 wisco::io::IHeadingSensor Class Reference . . . . .	246
5.48.1 Detailed Description . . . . .	246
5.48.2 Member Function Documentation . . . . .	247

---

5.48.2.1 initialize()	247
5.48.2.2 reset()	247
5.48.2.3 getHeading()	247
5.48.2.4 setHeading()	247
5.48.2.5 getRotation()	247
5.48.2.6 setRotation()	248
5.49 wisco::io::IMotor Class Reference	248
5.49.1 Detailed Description	249
5.49.2 Member Function Documentation	249
5.49.2.1 initialize()	249
5.49.2.2 getTorqueConstant()	249
5.49.2.3 getResistance()	249
5.49.2.4 getAngularVelocityConstant()	250
5.49.2.5 getGearRatio()	250
5.49.2.6 getAngularVelocity()	250
5.49.2.7 getPosition()	250
5.49.2.8 setVoltage()	250
5.50 wisco::io::IPiston Class Reference	251
5.50.1 Detailed Description	251
5.50.2 Member Function Documentation	252
5.50.2.1 extend()	252
5.50.2.2 retract()	252
5.50.2.3 toggle()	252
5.50.2.4 isExtended()	252
5.50.2.5 isRetracted()	252
5.51 wisco::io::IRotationSensor Class Reference	253
5.51.1 Detailed Description	253
5.51.2 Member Function Documentation	253
5.51.2.1 initialize()	253
5.51.2.2 reset()	254
5.51.2.3 getRotation()	254
5.51.2.4 setRotation()	254
5.51.2.5 getAngle()	254
5.52 wisco::IProfile Class Reference	255
5.52.1 Detailed Description	255
5.52.2 Member Function Documentation	255
5.52.2.1 getName()	255
5.52.2.2 getControlMode()	255
5.52.2.3 setControlMode()	256
5.52.2.4 getAnalogControlMapping()	256
5.52.2.5 getDigitalControlMapping()	256
5.53 wisco::MatchController Class Reference	257

5.53.1 Detailed Description	258
5.53.2 Constructor & Destructor Documentation	258
5.53.2.1 MatchController()	258
5.53.3 Member Function Documentation	258
5.53.3.1 initialize()	258
5.53.3.2 disabled()	259
5.53.3.3 competitionInitialize()	259
5.53.3.4 autonomous()	259
5.53.3.5 operatorControl()	260
5.53.4 Member Data Documentation	260
5.53.4.1 MENU_DELAY	260
5.53.4.2 m_menu	260
5.53.4.3 m_clock	260
5.53.4.4 m_delayer	260
5.53.4.5 autonomous_manager	261
5.53.4.6 opcontrol_manager	261
5.53.4.7 control_system	261
5.53.4.8 controller	261
5.53.4.9 robot	261
5.54 wisco::menu::LvglMenu Class Reference	261
5.54.1 Detailed Description	263
5.54.2 Member Function Documentation	263
5.54.2.1 initializeStyles()	263
5.54.2.2 addOption()	264
5.54.2.3 removeOption()	264
5.54.2.4 drawMainMenu()	265
5.54.2.5 drawSettingsMenu()	266
5.54.2.6 setComplete()	267
5.54.2.7 readConfiguration()	267
5.54.2.8 writeConfiguration()	268
5.54.2.9 displayMenu()	268
5.54.2.10 selectionComplete()	268
5.54.2.11 getSelection()	269
5.54.3 Member Data Documentation	269
5.54.3.1 CONFIGURATION_FILE	269
5.54.3.2 COLUMN_WIDTH	269
5.54.3.3 BUTTONS_PER_LINE	269
5.54.3.4 button_default_style	270
5.54.3.5 button_pressed_style	270
5.54.3.6 container_default_style	270
5.54.3.7 container_pressed_style	270
5.54.3.8 button_matrix_main_style	270

---

5.54.3.9 button_matrix_items_style . . . . .	270
5.54.3.10 styles_initialized . . . . .	271
5.54.3.11 options . . . . .	271
5.54.3.12 complete . . . . .	271
5.55 wisco::menu::MenuAdapter Class Reference . . . . .	271
5.55.1 Detailed Description . . . . .	272
5.55.2 Member Function Documentation . . . . .	272
5.55.2.1 addAlliance() . . . . .	272
5.55.2.2 addAutonomous() . . . . .	273
5.55.2.3 addConfiguration() . . . . .	273
5.55.2.4 addProfile() . . . . .	274
5.55.2.5 display() . . . . .	274
5.55.2.6 isStarted() . . . . .	275
5.55.2.7 getSystemConfiguration() . . . . .	275
5.55.3 Member Data Documentation . . . . .	276
5.55.3.1 ALLIANCE_OPTION_NAME . . . . .	276
5.55.3.2 AUTONOMOUS_OPTION_NAME . . . . .	276
5.55.3.3 CONFIGURATION_OPTION_NAME . . . . .	276
5.55.3.4 PROFILE_OPTION_NAME . . . . .	276
5.55.3.5 alliances . . . . .	276
5.55.3.6 autonomous_routines . . . . .	277
5.55.3.7 hardware_configurations . . . . .	277
5.55.3.8 driver_profiles . . . . .	277
5.55.3.9 lvgl_menu . . . . .	277
5.56 wisco::menu::Option Struct Reference . . . . .	277
5.56.1 Detailed Description . . . . .	278
5.56.2 Member Data Documentation . . . . .	278
5.56.2.1 name . . . . .	278
5.56.2.2 choices . . . . .	278
5.56.2.3 selected . . . . .	278
5.57 wisco::OPControlManager Class Reference . . . . .	278
5.57.1 Detailed Description . . . . .	279
5.57.2 Constructor & Destructor Documentation . . . . .	279
5.57.2.1 OPControlManager() . . . . .	279
5.57.3 Member Function Documentation . . . . .	280
5.57.3.1 setProfile() . . . . .	280
5.57.3.2 initializeOpcontrol() . . . . .	280
5.57.3.3 runOpcontrol() . . . . .	280
5.57.4 Member Data Documentation . . . . .	281
5.57.4.1 CONTROL_DELAY . . . . .	281
5.57.4.2 m_clock . . . . .	281
5.57.4.3 m_delayer . . . . .	282

---

5.57.4.4 m_profile . . . . .	282
5.58 wisco::profiles::HenryProfile Class Reference . . . . .	282
5.58.1 Detailed Description . . . . .	283
5.58.2 Member Function Documentation . . . . .	283
5.58.2.1 getName() . . . . .	283
5.58.2.2 getControlMode() . . . . .	283
5.58.2.3 setControlMode() . . . . .	284
5.58.2.4 getAnalogControlMapping() . . . . .	284
5.58.2.5 getDigitalControlMapping() . . . . .	285
5.58.3 Member Data Documentation . . . . .	285
5.58.3.1 PROFILE_NAME . . . . .	285
5.58.3.2 CONTROL_MODE_MAP . . . . .	286
5.58.3.3 ANALOG_CONTROL_MAP . . . . .	286
5.58.3.4 DIGITAL_CONTROL_MAP . . . . .	286
5.59 wisco::profiles::JohnProfile Class Reference . . . . .	287
5.59.1 Detailed Description . . . . .	288
5.59.2 Member Function Documentation . . . . .	288
5.59.2.1 getName() . . . . .	288
5.59.2.2 getControlMode() . . . . .	288
5.59.2.3 setControlMode() . . . . .	289
5.59.2.4 getAnalogControlMapping() . . . . .	289
5.59.2.5 getDigitalControlMapping() . . . . .	289
5.59.3 Member Data Documentation . . . . .	290
5.59.3.1 PROFILE_NAME . . . . .	290
5.59.3.2 CONTROL_MODE_MAP . . . . .	290
5.59.3.3 ANALOG_CONTROL_MAP . . . . .	291
5.59.3.4 DIGITAL_CONTROL_MAP . . . . .	291
5.60 wisco::robot::ASubsystem Class Reference . . . . .	291
5.60.1 Detailed Description . . . . .	292
5.60.2 Constructor & Destructor Documentation . . . . .	292
5.60.2.1 ASubsystem() [1/3] . . . . .	292
5.60.2.2 ASubsystem() [2/3] . . . . .	293
5.60.2.3 ASubsystem() [3/3] . . . . .	293
5.60.3 Member Function Documentation . . . . .	293
5.60.3.1 getName() . . . . .	293
5.60.3.2 initialize() . . . . .	294
5.60.3.3 run() . . . . .	294
5.60.3.4 command() . . . . .	294
5.60.3.5 state() . . . . .	294
5.60.3.6 operator=() [1/2] . . . . .	295
5.60.3.7 operator=() [2/2] . . . . .	295
5.60.4 Member Data Documentation . . . . .	295

---

5.60.4.1 <code>m_name</code>	295
5.61 <code>wisco::robot::Robot</code> Class Reference	296
5.61.1 Detailed Description	296
5.61.2 Member Function Documentation	296
5.61.2.1 <code>addSubsystem()</code>	296
5.61.2.2 <code>removeSubsystem()</code>	297
5.61.2.3 <code>initialize()</code>	297
5.61.2.4 <code>sendCommand()</code>	297
5.61.2.5 <code>getState()</code>	298
5.61.3 Member Data Documentation	298
5.61.3.1 <code>subsystems</code>	298
5.62 <code>wisco::robot::subsystems::drive::CurveVelocityProfile</code> Class Reference	299
5.62.1 Detailed Description	299
5.62.2 Constructor & Destructor Documentation	300
5.62.2.1 <code>CurveVelocityProfile()</code>	300
5.62.3 Member Function Documentation	300
5.62.3.1 <code>getAcceleration()</code>	300
5.62.3.2 <code>setAcceleration()</code>	301
5.62.4 Member Data Documentation	301
5.62.4.1 <code>m_clock</code>	301
5.62.4.2 <code>m_jerk_rate</code>	301
5.62.4.3 <code>m_max_acceleration</code>	302
5.62.4.4 <code>m_current_acceleration</code>	302
5.62.4.5 <code>last_time</code>	302
5.63 <code>wisco::robot::subsystems::drive::DifferentialDriveSubsystem</code> Class Reference	302
5.63.1 Detailed Description	303
5.63.2 Constructor & Destructor Documentation	303
5.63.2.1 <code>DifferentialDriveSubsystem()</code>	303
5.63.3 Member Function Documentation	304
5.63.3.1 <code>initialize()</code>	304
5.63.3.2 <code>run()</code>	304
5.63.3.3 <code>command()</code>	304
5.63.3.4 <code>state()</code>	305
5.63.4 Member Data Documentation	305
5.63.4.1 <code>SUBSYSTEM_NAME</code>	305
5.63.4.2 <code>SET_VELOCITY_COMMAND_NAME</code>	306
5.63.4.3 <code>SET_VOLTAGE_COMMAND_NAME</code>	306
5.63.4.4 <code>GET_VELOCITY_STATE_NAME</code>	306
5.63.4.5 <code>GET_RADIUS_STATE_NAME</code>	306
5.63.4.6 <code>m_differential_drive</code>	306
5.64 <code>wisco::robot::subsystems::drive::DirectDifferentialDrive</code> Class Reference	307
5.64.1 Detailed Description	308

---

5.64.2 Member Function Documentation . . . . .	308
5.64.2.1 initialize() . . . . .	308
5.64.2.2 run() . . . . .	308
5.64.2.3 getVelocity() . . . . .	309
5.64.2.4 setVelocity() . . . . .	309
5.64.2.5 setVoltage() . . . . .	309
5.64.2.6 getRadius() . . . . .	310
5.64.2.7 setLeftMotors() . . . . .	310
5.64.2.8 setRightMotors() . . . . .	310
5.64.2.9 setVelocityToVoltage() . . . . .	311
5.64.2.10 setGearRatio() . . . . .	311
5.64.2.11 setWheelRadius() . . . . .	311
5.64.2.12 setRadius() . . . . .	312
5.64.3 Member Data Documentation . . . . .	312
5.64.3.1 m_left_motors . . . . .	312
5.64.3.2 m_right_motors . . . . .	312
5.64.3.3 m_velocity_to_voltage . . . . .	312
5.64.3.4 m_gear_ratio . . . . .	313
5.64.3.5 m_wheel_radius . . . . .	313
5.64.3.6 m_radius . . . . .	313
5.65 wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder Class Reference . . . . .	313
5.65.1 Detailed Description . . . . .	314
5.65.2 Member Function Documentation . . . . .	314
5.65.2.1 withLeftMotor() . . . . .	314
5.65.2.2 withRightMotor() . . . . .	315
5.65.2.3 withVelocityToVoltage() . . . . .	315
5.65.2.4 withGearRatio() . . . . .	315
5.65.2.5 withWheelRadius() . . . . .	316
5.65.2.6 withRadius() . . . . .	316
5.65.2.7 build() . . . . .	317
5.65.3 Member Data Documentation . . . . .	317
5.65.3.1 m_left_motors . . . . .	317
5.65.3.2 m_right_motors . . . . .	317
5.65.3.3 m_velocity_to_voltage . . . . .	317
5.65.3.4 m_gear_ratio . . . . .	318
5.65.3.5 m_wheel_radius . . . . .	318
5.65.3.6 m_radius . . . . .	318
5.66 wisco::robot::subsystems::drive::IDifferentialDrive Class Reference . . . . .	318
5.66.1 Detailed Description . . . . .	319
5.66.2 Member Function Documentation . . . . .	319
5.66.2.1 initialize() . . . . .	319
5.66.2.2 run() . . . . .	319

---

5.66.2.3 getVelocity() . . . . .	319
5.66.2.4 setVelocity() . . . . .	319
5.66.2.5 setVoltage() . . . . .	320
5.66.2.6 getRadius() . . . . .	320
5.67 wisco::robot::subsystems::drive::IVelocityProfile Class Reference . . . . .	320
5.67.1 Detailed Description . . . . .	321
5.67.2 Member Function Documentation . . . . .	321
5.67.2.1 getAcceleration() . . . . .	321
5.67.2.2 setAcceleration() . . . . .	321
5.68 wisco::robot::subsystems::drive::KinematicDifferentialDrive Class Reference . . . . .	322
5.68.1 Detailed Description . . . . .	324
5.68.2 Member Function Documentation . . . . .	324
5.68.2.1 taskLoop() . . . . .	324
5.68.2.2 taskUpdate() . . . . .	325
5.68.2.3 updateAcceleration() . . . . .	325
5.68.2.4 initialize() . . . . .	326
5.68.2.5 run() . . . . .	326
5.68.2.6 getVelocity() . . . . .	326
5.68.2.7 setVelocity() . . . . .	327
5.68.2.8 setVoltage() . . . . .	327
5.68.2.9 getRadius() . . . . .	328
5.68.2.10 setDelayer() . . . . .	328
5.68.2.11 setMutex() . . . . .	328
5.68.2.12 setTask() . . . . .	329
5.68.2.13 setVelocityProfiles() . . . . .	329
5.68.2.14 setLeftMotors() . . . . .	329
5.68.2.15 setRightMotors() . . . . .	330
5.68.2.16 setMass() . . . . .	330
5.68.2.17 setRadius() . . . . .	330
5.68.2.18 setMomentOfInertia() . . . . .	330
5.68.2.19 setGearRatio() . . . . .	332
5.68.2.20 setWheelRadius() . . . . .	332
5.68.3 Member Data Documentation . . . . .	332
5.68.3.1 TASK_DELAY . . . . .	332
5.68.3.2 m_delayer . . . . .	333
5.68.3.3 m_mutex . . . . .	333
5.68.3.4 m_task . . . . .	333
5.68.3.5 m_left_velocity_profile . . . . .	333
5.68.3.6 m_right_velocity_profile . . . . .	333
5.68.3.7 m_left_motors . . . . .	334
5.68.3.8 m_right_motors . . . . .	334
5.68.3.9 m_mass . . . . .	334

---

5.68.3.10 m_radius . . . . .	334
5.68.3.11 m_moment_of_inertia . . . . .	334
5.68.3.12 m_gear_ratio . . . . .	335
5.68.3.13 m_wheel_radius . . . . .	335
5.68.3.14 c1 . . . . .	335
5.68.3.15 c2 . . . . .	335
5.68.3.16 c3 . . . . .	335
5.68.3.17 c4 . . . . .	335
5.68.3.18 c5 . . . . .	336
5.68.3.19 c6 . . . . .	336
5.68.3.20 c7 . . . . .	336
5.68.3.21 m_velocity . . . . .	336
5.69 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder Class Reference . . . . .	336
5.69.1 Detailed Description . . . . .	338
5.69.2 Member Function Documentation . . . . .	338
5.69.2.1 withDelay() . . . . .	338
5.69.2.2 withMutex() . . . . .	338
5.69.2.3 withTask() . . . . .	339
5.69.2.4 withLeftVelocityProfile() . . . . .	339
5.69.2.5 withRightVelocityProfile() . . . . .	340
5.69.2.6 withLeftMotor() . . . . .	340
5.69.2.7 withRightMotor() . . . . .	340
5.69.2.8 withMass() . . . . .	341
5.69.2.9 withRadius() . . . . .	341
5.69.2.10 withMomentOfInertia() . . . . .	342
5.69.2.11 withGearRatio() . . . . .	342
5.69.2.12 withWheelRadius() . . . . .	343
5.69.2.13 build() . . . . .	343
5.69.3 Member Data Documentation . . . . .	343
5.69.3.1 m_delayer . . . . .	343
5.69.3.2 m_mutex . . . . .	344
5.69.3.3 m_task . . . . .	344
5.69.3.4 m_left_velocity_profile . . . . .	344
5.69.3.5 m_right_velocity_profile . . . . .	344
5.69.3.6 m_left_motors . . . . .	344
5.69.3.7 m_right_motors . . . . .	345
5.69.3.8 m_mass . . . . .	345
5.69.3.9 m_radius . . . . .	345
5.69.3.10 m_moment_of_inertia . . . . .	345
5.69.3.11 m_gear_ratio . . . . .	345
5.69.3.12 m_wheel_radius . . . . .	346
5.70 wisco::robot::subsystems::drive::Velocity Struct Reference . . . . .	346

---

5.70.1 Detailed Description . . . . .	346
5.70.2 Member Data Documentation . . . . .	346
5.70.2.1 left_velocity . . . . .	346
5.70.2.2 right_velocity . . . . .	346
5.71 wisco::robot::subsystems::elevator::ElevatorSubsystem Class Reference . . . . .	347
5.71.1 Detailed Description . . . . .	348
5.71.2 Constructor & Destructor Documentation . . . . .	348
5.71.2.1 ElevatorSubsystem() . . . . .	348
5.71.3 Member Function Documentation . . . . .	349
5.71.3.1 initialize() . . . . .	349
5.71.3.2 run() . . . . .	349
5.71.3.3 command() . . . . .	349
5.71.3.4 state() . . . . .	350
5.71.4 Member Data Documentation . . . . .	350
5.71.4.1 SUBSYSTEM_NAME . . . . .	350
5.71.4.2 SET_POSITION_COMMAND_NAME . . . . .	350
5.71.4.3 GET_POSITION_STATE_NAME . . . . .	351
5.71.4.4 CAP_DISTANCE_STATE_NAME . . . . .	351
5.71.4.5 m_elevator . . . . .	351
5.71.4.6 m_distance_sensor . . . . .	351
5.72 wisco::robot::subsystems::elevator::IElevator Class Reference . . . . .	351
5.72.1 Detailed Description . . . . .	352
5.72.2 Member Function Documentation . . . . .	352
5.72.2.1 initialized() . . . . .	352
5.72.2.2 run() . . . . .	352
5.72.2.3 getPosition() . . . . .	352
5.72.2.4 setPosition() . . . . .	352
5.73 wisco::robot::subsystems::elevator::PIDElevator Class Reference . . . . .	353
5.73.1 Detailed Description . . . . .	354
5.73.2 Member Function Documentation . . . . .	354
5.73.2.1 taskLoop() . . . . .	354
5.73.2.2 taskUpdate() . . . . .	355
5.73.2.3 updatePosition() . . . . .	355
5.73.2.4 initialize() . . . . .	355
5.73.2.5 run() . . . . .	356
5.73.2.6 getPosition() . . . . .	356
5.73.2.7 setPosition() . . . . .	356
5.73.2.8 setClock() . . . . .	357
5.73.2.9 setDelayer() . . . . .	357
5.73.2.10 setMutex() . . . . .	357
5.73.2.11 setTask() . . . . .	358
5.73.2.12 setPID() . . . . .	358

5.73.2.13 setMotors() . . . . .	358
5.73.2.14 setRotationSensor() . . . . .	359
5.73.2.15 setInchesPerRadian() . . . . .	359
5.73.3 Member Data Documentation . . . . .	359
5.73.3.1 TASK_DELAY . . . . .	359
5.73.3.2 m_clock . . . . .	359
5.73.3.3 m_delayer . . . . .	360
5.73.3.4 m_mutex . . . . .	360
5.73.3.5 m_task . . . . .	360
5.73.3.6 m_pid . . . . .	360
5.73.3.7 m_motors . . . . .	360
5.73.3.8 m_rotation_sensor . . . . .	361
5.73.3.9 m_inches_per_radian . . . . .	361
5.73.3.10 m_position . . . . .	361
5.74 wisco::robot::subsystems::elevator::PIDElevatorBuilder Class Reference . . . . .	361
5.74.1 Detailed Description . . . . .	362
5.74.2 Member Function Documentation . . . . .	362
5.74.2.1 withClock() . . . . .	362
5.74.2.2 withDelayer() . . . . .	363
5.74.2.3 withMutex() . . . . .	363
5.74.2.4 withTask() . . . . .	363
5.74.2.5 withPID() . . . . .	364
5.74.2.6 withMotor() . . . . .	364
5.74.2.7 withRotationSensor() . . . . .	365
5.74.2.8 withInchesPerRadian() . . . . .	365
5.74.2.9 build() . . . . .	365
5.74.3 Member Data Documentation . . . . .	366
5.74.3.1 m_clock . . . . .	366
5.74.3.2 m_delayer . . . . .	366
5.74.3.3 m_mutex . . . . .	366
5.74.3.4 m_task . . . . .	366
5.74.3.5 m_pid . . . . .	367
5.74.3.6 m_motors . . . . .	367
5.74.3.7 m_rotation_sensor . . . . .	367
5.74.3.8 m_inches_per_radian . . . . .	367
5.75 wisco::robot::subsystems::hang::HangSubsystem Class Reference . . . . .	367
5.75.1 Detailed Description . . . . .	369
5.75.2 Constructor & Destructor Documentation . . . . .	369
5.75.2.1 HangSubsystem() . . . . .	369
5.75.3 Member Function Documentation . . . . .	370
5.75.3.1 initialize() . . . . .	370
5.75.3.2 run() . . . . .	370

---

5.75.3.3 command()	370
5.75.3.4 state()	371
5.75.4 Member Data Documentation	372
5.75.4.1 SUBSYSTEM_NAME	372
5.75.4.2 CLOSE_CLAW_COMMAND_NAME	372
5.75.4.3 OPEN_CLAW_COMMAND_NAME	373
5.75.4.4 LOWER_ARM_COMMAND_NAME	373
5.75.4.5 RAISE_ARM_COMMAND_NAME	373
5.75.4.6 ENGAGE_WINCH_COMMAND_NAME	373
5.75.4.7 DISENGAGE_WINCH_COMMAND_NAME	373
5.75.4.8 CLAW_CLOSED_STATE_NAME	374
5.75.4.9 CLAW_OPEN_STATE_NAME	374
5.75.4.10 ARM_DOWN_STATE_NAME	374
5.75.4.11 ARM_UP_STATE_NAME	374
5.75.4.12 WINCH_ENGAGED_STATE_NAME	374
5.75.4.13 WINCH_DISENGAGED_STATE_NAME	375
5.75.4.14 m_claw	375
5.75.4.15 m_toggle_arm	375
5.75.4.16 m_winch	375
5.76 wisco::robot::subsystems::hang::IClaw Class Reference	375
5.76.1 Detailed Description	376
5.76.2 Member Function Documentation	376
5.76.2.1 initialize()	376
5.76.2.2 run()	376
5.76.2.3 close()	377
5.76.2.4 open()	377
5.76.2.5 isClosed()	377
5.76.2.6 isOpen()	377
5.77 wisco::robot::subsystems::hang::IToggleArm Class Reference	377
5.77.1 Detailed Description	378
5.77.2 Member Function Documentation	378
5.77.2.1 initialize()	378
5.77.2.2 run()	378
5.77.2.3 setDown()	379
5.77.2.4 setUp()	379
5.77.2.5 isDown()	379
5.77.2.6 isUp()	379
5.78 wisco::robot::subsystems::hang::IWinch Class Reference	379
5.78.1 Detailed Description	380
5.78.2 Member Function Documentation	380
5.78.2.1 initialize()	380
5.78.2.2 run()	380

---

5.78.2.3 engage() . . . . .	381
5.78.2.4 disengage() . . . . .	381
5.78.2.5 isEngaged() . . . . .	381
5.78.2.6 isDisengaged() . . . . .	381
5.79 wisco::robot::subsystems::hang::PistonClaw Class Reference . . . . .	381
5.79.1 Detailed Description . . . . .	382
5.79.2 Member Function Documentation . . . . .	382
5.79.2.1 initialize() . . . . .	382
5.79.2.2 run() . . . . .	383
5.79.2.3 close() . . . . .	383
5.79.2.4 open() . . . . .	383
5.79.2.5 isClosed() . . . . .	383
5.79.2.6 isOpen() . . . . .	384
5.79.2.7 setPistons() . . . . .	384
5.79.2.8 setClosedState() . . . . .	384
5.79.3 Member Data Documentation . . . . .	385
5.79.3.1 m_pistons . . . . .	385
5.79.3.2 m_closed_state . . . . .	385
5.80 wisco::robot::subsystems::hang::PistonClawBuilder Class Reference . . . . .	385
5.80.1 Detailed Description . . . . .	385
5.80.2 Member Function Documentation . . . . .	386
5.80.2.1 withPiston() . . . . .	386
5.80.2.2 withClosedState() . . . . .	387
5.80.2.3 build() . . . . .	387
5.80.3 Member Data Documentation . . . . .	388
5.80.3.1 m_pistons . . . . .	388
5.80.3.2 m_closed_state . . . . .	388
5.81 wisco::robot::subsystems::hang::PistonToggleArm Class Reference . . . . .	388
5.81.1 Detailed Description . . . . .	389
5.81.2 Member Function Documentation . . . . .	389
5.81.2.1 initialize() . . . . .	389
5.81.2.2 run() . . . . .	389
5.81.2.3 setDown() . . . . .	390
5.81.2.4 setUp() . . . . .	390
5.81.2.5 isDown() . . . . .	390
5.81.2.6 isUp() . . . . .	390
5.81.2.7 setPistons() . . . . .	390
5.81.2.8 setUpState() . . . . .	391
5.81.3 Member Data Documentation . . . . .	391
5.81.3.1 m_pistons . . . . .	391
5.81.3.2 m_up_state . . . . .	391
5.82 wisco::robot::subsystems::hang::PistonToggleArmBuilder Class Reference . . . . .	391

---

5.82.1 Detailed Description . . . . .	392
5.82.2 Member Function Documentation . . . . .	392
5.82.2.1 withPiston() . . . . .	392
5.82.2.2 withUpState() . . . . .	393
5.82.2.3 build() . . . . .	393
5.82.3 Member Data Documentation . . . . .	393
5.82.3.1 m_pistons . . . . .	393
5.82.3.2 m_up_state . . . . .	394
5.83 wisco::robot::subsystems::hang::PistonWinch Class Reference . . . . .	394
5.83.1 Detailed Description . . . . .	395
5.83.2 Member Function Documentation . . . . .	395
5.83.2.1 initialize() . . . . .	395
5.83.2.2 run() . . . . .	395
5.83.2.3 engage() . . . . .	395
5.83.2.4 disengage() . . . . .	396
5.83.2.5 isEngaged() . . . . .	396
5.83.2.6 isDisengaged() . . . . .	396
5.83.2.7 setPistons() . . . . .	396
5.83.2.8 setEngagedState() . . . . .	397
5.83.3 Member Data Documentation . . . . .	397
5.83.3.1 m_pistons . . . . .	397
5.83.3.2 m_engaged_state . . . . .	397
5.84 wisco::robot::subsystems::hang::PistonWinchBuilder Class Reference . . . . .	397
5.84.1 Detailed Description . . . . .	398
5.84.2 Member Function Documentation . . . . .	398
5.84.2.1 withPiston() . . . . .	398
5.84.2.2 withEngagedState() . . . . .	399
5.84.2.3 build() . . . . .	399
5.84.3 Member Data Documentation . . . . .	399
5.84.3.1 m_pistons . . . . .	399
5.84.3.2 m_engaged_state . . . . .	400
5.85 wisco::robot::subsystems::intake::DistanceVisionBallDetector Class Reference . . . . .	400
5.85.1 Detailed Description . . . . .	401
5.85.2 Member Function Documentation . . . . .	401
5.85.2.1 initialize() . . . . .	401
5.85.2.2 run() . . . . .	401
5.85.2.3 getBallDistance() . . . . .	401
5.85.2.4 getBallAngle() . . . . .	402
5.85.2.5 setDistanceSensor() . . . . .	402
5.85.3 Member Data Documentation . . . . .	402
5.85.3.1 m_distance_sensor . . . . .	402
5.86 wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder Class Reference . . . . .	402

---

5.86.1 Detailed Description . . . . .	403
5.86.2 Member Function Documentation . . . . .	403
5.86.2.1 withDistanceSensor() . . . . .	403
5.86.2.2 build() . . . . .	404
5.86.3 Member Data Documentation . . . . .	404
5.86.3.1 m_distance_sensor . . . . .	404
5.87 wisco::robot::subsystems::intake::IBallDetector Class Reference . . . . .	404
5.87.1 Detailed Description . . . . .	405
5.87.2 Member Function Documentation . . . . .	405
5.87.2.1 initialize() . . . . .	405
5.87.2.2 run() . . . . .	405
5.87.2.3 getBallDistance() . . . . .	405
5.87.2.4 getBallAngle() . . . . .	405
5.88 wisco::robot::subsystems::intake::IIntake Class Reference . . . . .	406
5.88.1 Detailed Description . . . . .	406
5.88.2 Member Function Documentation . . . . .	406
5.88.2.1 initialize() . . . . .	406
5.88.2.2 run() . . . . .	407
5.88.2.3 getVelocity() . . . . .	407
5.88.2.4 setVelocity() . . . . .	407
5.88.2.5 setVoltage() . . . . .	407
5.89 wisco::robot::subsystems::intake::IntakeSubsystem Class Reference . . . . .	408
5.89.1 Detailed Description . . . . .	409
5.89.2 Constructor & Destructor Documentation . . . . .	409
5.89.2.1 IntakeSubsystem() . . . . .	409
5.89.3 Member Function Documentation . . . . .	410
5.89.3.1 initialize() . . . . .	410
5.89.3.2 run() . . . . .	410
5.89.3.3 command() . . . . .	410
5.89.3.4 state() . . . . .	411
5.89.4 Member Data Documentation . . . . .	411
5.89.4.1 SUBSYSTEM_NAME . . . . .	411
5.89.4.2 SET_VELOCITY_COMMAND_NAME . . . . .	412
5.89.4.3 SET_VOLTAGE_COMMAND_NAME . . . . .	412
5.89.4.4 GET_VELOCITY_STATE_NAME . . . . .	412
5.89.4.5 GET_BALL_DISTANCE_STATE_NAME . . . . .	412
5.89.4.6 GET_BALL_ANGLE_STATE_NAME . . . . .	412
5.89.4.7 m_intake . . . . .	413
5.89.4.8 m_ball_detector . . . . .	413
5.90 wisco::robot::subsystems::intake::PIDIntake Class Reference . . . . .	413
5.90.1 Detailed Description . . . . .	414
5.90.2 Member Function Documentation . . . . .	415

---

5.90.2.1 taskLoop()	415
5.90.2.2 taskUpdate()	415
5.90.2.3 updateVelocity()	415
5.90.2.4 initialize()	416
5.90.2.5 run()	416
5.90.2.6 getVelocity()	416
5.90.2.7 setVelocity()	416
5.90.2.8 setVoltage()	417
5.90.2.9 setClock()	417
5.90.2.10 setDelayer()	418
5.90.2.11 setMutex()	418
5.90.2.12 setTask()	418
5.90.2.13 setPID()	418
5.90.2.14 setMotors()	420
5.90.2.15 setRollerRadius()	420
5.90.3 Member Data Documentation	420
5.90.3.1 TASK_DELAY	420
5.90.3.2 m_clock	421
5.90.3.3 m_delayer	421
5.90.3.4 m_mutex	421
5.90.3.5 m_task	421
5.90.3.6 m_pid	421
5.90.3.7 m_motors	421
5.90.3.8 m_roller_radius	422
5.90.3.9 m_velocity	422
5.90.3.10 velocity_control	422
5.91 wisco::robot::subsystems::intake::PIDIntakeBuilder Class Reference	422
5.91.1 Detailed Description	423
5.91.2 Member Function Documentation	423
5.91.2.1 withClock()	423
5.91.2.2 withDelayer()	424
5.91.2.3 withMutex()	424
5.91.2.4 withTask()	424
5.91.2.5 withPID()	425
5.91.2.6 withMotor()	425
5.91.2.7 withRollerRadius()	426
5.91.2.8 build()	426
5.91.3 Member Data Documentation	426
5.91.3.1 m_clock	426
5.91.3.2 m_delayer	427
5.91.3.3 m_mutex	427
5.91.3.4 m_task	427

---

5.91.3.5 m_pid . . . . .	427
5.91.3.6 m_motors . . . . .	427
5.91.3.7 m_roller_radius . . . . .	428
5.92 wisco::robot::subsystems::loader::ILoader Class Reference . . . . .	428
5.92.1 Detailed Description . . . . .	428
5.92.2 Member Function Documentation . . . . .	429
5.92.2.1 initialize() . . . . .	429
5.92.2.2 run() . . . . .	429
5.92.2.3 doLoad() . . . . .	429
5.92.2.4 doReady() . . . . .	429
5.92.2.5 isLoaded() . . . . .	429
5.92.2.6 isReady() . . . . .	430
5.93 wisco::robot::subsystems::loader::LoaderSubsystem Class Reference . . . . .	430
5.93.1 Detailed Description . . . . .	431
5.93.2 Constructor & Destructor Documentation . . . . .	431
5.93.2.1 LoaderSubsystem() . . . . .	431
5.93.3 Member Function Documentation . . . . .	432
5.93.3.1 initialize() . . . . .	432
5.93.3.2 run() . . . . .	432
5.93.3.3 command() . . . . .	432
5.93.3.4 state() . . . . .	433
5.93.4 Member Data Documentation . . . . .	433
5.93.4.1 SUBSYSTEM_NAME . . . . .	433
5.93.4.2 DO_LOAD_COMMAND_NAME . . . . .	434
5.93.4.3 DO_READY_COMMAND_NAME . . . . .	434
5.93.4.4 IS_LOADED_STATE_NAME . . . . .	434
5.93.4.5 IS_READY_STATE_NAME . . . . .	434
5.93.4.6 m_loader . . . . .	434
5.94 wisco::robot::subsystems::loader::PIDLoader Class Reference . . . . .	435
5.94.1 Detailed Description . . . . .	437
5.94.2 Member Enumeration Documentation . . . . .	437
5.94.2.1 EState . . . . .	437
5.94.3 Member Function Documentation . . . . .	437
5.94.3.1 taskLoop() . . . . .	437
5.94.3.2 taskUpdate() . . . . .	438
5.94.3.3 updateState() . . . . .	438
5.94.3.4 updatePosition() . . . . .	438
5.94.3.5 getPosition() . . . . .	438
5.94.3.6 initialize() . . . . .	439
5.94.3.7 run() . . . . .	439
5.94.3.8 doLoad() . . . . .	439
5.94.3.9 doReady() . . . . .	440

---

5.94.3.10 isLoaded()	440
5.94.3.11 isReady()	440
5.94.3.12 setClock()	440
5.94.3.13 setDelay()	441
5.94.3.14 setMutex()	441
5.94.3.15 setTask()	441
5.94.3.16 setPID()	442
5.94.3.17 setMotors()	442
5.94.3.18 setMatchLoadPosition()	442
5.94.3.19 setReadyPosition()	443
5.94.3.20 setPositionTolerance()	443
5.94.4 Member Data Documentation	443
5.94.4.1 TASK_DELAY	443
5.94.4.2 m_clock	443
5.94.4.3 m_delayer	444
5.94.4.4 m_mutex	444
5.94.4.5 m_task	444
5.94.4.6 m_pid	444
5.94.4.7 m_motors	444
5.94.4.8 m_match_load_position	444
5.94.4.9 m_ready_position	445
5.94.4.10 m_position_tolerance	445
5.94.4.11 state	445
5.94.4.12 target_position	445
5.95 wisco::robot::subsystems::loader::PIDLoaderBuilder Class Reference	445
5.95.1 Detailed Description	446
5.95.2 Member Function Documentation	447
5.95.2.1 withClock()	447
5.95.2.2 withDelay()	447
5.95.2.3 withMutex()	447
5.95.2.4 withTask()	448
5.95.2.5 withPID()	448
5.95.2.6 withMotor()	449
5.95.2.7 withMatchLoadPosition()	449
5.95.2.8 withReadyPosition()	449
5.95.2.9 withPositionTolerance()	450
5.95.2.10 build()	450
5.95.3 Member Data Documentation	451
5.95.3.1 m_clock	451
5.95.3.2 m_delayer	451
5.95.3.3 m_mutex	451
5.95.3.4 m_task	451

5.95.3.5 m_pid . . . . .	451
5.95.3.6 m_motors . . . . .	452
5.95.3.7 m_match_load_position . . . . .	452
5.95.3.8 m_ready_position . . . . .	452
5.95.3.9 m_position_tolerance . . . . .	452
5.96 wisco::robot::subsystems::position::DistancePositionResetter Class Reference . . . . .	452
5.96.1 Detailed Description . . . . .	454
5.96.2 Member Function Documentation . . . . .	454
5.96.2.1 bindRadians() . . . . .	454
5.96.2.2 initialize() . . . . .	454
5.96.2.3 run() . . . . .	455
5.96.2.4 getResetX() . . . . .	455
5.96.2.5 getResetY() . . . . .	455
5.96.2.6 setDistanceSensor() . . . . .	456
5.96.2.7 setLocalX() . . . . .	456
5.96.2.8 setLocalY() . . . . .	456
5.96.2.9 setLocalTheta() . . . . .	457
5.96.3 Member Data Documentation . . . . .	457
5.96.3.1 NEAR_WALL . . . . .	457
5.96.3.2 FAR_WALL . . . . .	457
5.96.3.3 m_distance_sensor . . . . .	458
5.96.3.4 m_local_x . . . . .	458
5.96.3.5 m_local_y . . . . .	458
5.96.3.6 m_local_theta . . . . .	458
5.97 wisco::robot::subsystems::position::DistancePositionResetterBuilder Class Reference . . . . .	458
5.97.1 Detailed Description . . . . .	459
5.97.2 Member Function Documentation . . . . .	459
5.97.2.1 withDistanceSensor() . . . . .	459
5.97.2.2 withLocalX() . . . . .	459
5.97.2.3 withLocalY() . . . . .	460
5.97.2.4 withLocalTheta() . . . . .	460
5.97.2.5 build() . . . . .	461
5.97.3 Member Data Documentation . . . . .	461
5.97.3.1 m_distance_sensor . . . . .	461
5.97.3.2 m_local_x . . . . .	461
5.97.3.3 m_local_y . . . . .	462
5.97.3.4 m_local_theta . . . . .	462
5.98 wisco::robot::subsystems::position::InertialOdometry Class Reference . . . . .	462
5.98.1 Detailed Description . . . . .	464
5.98.2 Member Function Documentation . . . . .	464
5.98.2.1 taskLoop() . . . . .	464
5.98.2.2 taskUpdate() . . . . .	465

---

5.98.2.3 updatePosition()	465
5.98.2.4 initialize()	466
5.98.2.5 run()	466
5.98.2.6 setPosition()	466
5.98.2.7 getPosition()	467
5.98.2.8 setClock()	467
5.98.2.9 setDelayer()	468
5.98.2.10 setMutex()	468
5.98.2.11 setTask()	468
5.98.2.12 setHeadingSensor()	468
5.98.2.13 setLinearDistanceTrackingSensor()	470
5.98.2.14 setLinearDistanceTrackingOffset()	470
5.98.2.15 setStrafeDistanceTrackingSensor()	470
5.98.2.16 setStrafeDistanceTrackingOffset()	471
5.98.3 Member Data Documentation	471
5.98.3.1 TASK_DELAY	471
5.98.3.2 TIME_UNIT_CONVERTER	471
5.98.3.3 m_clock	472
5.98.3.4 m_delayer	472
5.98.3.5 m_mutex	472
5.98.3.6 m_task	472
5.98.3.7 m_heading_sensor	472
5.98.3.8 m_linear_distance_tracking_sensor	473
5.98.3.9 m_linear_distance_tracking_offset	473
5.98.3.10 m_strafe_distance_tracking_sensor	473
5.98.3.11 m_strafe_distance_tracking_offset	473
5.98.3.12 m_position	473
5.98.3.13 last_heading	474
5.98.3.14 last_linear_distance	474
5.98.3.15 last_strafe_distance	474
5.98.3.16 last_time	474
5.99 wisco::robot::subsystems::position::InertialOdometryBuilder Class Reference	474
5.99.1 Detailed Description	476
5.99.2 Member Function Documentation	476
5.99.2.1 withClock()	476
5.99.2.2 withDelayer()	476
5.99.2.3 withMutex()	477
5.99.2.4 withTask()	477
5.99.2.5 withHeadingSensor()	477
5.99.2.6 withLinearDistanceTrackingSensor()	478
5.99.2.7 withLinearDistanceTrackingOffset()	478
5.99.2.8 withStrafeDistance TrackingSensor()	479

---

5.99.2.9 withStrafeDistanceTrackingOffset()	479
5.99.2.10 build()	480
5.99.3 Member Data Documentation	480
5.99.3.1 m_clock	480
5.99.3.2 m_delayer	480
5.99.3.3 m_mutex	480
5.99.3.4 m_task	481
5.99.3.5 m_heading_sensor	481
5.99.3.6 m_linear_distance_tracking_sensor	481
5.99.3.7 m_linear_distance_tracking_offset	481
5.99.3.8 m_strafe_distance_tracking_sensor	481
5.99.3.9 m_strafe_distance_tracking_offset	482
5.100 wisco::robot::subsystems::position::IPositionResetter Class Reference	482
5.100.1 Detailed Description	482
5.100.2 Member Function Documentation	483
5.100.2.1 initialize()	483
5.100.2.2 run()	483
5.100.2.3 getResetX()	483
5.100.2.4 getResetY()	483
5.101 wisco::robot::subsystems::position::IPositionTracker Class Reference	484
5.101.1 Detailed Description	484
5.101.2 Member Function Documentation	484
5.101.2.1 initialize()	484
5.101.2.2 run()	485
5.101.2.3 setPosition()	485
5.101.2.4 getPosition()	485
5.102 wisco::robot::subsystems::position::Position Struct Reference	485
5.102.1 Detailed Description	486
5.102.2 Member Data Documentation	486
5.102.2.1 x	486
5.102.2.2 y	486
5.102.2.3 theta	486
5.102.2.4 xV	486
5.102.2.5 yV	487
5.102.2.6 thetaV	487
5.103 wisco::robot::subsystems::position::PositionSubsystem Class Reference	487
5.103.1 Detailed Description	488
5.103.2 Constructor & Destructor Documentation	489
5.103.2.1 PositionSubsystem()	489
5.103.3 Member Function Documentation	489
5.103.3.1 initialize()	489
5.103.3.2 run()	489

---

5.103.3.3 command()	489
5.103.3.4 state()	490
5.103.4 Member Data Documentation	491
5.103.4.1 SUBSYSTEM_NAME	491
5.103.4.2 SET_POSITION_COMMAND_NAME	491
5.103.4.3 RESET_X_COMMAND_NAME	491
5.103.4.4 RESET_Y_COMMAND_NAME	491
5.103.4.5 GET_POSITION_STATE_NAME	491
5.103.4.6 m_position_tracker	492
5.103.4.7 m_position_resetter	492
5.104 wisco::robot::subsystems::umbrella::IUmbrella Class Reference	492
5.104.1 Detailed Description	493
5.104.2 Member Function Documentation	493
5.104.2.1 initialize()	493
5.104.2.2 run()	493
5.104.2.3 setIn()	493
5.104.2.4 setOut()	493
5.104.2.5 isIn()	494
5.104.2.6 isOut()	494
5.105 wisco::robot::subsystems::umbrella::PistonUmbrella Class Reference	494
5.105.1 Detailed Description	495
5.105.2 Member Function Documentation	495
5.105.2.1 initialize()	495
5.105.2.2 run()	495
5.105.2.3 setIn()	496
5.105.2.4 setOut()	496
5.105.2.5 isIn()	496
5.105.2.6 isOut()	496
5.105.2.7 setPistons()	496
5.105.2.8 setOutState()	497
5.105.3 Member Data Documentation	497
5.105.3.1 m_pistons	497
5.105.3.2 m_out_state	497
5.106 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder Class Reference	497
5.106.1 Detailed Description	498
5.106.2 Member Function Documentation	498
5.106.2.1 withPiston()	498
5.106.2.2 withOutState()	499
5.106.2.3 build()	499
5.106.3 Member Data Documentation	499
5.106.3.1 m_pistons	499
5.106.3.2 m_out_state	500

---

5.107 wisco::robot::subsystems::umbrella::UmbrellaSubsystem Class Reference . . . . .	500
5.107.1 Detailed Description . . . . .	501
5.107.2 Constructor & Destructor Documentation . . . . .	501
5.107.2.1 UmbrellaSubsystem() . . . . .	501
5.107.3 Member Function Documentation . . . . .	502
5.107.3.1 initialize() . . . . .	502
5.107.3.2 run() . . . . .	502
5.107.3.3 command() . . . . .	502
5.107.3.4 state() . . . . .	503
5.107.4 Member Data Documentation . . . . .	503
5.107.4.1 SUBSYSTEM_NAME . . . . .	503
5.107.4.2 SET_IN_COMMAND_NAME . . . . .	503
5.107.4.3 SET_OUT_COMMAND_NAME . . . . .	504
5.107.4.4 IS_IN_STATE_NAME . . . . .	504
5.107.4.5 IS_OUT_STATE_NAME . . . . .	504
5.107.4.6 m_umbrella . . . . .	504
5.108 wisco::robot::subsystems::wings::IWings Class Reference . . . . .	504
5.108.1 Detailed Description . . . . .	505
5.108.2 Member Function Documentation . . . . .	505
5.108.2.1 initialize() . . . . .	505
5.108.2.2 run() . . . . .	505
5.108.2.3 setLeftWing() . . . . .	505
5.108.2.4 setRightWing() . . . . .	506
5.108.2.5 getLeftWing() . . . . .	506
5.108.2.6 getRightWing() . . . . .	506
5.109 wisco::robot::subsystems::wings::PistonWings Class Reference . . . . .	507
5.109.1 Detailed Description . . . . .	508
5.109.2 Member Function Documentation . . . . .	508
5.109.2.1 initialize() . . . . .	508
5.109.2.2 run() . . . . .	508
5.109.2.3 setLeftWing() . . . . .	508
5.109.2.4 setRightWing() . . . . .	509
5.109.2.5 getLeftWing() . . . . .	509
5.109.2.6 getRightWing() . . . . .	509
5.109.2.7 setLeftPistons() . . . . .	510
5.109.2.8 setRightPistons() . . . . .	510
5.109.2.9 setOutState() . . . . .	510
5.109.3 Member Data Documentation . . . . .	511
5.109.3.1 m_left_pistons . . . . .	511
5.109.3.2 m_right_pistons . . . . .	511
5.109.3.3 m_out_state . . . . .	511
5.110 wisco::robot::subsystems::wings::PistonWingsBuilder Class Reference . . . . .	511

---

5.110.1 Detailed Description . . . . .	512
5.110.2 Member Function Documentation . . . . .	512
5.110.2.1 withLeftPiston() . . . . .	512
5.110.2.2 withRightPiston() . . . . .	512
5.110.2.3 withOutState() . . . . .	513
5.110.2.4 build() . . . . .	513
5.110.3 Member Data Documentation . . . . .	514
5.110.3.1 m_left_pistons . . . . .	514
5.110.3.2 m_right_pistons . . . . .	514
5.110.3.3 m_out_state . . . . .	514
5.111 wisco::robot::subsystems::wings::WingsSubsystem Class Reference . . . . .	514
5.111.1 Detailed Description . . . . .	515
5.111.2 Constructor & Destructor Documentation . . . . .	515
5.111.2.1 WingsSubsystem() . . . . .	515
5.111.3 Member Function Documentation . . . . .	516
5.111.3.1 initialize() . . . . .	516
5.111.3.2 run() . . . . .	516
5.111.3.3 command() . . . . .	516
5.111.3.4 state() . . . . .	517
5.111.4 Member Data Documentation . . . . .	517
5.111.4.1 SUBSYSTEM_NAME . . . . .	517
5.111.4.2 SET_LEFT_WING_COMMAND_NAME . . . . .	518
5.111.4.3 SET_RIGHT_WING_COMMAND_NAME . . . . .	518
5.111.4.4 GET_LEFT_WING_STATE_NAME . . . . .	518
5.111.4.5 GET_RIGHT_WING_STATE_NAME . . . . .	518
5.111.4.6 m_wings . . . . .	518
5.112 wisco::rtos::IClock Class Reference . . . . .	519
5.112.1 Detailed Description . . . . .	519
5.112.2 Member Function Documentation . . . . .	519
5.112.2.1 clone() . . . . .	519
5.112.2.2 getTime() . . . . .	520
5.113 wisco::rtos::IDelayer Class Reference . . . . .	520
5.113.1 Detailed Description . . . . .	520
5.113.2 Member Function Documentation . . . . .	521
5.113.2.1 clone() . . . . .	521
5.113.2.2 delay() . . . . .	521
5.113.2.3 delayUntil() . . . . .	521
5.114 wisco::rtos::IMutex Class Reference . . . . .	521
5.114.1 Detailed Description . . . . .	522
5.114.2 Member Function Documentation . . . . .	522
5.114.2.1 take() . . . . .	522
5.114.2.2 give() . . . . .	522

---

5.115 wisco::rtos::ITask Class Reference . . . . .	523
5.115.1 Detailed Description . . . . .	523
5.115.2 Member Function Documentation . . . . .	523
5.115.2.1 start() . . . . .	523
5.115.2.2 remove() . . . . .	524
5.115.2.3 suspend() . . . . .	524
5.115.2.4 resume() . . . . .	524
5.115.2.5 join() . . . . .	524
5.116 wisco::SystemConfiguration Struct Reference . . . . .	524
5.116.1 Detailed Description . . . . .	525
5.116.2 Member Data Documentation . . . . .	525
5.116.2.1 alliance . . . . .	525
5.116.2.2 autonomous . . . . .	525
5.116.2.3 configuration . . . . .	525
5.116.2.4 profile . . . . .	526
5.117 wisco::testing::pros_testing::DriveTest Class Reference . . . . .	526
5.117.1 Detailed Description . . . . .	527
5.117.2 Constructor & Destructor Documentation . . . . .	527
5.117.2.1 DriveTest() . . . . .	527
5.117.3 Member Function Documentation . . . . .	528
5.117.3.1 initialize() . . . . .	528
5.117.3.2 runLinearTest() . . . . .	528
5.117.3.3 runTurningTest() . . . . .	529
5.117.4 Member Data Documentation . . . . .	529
5.117.4.1 LINEAR_FILE_NAME . . . . .	529
5.117.4.2 TURNING_FILE_NAME . . . . .	529
5.117.4.3 MILLIS_TO_S . . . . .	530
5.117.4.4 HEADING_TO_RADIANS . . . . .	530
5.117.4.5 INCHES_TO_METERS . . . . .	530
5.117.4.6 V_TO_MV . . . . .	530
5.117.4.7 TEST_V . . . . .	530
5.117.4.8 TEST_DURATION . . . . .	531
5.117.4.9 m_left_drive_motors . . . . .	531
5.117.4.10 m_right_drive_motors . . . . .	531
5.117.4.11 m_heading_sensor . . . . .	531
5.117.4.12 m_linear_sensor . . . . .	531
5.117.4.13 m_linear_counts_per_inch . . . . .	532
5.118 wisco::testing::TestFactory Class Reference . . . . .	532
5.118.1 Detailed Description . . . . .	532
5.118.2 Member Function Documentation . . . . .	533
5.118.2.1 createDriveTest() . . . . .	533
5.118.3 Member Data Documentation . . . . .	533

---

5.118.3.1 LEFT_DRIVE_PORTS . . . . .	533
5.118.3.2 RIGHT_DRIVE_PORTS . . . . .	533
5.118.3.3 INERTIAL_PORT . . . . .	533
5.118.3.4 LINEAR_TRACKING_PORT . . . . .	534
5.118.3.5 LINEAR_COUNTS_PER_INCH . . . . .	534
5.119 wisco::user::drive::DifferentialDriveOperator Class Reference . . . . .	534
5.119.1 Detailed Description . . . . .	535
5.119.2 Constructor & Destructor Documentation . . . . .	535
5.119.2.1 DifferentialDriveOperator() . . . . .	535
5.119.3 Member Function Documentation . . . . .	536
5.119.3.1 updateDriveVoltage() . . . . .	536
5.119.3.2 updateArcade() . . . . .	536
5.119.3.3 updateSingleArcadeLeft() . . . . .	536
5.119.3.4 updateSingleArcadeRight() . . . . .	537
5.119.3.5 updateSplitArcadeLeft() . . . . .	537
5.119.3.6 updateSplitArcadeRight() . . . . .	537
5.119.3.7 updateTank() . . . . .	537
5.119.3.8 setDriveVoltage() . . . . .	537
5.119.4 Member Data Documentation . . . . .	538
5.119.4.1 DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME . . . . .	538
5.119.4.2 SET_VOLTAGE_COMMAND . . . . .	538
5.119.4.3 VOLTAGE_CONVERSION . . . . .	538
5.119.4.4 m_controller . . . . .	539
5.119.4.5 m_robot . . . . .	539
5.120 wisco::user::elevator::ElevatorOperator Class Reference . . . . .	539
5.120.1 Detailed Description . . . . .	541
5.120.2 Member Enumeration Documentation . . . . .	541
5.120.2.1 EToggleState . . . . .	541
5.120.3 Constructor & Destructor Documentation . . . . .	541
5.120.3.1 ElevatorOperator() . . . . .	541
5.120.4 Member Function Documentation . . . . .	542
5.120.4.1 getElevatorPosition() . . . . .	542
5.120.4.2 getCapDistance() . . . . .	542
5.120.4.3 getHangArmUp() . . . . .	543
5.120.4.4 updateElevatorPosition() . . . . .	543
5.120.4.5 updatePoleHangPosition() . . . . .	543
5.120.4.6 updateManual() . . . . .	544
5.120.4.7 updatePresetSplit() . . . . .	545
5.120.4.8 updatePresetToggle() . . . . .	546
5.120.4.9 updatePresetLadder() . . . . .	546
5.120.4.10 updatePresetLadderIntake() . . . . .	547
5.120.4.11 setElevatorPosition() . . . . .	548

---

5.120.5 Member Data Documentation . . . . .	549
5.120.5.1 ELEVATOR_SUBSYSTEM_NAME . . . . .	549
5.120.5.2 HANG_SUBSYSTEM_NAME . . . . .	549
5.120.5.3 SET_POSITION_COMMAND . . . . .	549
5.120.5.4 GET_POSITION_STATE . . . . .	550
5.120.5.5 CAP_DISTANCE_STATE_NAME . . . . .	550
5.120.5.6 HANG_ARM_UP_STATE_NAME . . . . .	550
5.120.5.7 IN_POSITION . . . . .	550
5.120.5.8 FIELD_POSITION . . . . .	550
5.120.5.9 MATCH_LOAD_POSITION . . . . .	551
5.120.5.10 POLE_HANG_POSITION . . . . .	551
5.120.5.11 PARTNER_HANG_POSITION . . . . .	551
5.120.5.12 POLE_HANG_DISTANCE . . . . .	551
5.120.5.13 CAP_DETECTED_DISTANCE . . . . .	551
5.120.5.14 m_controller . . . . .	552
5.120.5.15 m_robot . . . . .	552
5.120.5.16 toggle_state . . . . .	552
5.120.5.17 manual_input . . . . .	552
5.121 wisco::user::hang::HangOperator Class Reference . . . . .	552
5.121.1 Detailed Description . . . . .	554
5.121.2 Member Enumeration Documentation . . . . .	554
5.121.2.1 EToggleState . . . . .	554
5.121.3 Constructor & Destructor Documentation . . . . .	554
5.121.3.1 HangOperator() . . . . .	554
5.121.4 Member Function Documentation . . . . .	555
5.121.4.1 closeClaw() . . . . .	555
5.121.4.2 openClaw() . . . . .	555
5.121.4.3 lowerArm() . . . . .	555
5.121.4.4 raiseArm() . . . . .	555
5.121.4.5 engageWinch() . . . . .	556
5.121.4.6 disengageWinch() . . . . .	556
5.121.4.7 setInactiveState() . . . . .	556
5.121.4.8 setRaisedState() . . . . .	556
5.121.4.9 setGrabbedState() . . . . .	556
5.121.4.10 setHungState() . . . . .	557
5.121.4.11 updatePresetSplit() . . . . .	557
5.121.4.12 updatePresetToggle() . . . . .	557
5.121.4.13 updatePresetLadder() . . . . .	558
5.121.4.14 updatePresetReset() . . . . .	559
5.121.4.15 setHangState() . . . . .	559
5.121.5 Member Data Documentation . . . . .	560
5.121.5.1 HANG_SUBSYSTEM_NAME . . . . .	560

---

5.121.5.2 CLOSE_CLAW_COMMAND_NAME . . . . .	560
5.121.5.3 OPEN_CLAW_COMMAND_NAME . . . . .	560
5.121.5.4 LOWER_ARM_COMMAND_NAME . . . . .	560
5.121.5.5 RAISE_ARM_COMMAND_NAME . . . . .	561
5.121.5.6 ENGAGE_WINCH_COMMAND_NAME . . . . .	561
5.121.5.7 DISENGAGE_WINCH_COMMAND_NAME . . . . .	561
5.121.5.8 m_controller . . . . .	561
5.121.5.9 m_robot . . . . .	561
5.121.5.10 toggle_state . . . . .	562
5.122 wisco::user::IController Class Reference . . . . .	562
5.122.1 Detailed Description . . . . .	562
5.122.2 Member Function Documentation . . . . .	563
5.122.2.1 initialize() . . . . .	563
5.122.2.2 run() . . . . .	563
5.122.2.3 getAnalog() . . . . .	563
5.122.2.4 getDigital() . . . . .	563
5.122.2.5 getNewDigital() . . . . .	564
5.122.2.6 rumble() . . . . .	564
5.123 wisco::user::intake::IntakeOperator Class Reference . . . . .	564
5.123.1 Detailed Description . . . . .	565
5.123.2 Member Enumeration Documentation . . . . .	566
5.123.2.1 EToggleState . . . . .	566
5.123.3 Constructor & Destructor Documentation . . . . .	566
5.123.3.1 IntakeOperator() . . . . .	566
5.123.4 Member Function Documentation . . . . .	566
5.123.4.1 updateIntakeVoltage() . . . . .	566
5.123.4.2 updateToggleVoltage() . . . . .	567
5.123.4.3 updateSingleToggle() . . . . .	567
5.123.4.4 updateSplitHold() . . . . .	567
5.123.4.5 updateSplitToggle() . . . . .	568
5.123.4.6 setIntakeVoltage() . . . . .	568
5.123.5 Member Data Documentation . . . . .	568
5.123.5.1 INTAKE_SUBSYSTEM_NAME . . . . .	568
5.123.5.2 SET_VOLTAGE_COMMAND . . . . .	569
5.123.5.3 VOLTAGE_SETTING . . . . .	569
5.123.5.4 m_controller . . . . .	569
5.123.5.5 m_robot . . . . .	569
5.123.5.6 toggle_state . . . . .	569
5.124 wisco::user::loader::LoaderOperator Class Reference . . . . .	570
5.124.1 Detailed Description . . . . .	571
5.124.2 Member Enumeration Documentation . . . . .	571
5.124.2.1 EToggleState . . . . .	571

---

5.124.3 Constructor & Destructor Documentation . . . . .	571
5.124.3.1 LoaderOperator() . . . . .	571
5.124.4 Member Function Documentation . . . . .	572
5.124.4.1 doLoad() . . . . .	572
5.124.4.2 doReady() . . . . .	572
5.124.4.3 isLoaded() . . . . .	572
5.124.4.4 isReady() . . . . .	573
5.124.4.5 updateHold() . . . . .	573
5.124.4.6 updateMacro() . . . . .	573
5.124.4.7 updateSingleToggle() . . . . .	574
5.124.4.8 updateSplitToggle() . . . . .	574
5.124.4.9 setLoaderPosition() . . . . .	575
5.124.5 Member Data Documentation . . . . .	575
5.124.5.1 SUBSYSTEM_NAME . . . . .	575
5.124.5.2 DO_LOAD_COMMAND_NAME . . . . .	575
5.124.5.3 DO_READY_COMMAND_NAME . . . . .	576
5.124.5.4 IS_LOADED_STATE_NAME . . . . .	576
5.124.5.5 IS_READY_STATE_NAME . . . . .	576
5.124.5.6 m_controller . . . . .	576
5.124.5.7 m_robot . . . . .	576
5.124.5.8 toggle_state . . . . .	577
5.125 wisco::user::umbrella::UmbrellaOperator Class Reference . . . . .	577
5.125.1 Detailed Description . . . . .	578
5.125.2 Member Enumeration Documentation . . . . .	578
5.125.2.1 EToggleState . . . . .	578
5.125.3 Constructor & Destructor Documentation . . . . .	578
5.125.3.1 UmbrellaOperator() . . . . .	578
5.125.4 Member Function Documentation . . . . .	579
5.125.4.1 setIn() . . . . .	579
5.125.4.2 setOut() . . . . .	579
5.125.4.3 updateHold() . . . . .	579
5.125.4.4 updateSplit() . . . . .	579
5.125.4.5 updateToggle() . . . . .	580
5.125.4.6 setUmbrellaPosition() . . . . .	580
5.125.5 Member Data Documentation . . . . .	581
5.125.5.1 UMBRELLA_SUBSYSTEM_NAME . . . . .	581
5.125.5.2 SET_IN_COMMAND . . . . .	581
5.125.5.3 SET_OUT_COMMAND . . . . .	581
5.125.5.4 m_controller . . . . .	581
5.125.5.5 m_robot . . . . .	582
5.125.5.6 toggle_state . . . . .	582
5.126 wisco::user::wings::WingsOperator Class Reference . . . . .	582

---

5.126.1 Detailed Description . . . . .	583
5.126.2 Member Enumeration Documentation . . . . .	583
5.126.2.1 EToggleState . . . . .	583
5.126.3 Constructor & Destructor Documentation . . . . .	583
5.126.3.1 WingsOperator() . . . . .	583
5.126.4 Member Function Documentation . . . . .	584
5.126.4.1 setLeftWing() . . . . .	584
5.126.4.2 setRightWing() . . . . .	584
5.126.4.3 setWings() . . . . .	584
5.126.4.4 updateDualHold() . . . . .	585
5.126.4.5 updateDualSplit() . . . . .	585
5.126.4.6 updateDualToggle() . . . . .	586
5.126.4.7 updateSingleHold() . . . . .	586
5.126.4.8 updateSingleSplit() . . . . .	586
5.126.4.9 updateSingleToggle() . . . . .	587
5.126.4.10 setWingsPosition() . . . . .	588
5.126.5 Member Data Documentation . . . . .	588
5.126.5.1 WINGS_SUBSYSTEM_NAME . . . . .	588
5.126.5.2 SET_LEFT_WING_COMMAND . . . . .	589
5.126.5.3 SET_RIGHT_WING_COMMAND . . . . .	589
5.126.5.4 m_controller . . . . .	589
5.126.5.5 m_robot . . . . .	589
5.126.5.6 toggle_state . . . . .	589
5.126.5.7 left_toggle_state . . . . .	590
5.126.5.8 right_toggle_state . . . . .	590
<b>Index</b> . . . . .	<b>591</b>



# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<code>pros_adapters</code>	Namespace for adapters from the pros library to the wisco library . . . . .	13
<code>wisco</code>	Namespace for all library code . . . . .	14
<code>wisco::alliances</code>	Namespace for all alliances . . . . .	15
<code>wisco::autons</code>	Namespace for autonomous routines . . . . .	15
<code>wisco::configs</code>	Namespace for hardware configurations . . . . .	16
<code>wisco::control</code>	Namespace for control algorithms . . . . .	16
<code>wisco::control::boomerang</code>	Namespace for boomerang controller components . . . . .	17
<code>wisco::control::motion</code>	Namespace for basic motion controls . . . . .	17
<code>wisco::control::path</code>	Namespace for path components . . . . .	18
<code>wisco::hal</code>	The namespace for the hardware abstraction layer . . . . .	19
<code>wisco::io</code>	Namespace for the io types . . . . .	20
<code>wisco::menu</code>	Interface for the menu system . . . . .	21
<code>wisco::profiles</code>	Namespace for the available driver profiles . . . . .	23
<code>wisco::robot</code>	The namespace that holds all robot classes . . . . .	23
<code>wisco::robot::subsystems</code>	Namespace for all robot subsystems . . . . .	24
<code>wisco::robot::subsystems::drive</code>	Namespace for drive classes . . . . .	25
<code>wisco::robot::subsystems::elevator</code>	Namespace for elevator classes . . . . .	25
<code>wisco::robot::subsystems::hang</code>	Namespace for hang classes . . . . .	26

wisco::robot::subsystems::intake	Namespace for intake classes . . . . .	27
wisco::robot::subsystems::loader	Namespace for all loader subsystem classes . . . . .	27
wisco::robot::subsystems::position	Namespace for all position subsystem classes . . . . .	28
wisco::robot::subsystems::umbrella	Namespace for umbrella classes . . . . .	29
wisco::robot::subsystems::wings	Namespace for all wings subsystem classes . . . . .	29
wisco::rtos	Namespace for the rtos interface of the library . . . . .	30
wisco::testing	Namespace for all testing functions . . . . .	30
wisco::testing::pros_testing	Namespace for pros-based testing functions . . . . .	31
wisco::user	Namespace for all user interactive components . . . . .	31
wisco::user::drive	Namespace for all drive user control components . . . . .	35
wisco::user::elevator	Namespace for all elevator user control components . . . . .	35
wisco::user::hang	Namespace for all hang user control components . . . . .	36
wisco::user::intake	Namespace for all intake user control components . . . . .	37
wisco::user::loader	Namespace for all loader user control components . . . . .	38
wisco::user::umbrella	Namespace for all umbrella user control components . . . . .	39
wisco::user::wings	Namespace for all wings user control components . . . . .	40

# Chapter 2

## Hierarchical Index

### 2.1 Class Hierarchy

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

MatchControllerFactory . . . . .	41
wisco::AutonomousManager . . . . .	89
wisco::control::AControl . . . . .	144
wisco::control::boomerang::BoomerangControl . . . . .	148
wisco::control::motion::MotionControl . . . . .	179
wisco::control::boomerang::IBoomerang . . . . .	153
wisco::control::boomerang::PIDBoomerang . . . . .	155
wisco::control::boomerang::PIDBoomerangBuilder . . . . .	168
wisco::control::ControlSystem . . . . .	173
wisco::control::motion::ITurn . . . . .	176
wisco::control::motion::PIDTurn . . . . .	184
wisco::control::motion::PIDTurnBuilder . . . . .	198
wisco::control::path::Point . . . . .	203
wisco::control::path::QuinticBezier . . . . .	210
wisco::control::path::QuinticBezierSpline . . . . .	214
wisco::control::PID . . . . .	216
wisco::hal::MotorGroup . . . . .	225
wisco::hal::PistonGroup . . . . .	229
wisco::IAlliance . . . . .	235
wisco::alliances::BlueAlliance . . . . .	85
wisco::alliances::RedAlliance . . . . .	86
wisco::alliances::SkillsAlliance . . . . .	87
wisco::IAutonomous . . . . .	236
wisco::autons::BlueMatchAuton . . . . .	91
wisco::autons::BlueSkillsAuton . . . . .	102
wisco::autons::OrangeMatchAuton . . . . .	105
wisco::autons::OrangeSkillsAuton . . . . .	107
wisco::IConfiguration . . . . .	238
wisco::configs::BlueConfiguration . . . . .	109
wisco::configs::OrangeConfiguration . . . . .	141
wisco::IMenu . . . . .	239
wisco::menu::MenuAdapter . . . . .	271
wisco::io::IBooleanSensor . . . . .	242

wisco::hal::DistanceBooleanSensor . . . . .	221
wisco::io::IDistanceSensor . . . . .	243
pros_adapters::ProsDistance . . . . .	54
wisco::io::IDistanceTrackingSensor . . . . .	244
wisco::hal::TrackingWheel . . . . .	232
wisco::io::IHeadingSensor . . . . .	246
pros_adapters::ProsHeading . . . . .	62
wisco::io::IMotor . . . . .	248
pros_adapters::ProsEXPMotor . . . . .	57
pros_adapters::ProsV5Motor . . . . .	78
wisco::io::IPiston . . . . .	251
pros_adapters::ProsPiston . . . . .	68
wisco::io::IRotationSensor . . . . .	253
pros_adapters::ProsRotation . . . . .	72
wisco::IProfile . . . . .	255
wisco::profiles::HenryProfile . . . . .	282
wisco::profiles::JohnProfile . . . . .	287
wisco::MatchController . . . . .	257
wisco::menu::LvglMenu . . . . .	261
wisco::menu::Option . . . . .	277
wisco::OPControlManager . . . . .	278
wisco::robot::ASubsystem . . . . .	291
wisco::robot::subsystems::drive::DifferentialDriveSubsystem . . . . .	302
wisco::robot::subsystems::elevator::ElevatorSubsystem . . . . .	347
wisco::robot::subsystems::hang::HangSubsystem . . . . .	367
wisco::robot::subsystems::intake::IntakeSubsystem . . . . .	408
wisco::robot::subsystems::loader::LoaderSubsystem . . . . .	430
wisco::robot::subsystems::position::PositionSubsystem . . . . .	487
wisco::robot::subsystems::umbrella::UmbrellaSubsystem . . . . .	500
wisco::robot::subsystems::wings::WingsSubsystem . . . . .	514
wisco::robot::Robot . . . . .	296
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder . . . . .	313
wisco::robot::subsystems::drive::IDifferentialDrive . . . . .	318
wisco::robot::subsystems::drive::DirectDifferentialDrive . . . . .	307
wisco::robot::subsystems::drive::KinematicDifferentialDrive . . . . .	322
wisco::robot::subsystems::drive::IVelocityProfile . . . . .	320
wisco::robot::subsystems::drive::CurveVelocityProfile . . . . .	299
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder . . . . .	336
wisco::robot::subsystems::drive::Velocity . . . . .	346
wisco::robot::subsystems::elevator::IElevator . . . . .	351
wisco::robot::subsystems::elevator::PIDElevator . . . . .	353
wisco::robot::subsystems::elevator::PIDElevatorBuilder . . . . .	361
wisco::robot::subsystems::hang::IClaw . . . . .	375
wisco::robot::subsystems::hang::PistonClaw . . . . .	381
wisco::robot::subsystems::hang::IToggleArm . . . . .	377
wisco::robot::subsystems::hang::PistonToggleArm . . . . .	388
wisco::robot::subsystems::hang::IWinch . . . . .	379
wisco::robot::subsystems::hang::PistonWinch . . . . .	394
wisco::robot::subsystems::hang::PistonClawBuilder . . . . .	385
wisco::robot::subsystems::hang::PistonToggleArmBuilder . . . . .	391
wisco::robot::subsystems::hang::PistonWinchBuilder . . . . .	397
wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder . . . . .	402
wisco::robot::subsystems::intake::IBallDetector . . . . .	404
wisco::robot::subsystems::intake::DistanceVisionBallDetector . . . . .	400

wisco::robot::subsystems::intake::IIntake . . . . .	406
wisco::robot::subsystems::intake::PIDIntake . . . . .	413
wisco::robot::subsystems::intake::PIDIntakeBuilder . . . . .	422
wisco::robot::subsystems::loader::ILoader . . . . .	428
wisco::robot::subsystems::loader::PIDLoader . . . . .	435
wisco::robot::subsystems::loader::PIDLoaderBuilder . . . . .	445
wisco::robot::subsystems::position::DistancePositionResetterBuilder . . . . .	458
wisco::robot::subsystems::position::InertialOdometryBuilder . . . . .	474
wisco::robot::subsystems::position::IPositionResetter . . . . .	482
wisco::robot::subsystems::position::DistancePositionResetter . . . . .	452
wisco::robot::subsystems::position::IPositionTracker . . . . .	484
wisco::robot::subsystems::position::InertialOdometry . . . . .	462
wisco::robot::subsystems::position::Position . . . . .	485
wisco::robot::subsystems::umbrella::IUmbrella . . . . .	492
wisco::robot::subsystems::umbrella::PistonUmbrella . . . . .	494
wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder . . . . .	497
wisco::robot::subsystems::wings::IWings . . . . .	504
wisco::robot::subsystems::wings::PistonWings . . . . .	507
wisco::robot::subsystems::wings::PistonWingsBuilder . . . . .	511
wisco::rtos::IClock . . . . .	519
pros_adapters::ProsClock . . . . .	42
wisco::rtos::IDelay . . . . .	520
pros_adapters::ProsDelay . . . . .	52
wisco::rtos::IMutex . . . . .	521
pros_adapters::ProsMutex . . . . .	66
wisco::rtos::ITask . . . . .	523
pros_adapters::ProsTask . . . . .	75
wisco::SystemConfiguration . . . . .	524
wisco::testing::pros_testing::DriveTest . . . . .	526
wisco::testing::TestFactory . . . . .	532
wisco::user::drive::DifferentialDriveOperator . . . . .	534
wisco::user::elevator::ElevatorOperator . . . . .	539
wisco::user::hang::HangOperator . . . . .	552
wisco::user::IController . . . . .	562
pros_adapters::ProsController . . . . .	44
wisco::user::intake::IntakeOperator . . . . .	564
wisco::user::loader::LoaderOperator . . . . .	570
wisco::user::umbrella::UmbrellaOperator . . . . .	577
wisco::user::wings::WingsOperator . . . . .	582



# Chapter 3

## Class Index

### 3.1 Class List

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

<a href="#">MatchControllerFactory</a>	
Class to create match controllers . . . . .	41
<a href="#">pros_adapters::ProsClock</a>	
Pros rtos clock adapter for the wisco rtos IClock interface . . . . .	42
<a href="#">pros_adapters::ProsController</a>	
Pros controller adapter for the wisco user IController interface . . . . .	44
<a href="#">pros_adapters::ProsDelayer</a>	
Pros rtos delay adapter for the wisco rtos IDelayer interface . . . . .	52
<a href="#">pros_adapters::ProsDistance</a>	
Pros distance sensor adapter for the wisco IDistanceSensor interface . . . . .	54
<a href="#">pros_adapters::ProsEXPMotor</a>	
Pros exp smart motor adapter for the wisco IMotor interface . . . . .	57
<a href="#">pros_adapters::ProsHeading</a>	
Pros inertial sensor adapter for the wisco IIHeadingSensor interface . . . . .	62
<a href="#">pros_adapters::ProsMutex</a>	
Pros rtos mutex adapter for the wisco rtos IMutex interface . . . . .	66
<a href="#">pros_adapters::ProsPiston</a>	
Pros piston adapter for the wisco IPiston interface . . . . .	68
<a href="#">pros_adapters::ProsRotation</a>	
Pros rotation sensor adapter for the wisco IRotationSensor interface . . . . .	72
<a href="#">pros_adapters::ProsTask</a>	
Pros rtos task adapter for the wisco rtos ITask interface . . . . .	75
<a href="#">pros_adapters::ProsV5Motor</a>	
Pros v5 smart motor adapter for the wisco IMotor interface . . . . .	78
<a href="#">wisco::alliances::BlueAlliance</a>	
The blue match alliance . . . . .	85
<a href="#">wisco::alliances::RedAlliance</a>	
The red match alliance . . . . .	86
<a href="#">wisco::alliances::SkillsAlliance</a>	
The skills alliance . . . . .	87
<a href="#">wisco::AutonomousManager</a>	
Manages the execution of the autonomous routine . . . . .	89
<a href="#">wisco::autons::BlueMatchAuton</a>	
The auton for the blue robot in matches . . . . .	91
<a href="#">wisco::autons::BlueSkillsAuton</a>	
The auton for the blue robot in skills . . . . .	102

wisco::autons::OrangeMatchAuton	The auton for the orange robot in matches . . . . .	105
wisco::autons::OrangeSkillsAuton	The auton for the orange robot in skills . . . . .	107
wisco::configs::BlueConfiguration	The hardware configuration of the blue robot . . . . .	109
wisco::configs::OrangeConfiguration	The hardware configuration of the orange robot . . . . .	141
wisco::control::AControl	An abstract class for control algorithms . . . . .	144
wisco::control::boomerang::BoomerangControl	Control adapter for the boomerang controller . . . . .	148
wisco::control::boomerang::IBoomerang	Interface for boomerang controllers . . . . .	153
wisco::control::boomerang::PIDBoomerang	Boomerang controller for the robot drivetrain . . . . .	155
wisco::control::boomerang::PIDBoomerangBuilder	Builder class for PID boomerang controllers . . . . .	168
wisco::control::ControlSystem	A container class for controls . . . . .	173
wisco::control::motion::ITurn	Interface for turn motion controls . . . . .	176
wisco::control::motion::MotionControl	Control adapter for the motion controller . . . . .	179
wisco::control::motion::PIDTurn	Interface for turn motion controls . . . . .	184
wisco::control::motion::PIDTurnBuilder	Builder class for PID turn controllers . . . . .	198
wisco::control::path::Point	Class for a point with an x and y coordinate . . . . .	203
wisco::control::path::QuinticBezier	A quintic bezier with a point function . . . . .	210
wisco::control::path::QuinticBezierSpline	A spline of quintic beziers . . . . .	214
wisco::control::PID	A general-purpose PID controller . . . . .	216
wisco::hal::DistanceBooleanSensor	A distance sensor used to create boolean outputs . . . . .	221
wisco::hal::MotorGroup	A group of motors on the same connected output SHOULD ONLY BE USED WITH IDENTICAL MOTORS . . . . .	225
wisco::hal::PistonGroup	A group of pistons on the same connected output . . . . .	229
wisco::hal::TrackingWheel	A tracking wheel sensor . . . . .	232
wisco::IAlliance	Interface for the alliances for the robot . . . . .	235
wisco::IAutonomous	Interface for the autonomous routines in the system . . . . .	236
wisco::IConfiguration	Interface for the configurations in the system . . . . .	238
wisco::IMenu	Interface for the menu system . . . . .	239
wisco::io::IBooleanSensor	Interface for sensors that generate a boolean value . . . . .	242
wisco::io::IDistanceSensor	Interface for distance tracking sensors . . . . .	243

wisco::io::IDistanceTrackingSensor	Interface for distance tracking sensors . . . . .	244
wisco::io::IHeadingSensor	Interface for heading sensors . . . . .	246
wisco::io::IMotor	Interface for electric motors controlled by voltage . . . . .	248
wisco::io::IPiston	Interface for a discrete state-based piston . . . . .	251
wisco::io::IRotationSensor	Interface for rotation sensors . . . . .	253
wisco::IProfile	Interface for the profiles in the system . . . . .	255
wisco::MatchController	Handles the field controller inputs during a match . . . . .	257
wisco::menu::LvglMenu	Controls an lvgl-based menu selection system . . . . .	261
wisco::menu::MenuAdapter	This class adapts the menu system to the <a href="#">IMenu</a> interface . . . . .	271
wisco::menu::Option	An option in the menu system . . . . .	277
wisco::OPControlManager	Manages the execution of the operator control . . . . .	278
wisco::profiles::HenryProfile	Driver profile for Henry . . . . .	282
wisco::profiles::JohnProfile	Driver profile for John . . . . .	287
wisco::robot::ASubsystem	An abstract class for robot subsystems . . . . .	291
wisco::robot::Robot	A container class for subsystems . . . . .	296
wisco::robot::subsystems::drive::CurveVelocityProfile	An s-curve velocity profile for the drive . . . . .	299
wisco::robot::subsystems::drive::DifferentialDriveSubsystem	The subsystem adapter for differential drives . . . . .	302
wisco::robot::subsystems::drive::DirectDifferentialDrive	A direct drive controller with independent left and right wheelsets . . . . .	307
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder	Builder class for the direct differential drive class . . . . .	313
wisco::robot::subsystems::drive::IDifferentialDrive	Interface for differential drivetrains . . . . .	318
wisco::robot::subsystems::drive::IVelocityProfile	Interface for drive velocity profiles . . . . .	320
wisco::robot::subsystems::drive::KinematicDifferentialDrive	A kinematic drive controller with independent left and right wheelsets . . . . .	322
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder	Builder class for the kinematic differential drive class . . . . .	336
wisco::robot::subsystems::drive::Velocity	Holds the velocity values for the drive . . . . .	346
wisco::robot::subsystems::elevator::ElevatorSubsystem	The subsystem adapter for elevators . . . . .	347
wisco::robot::subsystems::elevator::IElevator	Interface for elevators . . . . .	351
wisco::robot::subsystems::elevator::PIDElevator	An elevator controller with PID position control . . . . .	353
wisco::robot::subsystems::elevator::PIDElevatorBuilder	Builder class for a pid-based elevator system . . . . .	361
wisco::robot::subsystems::hang::HangSubsystem	The subsystem adapter for hangs . . . . .	367

wisco::robot::subsystems::hang::IClaw	Interface for claws with an open and closed position . . . . .	375
wisco::robot::subsystems::hang::IToggleArm	Interface for arms with an up and down position . . . . .	377
wisco::robot::subsystems::hang::IWinch	Interface for winches with an engaged and disengaged position . . . . .	379
wisco::robot::subsystems::hang::PistonClaw	A claw controlled using a piston . . . . .	381
wisco::robot::subsystems::hang::PistonClawBuilder	Builder for a piston-based claw . . . . .	385
wisco::robot::subsystems::hang::PistonToggleArm	Interface for arms with an up and down position . . . . .	388
wisco::robot::subsystems::hang::PistonToggleArmBuilder	Interface for arms with an up and down position . . . . .	391
wisco::robot::subsystems::hang::PistonWinch	A winch controlled using a piston . . . . .	394
wisco::robot::subsystems::hang::PistonWinchBuilder	Builder for a piston-based winch . . . . .	397
wisco::robot::subsystems::intake::DistanceVisionBallDetector	Ball detection system that uses a distance and vision sensor . . . . .	400
wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder	Ball detection system that uses a distance and vision sensor . . . . .	402
wisco::robot::subsystems::intake::IBallDetector	Interface for ball detection system . . . . .	404
wisco::robot::subsystems::intake::IIntake	Interface for intakes . . . . .	406
wisco::robot::subsystems::intake::IntakeSubsystem	The subsystem adapter for intakes . . . . .	408
wisco::robot::subsystems::intake::PIDIntake	An intake controller with PID velocity control . . . . .	413
wisco::robot::subsystems::intake::PIDIntakeBuilder	A builder class for a PID-based intake subsystem . . . . .	422
wisco::robot::subsystems::loader::ILoader	Interface for loader subsystems . . . . .	428
wisco::robot::subsystems::loader::LoaderSubsystem	The subsystem adapter for loaders . . . . .	430
wisco::robot::subsystems::loader::PIDLoader	An loader controller with PID position control . . . . .	435
wisco::robot::subsystems::loader::PIDLoaderBuilder	Builder class for a pid-based loader system . . . . .	445
wisco::robot::subsystems::position::DistancePositionResetter	Interface for position resetting subsystems . . . . .	452
wisco::robot::subsystems::position::DistancePositionResetterBuilder	Builder for distance position resetters . . . . .	458
wisco::robot::subsystems::position::InertialOdometry	An odometry system based on a heading sensor with two distance tracking sensors . . . . .	462
wisco::robot::subsystems::position::InertialOdometryBuilder	Builder class for the inertial odometry class . . . . .	474
wisco::robot::subsystems::position::IPositionResetter	Interface for position resetting subsystems . . . . .	482
wisco::robot::subsystems::position::IPositionTracker	Interface for position tracking subsystems . . . . .	484
wisco::robot::subsystems::position::Position	Holds a robot position . . . . .	485
wisco::robot::subsystems::position::PositionSubsystem	Adapter from a position tracker to a robot subsystem . . . . .	487
wisco::robot::subsystems::umbrella::IUmbrella	Interface for umbrellas with an out and in position . . . . .	492

wisco::robot::subsystems::umbrella::PistonUmbrella	A umbrella controlled using a piston . . . . .	494
wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder	Builder class for piston umbrellas . . . . .	497
wisco::robot::subsystems::umbrella::UmbrellaSubsystem	The subsystem adapter for umbrellas . . . . .	500
wisco::robot::subsystems::wings::IWings	Interface for wings subsystems . . . . .	504
wisco::robot::subsystems::wings::PistonWings	Wings controlled using pistons . . . . .	507
wisco::robot::subsystems::wings::PistonWingsBuilder	Builder class for the piston wings class . . . . .	511
wisco::robot::subsystems::wings::WingsSubsystem	The subsystem adapter for wings . . . . .	514
wisco::rtos::IClock	Interface for an rtos system clock . . . . .	519
wisco::rtos::IDelayer	Interface for rtos delay systems . . . . .	520
wisco::rtos::IMutex	Interface for rtos mutexes . . . . .	521
wisco::rtos::ITask	Interface for an rtos task system . . . . .	523
wisco::SystemConfiguration	Holds the system configuration information . . . . .	524
wisco::testing::pros_testing::DriveTest	Tests a pros-based drive . . . . .	526
wisco::testing::TestFactory	Factory to build test classes . . . . .	532
wisco::user::drive::DifferentialDriveOperator	Runs the operator-controlled differential drive voltage settings . . . . .	534
wisco::user::elevator::ElevatorOperator	Runs the operator-controlled elevator position settings . . . . .	539
wisco::user::hang::HangOperator	Runs the operator-controlled hang settings . . . . .	552
wisco::user::IController	Interface for a controller . . . . .	562
wisco::user::intake::IntakeOperator	Runs the operator-controlled intake voltage settings . . . . .	564
wisco::user::loader::LoaderOperator	Runs the operator-controlled loader settings . . . . .	570
wisco::user::umbrella::UmbrellaOperator	Runs the operator-controlled umbrella settings . . . . .	577
wisco::user::wings::WingsOperator	Runs the operator-controlled wings settings . . . . .	582



# Chapter 4

## Namespace Documentation

### 4.1 pros\_adapters Namespace Reference

Namespace for adapters from the pros library to the wisco library.

#### Classes

- class [ProsClock](#)  
*Pros rtos clock adapter for the wisco rtos IClock interface.*
- class [ProsController](#)  
*Pros controller adapter for the wisco user IController interface.*
- class [ProsDelayer](#)  
*Pros rtos delay adapter for the wisco rtos IDelayer interface.*
- class [ProsDistance](#)  
*Pros distance sensor adapter for the wisco IDistanceSensor interface.*
- class [ProsEXPMotor](#)  
*Pros exp smart motor adapter for the wisco IMotor interface.*
- class [ProsHeading](#)  
*Pros inertial sensor adapter for the wisco IHeadingSensor interface.*
- class [ProsMutex](#)  
*Pros rtos mutex adapter for the wisco rtos IMutex interface.*
- class [ProsPiston](#)  
*Pros piston adapter for the wisco IPiston interface.*
- class [ProsRotation](#)  
*Pros rotation sensor adapter for the wisco IRotationSensor interface.*
- class [ProsTask](#)  
*Pros rtos task adapter for the wisco rtos ITask interface.*
- class [ProsV5Motor](#)  
*Pros v5 smart motor adapter for the wisco IMotor interface.*

#### 4.1.1 Detailed Description

Namespace for adapters from the pros library to the wisco library.

#### Author

Nathan Sandvig

## 4.2 wisco Namespace Reference

Namespace for all library code.

### Namespaces

- namespace [alliances](#)  
*Namespace for all alliances.*
- namespace [autons](#)  
*Namespace for autonomous routines.*
- namespace [configs](#)  
*Namespace for hardware configurations.*
- namespace [control](#)  
*Namespace for control algorithms.*
- namespace [hal](#)  
*The namespace for the hardware abstraction layer.*
- namespace [io](#)  
*Namespace for the io types.*
- namespace [menu](#)  
*Interface for the menu system.*
- namespace [profiles](#)  
*Namespace for the available driver profiles.*
- namespace [robot](#)  
*The namespace that holds all robot classes.*
- namespace [rtos](#)  
*Namespace for the rtos interface of the library.*
- namespace [testing](#)  
*Namespace for all testing functions.*
- namespace [user](#)  
*Namespace for all user interactive components.*

### Classes

- class [AutonomousManager](#)  
*Manages the execution of the autonomous routine.*
- class [IAlliance](#)  
*Interface for the alliances for the robot.*
- class [IAutonomous](#)  
*Interface for the autonomous routines in the system.*
- class [IConfiguration](#)  
*Interface for the configurations in the system.*
- class [IMenu](#)  
*Interface for the menu system.*
- class [IProfile](#)  
*Interface for the profiles in the system.*
- class [MatchController](#)  
*Handles the field controller inputs during a match.*
- class [OPControlManager](#)  
*Manages the execution of the operator control.*
- struct [SystemConfiguration](#)  
*Holds the system configuration information.*

### 4.2.1 Detailed Description

Namespace for all library code.

Author

Nathan Sandvig

## 4.3 wisco::alliances Namespace Reference

Namespace for all alliances.

### Classes

- class [BlueAlliance](#)  
*The blue match alliance.*
- class [RedAlliance](#)  
*The red match alliance.*
- class [SkillsAlliance](#)  
*The skills alliance.*

### 4.3.1 Detailed Description

Namespace for all alliances.

Author

Nathan Sandvig

## 4.4 wisco::autons Namespace Reference

Namespace for autonomous routines.

### Classes

- class [BlueMatchAuton](#)  
*The auton for the blue robot in matches.*
- class [BlueSkillsAuton](#)  
*The auton for the blue robot in skills.*
- class [OrangeMatchAuton](#)  
*The auton for the orange robot in matches.*
- class [OrangeSkillsAuton](#)  
*The auton for the orange robot in skills.*

#### 4.4.1 Detailed Description

Namespace for autonomous routines.

##### Author

Nathan Sandvig

### 4.5 wisco::configs Namespace Reference

Namespace for hardware configurations.

#### Classes

- class [BlueConfiguration](#)  
*The hardware configuration of the blue robot.*
- class [OrangeConfiguration](#)  
*The hardware configuration of the orange robot.*

#### 4.5.1 Detailed Description

Namespace for hardware configurations.

##### Author

Nathan Sandvig

### 4.6 wisco::control Namespace Reference

Namespace for control algorithms.

#### Namespaces

- namespace [boomerang](#)  
*Namespace for boomerang controller components.*
- namespace [motion](#)  
*Namespace for basic motion controls.*
- namespace [path](#)  
*Namespace for path components.*

#### Classes

- class [AControl](#)  
*An abstract class for control algorithms.*
- class [ControlSystem](#)  
*A container class for controls.*
- class [PID](#)  
*A general-purpose PID controller.*

### 4.6.1 Detailed Description

Namespace for control algorithms.

#### Author

Nathan Sandvig

## 4.7 wisco::control::boomerang Namespace Reference

Namespace for boomerang controller components.

#### Classes

- class [BoomerangControl](#)  
*Control adapter for the boomerang controller.*
- class [IBoomerang](#)  
*Interface for boomerang controllers.*
- class [PIDBoomerang](#)  
*Boomerang controller for the robot drivetrain.*
- class [PIDBoomerangBuilder](#)  
*Builder class for PID boomerang controllers.*

### 4.7.1 Detailed Description

Namespace for boomerang controller components.

#### Author

Nathan Sandvig

## 4.8 wisco::control::motion Namespace Reference

Namespace for basic motion controls.

#### Classes

- class [ITurn](#)  
*Interface for turn motion controls.*
- class [MotionControl](#)  
*Control adapter for the motion controller.*
- class [PIDTurn](#)  
*Interface for turn motion controls.*
- class [PIDTurnBuilder](#)  
*Builder class for PID turn controllers.*

## Enumerations

- enum class [ETurnDirection](#) { **AUTO** , **CLOCKWISE** , **COUNTERCLOCKWISE** }
- Enum for turn directions.*

### 4.8.1 Detailed Description

Namespace for basic motion controls.

#### Author

Nathan Sandvig

### 4.8.2 Enumeration Type Documentation

#### 4.8.2.1 ETurnDirection

```
enum class wisco::control::motion::ETurnDirection [strong]
```

Enum for turn directions.

#### Author

Nathan Sandvig

Definition at line 33 of file [ETurnDirection.hpp](#).

```
00034 {
00035     AUTO,
00036     CLOCKWISE,
00037     COUNTERCLOCKWISE
00038 };
```

## 4.9 wisco::control::path Namespace Reference

Namespace for path components.

## Classes

- class [Point](#)  
*Class for a point with an x and y coordinate.*
- class [QuinticBezier](#)  
*A quintic bezier with a point function.*
- class [QuinticBezierSpline](#)  
*A spline of quintic beziers.*

## Functions

- [Point operator\\*](#) (double lhs, const [Point](#) &rhs)  
*Multiplication operator overload for doubles with points.*

### 4.9.1 Detailed Description

Namespace for path components.

#### Author

Nathan Sandvig

### 4.9.2 Function Documentation

#### 4.9.2.1 operator\*()

```
Point wisco::control::path::operator* (
    double lhs,
    const Point & rhs )
```

Multiplication operator overload for doubles with points.

#### Parameters

<i>lhs</i>	The multiplier on the left hand side of the operator
<i>rhs</i>	The point on the right hand side of the operator

#### Returns

[Point](#) A new point with the point coordinates multiplied by the multiplier

Definition at line 82 of file [Point.cpp](#).

```
00083 {
00084     return Point{lhs * rhs.getX(), lhs * rhs.getY()};
00085 }
```

## 4.10 wisco::hal Namespace Reference

The namespace for the hardware abstraction layer.

#### Classes

- class [DistanceBooleanSensor](#)  
*A distance sensor used to create boolean outputs.*
- class [MotorGroup](#)  
*A group of motors on the same connected output SHOULD ONLY BE USED WITH IDENTICAL MOTORS.*
- class [PistonGroup](#)  
*A group of pistons on the same connected output.*
- class [TrackingWheel](#)  
*A tracking wheel sensor.*

## Enumerations

- enum class [DistanceBooleanMode](#) { **ABOVE\_THRESHOLD** , **BELOW\_THRESHOLD** , **BETWEEN\_THRESHOLD** }

*The modes of a distance boolean sensor.*

### 4.10.1 Detailed Description

The namespace for the hardware abstraction layer.

#### Author

Nathan Sandvig

### 4.10.2 Enumeration Type Documentation

#### 4.10.2.1 DistanceBooleanMode

```
enum class wisco::hal::DistanceBooleanMode [strong]
```

The modes of a distance boolean sensor.

#### Author

Nathan Sandvig

Definition at line 25 of file [DistanceBooleanMode.hpp](#).

```
00026 {
00027     ABOVE_THRESHOLD,
00028     BELOW_THRESHOLD,
00029     BETWEEN_THRESHOLD
00030 };
```

## 4.11 wisco::io Namespace Reference

Namespace for the io types.

### Classes

- class [IBooleanSensor](#)  
*Interface for sensors that generate a boolean value.*
- class [IDistanceSensor](#)  
*Interface for distance tracking sensors.*
- class [IDistanceTrackingSensor](#)  
*Interface for distance tracking sensors.*
- class [IHeadingSensor](#)  
*Interface for heading sensors.*
- class [IMotor](#)  
*Interface for electric motors controlled by voltage.*
- class [IPiston](#)  
*Interface for a discrete state-based piston.*
- class [IRotationSensor](#)  
*Interface for rotation sensors.*

### 4.11.1 Detailed Description

Namespace for the io types.

Author

Nathan Sandvig

## 4.12 wisco::menu Namespace Reference

Interface for the menu system.

### Classes

- class [LvglMenu](#)  
*Controls an lvgl-based menu selection system.*
- class [MenuAdapter](#)  
*This class adapts the menu system to the [IMenu](#) interface.*
- struct [Option](#)  
*An option in the menu system.*

### Functions

- void [startButtonEventHandler](#) (lv\_event\_t \*event)  
*Event handler function for the start button.*
- void [settingsButtonEventHandler](#) (lv\_event\_t \*event)  
*Event handler function for the settings button.*
- void [settingsBackButtonEventHandler](#) (lv\_event\_t \*event)  
*Event handler function for the back button in the settings menu.*
- void [settingsButtonMatrixEventHandler](#) (lv\_event\_t \*event)  
*Event handler function for the button matrices in settings.*

### 4.12.1 Detailed Description

Interface for the menu system.

Program data related to the menu system.

Author

Nathan Sandvig

### 4.12.2 Function Documentation

#### 4.12.2.1 [startButtonEventHandler\(\)](#)

```
void wisco::menu::startButtonEventHandler (
    lv_event_t * event ) [extern]
```

Event handler function for the start button.

**Parameters**

<i>event</i>	The event data
--------------	----------------

Definition at line 16 of file LvglMenu.cpp.

```
00017 {
00018     void** user_data{static_cast<void**>(lv_event_get_user_data(event))};
00019     LvglMenu* lvgl_menu{static_cast<LvglMenu*>(user_data[0])};
00020
00021     lv_obj_clean(lv_scr_act());
00022     if (lvgl_menu)
00023     {
00024         lvgl_menu->writeConfiguration();
00025         lvgl_menu->setComplete();
00026     }
00027 }
```

**4.12.2.2 settingsButtonEventHandler()**

```
void wisco::menu::settingsButtonEventHandler (
    lv_event_t * event ) [extern]
```

Event handler function for the settings button.

**Parameters**

<i>event</i>	The event data
--------------	----------------

Definition at line 29 of file LvglMenu.cpp.

```
00030 {
00031     void** user_data{static_cast<void**>(lv_event_get_user_data(event))};
00032     LvglMenu* lvgl_menu{static_cast<LvglMenu*>(user_data[0])};
00033
00034     lv_obj_clean(lv_scr_act());
00035     if (lvgl_menu)
00036         lvgl_menu->drawSettingsMenu();
00037 }
```

**4.12.2.3 settingsBackButtonEventHandler()**

```
void wisco::menu::settingsBackButtonEventHandler (
    lv_event_t * event ) [extern]
```

Event handler function for the back button in the settings menu.

**Parameters**

<i>event</i>	The event data
--------------	----------------

Definition at line 39 of file LvglMenu.cpp.

```
00040 {
00041     lv_obj_t* obj{lv_event_get_target(event)};
00042     void** user_data{static_cast<void**>(lv_event_get_user_data(event))};
00043     lv_obj_t* menu{static_cast<lv_obj_t*>(user_data[0])};
00044     LvglMenu* lvgl_menu{static_cast<LvglMenu*>(user_data[1])};
00045
00046     if (obj == lv_menu_get_sidebar_header_back_btn(menu))
00047     {
00048         lv_obj_clean(lv_scr_act());
00049         if (lvgl_menu)
```

```
00050         lvgl_menu->drawMainMenu();  
00051     }  
00052 }
```

#### 4.12.2.4 settingsButtonMatrixEventHandler()

```
void wisco::menu::settingsButtonMatrixEventHandler (  
    lv_event_t * event ) [extern]
```

Event handler function for the button matrices in settings.

##### Parameters

<code>event</code>	The event data
--------------------	----------------

Definition at line 54 of file [LvglMenu.cpp](#).

```
00055 {  
00056     lv_obj_t* obj {lv_event_get_target(event)};  
00057     uint32_t button_id{lv_btnmatrix_get_selected_btn(obj)};  
00058     Option* option{static_cast<Option*>(lv_event_get_user_data(event))};  
00059     option->selected = button_id;  
00060 }
```

## 4.13 wisco::profiles Namespace Reference

Namespace for the available driver profiles.

### Classes

- class [HenryProfile](#)  
*Driver profile for Henry.*
- class [JohnProfile](#)  
*Driver profile for John.*

#### 4.13.1 Detailed Description

Namespace for the available driver profiles.

##### Author

Nathan Sandvig

## 4.14 wisco::robot Namespace Reference

The namespace that holds all robot classes.

## Namespaces

- namespace [subsystems](#)

*Namespace for all robot subsystems.*

## Classes

- class [ASubsystem](#)

*An abstract class for robot subsystems.*

- class [Robot](#)

*A container class for subsystems.*

### 4.14.1 Detailed Description

The namespace that holds all robot classes.

#### Author

Nathan Sandvig

## 4.15 `wisco::robot::subsystems` Namespace Reference

Namespace for all robot subsystems.

## Namespaces

- namespace [drive](#)

*Namespace for drive classes.*

- namespace [elevator](#)

*Namespace for elevator classes.*

- namespace [hang](#)

*Namespace for hang classes.*

- namespace [intake](#)

*Namespace for intake classes.*

- namespace [loader](#)

*Namespace for all loader subsystem classes.*

- namespace [position](#)

*Namespace for all position subsystem classes.*

- namespace [umbrella](#)

*Namespace for umbrella classes.*

- namespace [wings](#)

*Namespace for all wings subsystem classes.*

### 4.15.1 Detailed Description

Namespace for all robot subsystems.

#### Author

Nathan Sandvig

## 4.16 wisco::robot::subsystems::drive Namespace Reference

Namespace for drive classes.

#### Classes

- class [CurveVelocityProfile](#)  
*An s-curve velocity profile for the drive.*
- class [DifferentialDriveSubsystem](#)  
*The subsystem adapter for differential drives.*
- class [DirectDifferentialDrive](#)  
*A direct drive controller with independent left and right wheelsets.*
- class [DirectDifferentialDriveBuilder](#)  
*Builder class for the direct differential drive class.*
- class [IDifferentialDrive](#)  
*Interface for differential drivetrains.*
- class [IVelocityProfile](#)  
*Interface for drive velocity profiles.*
- class [KinematicDifferentialDrive](#)  
*A kinematic drive controller with independent left and right wheelsets.*
- class [KinematicDifferentialDriveBuilder](#)  
*Builder class for the kinematic differential drive class.*
- struct [Velocity](#)  
*Holds the velocity values for the drive.*

### 4.16.1 Detailed Description

Namespace for drive classes.

#### Author

Nathan Sandvig

## 4.17 wisco::robot::subsystems::elevator Namespace Reference

Namespace for elevator classes.

## Classes

- class [ElevatorSubsystem](#)  
*The subsystem adapter for elevators.*
- class [IElevator](#)  
*Interface for elevators.*
- class [PIDElevator](#)  
*An elevator controller with PID position control.*
- class [PIDElevatorBuilder](#)  
*Builder class for a pid-based elevator system.*

### 4.17.1 Detailed Description

Namespace for elevator classes.

#### Author

Nathan Sandvig

## 4.18 wisco::robot::subsystems::hang Namespace Reference

Namespace for hang classes.

## Classes

- class [HangSubsystem](#)  
*The subsystem adapter for hangs.*
- class [IClaw](#)  
*Interface for claws with an open and closed position.*
- class [IToggleArm](#)  
*Interface for arms with an up and down position.*
- class [IWinch](#)  
*Interface for winches with an engaged and disengaged position.*
- class [PistonClaw](#)  
*A claw controlled using a piston.*
- class [PistonClawBuilder](#)  
*Builder for a piston-based claw.*
- class [PistonToggleArm](#)  
*Interface for arms with an up and down position.*
- class [PistonToggleArmBuilder](#)  
*Interface for arms with an up and down position.*
- class [PistonWinch](#)  
*A winch controlled using a piston.*
- class [PistonWinchBuilder](#)  
*Builder for a piston-based winch.*

### 4.18.1 Detailed Description

Namespace for hang classes.

#### Author

Nathan Sandvig

## 4.19 wisco::robot::subsystems::intake Namespace Reference

Namespace for intake classes.

#### Classes

- class [DistanceVisionBallDetector](#)  
*Ball detection system that uses a distance and vision sensor.*
- class [DistanceVisionBallDetectorBuilder](#)  
*Ball detection system that uses a distance and vision sensor.*
- class [IBallDetector](#)  
*Interface for ball detection system.*
- class [IIntake](#)  
*Interface for intakes.*
- class [IntakeSubsystem](#)  
*The subsystem adapter for intakes.*
- class [PIDIntake](#)  
*An intake controller with PID velocity control.*
- class [PIDIntakeBuilder](#)  
*A builder class for a PID-based intake subsystem.*

### 4.19.1 Detailed Description

Namespace for intake classes.

#### Author

Nathan Sandvig

## 4.20 wisco::robot::subsystems::loader Namespace Reference

Namespace for all loader subsystem classes.

## Classes

- class [ILoader](#)  
*Interface for loader subsystems.*
- class [LoaderSubsystem](#)  
*The subsystem adapter for loaders.*
- class [PIDLoader](#)  
*An loader controller with PID position control.*
- class [PIDLoaderBuilder](#)  
*Builder class for a pid-based loader system.*

### 4.20.1 Detailed Description

Namespace for all loader subsystem classes.

Namespace for loader classes.

#### Author

Nathan Sandvig

## 4.21 wisco::robot::subsystems::position Namespace Reference

Namespace for all position subsystem classes.

## Classes

- class [DistancePositionResetter](#)  
*Interface for position resetting subsystems.*
- class [DistancePositionResetterBuilder](#)  
*Builder for distance position resetters.*
- class [InertialOdometry](#)  
*An odometry system based on a heading sensor with two distance tracking sensors.*
- class [InertialOdometryBuilder](#)  
*Builder class for the inertial odometry class.*
- class [IPositionResetter](#)  
*Interface for position resetting subsystems.*
- class [IPositionTracker](#)  
*Interface for position tracking subsystems.*
- struct [Position](#)  
*Holds a robot position.*
- class [PositionSubsystem](#)  
*Adapter from a position tracker to a robot subsystem.*

### 4.21.1 Detailed Description

Namespace for all position subsystem classes.

#### Author

Nathan Sandvig

## 4.22 wisco::robot::subsystems::umbrella Namespace Reference

Namespace for umbrella classes.

#### Classes

- class [IUmbrella](#)  
*Interface for umbrellas with an out and in position.*
- class [PistonUmbrella](#)  
*A umbrella controlled using a piston.*
- class [PistonUmbrellaBuilder](#)  
*Builder class for piston umbrellas.*
- class [UmbrellaSubsystem](#)  
*The subsystem adapter for umbrellas.*

### 4.22.1 Detailed Description

Namespace for umbrella classes.

#### Author

Nathan Sandvig

## 4.23 wisco::robot::subsystems::wings Namespace Reference

Namespace for all wings subsystem classes.

#### Classes

- class [IWings](#)  
*Interface for wings subsystems.*
- class [PistonWings](#)  
*Wings controlled using pistons.*
- class [PistonWingsBuilder](#)  
*Builder class for the piston wings class.*
- class [WingsSubsystem](#)  
*The subsystem adapter for wings.*

### 4.23.1 Detailed Description

Namespace for all wings subsystem classes.

#### Author

Nathan Sandvig

## 4.24 wisco::rtos Namespace Reference

Namespace for the rtos interface of the library.

#### Classes

- class [IClock](#)  
*Interface for an rtos system clock.*
- class [IDelayer](#)  
*Interface for rtos delay systems.*
- class [IMutex](#)  
*Interface for rtos mutexes.*
- class [ITask](#)  
*Interface for an rtos task system.*

### 4.24.1 Detailed Description

Namespace for the rtos interface of the library.

#### Author

Nathan Sandvig

## 4.25 wisco::testing Namespace Reference

Namespace for all testing functions.

#### Namespaces

- namespace [pros\\_testing](#)  
*Namespace for pros-based testing functions.*

#### Classes

- class [TestFactory](#)  
*Factory to build test classes.*

### 4.25.1 Detailed Description

Namespace for all testing functions.

#### Author

Nathan Sandvig

## 4.26 wisco::testing::pros\_testing Namespace Reference

Namespace for pros-based testing functions.

#### Classes

- class [DriveTest](#)  
*Tests a pros-based drive.*

#### Variables

- static constexpr char [FILE\\_PATH](#) [] {"/usd/testing/"}  
*The path for writing all pros testing files.*

### 4.26.1 Detailed Description

Namespace for pros-based testing functions.

#### Author

Nathan Sandvig

### 4.26.2 Variable Documentation

#### 4.26.2.1 FILE\_PATH

```
constexpr char wisco::testing::pros_testing::FILE_PATH[] {"/usd/testing/"} [static], [constexpr]
```

The path for writing all pros testing files.

Definition at line [41](#) of file [DriveTest.hpp](#).  
00041 {"/usd/testing/"};

## 4.27 wisco::user Namespace Reference

Namespace for all user interactive components.

## Namespaces

- namespace [drive](#)  
*Namespace for all drive user control components.*
- namespace [elevator](#)  
*Namespace for all elevator user control components.*
- namespace [hang](#)  
*Namespace for all hang user control components.*
- namespace [intake](#)  
*Namespace for all intake user control components.*
- namespace [loader](#)  
*Namespace for all loader user control components.*
- namespace [umbrella](#)  
*Namespace for all umbrella user control components.*
- namespace [wings](#)  
*Namespace for all wings user control components.*

## Classes

- class [IController](#)  
*Interface for a controller.*

## Enumerations

- enum class [EControl](#) {
   
**ELEVATOR\_IN** , **ELEVATOR\_FIELD** , **ELEVATOR\_MATCH\_LOAD** , **ELEVATOR\_POLE\_HANG** ,
 **ELEVATOR\_PARTNER\_HANG** , **ELEVATOR\_OUT** , **ELEVATOR\_TOGGLE** , **HANG\_GRAB** ,
 **HANG\_HANG** , **HANG\_INACTIVE** , **HANG\_NEXT** , **HANG\_PREVIOUS** ,
 **HANG\_RAISE** , **HANG\_RESET** , **HANG\_TOGGLE** , **INTAKE\_IN** ,
 **INTAKE\_OUT** , **INTAKE\_TOGGLE** , **LOADER\_HOLD** , **LOADER\_LOAD** ,
 **LOADER\_READY** , **LOADER\_TOGGLE** , **UMBRELLA\_HOLD** , **UMBRELLA\_IN** ,
 **UMBRELLA\_OUT** , **UMBRELLA\_TOGGLE** , **WINGS\_HOLD** , **WINGS\_IN** ,
 **WINGS\_OUT** , **WINGS\_TOGGLE** , **WINGS\_LEFT\_HOLD** , **WINGS\_LEFT\_IN** ,
 **WINGS\_LEFT\_OUT** , **WINGS\_LEFT\_TOGGLE** , **WINGS\_RIGHT\_HOLD** , **WINGS\_RIGHT\_IN** ,
 **WINGS\_RIGHT\_OUT** , **WINGS\_RIGHT\_TOGGLE** }
   
*Defines all different control inputs.*
- enum class [EControllerAnalog](#) {
   
**JOYSTICK\_LEFT\_X** , **JOYSTICK\_LEFT\_Y** , **JOYSTICK\_RIGHT\_X** , **JOYSTICK\_RIGHT\_Y** ,
 **TRIGGER\_LEFT** , **TRIGGER\_RIGHT** , **NONE** }
   
*Defines all different controller analog inputs.*
- enum class [EControllerDigital](#) {
   
**BUTTON\_A** , **BUTTON\_B** , **BUTTON\_X** , **BUTTON\_Y** ,
 **DPAD\_DOWN** , **DPAD\_LEFT** , **DPAD\_RIGHT** , **DPAD\_UP** ,
 **JOYSTICK\_LEFT** , **JOYSTICK\_RIGHT** , **SCUFF\_LEFT\_REAR** , **SCUFF\_LEFT\_UNDER** ,
 **SCUFF\_RIGHT\_REAR** , **SCUFF\_RIGHT\_UNDER** , **TRIGGER\_LEFT\_BOTTOM** , **TRIGGER\_LEFT\_TOP** ,
 **TRIGGER\_RIGHT\_BOTTOM** , **TRIGGER\_RIGHT\_TOP** , **NONE** }
   
*Defines all different controller digital inputs.*
- enum class [EControlType](#) {
   
**DRIVE** , **ELEVATOR** , **HANG** , **INTAKE** ,
 **LOADER** , **UMBRELLA** , **WINGS** }
   
*Defines all different control types.*

### 4.27.1 Detailed Description

Namespace for all user interactive components.

#### Author

Nathan Sandvig

### 4.27.2 Enumeration Type Documentation

#### 4.27.2.1 EControl

```
enum class wisco::user::EControl [strong]
```

Defines all different control inputs.

#### Author

Nathan Sandvig

Definition at line 25 of file [EControl.hpp](#).

```
00026 {  
00027     ELEVATOR_IN,  
00028     ELEVATOR_FIELD,  
00029     ELEVATOR_MATCH_LOAD,  
00030     ELEVATOR_POLE_HANG,  
00031     ELEVATOR_PARTNER_HANG,  
00032     ELEVATOR_OUT,  
00033     ELEVATOR_TOGGLE,  
00034     HANG_GRAB,  
00035     HANG_HANG,  
00036     HANG_INACTIVE,  
00037     HANG_NEXT,  
00038     HANG_PREVIOUS,  
00039     HANG_RAISE,  
00040     HANG_RESET,  
00041     HANG_TOGGLE,  
00042     INTAKE_IN,  
00043     INTAKE_OUT,  
00044     INTAKE_TOGGLE,  
00045     LOADER_HOLD,  
00046     LOADER_LOAD,  
00047     LOADER_READY,  
00048     LOADER_TOGGLE,  
00049     UMBRELLA_HOLD,  
00050     UMBRELLA_IN,  
00051     UMBRELLA_OUT,  
00052     UMBRELLA_TOGGLE,  
00053     WINGS_HOLD,  
00054     WINGS_IN,  
00055     WINGS_OUT,  
00056     WINGS_TOGGLE,  
00057     WINGS_LEFT_HOLD,  
00058     WINGS_LEFT_IN,  
00059     WINGS_LEFT_OUT,  
00060     WINGS_LEFT_TOGGLE,  
00061     WINGS_RIGHT_HOLD,  
00062     WINGS_RIGHT_IN,  
00063     WINGS_RIGHT_OUT,  
00064     WINGS_RIGHT_TOGGLE  
00065 };
```

#### 4.27.2.2 EControllerAnalog

```
enum class wisco::user::EControllerAnalog [strong]
```

Defines all different controller analog inputs.

##### Author

Nathan Sandvig

Definition at line 25 of file [EControllerAnalog.hpp](#).

```
00026 {  
00027     JOYSTICK_LEFT_X,  
00028     JOYSTICK_LEFT_Y,  
00029     JOYSTICK_RIGHT_X,  
00030     JOYSTICK_RIGHT_Y,  
00031     TRIGGER_LEFT,  
00032     TRIGGER_RIGHT,  
00033     NONE  
00034 };
```

#### 4.27.2.3 EControllerDigital

```
enum class wisco::user::EControllerDigital [strong]
```

Defines all different controller digital inputs.

##### Author

Nathan Sandvig

Definition at line 25 of file [EControllerDigital.hpp](#).

```
00026 {  
00027     BUTTON_A,  
00028     BUTTON_B,  
00029     BUTTON_X,  
00030     BUTTON_Y,  
00031     DPAD_DOWN,  
00032     DPAD_LEFT,  
00033     DPAD_RIGHT,  
00034     DPAD_UP,  
00035     JOYSTICK_LEFT,  
00036     JOYSTICK_RIGHT,  
00037     SCUFF_LEFT_REAR,  
00038     SCUFF_LEFT_UNDER,  
00039     SCUFF_RIGHT_REAR,  
00040     SCUFF_RIGHT_UNDER,  
00041     TRIGGER_LEFT_BOTTOM,  
00042     TRIGGER_LEFT_TOP,  
00043     TRIGGER_RIGHT_BOTTOM,  
00044     TRIGGER_RIGHT_TOP,  
00045     NONE  
00046 };
```

#### 4.27.2.4 EControlType

```
enum class wisco::user::EControlType [strong]
```

Defines all different control types.

##### Author

Nathan Sandvig

Definition at line 25 of file [EControlType.hpp](#).

```
00026 {  
00027     DRIVE,  
00028     ELEVATOR,  
00029     HANG,  
00030     INTAKE,  
00031     LOADER,  
00032     UMBRELLA,  
00033     WINGS  
00034 };
```

## 4.28 wisco::user::drive Namespace Reference

Namespace for all drive user control components.

### Classes

- class [DifferentialDriveOperator](#)  
*Runs the operator-controlled differential drive voltage settings.*

### Enumerations

- enum class [EChassisControlMode](#) {  
    SINGLE\_ARCADE\_LEFT , SINGLE\_ARCADE\_RIGHT , SPLIT\_ARCADE\_LEFT , SPLIT\_ARCADE\_←  
    RIGHT ,  
    TANK }  
*Defines all different chassis control formats.*

### 4.28.1 Detailed Description

Namespace for all drive user control components.

#### Author

Nathan Sandvig

### 4.28.2 Enumeration Type Documentation

#### 4.28.2.1 EChassisControlMode

```
enum class wisco::user::drive::EChassisControlMode [strong]
```

Defines all different chassis control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EChassisControlMode.hpp](#).

```
00034 {  
00035     SINGLE_ARCADE_LEFT,  
00036     SINGLE_ARCADE_RIGHT,  
00037     SPLIT_ARCADE_LEFT,  
00038     SPLIT_ARCADE_RIGHT,  
00039     TANK  
00040 };
```

## 4.29 wisco::user::elevator Namespace Reference

Namespace for all elevator user control components.

## Classes

- class [ElevatorOperator](#)  
*Runs the operator-controlled elevator position settings.*

## Enumerations

- enum class [EElevatorControlMode](#) {
 **MANUAL** , **PRESET\_SPLIT** , **PRESET\_TOGGLE\_SINGLE** , **PRESET\_TOGGLE\_LADDER** ,  
**PRESET\_TOGGLE\_LADDER\_INTAKE** }  
*Defines all different elevator control formats.*

### 4.29.1 Detailed Description

Namespace for all elevator user control components.

#### Author

Nathan Sandvig

### 4.29.2 Enumeration Type Documentation

#### 4.29.2.1 EElevatorControlMode

```
enum class wisco::user::elevator::EElevatorControlMode [strong]
```

Defines all different elevator control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EElevatorControlMode.hpp](#).

```
00034 {  
00035     MANUAL,  
00036     PRESET_SPLIT,  
00037     PRESET_TOGGLE_SINGLE,  
00038     PRESET_TOGGLE_LADDER,  
00039     PRESET_TOGGLE_LADDER_INTAKE  
00040};
```

## 4.30 wisco::user::hang Namespace Reference

Namespace for all hang user control components.

## Classes

- class [HangOperator](#)  
*Runs the operator-controlled hang settings.*

## Enumerations

- enum class [EHangControlMode](#) { **PRESET\_SPLIT** , **PRESET\_TOGGLE\_SINGLE** , **PRESET\_TOGGLE\_LADDER** , **PRESET\_TOGGLE\_RESET** }  
*Defines all different hang control formats.*

### 4.30.1 Detailed Description

Namespace for all hang user control components.

#### Author

Nathan Sandvig

### 4.30.2 Enumeration Type Documentation

#### 4.30.2.1 EHangControlMode

```
enum class wisco::user::hang::EHangControlMode [strong]
```

Defines all different hang control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EHangControlMode.hpp](#).

```
00034 {  
00035     PRESET_SPLIT,  
00036     PRESET_TOGGLE_SINGLE,  
00037     PRESET_TOGGLE_LADDER,  
00038     PRESET_TOGGLE_RESET  
00039 };
```

## 4.31 wisco::user::intake Namespace Reference

Namespace for all intake user control components.

## Classes

- class [IntakeOperator](#)  
*Runs the operator-controlled intake voltage settings.*

## Enumerations

- enum class [EIntakeControlMode](#) { **SINGLE\_TOGGLE** , **SPLIT\_HOLD** , **SPLIT\_TOGGLE** }  
*Defines all different intake control formats.*

### 4.31.1 Detailed Description

Namespace for all intake user control components.

#### Author

Nathan Sandvig

### 4.31.2 Enumeration Type Documentation

#### 4.31.2.1 EIntakeControlMode

```
enum class wisco::user::intake::EIntakeControlMode [strong]
```

Defines all different intake control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EIntakeControlMode.hpp](#).

```
00034 {  
00035     SINGLE_TOGGLE,  
00036     SPLIT_HOLD,  
00037     SPLIT_TOGGLE  
00038 };
```

## 4.32 wisco::user::loader Namespace Reference

Namespace for all loader user control components.

### Classes

- class [LoaderOperator](#)  
*Runs the operator-controlled loader settings.*

### Enumerations

- enum class [ELoaderControlMode](#) { **HOLD** , **MACRO** , **SINGLE\_TOGGLE** , **SPLIT\_TOGGLE** }  
*Defines all different loader control formats.*

### 4.32.1 Detailed Description

Namespace for all loader user control components.

#### Author

Nathan Sandvig

## 4.32.2 Enumeration Type Documentation

### 4.32.2.1 ELoaderControlMode

```
enum class wisco::user::loader::ELoaderControlMode [strong]
```

Defines all different loader control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [ELoaderControlMode.hpp](#).

```
00034 {  
00035     HOLD,  
00036     MACRO,  
00037     SINGLE_TOGGLE,  
00038     SPLIT_TOGGLE  
00039 };
```

## 4.33 wisco::user::umbrella Namespace Reference

Namespace for all umbrella user control components.

#### Classes

- class [UmbrellaOperator](#)  
*Runs the operator-controlled umbrella settings.*

#### Enumerations

- enum class [EUmbrellaControlMode](#) { **HOLD** , **SINGLE\_TOGGLE** , **SPLIT\_TOGGLE** }  
*Defines all different umbrella control formats.*

### 4.33.1 Detailed Description

Namespace for all umbrella user control components.

#### Author

Nathan Sandvig

## 4.33.2 Enumeration Type Documentation

### 4.33.2.1 EUmbrellaControlMode

enum class [wisco::user::umbrella::EUmbrellaControlMode](#) [strong]

Defines all different umbrella control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EUmbrellaControlMode.hpp](#).

```
00034 {
00035     HOLD,
00036     SINGLE_TOGGLE,
00037     SPLIT_TOGGLE
00038 };
```

## 4.34 wisco::user::wings Namespace Reference

Namespace for all wings user control components.

#### Classes

- class [WingsOperator](#)  
*Runs the operator-controlled wings settings.*

#### Enumerations

- enum class [EWingsControlMode](#) {
 DUAL\_HOLD, DUAL\_SPLIT, DUAL\_TOGGLE, SINGLE\_HOLD,
 SINGLE\_SPLIT, SINGLE\_TOGGLE }

*Defines all different wings control formats.*

### 4.34.1 Detailed Description

Namespace for all wings user control components.

#### Author

Nathan Sandvig

## 4.34.2 Enumeration Type Documentation

### 4.34.2.1 EWingsControlMode

enum class [wisco::user::wings::EWingsControlMode](#) [strong]

Defines all different wings control formats.

#### Author

Nathan Sandvig

Definition at line 33 of file [EWingsControlMode.hpp](#).

```
00034 {
00035     DUAL_HOLD,
00036     DUAL_SPLIT,
00037     DUAL_TOGGLE,
00038     SINGLE_HOLD,
00039     SINGLE_SPLIT,
00040     SINGLE_TOGGLE
00041 };
```

# Chapter 5

## Class Documentation

### 5.1 MatchControllerFactory Class Reference

Class to create match controllers.

#### Static Public Member Functions

- static `wisco::MatchController createMatchController ()`

*Create a Match Controller.*

#### 5.1.1 Detailed Description

Class to create match controllers.

#### Author

Nathan Sandvig

Definition at line 28 of file [MatchControllerFactory.hpp](#).

#### 5.1.2 Member Function Documentation

##### 5.1.2.1 `createMatchController()`

```
wisco::MatchController MatchControllerFactory::createMatchController ( ) [static]
```

Create a Match Controller.

**Returns**

MatchController The new match controller

Definition at line 3 of file [MatchControllerFactory.cpp](#).

```

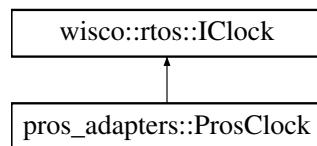
00004 {
00005     // Menu creation
00006     std::unique_ptr<wisco::IMenu> lvgl_menu{std::make_unique<wisco::menu::MenuAdapter>()};
00007     std::unique_ptr<wisco::IAlliance>
00008         blue_alliance{std::make_unique<wisco::alliances::BlueAlliance>()};
00009         lvgl_menu->addAlliance(blue_alliance);
00010     std::unique_ptr<wisco::IAlliance> red_alliance{std::make_unique<wisco::alliances::RedAlliance>()};
00011     lvgl_menu->addAlliance(red_alliance);
00012     std::unique_ptr<wisco::IAlliance>
00013         skills_alliance{std::make_unique<wisco::alliances::SkillsAlliance>()};
00014     lvgl_menu->addAlliance.skills_alliance();
00015     std::unique_ptr<wisco::IAutonomous>
00016         blue_match_autonomous{std::make_unique<wisco::autons::BlueMatchAuton>()};
00017     lvgl_menu->addAutonomous(blue_match_autonomous);
00018     std::unique_ptr<wisco::IAutonomous>
00019         blue_skills_autonomous{std::make_unique<wisco::autons::BlueSkillsAuton>()};
00020     lvgl_menu->addAutonomous(blue_skills_autonomous);
00021     std::unique_ptr<wisco::IAutonomous>
00022         orange_match_autonomous{std::make_unique<wisco::autons::OrangeMatchAuton>()};
00023     lvgl_menu->addAutonomous(orange_match_autonomous);
00024     std::unique_ptr<wisco::IAutonomous>
00025         orange_skills_autonomous{std::make_unique<wisco::autons::OrangeSkillsAuton>()};
00026     lvgl_menu->addAutonomous(orange_skills_autonomous);
00027     std::unique_ptr<wisco::IConfiguration>
00028         blue_configuration{std::make_unique<wisco::configs::BlueConfiguration>()};
00029     lvgl_menu->addConfiguration(blue_configuration);
00030     std::unique_ptr<wisco::IConfiguration>
00031         orange_configuration{std::make_unique<wisco::configs::OrangeConfiguration>()};
00032     lvgl_menu->addConfiguration(orange_configuration);
00033     std::unique_ptr<wisco::IProfile> henry_profile{std::make_unique<wisco::profiles::HenryProfile>()};
00034     lvgl_menu->addProfile(henry_profile);
00035     std::unique_ptr<wisco::IProfile> john_profile{std::make_unique<wisco::profiles::JohnProfile>()};
00036     lvgl_menu->addProfile(john_profile);

00037     // RTOS creation
00038     std::shared_ptr<wisco::rtos::IClock> pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00039     std::unique_ptr<wisco::rtos::IDelayLayer>
00040         pros_delayer{std::make_unique<pros_adapters::ProsDelayLayer>()};
00041     return wisco::MatchController{lvgl_menu, pros_clock, pros_delayer};
00042 }
```

## 5.2 pros\_adapters::ProsClock Class Reference

Pros rtos clock adapter for the wisco rtos IClock interface.

Inheritance diagram for pros\_adapters::ProsClock:



### Public Member Functions

- std::unique\_ptr< wisco::rtos::IClock > **clone** () const override  
*Clones the IClock object.*
- uint32\_t **getTime** () override  
*Get the system clock time in milliseconds.*

## Public Member Functions inherited from [wisco::rtos::IClock](#)

- virtual ~**IClock** ()=default

*Destroy the [IClock](#) object.*

### 5.2.1 Detailed Description

Pros rtos clock adapter for the wisco rtos IClock interface.

#### Author

Nathan Sandvig

Definition at line 21 of file [ProsClock.hpp](#).

### 5.2.2 Member Function Documentation

#### 5.2.2.1 clone()

```
std::unique_ptr<wisco::rtos::IClock> pros_adapters::ProsClock::clone() const [override],  
[virtual]
```

Clones the IClock object.

#### Returns

`std::unique_ptr<IClock>` The cloned IClock object

Implements [wisco::rtos::IClock](#).

Definition at line 5 of file [ProsClock.cpp](#).

```
00006 {  
00007     return std::unique_ptr<wisco::rtos::IClock>(std::make_unique<ProsClock>(*this));  
00008 }
```

#### 5.2.2.2 getTime()

```
uint32_t pros_adapters::ProsClock::getTime() [override], [virtual]
```

Get the system clock time in milliseconds.

#### Returns

`uint32_t` The system clock time in milliseconds

Implements [wisco::rtos::IClock](#).

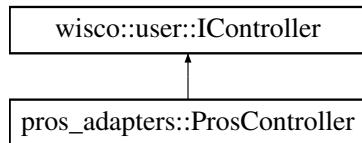
Definition at line 10 of file [ProsClock.cpp](#).

```
00011 {  
00012     return pros::millis();  
00013 }
```

## 5.3 pros\_adapters::ProsController Class Reference

Pros controller adapter for the wisco user IController interface.

Inheritance diagram for pros\_adapters::ProsController:



### Public Member Functions

- **ProsController** (std::unique\_ptr< pros::Controller > &controller)  
*Construct a new Pros Controller object.*
- void **initialize** () override  
*Initializes the controller.*
- void **run** () override  
*Runs the controller.*
- double **getAnalog** (wisco::user::EControllerAnalog analog\_channel) override  
*Get the analog input of a channel from the controller.*
- bool **getDigital** (wisco::user::EControllerDigital digital\_channel) override  
*Get the digital input of a channel from the controller.*
- bool **getNewDigital** (wisco::user::EControllerDigital digital\_channel) override  
*Check for a new digital input of a channel from the controller.*
- void **rumble** (std::string pattern) override  
*Rumbles the controller.*

### Public Member Functions inherited from [wisco::user::IController](#)

- virtual ~**IController** ()=default  
*Destroy the **IController** object.*

### Private Member Functions

- void **updateRumble** ()  
*Updates the rumble.*
- void **taskUpdate** ()  
*Runs in the task loop to update the controller.*

### Static Private Member Functions

- static void **taskLoop** (void \*params)  
*The task loop function for background updates.*

### Private Attributes

- const std::map< [wisco::user::EControllerAnalog](#), pros::controller\_analog\_e\_t > **ANALOG\_MAP**  
*The mapping of analog controls.*
- const std::map< [wisco::user::EControllerDigital](#), pros::controller\_digital\_e\_t > **DIGITAL\_MAP**  
*The mapping of digital controls.*
- std::unique\_ptr< pros::Controller > **m\_controller** {}  
*The controller being adapted.*
- pros::Mutex **mutex** {}  
*The mutex for thread safety.*
- char **rumble\_pattern** [MAX\_RUMBLE\_LENGTH] {}  
*The current rumble pattern.*
- bool **new\_rumble\_pattern** {}  
*Whether or not there is a new rumble pattern.*
- uint32\_t **last\_rumble\_refresh** {}  
*The last time the rumble was refreshed.*

### Static Private Attributes

- static constexpr uint8\_t **TASK\_DELAY** {10}  
*The delay in the task loop.*
- static constexpr uint8\_t **RUMBLE\_REFRESH\_RATE** {50}  
*The refresh rate of the rumble output.*
- static constexpr double **ANALOG\_CONVERSION** {1.0 / 127}  
*Converts the analog values to [-1, 1].*
- static constexpr uint8\_t **MAX\_RUMBLE\_LENGTH** {8}  
*The maximum length of a rumble pattern.*

### 5.3.1 Detailed Description

Pros controller adapter for the wisco user IController interface.

#### Author

Nathan Sandvig

Definition at line [26](#) of file [ProsController.hpp](#).

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 ProsController()

```
pros_adapters::ProsController::ProsController (
    std::unique_ptr< pros::Controller > & controller )
```

Construct a new Pros Controller object.

**Parameters**

<i>controller</i>	The controller being adapted
-------------------	------------------------------

Definition at line 5 of file ProsController.cpp.

```
00005     m_controller{std::move(controller)} : 
00006 {
00007
00008 }
```

### 5.3.3 Member Function Documentation

#### 5.3.3.1 taskLoop()

```
void pros_adapters::ProsController::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

**Parameters**

<i>params</i>	
---------------	--

Definition at line 10 of file ProsController.cpp.

```
00011 {
00012     void** parameters{static_cast<void**>(params)};
00013     ProsController* controller{static_cast<ProsController*>(parameters[0])};
00014
00015     while (true)
00016     {
00017         controller->taskUpdate();
00018         pros::delay(TASK_DELAY);
00019     }
00020 }
```

#### 5.3.3.2 updateRumble()

```
void pros_adapters::ProsController::updateRumble ( ) [private]
```

Updates the rumble.

Definition at line 22 of file ProsController.cpp.

```
00023 {
00024     uint32_t time{pros::millis()};
00025     if (new_rumble_pattern && time - last_rumble_refresh >= RUMBLE_REFRESH_RATE)
00026     {
00027         if (m_controller)
00028             m_controller->rumble(rumble_pattern);
00029         new_rumble_pattern = false;
00030         last_rumble_refresh = time;
00031     }
00032 }
```

### 5.3.3.3 taskUpdate()

```
void pros_adapters::ProsController::taskUpdate () [private]
```

Runs in the task loop to update the controller.

Definition at line 34 of file [ProsController.cpp](#).

```
00035 {  
00036     mutex.take();  
00037     updateRumble();  
00038     mutex.give();  
00039 }
```

### 5.3.3.4 initialize()

```
void pros_adapters::ProsController::initialize () [override], [virtual]
```

Initializes the controller.

Implements [wisco::user::IController](#).

Definition at line 41 of file [ProsController.cpp](#).

```
00042 {  
00043  
00044 }
```

### 5.3.3.5 run()

```
void pros_adapters::ProsController::run () [override], [virtual]
```

Runs the controller.

Implements [wisco::user::IController](#).

Definition at line 46 of file [ProsController.cpp](#).

```
00047 {  
00048     void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};  
00049     params[0] = this;  
00050     pros::Task controllerTask{&ProsController::taskLoop, params};  
00051 }
```

### 5.3.3.6 getAnalog()

```
double pros_adapters::ProsController::getAnalog (  
    wisco::user::EControllerAnalog analog_channel) [override], [virtual]
```

Get the analog input of a channel from the controller.

#### Parameters

<i>analog_channel</i>	The channel to read analog input from
-----------------------	---------------------------------------

**Returns**

double The value of the analog channel

Implements [wisco::user::IController](#).

Definition at line 53 of file [ProsController.cpp](#).

```
00054 {
00055     double value{};
00056     if (ANALOG_MAP.contains(analog_channel))
00057         if (m_controller)
00058             value = m_controller->get_analog(ANALOG_MAP.at(analog_channel)) * ANALOG_CONVERSION;
00059     return value;
00060 }
```

**5.3.3.7 getDigital()**

```
bool pros_adapters::ProsController::getDigital (
    wisco::user::EControllerDigital digital_channel ) [override], [virtual]
```

Get the digital input of a channel from the controller.

**Parameters**

<i>digital_channel</i>	The channel to read digital input from
------------------------	--

**Returns**

true The digital channel is active

false The digital channel is not active

Implements [wisco::user::IController](#).

Definition at line 62 of file [ProsController.cpp](#).

```
00063 {
00064     bool value{};
00065     if (DIGITAL_MAP.contains(digital_channel))
00066         if (m_controller)
00067             value = m_controller->get_digital(DIGITAL_MAP.at(digital_channel));
00068     return value;
00069 }
```

**5.3.3.8 getNewDigital()**

```
bool pros_adapters::ProsController::getNewDigital (
    wisco::user::EControllerDigital digital_channel ) [override], [virtual]
```

Check for a new digital input of a channel from the controller.

**Parameters**

<i>digital_channel</i>	The channel to read digital input from
------------------------	--

**Returns**

true The digital channel has a new input  
 false The digital channel does not have a new input

Implements [wisco::user::IController](#).

Definition at line 71 of file [ProsController.cpp](#).

```
00072 {
00073     bool value{};
00074     if (DIGITAL_MAP.contains(digital_channel))
00075         if (m_controller)
00076             value = m_controller->get_digital_new_press(DIGITAL_MAP.at(digital_channel));
00077     return value;
00078 }
```

**5.3.3.9 rumble()**

```
void pros_adapters::ProsController::rumble (
    std::string pattern) [override], [virtual]
```

Rumbles the controller.

**Parameters**

<i>pattern</i>	The rumble pattern to follow Up to 8 characters, '.' short, '-' long, '' pause
----------------	--

Implements [wisco::user::IController](#).

Definition at line 80 of file [ProsController.cpp](#).

```
00081 {
00082     mutex.take();
00083     for (uint8_t i{0}; i < MAX_RUMBLE_LENGTH; ++i)
00084         rumble_pattern[i] = pattern[i];
00085     new_rumble_pattern = true;
00086     mutex.give();
00087 }
```

**5.3.4 Member Data Documentation****5.3.4.1 TASK\_DELAY**

```
constexpr uint8_t pros_adapters::ProsController::TASK_DELAY {10} [static], [constexpr], [private]
```

The delay in the task loop.

Definition at line 33 of file [ProsController.hpp](#).

```
00033 {10};
```

**5.3.4.2 RUMBLE\_REFRESH\_RATE**

```
constexpr uint8_t pros_adapters::ProsController::RUMBLE_REFRESH_RATE {50} [static], [constexpr], [private]
```

The refresh rate of the rumble output.

Definition at line 39 of file [ProsController.hpp](#).

```
00039 {50};
```

### 5.3.4.3 ANALOG\_CONVERSION

```
constexpr double pros_adapters::ProsController::ANALOG_CONVERSION {1.0 / 127} [static], [constexpr], [private]
```

Converts the analog values to [-1, 1].

Definition at line 45 of file [ProsController.hpp](#).

```
00045 {1.0 / 127};
```

### 5.3.4.4 MAX\_RUMBLE\_LENGTH

```
constexpr uint8_t pros_adapters::ProsController::MAX_RUMBLE_LENGTH {8} [static], [constexpr], [private]
```

The maximum length of a rumble pattern.

Definition at line 51 of file [ProsController.hpp](#).

```
00051 {8};
```

### 5.3.4.5 ANALOG\_MAP

```
const std::map<wisco::user::EControllerAnalog, pros::controller_analog_e_t> pros_adapters::ProsController::ANALOG_MAP [private]
```

**Initial value:**

```
{
    {wisco::user::EControllerAnalog::JOYSTICK\_LEFT\_X, pros::E_CONTROLLER_ANALOG_LEFT_X},
    {wisco::user::EControllerAnalog::JOYSTICK\_LEFT\_Y, pros::E_CONTROLLER_ANALOG_LEFT_Y},
    {wisco::user::EControllerAnalog::JOYSTICK\_RIGHT\_X, pros::E_CONTROLLER_ANALOG_RIGHT_X},
    {wisco::user::EControllerAnalog::JOYSTICK\_RIGHT\_Y, pros::E_CONTROLLER_ANALOG_RIGHT_Y}
}
```

The mapping of analog controls.

Definition at line 64 of file [ProsController.hpp](#).

```
00065 {
00066     {wisco::user::EControllerAnalog::JOYSTICK\_LEFT\_X, pros::E_CONTROLLER_ANALOG_LEFT_X},
00067     {wisco::user::EControllerAnalog::JOYSTICK\_LEFT\_Y, pros::E_CONTROLLER_ANALOG_LEFT_Y},
00068     {wisco::user::EControllerAnalog::JOYSTICK\_RIGHT\_X, pros::E_CONTROLLER_ANALOG_RIGHT_X},
00069     {wisco::user::EControllerAnalog::JOYSTICK\_RIGHT\_Y, pros::E_CONTROLLER_ANALOG_RIGHT_Y}
00070 };
```

### 5.3.4.6 DIGITAL\_MAP

```
const std::map<wisco::user::EControllerDigital, pros::controller_digital_e_t> pros_adapters::ProsController::DIGITAL_MAP [private]
```

**Initial value:**

```
{
    {wisco::user::EControllerDigital::BUTTON\_A, pros::E_CONTROLLER_DIGITAL_A},
    {wisco::user::EControllerDigital::BUTTON\_B, pros::E_CONTROLLER_DIGITAL_B},
    {wisco::user::EControllerDigital::BUTTON\_X, pros::E_CONTROLLER_DIGITAL_X},
    {wisco::user::EControllerDigital::BUTTON\_Y, pros::E_CONTROLLER_DIGITAL_Y},
    {wisco::user::EControllerDigital::DPAD\_DOWN, pros::E_CONTROLLER_DIGITAL_DOWN},
    {wisco::user::EControllerDigital::DPAD\_LEFT, pros::E_CONTROLLER_DIGITAL_LEFT},
    {wisco::user::EControllerDigital::DPAD\_RIGHT, pros::E_CONTROLLER_DIGITAL_RIGHT},
    {wisco::user::EControllerDigital::DPAD\_UP, pros::E_CONTROLLER_DIGITAL_UP},
    {wisco::user::EControllerDigital::SCUFF\_LEFT\_REAR, pros::E_CONTROLLER_DIGITAL_RIGHT},
    {wisco::user::EControllerDigital::SCUFF\_RIGHT\_REAR, pros::E_CONTROLLER_DIGITAL_Y},
    {wisco::user::EControllerDigital::TRIGGER\_LEFT\_BOTTOM, pros::E_CONTROLLER_DIGITAL_L2},
    {wisco::user::EControllerDigital::TRIGGER\_LEFT\_TOP, pros::E_CONTROLLER_DIGITAL_L1},
```

```

{wisco::user::EControllerDigital::TRIGGER_RIGHT_BOTTOM, pros::E_CONTROLLER_DIGITAL_R2},
{wisco::user::EControllerDigital::TRIGGER_RIGHT_TOP, pros::E_CONTROLLER_DIGITAL_R1}
}
```

The mapping of digital controls.

Definition at line 76 of file ProsController.hpp.

```

00077    {
00078        {wisco::user::EControllerDigital::BUTTON_A, pros::E_CONTROLLER_DIGITAL_A},
00079        {wisco::user::EControllerDigital::BUTTON_B, pros::E_CONTROLLER_DIGITAL_B},
00080        {wisco::user::EControllerDigital::BUTTON_X, pros::E_CONTROLLER_DIGITAL_X},
00081        {wisco::user::EControllerDigital::BUTTON_Y, pros::E_CONTROLLER_DIGITAL_Y},
00082        {wisco::user::EControllerDigital::DPAD_DOWN, pros::E_CONTROLLER_DIGITAL_DOWN},
00083        {wisco::user::EControllerDigital::DPAD_LEFT, pros::E_CONTROLLER_DIGITAL_LEFT},
00084        {wisco::user::EControllerDigital::DPAD_RIGHT, pros::E_CONTROLLER_DIGITAL_RIGHT},
00085        {wisco::user::EControllerDigital::DPAD_UP, pros::E_CONTROLLER_DIGITAL_UP},
00086        {wisco::user::EControllerDigital::SCUFF_LEFT_REAR, pros::E_CONTROLLER_DIGITAL_RIGHT},
00087        {wisco::user::EControllerDigital::SCUFF_RIGHT_REAR, pros::E_CONTROLLER_DIGITAL_Y},
00088        {wisco::user::EControllerDigital::TRIGGER_LEFT_BOTTOM, pros::E_CONTROLLER_DIGITAL_L2},
00089        {wisco::user::EControllerDigital::TRIGGER_LEFT_TOP, pros::E_CONTROLLER_DIGITAL_L1},
00090        {wisco::user::EControllerDigital::TRIGGER_RIGHT_BOTTOM, pros::E_CONTROLLER_DIGITAL_R2},
00091        {wisco::user::EControllerDigital::TRIGGER_RIGHT_TOP, pros::E_CONTROLLER_DIGITAL_R1}
00092    };

```

#### 5.3.4.7 m\_controller

```
std::unique_ptr<pros::Controller> pros_adapters::ProsController::m_controller {} [private]
```

The controller being adapted.

Definition at line 98 of file ProsController.hpp.

```
00098 {};
```

#### 5.3.4.8 mutex

```
pros::Mutex pros_adapters::ProsController::mutex {} [private]
```

The mutex for thread safety.

Definition at line 104 of file ProsController.hpp.

```
00104 {};
```

#### 5.3.4.9 rumble\_pattern

```
char pros_adapters::ProsController::rumble_pattern[MAX_RUMBLE_LENGTH] {} [private]
```

The current rumble pattern.

Definition at line 110 of file ProsController.hpp.

```
00110 {};
```

#### 5.3.4.10 new\_rumble\_pattern

```
bool pros_adapters::ProsController::new_rumble_pattern {} [private]
```

Whether or not there is a new rumble pattern.

Definition at line 116 of file ProsController.hpp.

```
00116 {};
```

### 5.3.4.11 last\_rumble\_refresh

```
uint32_t pros_adapters::ProsController::last_rumble_refresh {} [private]
```

The last time the rumble was refreshed.

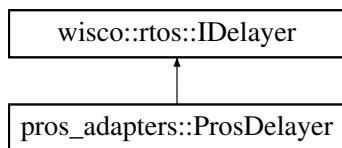
Definition at line 122 of file [ProsController.hpp](#).

```
00122 {};
```

## 5.4 pros\_adapters::ProsDelay Class Reference

Pros rtos delay adapter for the wisco rtos IDelay interface.

Inheritance diagram for pros\_adapters::ProsDelay:



### Public Member Functions

- std::unique\_ptr<[wisco::rtos::IDelay](#)> [clone](#) () const override  
*Clones the IDelay object.*
- void [delay](#) (uint32\_t millis) override  
*Delays the rtos system for a number of milliseconds.*
- void [delayUntil](#) (uint32\_t time) override  
*Delays the rtos system until a certain system time in milliseconds.*

### Public Member Functions inherited from [wisco::rtos::IDelay](#)

- virtual ~[IDelay](#) ()=default  
*Destroy the IDelay object.*

### 5.4.1 Detailed Description

Pros rtos delay adapter for the wisco rtos IDelay interface.

#### Author

Nathan Sandvig

Definition at line 20 of file [ProsDelay.hpp](#).

## 5.4.2 Member Function Documentation

### 5.4.2.1 clone()

```
std::unique_ptr< wisco::rtos::IDelay > pros_adapters::ProsDelay::clone ( ) const [override],  
[virtual]
```

Clones the IDelay object.

#### Returns

```
std::unique_ptr<IDelay> The cloned IDelay object
```

Implements [wisco::rtos::IDelay](#).

Definition at line 5 of file [ProsDelay.cpp](#).

```
00006 {  
00007     return std::unique_ptr<wisco::rtos::IDelay>(std::make_unique<ProsDelay>(*this));  
00008 }
```

### 5.4.2.2 delay()

```
void pros_adapters::ProsDelay::delay (uint32_t millis) [override], [virtual]
```

Delays the rtos system for a number of milliseconds.

#### Parameters

<i>millis</i>	The number of milliseconds to delay
---------------	-------------------------------------

Implements [wisco::rtos::IDelay](#).

Definition at line 10 of file [ProsDelay.cpp](#).

```
00011 {  
00012     pros::Task::delay(millis);  
00013 }
```

### 5.4.2.3 delayUntil()

```
void pros_adapters::ProsDelay::delayUntil (uint32_t time) [override], [virtual]
```

Delays the rtos system until a certain system time in milliseconds.

#### Parameters

<i>time</i>	The time in milliseconds to delay until
-------------	---

Implements [wisco::rtos::IDelay](#).

Definition at line 15 of file [ProsDelay.cpp](#).

```

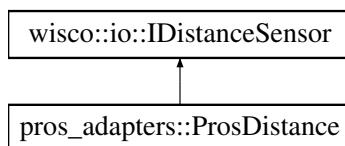
00016 {
00017     uint32_t current_time{pros::millis()};
00018     pros::Task::delay_until(&current_time, time - current_time);
00019 }

```

## 5.5 pros\_adapters::ProsDistance Class Reference

Pros distance sensor adapter for the wisco IDistanceSensor interface.

Inheritance diagram for pros\_adapters::ProsDistance:



### Public Member Functions

- **ProsDistance** (std::unique\_ptr< pros::Distance > &sensor, double tuning\_constant=1, double tuning\_offset=0)  
*Construct a new Pros Heading object.*
- void **initialize** () override  
*Initializes the sensor.*
- void **reset** () override  
*Resets the sensor.*
- double **getDistance** () override  
*Get the distance of the sensor in inches.*

### Public Member Functions inherited from [wisco::io::IDistanceSensor](#)

- virtual ~**IDistanceSensor** ()=default  
*Destroy the [IDistanceSensor](#) object.*

### Private Attributes

- std::unique\_ptr< pros::Distance > **m\_sensor** {}  
*The sensor being adapted.*
- double **m\_tuning\_constant** {1}  
*The tuning constant for the sensor.*
- double **m\_tuning\_offset** {}  
*The tuning offset for the sensor.*

### Static Private Attributes

- static constexpr double **UNIT\_CONVERTER** {1.0 / 25.4}  
*Converts the units for the sensor.*

### 5.5.1 Detailed Description

Pros distance sensor adapter for the wisco IDistanceSensor interface.

#### Author

Nathan Sandvig

Definition at line 21 of file [ProsDistance.hpp](#).

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 ProsDistance()

```
pros_adapters::ProsDistance::ProsDistance (
    std::unique_ptr< pros::Distance > & sensor,
    double tuning_constant = 1,
    double tuning_offset = 0 )
```

Construct a new Pros Heading object.

#### Parameters

<i>sensor</i>	The sensor being adapted
<i>tuning_constant</i>	The tuning constant multiplier for the sensor
<i>tuning_offset</i>	The tuning offset for the sensor

Definition at line 5 of file [ProsDistance.cpp](#).

```
00005 :
00006     m_sensor{std::move(sensor)}, m_tuning_constant{tuning_constant}, m_tuning_offset{tuning_offset}
00007 {
00008
00009 }
```

### 5.5.3 Member Function Documentation

#### 5.5.3.1 initialize()

```
void pros_adapters::ProsDistance::initialize () [override], [virtual]
```

Initializes the sensor.

Implements [wisco::io::IDistanceSensor](#).

Definition at line 11 of file [ProsDistance.cpp](#).

```
00012 {
00013
00014 }
```

### 5.5.3.2 reset()

```
void pros_adapters::ProsDistance::reset ( ) [override], [virtual]
```

Resets the sensor.

Implements [wisco::io::IDistanceSensor](#).

Definition at line 16 of file [ProsDistance.cpp](#).

```
00017 {
00018
00019 }
```

### 5.5.3.3 getDistance()

```
double pros_adapters::ProsDistance::getDistance ( ) [override], [virtual]
```

Get the distance of the sensor in inches.

Returns

double The rotation in inches

Implements [wisco::io::IDistanceSensor](#).

Definition at line 21 of file [ProsDistance.cpp](#).

```
00022 {
00023     double distance{};
00024     if (m_sensor)
00025     {
00026         distance = (m_sensor->get() * UNIT_CONVERTER * m_tuning_constant) + m_tuning_offset;
00027     }
00028     return distance;
00029 }
```

## 5.5.4 Member Data Documentation

### 5.5.4.1 UNIT\_CONVERTER

```
constexpr double pros_adapters::ProsDistance::UNIT_CONVERTER {1.0 / 25.4} [static], [constexpr], [private]
```

Converts the units for the sensor.

Definition at line 28 of file [ProsDistance.hpp](#).

```
00028 {1.0 / 25.4};
```

### 5.5.4.2 m\_sensor

```
std::unique_ptr<pros::Distance> pros_adapters::ProsDistance::m_sensor {} [private]
```

The sensor being adapted.

Definition at line 34 of file [ProsDistance.hpp](#).

```
00034 {};
```

### 5.5.4.3 m\_tuning\_constant

```
double pros_adapters::ProsDistance::m_tuning_constant {1} [private]
```

The tuning constant for the sensor.

Definition at line 40 of file [ProsDistance.hpp](#).  
00040 {1};

### 5.5.4.4 m\_tuning\_offset

```
double pros_adapters::ProsDistance::m_tuning_offset {} [private]
```

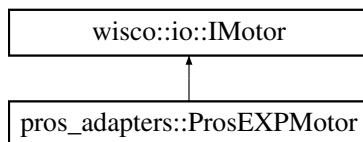
The tuning offset for the sensor.

Definition at line 46 of file [ProsDistance.hpp](#).  
00046 {};

## 5.6 pros\_adapters::ProsEXPMotor Class Reference

Pros exp smart motor adapter for the wisco IMotor interface.

Inheritance diagram for pros\_adapters::ProsEXPMotor:



### Public Member Functions

- [ProsEXPMotor](#) (std::unique\_ptr< pros::Motor > &motor)  
*Construct a new Pros EXP Motor object.*
- void [initialize](#) () override  
*Initializes the motor.*
- double [getTorqueConstant](#) () override  
*Get the torque constant of the motor.*
- double [getResistance](#) () override  
*Get the resistance of the motor.*
- double [getAngularVelocityConstant](#) () override  
*Get the angular velocity constant of the motor.*
- double [getGearRatio](#) () override  
*Get the gear ratio of the motor (1 if n/a)*
- double [getAngularVelocity](#) () override  
*Get the angular velocity of the motor in radians/second.*
- void [setVoltage](#) (double volts) override  
*Set the voltage input to the motor in Volts.*

## Public Member Functions inherited from `wisco::io::IMotor`

- virtual ~`IMotor` ()=default  
*Destroy the `IMotor` object.*
- virtual double `getPosition` ()=0  
*Get the position of the motor in total radians.*

## Private Attributes

- std::unique\_ptr< pros::Motor > `m_motor` {}  
*The motor being adapted.*

## Static Private Attributes

- static constexpr double `TORQUE_CONSTANT` {(0.5 / 18) / 2.5}  
*The torque constant of the motor.*
- static constexpr double `RESISTANCE` {}  
*The resistance of the motor.*
- static constexpr double `ANGULAR_VELOCITY_CONSTANT` {}  
*The angular velocity constant of the motor.*
- static constexpr double `GEAR_RATIO` {18}  
*The internal gear ratio for the motor.*
- static constexpr double `VELOCITY_CONVERSION` {2 \* M\_PI / 60}  
*Converts motor velocity to radians/second.*
- static constexpr double `VOLTAGE_CONVERSION` {1000}  
*Converts input voltage to millivolts.*

### 5.6.1 Detailed Description

Pros exp smart motor adapter for the wisco `IMotor` interface.

#### Author

Nathan Sandvig

Definition at line 23 of file `ProsEXPMotor.hpp`.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 `ProsEXPMotor()`

```
pros_adapters::ProsEXPMotor::ProsEXPMotor (
    std::unique_ptr< pros::Motor > & motor )
```

Construct a new Pros EXP Motor object.

**Parameters**

<i>motor</i>	The motor being adapted
--------------	-------------------------

Definition at line 5 of file ProsEXPMotor.cpp.

```
00005
00006 {
00007
00008 } : m_motor{std::move(motor)}
```

### 5.6.3 Member Function Documentation

#### 5.6.3.1 initialize()

```
void pros_adapters::ProsEXPMotor::initialize () [override], [virtual]
```

Initializes the motor.

Implements [wisco::io::IMotor](#).

Definition at line 10 of file ProsEXPMotor.cpp.

```
00011 {
00012     if (m_motor)
00013     {
00014         m_motor->move_voltage(0);
00015         m_motor->tare_position();
00016     }
00017 }
```

#### 5.6.3.2 getTorqueConstant()

```
double pros_adapters::ProsEXPMotor::getTorqueConstant () [override], [virtual]
```

Get the torque constant of the motor.

**Returns**

```
double The torque constant of the motor
```

Implements [wisco::io::IMotor](#).

Definition at line 19 of file ProsEXPMotor.cpp.

```
00020 {
00021     return TORQUE_CONSTANT;
00022 }
```

#### 5.6.3.3 getResistance()

```
double pros_adapters::ProsEXPMotor::getResistance () [override], [virtual]
```

Get the resistance of the motor.

**Returns**

```
double The resistance of the motor
```

Implements [wisco::io::IMotor](#).

Definition at line 24 of file ProsEXPMotor.cpp.

```
00025 {
00026     return RESISTANCE;
00027 }
```

### 5.6.3.4 getAngularVelocityConstant()

```
double pros_adapters::ProsEXPMotor::getAngularVelocityConstant() [override], [virtual]
```

Get the angular velocity constant of the motor.

#### Returns

double The angular velocity constant of the motor

Implements [wisco::io::IMotor](#).

Definition at line 29 of file [ProsEXPMotor.cpp](#).

```
00030 {
00031     return ANGULAR_VELOCITY_CONSTANT;
00032 }
```

### 5.6.3.5 getGearRatio()

```
double pros_adapters::ProsEXPMotor::getGearRatio() [override], [virtual]
```

Get the gear ratio of the motor (1 if n/a)

#### Returns

double The gear ratio of the motor

Implements [wisco::io::IMotor](#).

Definition at line 34 of file [ProsEXPMotor.cpp](#).

```
00035 {
00036     return GEAR_RATIO;
00037 }
```

### 5.6.3.6 getAngularVelocity()

```
double pros_adapters::ProsEXPMotor::getAngularVelocity() [override], [virtual]
```

Get the angular velocity of the motor in radians/second.

#### Returns

double The angular velocity of the motor in radians/second

Implements [wisco::io::IMotor](#).

Definition at line 39 of file [ProsEXPMotor.cpp](#).

```
00040 {
00041     double angular_velocity();
00042
00043     if (m_motor)
00044         angular_velocity = m_motor->get_actual_velocity() * VELOCITY_CONVERSION;
00045
00046     return angular_velocity;
00047 }
```

### 5.6.3.7 setVoltage()

```
void pros_adapters::ProsEXPMotor::setVoltage(
    double volts) [override], [virtual]
```

Set the voltage input to the motor in Volts.

**Parameters**

<i>volts</i>	The voltage input in Volts
--------------	----------------------------

Implements [wisco::io::IMotor](#).

Definition at line 49 of file [ProsEXPMotor.cpp](#).

```
00050 {  
00051     if (m_motor)  
00052         m_motor->move_voltage(volts * VOLTAGE_CONVERSION);  
00053 }
```

## 5.6.4 Member Data Documentation

### 5.6.4.1 TORQUE\_CONSTANT

```
constexpr double pros_adapters::ProsEXPMotor::TORQUE_CONSTANT {(0.5 / 18) / 2.5} [static],  
[constexpr], [private]
```

The torque constant of the motor.

Definition at line 30 of file [ProsEXPMotor.hpp](#).

```
00030 {(0.5 / 18) / 2.5};
```

### 5.6.4.2 RESISTANCE

```
constexpr double pros_adapters::ProsEXPMotor::RESISTANCE {} [static], [constexpr], [private]
```

The resistance of the motor.

Definition at line 36 of file [ProsEXPMotor.hpp](#).

```
00036 {};
```

### 5.6.4.3 ANGULAR\_VELOCITY\_CONSTANT

```
constexpr double pros_adapters::ProsEXPMotor::ANGULAR_VELOCITY_CONSTANT {} [static], [constexpr],  
[private]
```

The angular velocity constant of the motor.

Definition at line 42 of file [ProsEXPMotor.hpp](#).

```
00042 {};
```

### 5.6.4.4 GEAR\_RATIO

```
constexpr double pros_adapters::ProsEXPMotor::GEAR_RATIO {18} [static], [constexpr], [private]
```

The internal gear ratio for the motor.

Definition at line 48 of file [ProsEXPMotor.hpp](#).

```
00048 {18};
```

### 5.6.4.5 VELOCITY\_CONVERSION

```
constexpr double pros_adapters::ProsEXPMotor::VELOCITY_CONVERSION {2 * M_PI / 60} [static],  
[constexpr], [private]
```

Converts motor velocity to radians/second.

Definition at line 54 of file [ProsEXPMotor.hpp](#).  
00054 {2 \* M\_PI / 60};

### 5.6.4.6 VOLTAGE\_CONVERSION

```
constexpr double pros_adapters::ProsEXPMotor::VOLTAGE_CONVERSION {1000} [static], [constexpr],  
[private]
```

Converts input voltage to millivolts.

Definition at line 60 of file [ProsEXPMotor.hpp](#).  
00060 {1000};

### 5.6.4.7 m\_motor

```
std::unique_ptr<pros::Motor> pros_adapters::ProsEXPMotor::m_motor {} [private]
```

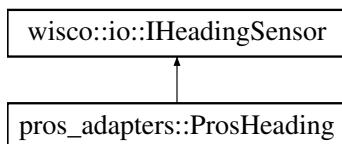
The motor being adapted.

Definition at line 66 of file [ProsEXPMotor.hpp](#).  
00066 {};

## 5.7 pros\_adapters::ProsHeading Class Reference

Pros inertial sensor adapter for the wisco IHeadingSensor interface.

Inheritance diagram for pros\_adapters::ProsHeading:



### Public Member Functions

- **ProsHeading** (std::unique\_ptr<pros::IMU> &sensor, double tuning\_constant=1)  
*Construct a new Pros Heading object.*
- void **initialize** () override  
*Initializes the sensor.*
- void **reset** () override  
*Resets the sensor.*
- double **getHeading** () override  
*Get the heading of the sensor in radians.*
- void **setHeading** (double heading) override  
*Set the heading of the sensor in radians.*
- double **getRotation** () override  
*Get the rotation of the sensor in radians.*
- void **setRotation** (double rotation) override  
*Set the rotation of the sensor in radians.*

## Public Member Functions inherited from [wisco::io::IHeadingSensor](#)

- virtual ~[IHeadingSensor](#) ()=default

*Destroy the [IHeadingSensor](#) object.*

### Private Attributes

- std::unique\_ptr< pros::Imu > [m\\_sensor](#) {}

*The sensor being adapted.*

- double [m\\_tuning\\_constant](#) {1}

*The tuning constant for the sensor.*

### Static Private Attributes

- static constexpr double [UNIT\\_CONVERTER](#) {-180 / M\_PI}

*Converts the units for the sensor.*

## 5.7.1 Detailed Description

Pros inertial sensor adapter for the wisco IHeadingSensor interface.

### Author

Nathan Sandvig

Definition at line 22 of file [ProsHeading.hpp](#).

## 5.7.2 Constructor & Destructor Documentation

### 5.7.2.1 ProsHeading()

```
pros_adapters::ProsHeading::ProsHeading (
    std::unique_ptr< pros::Imu > & sensor,
    double tuning_constant = 1 )
```

Construct a new Pros Heading object.

#### Parameters

<i>sensor</i>	The sensor being adapted
<i>tuning_constant</i>	The tuning constant multiplier for the sensor

Definition at line 5 of file [ProsHeading.cpp](#).

```
00005
00006     m\_sensor{std::move(sensor)}, m\_tuning\_constant{tuning_constant}
00007 {
00008
00009 }
```

## 5.7.3 Member Function Documentation

### 5.7.3.1 initialize()

```
void pros_adapters::ProsHeading::initialize () [override], [virtual]
```

Initializes the sensor.

Implements [wisco::io::IHeadingSensor](#).

Definition at line 11 of file [ProsHeading.cpp](#).

```
00012 {
00013     if (m_sensor)
00014     {
00015         if (m_sensor->is_installed())
00016         {
00017             m_sensor->reset();
00018             pros::delay(3000);
00019         }
00020     }
00021 }
```

### 5.7.3.2 reset()

```
void pros_adapters::ProsHeading::reset () [override], [virtual]
```

Resets the sensor.

Implements [wisco::io::IHeadingSensor](#).

Definition at line 23 of file [ProsHeading.cpp](#).

```
00024 {
00025     if (m_sensor)
00026     {
00027         uint8_t port{m_sensor->get_port()};
00028         pros::Device device{port};
00029         pros::DeviceType sensor_type{device.get_plugged_type()};
00030         if (sensor_type == pros::DeviceType::imu)
00031         {
00032             m_sensor->reset();
00033             pros::delay(3000);
00034         }
00035     }
00036 }
```

### 5.7.3.3 getHeading()

```
double pros_adapters::ProsHeading::getHeading () [override], [virtual]
```

Get the heading of the sensor in radians.

Returns

`double` The heading in radians

Implements [wisco::io::IHeadingSensor](#).

Definition at line 38 of file [ProsHeading.cpp](#).

```
00039 {
00040     double heading{};
00041     if (m_sensor)
00042     {
00043         heading = m_sensor->get_heading() / UNIT_CONVERTER;
00044     }
00045     return heading;
00046 }
```

### 5.7.3.4 setHeading()

```
void pros_adapters::ProsHeading::setHeading (
    double heading) [override], [virtual]
```

Set the heading of the sensor in radians.

**Parameters**

<i>heading</i>	The heading in radians
----------------	------------------------

Implements [wisco::io::IHeadingSensor](#).

Definition at line 48 of file [ProsHeading.cpp](#).

```
00049 {  
00050     if (m_sensor)  
00051         m_sensor->set_heading(heading * UNIT_CONVERTER);  
00052 }
```

### 5.7.3.5 getRotation()

```
double pros_adapters::ProsHeading::getRotation () [override], [virtual]
```

Get the rotation of the sensor in radians.

**Returns**

```
double The rotation in radians
```

Implements [wisco::io::IHeadingSensor](#).

Definition at line 54 of file [ProsHeading.cpp](#).

```
00055 {  
00056     double rotation{};  
00057     if (m_sensor)  
00058     {  
00059         rotation = m_sensor->get_rotation() / UNIT_CONVERTER;  
00060     }  
00061     return rotation;  
00062 }
```

### 5.7.3.6 setRotation()

```
void pros_adapters::ProsHeading::setRotation (  
    double rotation) [override], [virtual]
```

Set the rotation of the sensor in radians.

**Parameters**

<i>rotation</i>	The rotation in radians
-----------------	-------------------------

Implements [wisco::io::IHeadingSensor](#).

Definition at line 64 of file [ProsHeading.cpp](#).

```
00065 {  
00066     if (m_sensor)  
00067         m_sensor->set_rotation(rotation * UNIT_CONVERTER);  
00068 }
```

## 5.7.4 Member Data Documentation

### 5.7.4.1 UNIT\_CONVERTER

```
constexpr double pros_adapters::ProsHeading::UNIT_CONVERTER {-180 / M_PI} [static], [constexpr], [private]
```

Converts the units for the sensor.

Definition at line 29 of file [ProsHeading.hpp](#).  
00029 {-180 / M\_PI};

### 5.7.4.2 m\_sensor

```
std::unique_ptr<pros::Imu> pros_adapters::ProsHeading::m_sensor {} [private]
```

The sensor being adapted.

Definition at line 35 of file [ProsHeading.hpp](#).  
00035 {};

### 5.7.4.3 m\_tuning\_constant

```
double pros_adapters::ProsHeading::m_tuning_constant {1} [private]
```

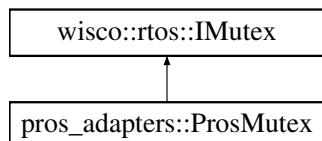
The tuning constant for the sensor.

Definition at line 41 of file [ProsHeading.hpp](#).  
00041 {1};

## 5.8 pros\_adapters::ProsMutex Class Reference

Pros rtos mutex adapter for the wisco rtos IMutex interface.

Inheritance diagram for pros\_adapters::ProsMutex:



### Public Member Functions

- void [take \(\)](#) override  
*Takes and locks the mutex.*
- void [give \(\)](#) override  
*Gives and unlocks the mutex.*

## Public Member Functions inherited from [wisco::rtos::IMutex](#)

- virtual ~[IMutex](#) ()=default

*Destroy the [IMutex](#) object.*

## Private Attributes

- pros::Mutex [mutex](#) {}  
*The mutex being adapted.*

### 5.8.1 Detailed Description

Pros rtos mutex adapter for the wisco rtos IMutex interface.

#### Author

Nathan Sandvig

Definition at line 22 of file [ProsMutex.hpp](#).

### 5.8.2 Member Function Documentation

#### 5.8.2.1 take()

```
void pros_adapters::ProsMutex::take ( ) [override], [virtual]
```

Takes and locks the mutex.

Implements [wisco::rtos::IMutex](#).

Definition at line 5 of file [ProsMutex.cpp](#).

```
00006 {  
00007     mutex.take();  
00008 }
```

#### 5.8.2.2 give()

```
void pros_adapters::ProsMutex::give ( ) [override], [virtual]
```

Gives and unlocks the mutex.

Implements [wisco::rtos::IMutex](#).

Definition at line 10 of file [ProsMutex.cpp](#).

```
00011 {  
00012     mutex.give();  
00013 }
```

## 5.8.3 Member Data Documentation

### 5.8.3.1 mutex

```
pros::Mutex pros_adapters::ProsMutex::mutex {} [private]
```

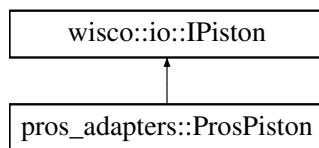
The mutex being adapted.

Definition at line 29 of file [ProsMutex.hpp](#).  
00029 {};

## 5.9 pros\_adapters::ProsPiston Class Reference

Pros piston adapter for the wisco IPiston interface.

Inheritance diagram for pros\_adapters::ProsPiston:



### Public Member Functions

- **ProsPiston** (std::unique\_ptr< pros::adi::DigitalOut > &digital\_out, bool extended\_value=true)  
*Construct a new Pros Piston object.*
- void **extend** () override  
*Extends the piston.*
- void **retract** () override  
*Retracts the piston.*
- void **toggle** () override  
*Toggles the piston state.*
- bool **isExtended** () override  
*Checks if the piston is extended.*
- bool **isRetracted** () override  
*Checks if the piston is retracted.*

### Public Member Functions inherited from [wisco::io::IPiston](#)

- virtual ~IPiston ()=default  
*Destroy the IPiston object.*

### Private Attributes

- std::unique\_ptr< pros::adi::DigitalOut > **m\_digital\_out** {}  
*The digital output for the piston.*
- bool **m\_extended\_value** {}  
*The value to use for extended.*
- bool **extended** {}  
*Whether the piston is currently extended or not.*

### 5.9.1 Detailed Description

Pros piston adapter for the wisco IPiston interface.

#### Author

Nathan Sandvig

Definition at line 23 of file [ProsPiston.hpp](#).

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 ProsPiston()

```
pros_adapters::ProsPiston::ProsPiston (
    std::unique_ptr< pros::adi::DigitalOut > & digital_out,
    bool extended_value = true )
```

Construct a new Pros Piston object.

#### Parameters

<code>digital_out</code>	The digital output for the piston
<code>extended_value</code>	The value to use for extended

Definition at line 5 of file [ProsPiston.cpp](#).

```
00006     : m_digital_out{std::move(digital_out)}, m_extended_value{extended_value}
00007 {
00008
00009 }
```

### 5.9.3 Member Function Documentation

#### 5.9.3.1 extend()

```
void pros_adapters::ProsPiston::extend ( ) [override], [virtual]
```

Extends the piston.

Implements [wisco::io::IPiston](#).

Definition at line 11 of file [ProsPiston.cpp](#).

```
00012 {
00013     if (m_digital_out)
00014     {
00015         extended = true;
00016         m_digital_out->set_value(m_extended_value);
00017     }
00018 }
```

### 5.9.3.2 retract()

```
void pros_adapters::ProsPiston::retract ( ) [override], [virtual]
```

Retracts the piston.

Implements [wisco::io::IPiston](#).

Definition at line 20 of file [ProsPiston.cpp](#).

```
00021 {
00022     if (m_digital_out)
00023     {
00024         extended = false;
00025         m_digital_out->set_value(!m_extended_value);
00026     }
00027 }
```

### 5.9.3.3 toggle()

```
void pros_adapters::ProsPiston::toggle ( ) [override], [virtual]
```

Toggles the piston state.

Implements [wisco::io::IPiston](#).

Definition at line 29 of file [ProsPiston.cpp](#).

```
00030 {
00031     if (m_digital_out)
00032     {
00033         m_digital_out->set_value(m_extended_value ^ extended);
00034         extended = !extended;
00035     }
00036 }
```

### 5.9.3.4 isExtended()

```
bool pros_adapters::ProsPiston::isExtended ( ) [override], [virtual]
```

Checks if the piston is extended.

#### Returns

true The piston is extended

false The piston is not extended

Implements [wisco::io::IPiston](#).

Definition at line 38 of file [ProsPiston.cpp](#).

```
00039 {
00040     return extended;
00041 }
```

### 5.9.3.5 isRetracted()

```
bool pros_adapters::ProsPiston::isRetracted () [override], [virtual]
```

Checks if the piston is retracted.

#### Returns

true The piston is retracted  
false The piston is not retracted

Implements [wisco::io::IPiston](#).

Definition at line 43 of file [ProsPiston.cpp](#).

```
00044 {  
00045     return !extended;  
00046 }
```

## 5.9.4 Member Data Documentation

### 5.9.4.1 m\_digital\_out

```
std::unique_ptr<pros::adi::DigitalOut> pros_adapters::ProsPiston::m_digital_out {} [private]
```

The digital output for the piston.

Definition at line 30 of file [ProsPiston.hpp](#).

```
00030 {};
```

### 5.9.4.2 m\_extended\_value

```
bool pros_adapters::ProsPiston::m_extended_value {} [private]
```

The value to use for extended.

Definition at line 36 of file [ProsPiston.hpp](#).

```
00036 {};
```

### 5.9.4.3 extended

```
bool pros_adapters::ProsPiston::extended {} [private]
```

Whether the piston is currently extended or not.

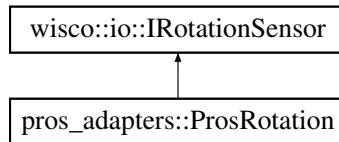
Definition at line 42 of file [ProsPiston.hpp](#).

```
00042 {};
```

## 5.10 pros\_adapters::ProsRotation Class Reference

Pros rotation sensor adapter for the wisco IRotationSensor interface.

Inheritance diagram for pros\_adapters::ProsRotation:



### Public Member Functions

- **ProsRotation** (std::unique\_ptr< pros::Rotation > &sensor)  
*Construct a new Pros Rotation object.*
- void **initialize** () override  
*Initializes the rotation sensor.*
- void **reset** () override  
*Resets the rotation sensor.*
- double **getRotation** () override  
*Get the rotation of the sensor in radians.*
- void **setRotation** (double rotation) override  
*Set the rotation of the sensor in radians.*
- double **getAngle** () override  
*Get the angle of the sensor in radians.*

### Public Member Functions inherited from [wisco::io::IRotationSensor](#)

- virtual ~**IRotationSensor** ()=default  
*Destroy the [IRotationSensor](#) object.*

### Private Attributes

- std::unique\_ptr< pros::Rotation > **m\_sensor** {}  
*The rotation sensor being adapted.*

### Static Private Attributes

- static constexpr double **UNIT\_CONVERSION** {18000 / M\_PI}  
*Conversion factor for the input and outputs units.*

#### 5.10.1 Detailed Description

Pros rotation sensor adapter for the wisco IRotationSensor interface.

##### Author

Nathan Sandvig

Definition at line 23 of file [ProsRotation.hpp](#).

## 5.10.2 Constructor & Destructor Documentation

### 5.10.2.1 ProsRotation()

```
pros_adapters::ProsRotation::ProsRotation (
    std::unique_ptr< pros::Rotation > & sensor )
```

Construct a new Pros Rotation object.

#### Parameters

sensor	The sensor to adapt
--------	---------------------

Definition at line 5 of file [ProsRotation.cpp](#).

```
00005
00006 {
00007
00008 }
```

: `m_sensor{std::move(sensor)}`

## 5.10.3 Member Function Documentation

### 5.10.3.1 initialize()

```
void pros_adapters::ProsRotation::initialize () [override], [virtual]
```

Initializes the rotation sensor.

Implements [wisco::io::IRotationSensor](#).

Definition at line 10 of file [ProsRotation.cpp](#).

```
00011 {
00012     if (m_sensor)
00013     {
00014         m_sensor->reset ();
00015     }
00016 }
```

### 5.10.3.2 reset()

```
void pros_adapters::ProsRotation::reset () [override], [virtual]
```

Resets the rotation sensor.

Implements [wisco::io::IRotationSensor](#).

Definition at line 18 of file [ProsRotation.cpp](#).

```
00019 {
00020     if (m_sensor)
00021     {
00022         m_sensor->reset ();
00023     }
00024 }
```

### 5.10.3.3 getRotation()

```
double pros_adapters::ProsRotation::getRotation ( ) [override], [virtual]
```

Get the rotation of the sensor in radians.

#### Returns

double The number of radians of rotation

Implements [wisco::io::IRotationSensor](#).

Definition at line 26 of file [ProsRotation.cpp](#).

```
00027 {
00028     double rotation{};
00029     if (m_sensor)
00030     {
00031         rotation = m_sensor->get_position() / UNIT_CONVERSION;
00032     }
00033     return rotation;
00034 }
```

### 5.10.3.4 setRotation()

```
void pros_adapters::ProsRotation::setRotation (
    double rotation) [override], [virtual]
```

Set the rotation of the sensor in radians.

#### Parameters

<i>rotation</i>	The number of radians of rotation
-----------------	-----------------------------------

Implements [wisco::io::IRotationSensor](#).

Definition at line 36 of file [ProsRotation.cpp](#).

```
00037 {
00038     if (m_sensor)
00039     {
00040         m_sensor->set_position(rotation * UNIT_CONVERSION);
00041     }
00042 }
```

### 5.10.3.5 getAngle()

```
double pros_adapters::ProsRotation::getAngle ( ) [override], [virtual]
```

Get the angle of the sensor in radians.

#### Returns

double The angle in radians

Implements [wisco::io::IRotationSensor](#).

Definition at line 44 of file [ProsRotation.cpp](#).

```
00045 {
00046     double angle{};
00047     if (m_sensor)
00048     {
00049         angle = m_sensor->get_angle() / UNIT_CONVERSION;
00050     }
00051     return angle;
00052 }
```

## 5.10.4 Member Data Documentation

### 5.10.4.1 UNIT\_CONVERSION

```
constexpr double pros_adapters::ProsRotation::UNIT_CONVERSION {18000 / M_PI} [static], [constexpr], [private]
```

Conversion factor for the input and outputs units.

Definition at line 30 of file [ProsRotation.hpp](#).  
00030 {18000 / M\_PI};

### 5.10.4.2 m\_sensor

```
std::unique_ptr<pros::Rotation> pros_adapters::ProsRotation::m_sensor {} [private]
```

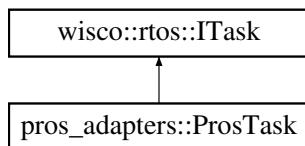
The rotation sensor being adapted.

Definition at line 36 of file [ProsRotation.hpp](#).  
00036 {};

## 5.11 pros\_adapters::ProsTask Class Reference

Pros rtos task adapter for the wisco rtos ITask interface.

Inheritance diagram for pros\_adapters::ProsTask:



### Public Member Functions

- void [start](#) (void(\*function)(void \*), void \*parameters) override  
*Starts the task.*
- void [remove](#) () override  
*Removes the task from the system.*
- void [suspend](#) () override  
*Suspends the task in the scheduler.*
- void [resume](#) () override  
*Resumes the task in the scheduler.*
- void [join](#) () override  
*Waits for the task to finish.*

### Public Member Functions inherited from [wisco::rtos::ITask](#)

- virtual ~[ITask](#) ()=default  
*Destroy the ITask object.*

### Private Attributes

- std::unique\_ptr< pros::Task > **task** {}  
*The pros task being adapted.*

### 5.11.1 Detailed Description

Pros rtos task adapter for the wisco rtos ITask interface.

#### Author

Nathan Sandvig

Definition at line 22 of file [ProsTask.hpp](#).

### 5.11.2 Member Function Documentation

#### 5.11.2.1 start()

```
void pros_adapters::Prostask::start (
    void(*) (void *) function,
    void * parameters ) [override], [virtual]
```

Starts the task.

#### Parameters

<i>function</i>	The function to run in the task
<i>parameters</i>	The parameters for the function

Implements [wisco::rtos::ITask](#).

Definition at line 5 of file [ProsTask.cpp](#).

```
00006 {
00007     task = std::make_unique<pros::Task>(function, parameters);
00008 }
```

#### 5.11.2.2 remove()

```
void pros_adapters::Prostask::remove () [override], [virtual]
```

Removes the task from the system.

Implements [wisco::rtos::ITask](#).

Definition at line 10 of file [ProsTask.cpp](#).

```
00011 {
00012     task->remove();
00013 }
```

### 5.11.2.3 suspend()

```
void pros_adapters::ProsTask::suspend ( ) [override], [virtual]
```

Suspends the task in the scheduler.

Implements [wisco::rtos::ITask](#).

Definition at line 15 of file [ProsTask.cpp](#).

```
00016 {  
00017     task->suspend();  
00018 }
```

### 5.11.2.4 resume()

```
void pros_adapters::ProsTask::resume ( ) [override], [virtual]
```

Resumes the task in the scheduler.

Implements [wisco::rtos::ITask](#).

Definition at line 20 of file [ProsTask.cpp](#).

```
00021 {  
00022     task->resume();  
00023 }
```

### 5.11.2.5 join()

```
void pros_adapters::ProsTask::join ( ) [override], [virtual]
```

Waits for the task to finish.

Implements [wisco::rtos::ITask](#).

Definition at line 25 of file [ProsTask.cpp](#).

```
00026 {  
00027     task->join();  
00028 }
```

## 5.11.3 Member Data Documentation

### 5.11.3.1 task

```
std::unique_ptr<pros::Task> pros_adapters::ProsTask::task {} [private]
```

The pros task being adapted.

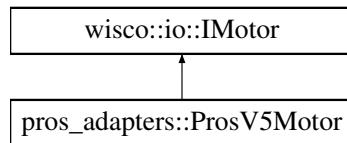
Definition at line 29 of file [ProsTask.hpp](#).

```
00029 {};
```

## 5.12 pros\_adapters::ProsV5Motor Class Reference

Pros v5 smart motor adapter for the wisco IMotor interface.

Inheritance diagram for pros\_adapters::ProsV5Motor:



### Public Member Functions

- **ProsV5Motor** (std::unique\_ptr< pros::Motor > &motor)  
*Construct a new Pros V5 Motor object.*
- void **initialize** () override  
*Initializes the motor.*
- double **getTorqueConstant** () override  
*Get the torque constant of the motor.*
- double **getResistance** () override  
*Get the resistance of the motor.*
- double **getAngularVelocityConstant** () override  
*Get the angular velocity constant of the motor.*
- double **getGearRatio** () override  
*Get the gear ratio of the motor (1 if n/a)*
- double **getAngularVelocity** () override  
*Get the angular velocity of the motor in radians/second.*
- double **getPosition** () override  
*Get the position of the motor in total radians.*
- void **setVoltage** (double volts) override  
*Set the voltage input to the motor in Volts.*

### Public Member Functions inherited from [wisco::io::IMotor](#)

- virtual ~IMotor ()=default  
*Destroy the IMotor object.*

### Private Attributes

- const std::map< pros::MotorGears, double > **cartridge\_map**  
*Map from v5 motor cartridges to gear ratios.*
- std::unique\_ptr< pros::Motor > **m\_motor** {}  
*The motor being adapted.*

## Static Private Attributes

- static constexpr double **NO\_CARTRIDGE** {1.0}  
*The gear ratio if no cartridge is present.*
- static constexpr double **TORQUE\_CONSTANT** {(2.1 / 36) / 2.5}  
*The torque constant of the motor.*
- static constexpr double **RESISTANCE** {3.2}  
*The resistance of the motor in ohms.*
- static constexpr double **ANGULAR\_VELOCITY\_CONSTANT** {2.1}  
*The angular velocity constant of the motor.*
- static constexpr double **VELOCITY\_CONVERSION** {2 \* M\_PI / 60}  
*Converts motor velocity to radians/second.*
- static constexpr double **POSITION\_CONVERSION** {M\_PI / 25}  
*Converts motor position to radians.*
- static constexpr double **VOLTAGE\_CONVERSION** {1000}  
*Converts input voltage to millivolts.*
- static constexpr int **MAX\_MILLIVOLTS** {12000}  
*The maximum output to the motor in millivolts.*

## 5.12.1 Detailed Description

Pros v5 smart motor adapter for the wisco IMotor interface.

### Author

Nathan Sandvig

Definition at line 24 of file [ProsV5Motor.hpp](#).

## 5.12.2 Constructor & Destructor Documentation

### 5.12.2.1 ProsV5Motor()

```
pros_adapters::ProsV5Motor::ProsV5Motor (
    std::unique_ptr< pros::Motor > & motor )
```

Construct a new Pros V5 Motor object.

#### Parameters

<i>motor</i>	The motor being adapted
--------------	-------------------------

Definition at line 7 of file [ProsV5Motor.cpp](#).

```
00007
00008 {
00009
00010 }
```

```
: m_motor{std::move(motor)}
```

### 5.12.3 Member Function Documentation

#### 5.12.3.1 initialize()

```
void pros_adapters::ProsV5Motor::initialize ( ) [override], [virtual]
```

Initializes the motor.

Implements [wisco::io::IMotor](#).

Definition at line 12 of file [ProsV5Motor.cpp](#).

```
00013 {
00014     if (m_motor)
00015     {
00016         m_motor->move_voltage(0);
00017         m_motor->tare_position();
00018         m_motor->set_brake_mode(pros::E_MOTOR_BRAKE_BRAKE);
00019     }
00020 }
```

#### 5.12.3.2 getTorqueConstant()

```
double pros_adapters::ProsV5Motor::getTorqueConstant ( ) [override], [virtual]
```

Get the torque constant of the motor.

##### Returns

double The torque constant of the motor

Implements [wisco::io::IMotor](#).

Definition at line 22 of file [ProsV5Motor.cpp](#).

```
00023 {
00024     return TORQUE_CONSTANT;
00025 }
```

#### 5.12.3.3 getResistance()

```
double pros_adapters::ProsV5Motor::getResistance ( ) [override], [virtual]
```

Get the resistance of the motor.

##### Returns

double The resistance of the motor

Implements [wisco::io::IMotor](#).

Definition at line 27 of file [ProsV5Motor.cpp](#).

```
00028 {
00029     return RESISTANCE;
00030 }
```

### 5.12.3.4 getAngularVelocityConstant()

```
double pros_adapters::ProsV5Motor::getAngularVelocityConstant() [override], [virtual]
```

Get the angular velocity constant of the motor.

#### Returns

```
double The angular velocity constant of the motor
```

Implements [wisco::io::IMotor](#).

Definition at line 32 of file [ProsV5Motor.cpp](#).

```
00033 {
00034     return ANGULAR_VELOCITY_CONSTANT;
00035 }
```

### 5.12.3.5 getGearRatio()

```
double pros_adapters::ProsV5Motor::getGearRatio() [override], [virtual]
```

Get the gear ratio of the motor (1 if n/a)

#### Returns

```
double The gear ratio of the motor
```

Implements [wisco::io::IMotor](#).

Definition at line 37 of file [ProsV5Motor.cpp](#).

```
00038 {
00039     double ratio{};
00040
00041     if (m_motor)
00042     {
00043         pros::MotorGears gearing{m_motor->get_gearing()};
00044         if (cartridge_map.contains(gearing))
00045             ratio = cartridge_map.at(gearing);
00046         else
00047             ratio = NO_CARTRIDGE;
00048     }
00049
00050     return ratio;
00051 }
```

### 5.12.3.6 getAngularVelocity()

```
double pros_adapters::ProsV5Motor::getAngularVelocity() [override], [virtual]
```

Get the angular velocity of the motor in radians/second.

#### Returns

```
double The angular velocity of the motor in radians/second
```

Implements [wisco::io::IMotor](#).

Definition at line 53 of file [ProsV5Motor.cpp](#).

```
00054 {
00055     double angular_velocity{};
00056
00057     if (m_motor)
00058         angular_velocity = m_motor->get_actual_velocity() * VELOCITY_CONVERSION;
00059
00060     return angular_velocity;
00061 }
```

### 5.12.3.7 `getPosition()`

```
double pros_adapters::ProsV5Motor::getPosition () [override], [virtual]
```

Get the position of the motor in total radians.

#### Returns

`double` The total number of radians moved since last reset

Implements [wisco::io::IMotor](#).

Definition at line 63 of file [ProsV5Motor.cpp](#).

```
00064 {
00065     double position{};
00066
00067     if (m_motor)
00068     {
00069         position = (m_motor->get_position() / getGearRatio()) * POSITION_CONVERSION;
00070     }
00071
00072     return position;
00073 }
```

### 5.12.3.8 `setVoltage()`

```
void pros_adapters::ProsV5Motor::setVoltage (
    double volts) [override], [virtual]
```

Set the voltage input to the motor in Volts.

#### Parameters

<code>volts</code>	The voltage input in Volts
--------------------	----------------------------

Implements [wisco::io::IMotor](#).

Definition at line 75 of file [ProsV5Motor.cpp](#).

```
00076 {
00077     int millivolts{static_cast<int>(volts * VOLTAGE_CONVERSION)};
00078     millivolts = std::min(millivolts, MAX_MILLIVOLTS);
00079     millivolts = std::max(millivolts, -MAX_MILLIVOLTS);
00080
00081     if (m_motor)
00082         m_motor->move_voltage(millivolts);
00083 }
```

## 5.12.4 Member Data Documentation

### 5.12.4.1 `cartridge_map`

```
const std::map<pros::MotorGears, double> pros_adapters::ProsV5Motor::cartridge_map [private]
```

#### Initial value:

```
{
    {pros::MotorGears::rpm_100, 36.0},
    {pros::MotorGears::rpm_200, 18.0},
    {pros::MotorGears::rpm_600, 6.0}
```

```
}
```

Map from v5 motor cartridges to gear ratios.

Definition at line 31 of file [ProsV5Motor.hpp](#).

```
00032     {  
00033         {pros::MotorGears::rpm_100, 36.0},  
00034         {pros::MotorGears::rpm_200, 18.0},  
00035         {pros::MotorGears::rpm_600, 6.0}  
00036     };
```

#### 5.12.4.2 NO\_CARTRIDGE

```
constexpr double pros_adapters::ProsV5Motor::NO_CARTRIDGE {1.0} [static], [constexpr], [private]
```

The gear ratio if no cartridge is present.

Definition at line 42 of file [ProsV5Motor.hpp](#).

```
00042 {1.0};
```

#### 5.12.4.3 TORQUE\_CONSTANT

```
constexpr double pros_adapters::ProsV5Motor::TORQUE_CONSTANT {(2.1 / 36) / 2.5} [static],  
[constexpr], [private]
```

The torque constant of the motor.

Definition at line 48 of file [ProsV5Motor.hpp](#).

```
00048 {(2.1 / 36) / 2.5};
```

#### 5.12.4.4 RESISTANCE

```
constexpr double pros_adapters::ProsV5Motor::RESISTANCE {3.2} [static], [constexpr], [private]
```

The resistance of the motor in ohms.

Definition at line 54 of file [ProsV5Motor.hpp](#).

```
00054 {3.2};
```

#### 5.12.4.5 ANGULAR\_VELOCITY\_CONSTANT

```
constexpr double pros_adapters::ProsV5Motor::ANGULAR_VELOCITY_CONSTANT {2.1} [static], [constexpr],  
[private]
```

The angular velocity constant of the motor.

Definition at line 60 of file [ProsV5Motor.hpp](#).

```
00060 {2.1};
```

#### 5.12.4.6 VELOCITY\_CONVERSION

```
constexpr double pros_adapters::ProsV5Motor::VELOCITY_CONVERSION {2 * M_PI / 60} [static],  
[constexpr], [private]
```

Converts motor velocity to radians/second.

Definition at line [66](#) of file [ProsV5Motor.hpp](#).

```
00066 {2 * M_PI / 60};
```

#### 5.12.4.7 POSITION\_CONVERSION

```
constexpr double pros_adapters::ProsV5Motor::POSITION_CONVERSION {M_PI / 25} [static], [constexpr],  
[private]
```

Converts motor position to radians.

Definition at line [72](#) of file [ProsV5Motor.hpp](#).

```
00072 {M_PI / 25};
```

#### 5.12.4.8 VOLTAGE\_CONVERSION

```
constexpr double pros_adapters::ProsV5Motor::VOLTAGE_CONVERSION {1000} [static], [constexpr],  
[private]
```

Converts input voltage to millivolts.

Definition at line [78](#) of file [ProsV5Motor.hpp](#).

```
00078 {1000};
```

#### 5.12.4.9 MAX\_MILLIVOLTS

```
constexpr int pros_adapters::ProsV5Motor::MAX_MILLIVOLTS {12000} [static], [constexpr], [private]
```

The maximum output to the motor in millivolts.

Definition at line [84](#) of file [ProsV5Motor.hpp](#).

```
00084 {12000};
```

#### 5.12.4.10 m\_motor

```
std::unique_ptr<pros::Motor> pros_adapters::ProsV5Motor::m_motor {} [private]
```

The motor being adapted.

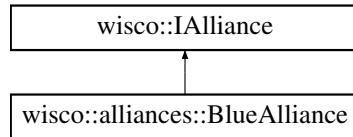
Definition at line [90](#) of file [ProsV5Motor.hpp](#).

```
00090 {};
```

## 5.13 wisco::alliances::BlueAlliance Class Reference

The blue match alliance.

Inheritance diagram for wisco::alliances::BlueAlliance:



### Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the alliance.*

### Public Member Functions inherited from [wisco::IAlliance](#)

- virtual ~[IAlliance \(\)](#)=default  
*Destroy the [IAlliance](#) object.*

### Static Private Attributes

- static constexpr char [ALLIANCE\\_NAME \[\]](#) {"BLUE"}  
*The name of the alliance.*

### 5.13.1 Detailed Description

The blue match alliance.

#### Author

Nathan Sandvig

Definition at line [28](#) of file [BlueAlliance.hpp](#).

### 5.13.2 Member Function Documentation

#### 5.13.2.1 [getName\(\)](#)

```
std::string wisco::alliances::BlueAlliance::getName ( ) [override], [virtual]
```

Get the name of the alliance.

#### Returns

std::string The name of the alliance

Implements [wisco::IAlliance](#).

Definition at line [7](#) of file [BlueAlliance.cpp](#).

```
00008 {  
00009     return ALLIANCE_NAME;  
00010 }
```

### 5.13.3 Member Data Documentation

#### 5.13.3.1 ALLIANCE\_NAME

```
constexpr char wisco::alliances::BlueAlliance::ALLIANCE_NAME[ ] { "BLUE" } [static], [constexpr], [private]
```

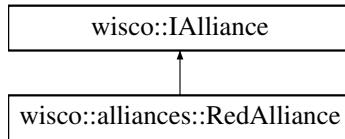
The name of the alliance.

Definition at line 35 of file [BlueAlliance.hpp](#).  
00035 {"BLUE"};

## 5.14 wisco::alliances::RedAlliance Class Reference

The red match alliance.

Inheritance diagram for wisco::alliances::RedAlliance:



### Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the alliance.*

### Public Member Functions inherited from [wisco::IAlliance](#)

- virtual ~IAlliance ()=default  
*Destroy the [IAlliance](#) object.*

### Static Private Attributes

- static constexpr char [ALLIANCE\\_NAME](#) [] {"RED"}  
*The name of the alliance.*

### 5.14.1 Detailed Description

The red match alliance.

#### Author

Nathan Sandvig

Definition at line 28 of file [RedAlliance.hpp](#).

## 5.14.2 Member Function Documentation

### 5.14.2.1 getName()

```
std::string wisco::alliances::RedAlliance::getName ( ) [override], [virtual]
```

Get the name of the alliance.

#### Returns

`std::string` The name of the alliance

Implements [wisco::IAlliance](#).

Definition at line 7 of file [RedAlliance.cpp](#).

```
00008 {
00009     return ALLIANCE_NAME;
00010 }
```

## 5.14.3 Member Data Documentation

### 5.14.3.1 ALLIANCE\_NAME

```
constexpr char wisco::alliances::RedAlliance::ALLIANCE_NAME[ ] {"RED"} [static], [constexpr],
[private]
```

The name of the alliance.

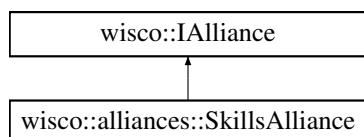
Definition at line 35 of file [RedAlliance.hpp](#).

```
00035 {"RED"};
```

## 5.15 wisco::alliances::SkillsAlliance Class Reference

The skills alliance.

Inheritance diagram for wisco::alliances::SkillsAlliance:



### Public Member Functions

- `std::string getName () override`

*Get the name of the alliance.*

## Public Member Functions inherited from [wisco::IAlliance](#)

- virtual ~[IAlliance](#) ()=default

*Destroy the [IAlliance](#) object.*

### Static Private Attributes

- static constexpr char [ALLIANCE\\_NAME](#) [] {"SKILLS"}

*The name of the alliance.*

## 5.15.1 Detailed Description

The skills alliance.

### Author

Nathan Sandvig

Definition at line 28 of file [SkillsAlliance.hpp](#).

## 5.15.2 Member Function Documentation

### 5.15.2.1 [getName\(\)](#)

```
std::string wisco::alliances::SkillsAlliance::getName ( ) [override], [virtual]
```

Get the name of the alliance.

### Returns

std::string The name of the alliance

Implements [wisco::IAlliance](#).

Definition at line 7 of file [SkillsAlliance.cpp](#).

```
00008 {  
00009     return ALLIANCE_NAME;  
00010 }
```

## 5.15.3 Member Data Documentation

### 5.15.3.1 [ALLIANCE\\_NAME](#)

```
constexpr char wisco::alliances::SkillsAlliance::ALLIANCE_NAME[ ] {"SKILLS"} [static], [constexpr], [private]
```

The name of the alliance.

Definition at line 35 of file [SkillsAlliance.hpp](#).

```
00035 {"SKILLS"};
```

## 5.16 wisco::AutonomousManager Class Reference

Manages the execution of the autonomous routine.

### Public Member Functions

- `AutonomousManager` (const std::shared\_ptr<`rtos::IClock`> &`clock`, const std::unique\_ptr<`rtos::IDelay`> &`delayer`)  
*Construct a new AutonomousManager object.*
- `void setAutonomous` (std::unique\_ptr<`IAutonomous`> &`autonomous`)  
*Set the autonomous routine.*
- `void initializeAutonomous` (std::shared\_ptr<`control::ControlSystem`> `control_system`, std::shared\_ptr<`robot::Robot`> `robot`)  
*Initialize the autonomous routine.*
- `void runAutonomous` (std::shared\_ptr<`control::ControlSystem`> `control_system`, std::shared\_ptr<`robot::Robot`> `robot`)  
*Run the autonomous routine.*

### Private Attributes

- `std::unique_ptr<IAutonomous> m_autonomous {}  
The autonomous routine.`
- `std::shared_ptr<rtos::IClock> m_clock {}  
The rtos clock.`
- `std::unique_ptr<rtos::IDelay> m_delay {}  
The rtos delay.`

### 5.16.1 Detailed Description

Manages the execution of the autonomous routine.

#### Author

Nathan Sandvig

Definition at line 23 of file `AutonomousManager.hpp`.

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 AutonomousManager()

```
wisco::AutonomousManager::AutonomousManager (
    const std::shared_ptr<rtos::IClock> &clock,
    const std::unique_ptr<rtos::IDelay> &delayer)
```

Construct a new `AutonomousManager` object.

**Parameters**

<i>clock</i>	The rtos clock
<i>delayer</i>	The rtos delayer

Definition at line 5 of file AutonomousManager.cpp.

```
00006     : m_clock{clock}, m_delayer{delayer->clone()}  
00007 {  
00008  
00009 }
```

### 5.16.3 Member Function Documentation

#### 5.16.3.1 setAutonomous()

```
void wisco::AutonomousManager::setAutonomous (  
    std::unique_ptr< IAutonomous > & autonomous )
```

Set the autonomous routine.

**Parameters**

<i>autonomous</i>	The autonomous routine
-------------------	------------------------

Definition at line 11 of file AutonomousManager.cpp.

```
00012 {  
00013     m_autonomous = std::move(autonomous);  
00014 }
```

#### 5.16.3.2 initializeAutonomous()

```
void wisco::AutonomousManager::initializeAutonomous (   
    std::shared_ptr< control::ControlSystem > control_system,  
    std::shared_ptr< robot::Robot > robot )
```

Initialize the autonomous routine.

**Parameters**

<i>control_system</i>	The control system
<i>robot</i>	The robot being controlled

Definition at line 16 of file AutonomousManager.cpp.

```
00018 {  
00019     if (m_autonomous)  
00020         m_autonomous->initialize(control_system, robot);  
00021 }
```

#### 5.16.3.3 runAutonomous()

```
void wisco::AutonomousManager::runAutonomous (   
    std::shared_ptr< control::ControlSystem > control_system,  
    std::shared_ptr< robot::Robot > robot )
```

Run the autonomous routine.

#### Parameters

<i>control_system</i>	The control system
<i>robot</i>	The robot being controlled

Definition at line 23 of file AutonomousManager.cpp.

```
00025 {  
00026     if (m_autonomous)  
00027         m_autonomous->run(m_clock, m_delayer, control_system, robot);  
00028 }
```

## 5.16.4 Member Data Documentation

### 5.16.4.1 m\_autonomous

```
std::unique_ptr<IAutonomous> wisco::AutonomousManager::m_autonomous {} [private]
```

The autonomous routine.

Definition at line 30 of file AutonomousManager.hpp.

```
00030 {};
```

### 5.16.4.2 m\_clock

```
std::shared_ptr<rtos::IClock> wisco::AutonomousManager::m_clock {} [private]
```

The rtos clock.

Definition at line 36 of file AutonomousManager.hpp.

```
00036 {};
```

### 5.16.4.3 m\_delayer

```
std::unique_ptr<rtos::IDelay> wisco::AutonomousManager::m_delayer {} [private]
```

The rtos delayer.

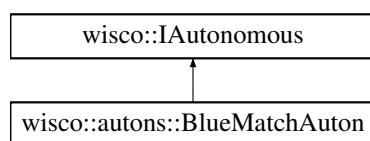
Definition at line 42 of file AutonomousManager.hpp.

```
00042 {};
```

## 5.17 wisco::autons::BlueMatchAuton Class Reference

The auton for the blue robot in matches.

Inheritance diagram for wisco::autons::BlueMatchAuton:



## Public Member Functions

- std::string `getName ()` override  
*Get the name of the autonomous.*
- void `initialize (std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot)` override  
*Initialize the autonomous.*
- void `run (std::shared_ptr< rtos::IClock > clock, std::unique_ptr< rtos::IDelay > &delayer, std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot)` override  
*Run the autonomous.*

## Public Member Functions inherited from [wisco::IAutonomous](#)

- virtual ~**IAutonomous** ()=default  
*Destroy the `IAutonomous` object.*

## Private Member Functions

- void `boomerangGoToPoint (std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot, double velocity, double x, double y, double theta)`  
*Calls the boomerang go to point command.*
- void `boomerangPause (std::shared_ptr< control::ControlSystem > control_system)`  
*Pauses the boomerang motion.*
- void `boomerangResume (std::shared_ptr< control::ControlSystem > control_system)`  
*Resumes the boomerang motion.*
- bool `boomerangTargetReached (std::shared_ptr< control::ControlSystem > control_system)`  
*Checks if the boomerang target has been reached.*
- void `odometrySetPosition (std::shared_ptr< robot::Robot > robot, double x, double y, double theta)`  
*Sets the odometry position.*
- `robot::subsystems::position::Position odometryGetPosition (std::shared_ptr< robot::Robot > robot)`  
*Gets the odometry position.*
- void `motionTurnToAngle (std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot, double velocity, double theta, control::motion::ETurnDirection direction=control::motion::ETurnDirection::AUTO)`  
*Calls the motion turn to angle command.*
- void `motionTurnToPoint (std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot, double velocity, double x, double y, control::motion::ETurnDirection direction=control::motion::ETurnDirection::AUTO)`  
*Calls the motion turn to point command.*
- void `motionPauseTurn (std::shared_ptr< control::ControlSystem > control_system)`  
*Pauses the motion turn.*
- void `motionResumeTurn (std::shared_ptr< control::ControlSystem > control_system)`  
*Resumes the motion turn.*
- bool `motionTurnTargetReached (std::shared_ptr< control::ControlSystem > control_system)`  
*Checks if the motion turn target has been reached.*

**Static Private Attributes**

- static constexpr char **AUTONOMOUS\_NAME** [] {"B\_MATCH"}  
*The name of the autonomous.*
- static constexpr char **BOOMERANG\_CONTROL\_NAME** [] {"BOOMERANG"}  
*The name of the boomerang control.*
- static constexpr char **ODOMETRY\_SUBSYSTEM\_NAME** [] {"POSITION TRACKER"}  
*The name of the odometry subsystem.*
- static constexpr char **BOOMERANG\_GO\_TO\_POSITION\_COMMAND\_NAME** [] {"GO TO POSITION"}  
*The name of the boomerang go to position command.*
- static constexpr char **BOOMERANG\_PAUSE\_COMMAND\_NAME** [] {"PAUSE"}  
*The name of the boomerang pause command.*
- static constexpr char **BOOMERANG\_RESUME\_COMMAND\_NAME** [] {"RESUME"}  
*The name of the boomerang resume command.*
- static constexpr char **ODOMETRY\_SET\_POSITION\_COMMAND\_NAME** [] {"SET POSITION"}  
*The name of the odometry set position command.*
- static constexpr char **BOOMERANG\_TARGET\_REACHED\_STATE\_NAME** [] {"TARGET REACHED"}  
*The name of the boomerang target reached state.*
- static constexpr char **ODOMETRY\_GET\_POSITION\_STATE\_NAME** [] {"GET POSITION"}  
*The name of the odometry get position state.*
- static constexpr char **MOTION\_CONTROL\_NAME** [] {"MOTION"}  
*The name of the motion control.*
- static constexpr char **MOTION\_TURN\_TO\_ANGLE\_COMMAND\_NAME** [] {"TURN TO ANGLE"}  
*The name of the motion turn to angle command.*
- static constexpr char **MOTION\_TURN\_TO\_POINT\_COMMAND\_NAME** [] {"TURN TO POINT"}  
*The name of the motion turn to point command.*
- static constexpr char **MOTION\_PAUSE\_TURN\_COMMAND\_NAME** [] {"PAUSE TURN"}  
*The name of the motion pause turn command.*
- static constexpr char **MOTION\_RESUME\_TURN\_COMMAND\_NAME** [] {"RESUME TURN"}  
*The name of the motion resume turn command.*
- static constexpr char **MOTION\_TURN\_TARGET\_REACHED\_STATE\_NAME** [] {"TURN TARGET REACHED"}  
*The name of the motion turn target reached state.*

**5.17.1 Detailed Description**

The auton for the blue robot in matches.

**Author**

Nathan Sandvig

Definition at line 31 of file [BlueMatchAuton.hpp](#).

**5.17.2 Member Function Documentation****5.17.2.1 boomerangGoToPoint()**

```
void wisco::autons::BlueMatchAuton::boomerangGoToPoint (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot,
    double velocity,
    double x,
    double y,
    double theta ) [private]
```

Calls the boomerang go to point command.

**Parameters**

<i>control_system</i>	The control system
<i>robot</i>	The robot
<i>velocity</i>	The motion velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>theta</i>	The target angle

Definition at line 7 of file BlueMatchAuton.cpp.

```
00010 {
00011     if (control_system)
00012         control_system->sendCommand(BOOMERANG_CONTROL_NAME, BOOMERANG_GO_TO_POSITION_COMMAND_NAME,
00013         &robot, velocity, x, y, theta);
```

**5.17.2.2 boomerangPause()**

```
void wisco::autons::BlueMatchAuton::boomerangPause (
    std::shared_ptr< control::ControlSystem > control_system ) [private]
```

Pauses the boomerang motion.

**Parameters**

<i>control_system</i>	The control system
-----------------------	--------------------

Definition at line 15 of file BlueMatchAuton.cpp.

```
00016 {
00017     if (control_system)
00018         control_system->sendCommand(BOOMERANG_CONTROL_NAME, BOOMERANG_PAUSE_COMMAND_NAME);
00019 }
```

**5.17.2.3 boomerangResume()**

```
void wisco::autons::BlueMatchAuton::boomerangResume (
    std::shared_ptr< control::ControlSystem > control_system ) [private]
```

Resumes the boomerang motion.

**Parameters**

<i>control_system</i>	The control system
-----------------------	--------------------

Definition at line 21 of file BlueMatchAuton.cpp.

```
00022 {
00023     if (control_system)
00024         control_system->sendCommand(BOOMERANG_CONTROL_NAME, BOOMERANG_RESUME_COMMAND_NAME);
00025 }
```

**5.17.2.4 boomerangTargetReached()**

```
bool wisco::autons::BlueMatchAuton::boomerangTargetReached (
    std::shared_ptr< control::ControlSystem > control_system ) [private]
```

Checks if the boomerang target has been reached.

#### Parameters

<i>control_system</i>	The control system
-----------------------	--------------------

#### Returns

**true** The boomerang target has been reached  
**false** The boomerang target has not been reached

Definition at line 27 of file [BlueMatchAuton.cpp](#).

```
00028 {
00029     bool target_reached{};
00030     if (control_system)
00031     {
00032         bool* result{static_cast<bool*>(control_system->getState(BOOMERANG_CONTROL_NAME,
00033             BOOMERANG_TARGET_REACHED_STATE_NAME))};
00034         if (result)
00035         {
00036             target_reached = *result;
00037             delete result;
00038         }
00039     return target_reached;
00040 }
```

### 5.17.2.5 odometrySetPosition()

```
void wisco::autons::BlueMatchAuton::odometrySetPosition (
    std::shared_ptr< robot::Robot > robot,
    double x,
    double y,
    double theta ) [private]
```

Sets the odometry position.

#### Parameters

<i>robot</i>	The robot
<i>x</i>	The x-coordinate
<i>y</i>	The y-coordinate
<i>theta</i>	The angle

Definition at line 42 of file [BlueMatchAuton.cpp](#).

```
00044 {
00045     if (robot)
00046     {
00047         robot::subsystems::position::Position position{x, y, theta};
00048         robot->sendCommand(ODOMETRY_SUBSYSTEM_NAME, ODOMETRY_SET_POSITION_COMMAND_NAME, position);
00049     }
00050 }
```

### 5.17.2.6 odometryGetPosition()

```
robot::subsystems::position::Position wisco::autons::BlueMatchAuton::odometryGetPosition (
    std::shared_ptr< robot::Robot > robot ) [private]
```

Gets the odometry position.

**Parameters**

<i>robot</i>	The robot
--------------	-----------

**Returns**

`robot::subsystems::position::Position` The odometry position

Definition at line 52 of file [BlueMatchAuton.cpp](#).

```
00053 {
00054     robot::subsystems::position::Position position{};
00055
00056     if (robot)
00057     {
00058         robot::subsystems::position::Position*
00059             result{static_cast<robot::subsystems::position::Position*>(robot->getState(ODOMETRY_SUBSYSTEM_NAME,
00060                                         ODOMETRY_GET_POSITION_STATE_NAME))};
00061         if (result)
00062         {
00063             position = *result;
00064             delete result;
00065         }
00066     }
00067 }
```

**5.17.2.7 motionTurnToAngle()**

```
void wisco::autons::BlueMatchAuton::motionTurnToAngle (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot,
    double velocity,
    double theta,
    control::motion::ETurnDirection direction = control::motion::ETurnDirection::AUTO
) [private]
```

Calls the motion turn to angle command.

**Parameters**

<i>control_system</i>	The control system
<i>robot</i>	The robot
<i>velocity</i>	The turn velocity
<i>theta</i>	The target angle
<i>direction</i>	The turn direction (default auto)

Definition at line 69 of file [BlueMatchAuton.cpp](#).

```
00073 {
00074     if (control_system)
00075         control_system->sendCommand(MOTION_CONTROL_NAME, MOTION_TURN_TO_ANGLE_COMMAND_NAME, &robot,
00076                                     velocity, theta, direction);
00076 }
```

**5.17.2.8 motionTurnToPoint()**

```
void wisco::autons::BlueMatchAuton::motionTurnToPoint (
    std::shared_ptr< control::ControlSystem > control_system,
```

```

    std::shared_ptr< robot::Robot > robot,
    double velocity,
    double x,
    double y,
    control::motion::ETurnDirection direction = control::motion::ETurnDirection::AUTO
) [private]

```

Calls the motion turn to point command.

#### Parameters

<i>control_system</i>	The control system
<i>robot</i>	The robot
<i>velocity</i>	The turn velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>direction</i>	The turn direction (default auto)

Definition at line 78 of file [BlueMatchAuton.cpp](#).

```

00082 {
00083     if (control_system)
00084         control_system->sendCommand(MOTION_CONTROL_NAME, MOTION_TURN_TO_POINT_COMMAND_NAME, &robot,
00085             velocity, x, y, direction);
00085 }

```

#### 5.17.2.9 motionPauseTurn()

```

void wisco::autons::BlueMatchAuton::motionPauseTurn (
    std::shared_ptr< control::ControlSystem > control_system ) [private]

```

Pauses the motion turn.

#### Parameters

<i>control_system</i>	The control system
-----------------------	--------------------

Definition at line 87 of file [BlueMatchAuton.cpp](#).

```

00088 {
00089     if (control_system)
00090         control_system->sendCommand(MOTION_CONTROL_NAME, MOTION_PAUSE_TURN_COMMAND_NAME);
00091 }

```

#### 5.17.2.10 motionResumeTurn()

```

void wisco::autons::BlueMatchAuton::motionResumeTurn (
    std::shared_ptr< control::ControlSystem > control_system ) [private]

```

Resumes the motion turn.

#### Parameters

<i>control_system</i>	The control system
-----------------------	--------------------

Definition at line 93 of file [BlueMatchAuton.cpp](#).

```
00094 {
00095     if (control_system)
00096         control_system->sendCommand(MOTION_CONTROL_NAME, MOTION_RESUME_TURN_COMMAND_NAME);
00097 }
```

### 5.17.2.11 motionTurnTargetReached()

```
bool wisco::autons::BlueMatchAuton::motionTurnTargetReached (
    std::shared_ptr< control::ControlSystem > control_system ) [private]
```

Checks if the motion turn target has been reached.

#### Parameters

<i>control_system</i>	The control system
-----------------------	--------------------

#### Returns

true The motion turn target has been reached  
 false The motion turn target has not been reached

Definition at line 99 of file [BlueMatchAuton.cpp](#).

```
00100 {
00101     bool target_reached{};
00102     if (control_system)
00103     {
00104         bool* result{static_cast<bool*>(control_system->getState(MOTION_CONTROL_NAME,
MOTION_TURN_TARGET_REACHED_STATE_NAME))};
00105         if (result)
00106         {
00107             target_reached = *result;
00108             delete result;
00109         }
00110     }
00111     return target_reached;
00112 }
```

### 5.17.2.12 getName()

```
std::string wisco::autons::BlueMatchAuton::getName ( ) [override], [virtual]
```

Get the name of the autonomous.

#### Returns

std::string The name of the autonomous

Implements [wisco::IAutonomous](#).

Definition at line 114 of file [BlueMatchAuton.cpp](#).

```
00115 {
00116     return AUTONOMOUS_NAME;
00117 }
```

### 5.17.2.13 initialize()

```
void wisco::autons::BlueMatchAuton::initialize (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Initialize the autonomous.

**Parameters**

<i>control_system</i>	The control system
<i>robot</i>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 119 of file [BlueMatchAuton.cpp](#).

```
00121 {
00122
00123 }
```

**5.17.2.14 run()**

```
void wisco::autons::BlueMatchAuton::run (
    std::shared_ptr< rtos::IClock > clock,
    std::unique_ptr< rtos::IDelayer > & delayer,
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Run the autonomous.

**Parameters**

<i>clock</i>	The rtos clock
<i>delayer</i>	The rtos delayer
<i>control_system</i>	The control system
<i>robot</i>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 125 of file [BlueMatchAuton.cpp](#).

```
00129 {
00130     odometrySetPosition(robot, 0, 0, 0);
00131     motionTurnToPoint(control_system, robot, 2 * M_PI, 48.0, 48.0,
00132         control::motion::ETurnDirection::CLOCKWISE);
00132 }
```

**5.17.3 Member Data Documentation****5.17.3.1 AUTONOMOUS\_NAME**

```
constexpr char wisco::autons::BlueMatchAuton::AUTONOMOUS_NAME[ ] {"B_MATCH"} [static], [constexpr],
[private]
```

The name of the autonomous.

Definition at line 38 of file [BlueMatchAuton.hpp](#).

```
00038 {"B_MATCH"};
```

### 5.17.3.2 BOOMERANG\_CONTROL\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::BOOMERANG_CONTROL_NAME[ ] { "BOOMERANG" } [static],  
[constexpr], [private]
```

The name of the boomerang control.

Definition at line 44 of file [BlueMatchAuton.hpp](#).

```
00044 {"BOOMERANG"};
```

### 5.17.3.3 ODOMETRY\_SUBSYSTEM\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::ODOMETRY_SUBSYSTEM_NAME[ ] { "POSITION TRACKER" }  
[static], [constexpr], [private]
```

The name of the odometry subsystem.

Definition at line 50 of file [BlueMatchAuton.hpp](#).

```
00050 {"POSITION TRACKER"};
```

### 5.17.3.4 BOOMERANG\_GO\_TO\_POSITION\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::BOOMERANG_GO_TO_POSITION_COMMAND_NAME[ ] { "GO TO  
POSITION" } [static], [constexpr], [private]
```

The name of the boomerang go to position command.

Definition at line 56 of file [BlueMatchAuton.hpp](#).

```
00056 {"GO TO POSITION"};
```

### 5.17.3.5 BOOMERANG\_PAUSE\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::BOOMERANG_PAUSE_COMMAND_NAME[ ] { "PAUSE" } [static],  
[constexpr], [private]
```

The name of the boomerang pause command.

Definition at line 62 of file [BlueMatchAuton.hpp](#).

```
00062 {"PAUSE"};
```

### 5.17.3.6 BOOMERANG\_RESUME\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::BOOMERANG_RESUME_COMMAND_NAME[ ] { "RESUME" } [static],  
[constexpr], [private]
```

The name of the boomerang resume command.

Definition at line 68 of file [BlueMatchAuton.hpp](#).

```
00068 {"RESUME"};
```

### 5.17.3.7 ODOMETRY\_SET\_POSITION\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::ODOMETRY_SET_POSITION_COMMAND_NAME[ ] { "SET POSITION" }  
[static], [constexpr], [private]
```

The name of the odometry set position command.

Definition at line 74 of file [BlueMatchAuton.hpp](#).

```
00074 {"SET POSITION"};
```

### 5.17.3.8 BOOMERANG\_TARGET\_REACHED\_STATE\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::BOOMERANG_TARGET_REACHED_STATE_NAME[ ] { "TARGET  
REACHED" } [static], [constexpr], [private]
```

The name of the boomerang target reached state.

Definition at line 80 of file [BlueMatchAuton.hpp](#).

```
00080 {"TARGET REACHED"};
```

### 5.17.3.9 ODOMETRY\_GET\_POSITION\_STATE\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::ODOMETRY_GET_POSITION_STATE_NAME[ ] { "GET POSITION" }  
[static], [constexpr], [private]
```

The name of the odometry get position state.

Definition at line 86 of file [BlueMatchAuton.hpp](#).

```
00086 {"GET POSITION"};
```

### 5.17.3.10 MOTION\_CONTROL\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_CONTROL_NAME[ ] { "MOTION" } [static],  
[constexpr], [private]
```

The name of the motion control.

Definition at line 92 of file [BlueMatchAuton.hpp](#).

```
00092 {"MOTION"};
```

### 5.17.3.11 MOTION\_TURN\_TO\_ANGLE\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_TURN_TO_ANGLE_COMMAND_NAME[ ] { "TURN TO  
ANGLE" } [static], [constexpr], [private]
```

The name of the motion turn to angle command.

Definition at line 98 of file [BlueMatchAuton.hpp](#).

```
00098 {"TURN TO ANGLE"};
```

### 5.17.3.12 MOTION\_TURN\_TO\_POINT\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_TURN_TO_POINT_COMMAND_NAME[ ] { "TURN TO POINT" } [static], [constexpr], [private]
```

The name of the motion turn to point command.

Definition at line 104 of file [BlueMatchAuton.hpp](#).

```
00104 {"TURN TO POINT"};
```

### 5.17.3.13 MOTION\_PAUSE\_TURN\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_PAUSE_TURN_COMMAND_NAME[ ] { "PAUSE TURN" } [static], [constexpr], [private]
```

The name of the motion pause turn command.

Definition at line 110 of file [BlueMatchAuton.hpp](#).

```
00110 {"PAUSE TURN"};
```

### 5.17.3.14 MOTION\_RESUME\_TURN\_COMMAND\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_RESUME_TURN_COMMAND_NAME[ ] { "RESUME TURN" } [static], [constexpr], [private]
```

The name of the motion resume turn command.

Definition at line 116 of file [BlueMatchAuton.hpp](#).

```
00116 {"RESUME TURN"};
```

### 5.17.3.15 MOTION\_TURN\_TARGET\_REACHED\_STATE\_NAME

```
constexpr char wisco::autons::BlueMatchAuton::MOTION_TURN_TARGET_REACHED_STATE_NAME[ ] { "TURN TARGET REACHED" } [static], [constexpr], [private]
```

The name of the motion turn target reached state.

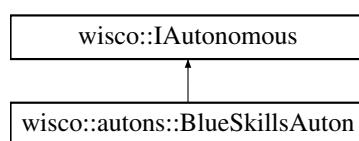
Definition at line 122 of file [BlueMatchAuton.hpp](#).

```
00122 {"TURN TARGET REACHED"};
```

## 5.18 wisco::autons::BlueSkillsAuton Class Reference

The auton for the blue robot in skills.

Inheritance diagram for wisco::autons::BlueSkillsAuton:



**Public Member Functions**

- std::string [getName \(\)](#) override  
*Get the name of the autonomous.*
- void [initialize \(std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Initialize the autonomous.*
- void [run \(std::shared\\_ptr< rtos::IClock > clock, std::unique\\_ptr< rtos::IDelay > &delayer, std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Run the autonomous.*

**Public Member Functions inherited from [wisco::IAutonomous](#)**

- virtual ~[IAutonomous \(\)](#)=default  
*Destroy the [IAutonomous](#) object.*

**Static Private Attributes**

- static constexpr char [AUTONOMOUS\\_NAME \[\]](#) {"B\_SKILLS"}  
*The name of the autonomous.*

**5.18.1 Detailed Description**

The auton for the blue robot in skills.

**Author**

Nathan Sandvig

Definition at line 26 of file [BlueSkillsAuton.hpp](#).

**5.18.2 Member Function Documentation****5.18.2.1 getName()**

```
std::string wisco::autons::BlueSkillsAuton::getName ( ) [override], [virtual]
```

Get the name of the autonomous.

**Returns**

std::string The name of the autonomous

Implements [wisco::IAutonomous](#).

Definition at line 7 of file [BlueSkillsAuton.cpp](#).

```
00008 {
00009     return AUTONOMOUS_NAME;
0010 }
```

**5.18.2.2 initialize()**

```
void wisco::autons::BlueSkillsAuton::initialize (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot) [override], [virtual]
```

Initialize the autonomous.

**Parameters**

<i>control_system</i>	The control system
<i>robot</i>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 12 of file [BlueSkillsAuton.cpp](#).

```
00014 {
00015
00016 }
```

**5.18.2.3 run()**

```
void wisco::autons::BlueSkillsAuton::run (
    std::shared_ptr< rtos::IClock > clock,
    std::unique_ptr< rtos::IDelayer > & delayer,
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Run the autonomous.

**Parameters**

<i>clock</i>	The rtos clock
<i>delayer</i>	The rtos delayer
<i>control_system</i>	The control system
<i>robot</i>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 18 of file [BlueSkillsAuton.cpp](#).

```
00022 {
00023
00024 }
```

**5.18.3 Member Data Documentation****5.18.3.1 AUTONOMOUS\_NAME**

```
constexpr char wisco::autons::BlueSkillsAuton::AUTONOMOUS_NAME[ ] {"B_SKILLS"} [static], [constexpr],
[private]
```

The name of the autonomous.

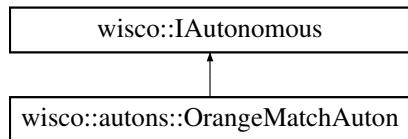
Definition at line 33 of file [BlueSkillsAuton.hpp](#).

```
00033 {"B_SKILLS"};
```

## 5.19 wisco::autons::OrangeMatchAuton Class Reference

The auton for the orange robot in matches.

Inheritance diagram for wisco::autons::OrangeMatchAuton:



### Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the autonomous.*
- void [initialize \(std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Initialize the autonomous.*
- void [run \(std::shared\\_ptr< rtos::IClock > clock, std::unique\\_ptr< rtos::IDelay > &delayer, std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Run the autonomous.*

### Public Member Functions inherited from [wisco::IAutonomous](#)

- virtual ~[IAutonomous \(\)](#)=default  
*Destroy the [IAutonomous](#) object.*

### Static Private Attributes

- static constexpr char [AUTONOMOUS\\_NAME \[\]](#) {"O\_MATCH"}  
*The name of the autonomous.*

### 5.19.1 Detailed Description

The auton for the orange robot in matches.

#### Author

Nathan Sandvig

Definition at line [26](#) of file [OrangeMatchAuton.hpp](#).

## 5.19.2 Member Function Documentation

### 5.19.2.1 getName()

```
std::string wisco::autons::OrangeMatchAuton::getName ( ) [override], [virtual]
```

Get the name of the autonomous.

#### Returns

`std::string` The name of the autonomous

Implements [wisco::IAutonomous](#).

Definition at line 7 of file [OrangeMatchAuton.cpp](#).

```
00008 {
00009     return AUTONOMOUS_NAME;
00010 }
```

### 5.19.2.2 initialize()

```
void wisco::autons::OrangeMatchAuton::initialize (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Initialize the autonomous.

#### Parameters

<code>control_system</code>	The control system
<code>robot</code>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 12 of file [OrangeMatchAuton.cpp](#).

```
00014 {
00015
00016 }
```

### 5.19.2.3 run()

```
void wisco::autons::OrangeMatchAuton::run (
    std::shared_ptr< rtos::IClock > clock,
    std::unique_ptr< rtos::IDelayer > & delayer,
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Run the autonomous.

#### Parameters

<code>clock</code>	The rtos clock
<code>delayer</code>	The rtos delayer
<code>control_system</code>	The control system
<code>robot</code>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 18 of file [OrangeMatchAuton.cpp](#).

```
00022 {
00023
00024 }
```

### 5.19.3 Member Data Documentation

#### 5.19.3.1 AUTONOMOUS\_NAME

```
constexpr char wisco::autons::OrangeMatchAuton::AUTONOMOUS_NAME[] {"O_MATCH"} [static], [constexpr], [private]
```

The name of the autonomous.

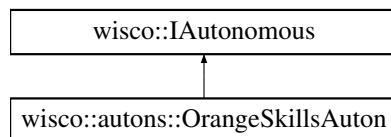
Definition at line 33 of file [OrangeMatchAuton.hpp](#).

```
00033 {"O_MATCH"};
```

## 5.20 **wisco::autons::OrangeSkillsAuton Class Reference**

The auton for the orange robot in skills.

Inheritance diagram for **wisco::autons::OrangeSkillsAuton**:



### Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the autonomous.*
- void [initialize \(std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Initialize the autonomous.*
- void [run \(std::shared\\_ptr< rtos::IClock > clock, std::unique\\_ptr< rtos::IDelayer > &delayer, std::shared\\_ptr< control::ControlSystem > control\\_system, std::shared\\_ptr< robot::Robot > robot\)](#) override  
*Run the autonomous.*

### Public Member Functions inherited from [wisco::IAutonomous](#)

- virtual ~[IAutonomous \(\)](#)=default  
*Destroy the [IAutonomous](#) object.*

## Static Private Attributes

- static constexpr char `AUTONOMOUS_NAME` [] {"R\_SKILLS"}  
*The name of the autonomous.*

### 5.20.1 Detailed Description

The auton for the orange robot in skills.

#### Author

Nathan Sandvig

Definition at line 26 of file [OrangeSkillsAuton.hpp](#).

### 5.20.2 Member Function Documentation

#### 5.20.2.1 getName()

```
std::string wisco::autons::OrangeSkillsAuton::getName ( ) [override], [virtual]
```

Get the name of the autonomous.

#### Returns

`std::string` The name of the autonomous

Implements [wisco::IAutonomous](#).

Definition at line 7 of file [OrangeSkillsAuton.cpp](#).

```
00008 {
00009     return AUTONOMOUS_NAME;
00010 }
```

#### 5.20.2.2 initialize()

```
void wisco::autons::OrangeSkillsAuton::initialize (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Initialize the autonomous.

#### Parameters

<code>control_system</code>	The control system
<code>robot</code>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 12 of file [OrangeSkillsAuton.cpp](#).

```
00014 {
00015
00016 }
```

### 5.20.2.3 run()

```
void wisco::autons::OrangeSkillsAuton::run (
    std::shared_ptr< rtos::IClock > clock,
    std::unique_ptr< rtos::IDelayer > & delayer,
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [override], [virtual]
```

Run the autonomous.

#### Parameters

<i>clock</i>	The rtos clock
<i>delayer</i>	The rtos delayer
<i>control_system</i>	The control system
<i>robot</i>	The robot

Implements [wisco::IAutonomous](#).

Definition at line 18 of file [OrangeSkillsAuton.cpp](#).

```
00022 {
00023
00024 }
```

## 5.20.3 Member Data Documentation

### 5.20.3.1 AUTONOMOUS\_NAME

```
constexpr char wisco::autons::OrangeSkillsAuton::AUTONOMOUS_NAME[ ] {"R_SKILLS"} [static],
[constexpr], [private]
```

The name of the autonomous.

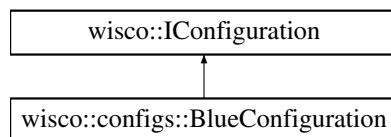
Definition at line 33 of file [OrangeSkillsAuton.hpp](#).

```
00033 {"R_SKILLS"};
```

## 5.21 wisco::configs::BlueConfiguration Class Reference

The hardware configuration of the blue robot.

Inheritance diagram for wisco::configs::BlueConfiguration:



## Public Member Functions

- std::string `getName ()` override  
*Get the name of the configuration.*
- std::shared\_ptr<control::ControlSystem> `buildControlSystem ()` override  
*Build a control system using this configuration.*
- std::shared\_ptr<user::IController> `buildController ()` override  
*Build a controller using this configuration.*
- std::shared\_ptr<robot::Robot> `buildRobot ()` override  
*Build a robot using this configuration.*

## Public Member Functions inherited from `wisco::IConfiguration`

- virtual ~`IConfiguration` ()=default  
*Destroy the `IConfiguration` object.*

## Static Private Attributes

- static constexpr char `CONFIGURATION_NAME []` {"BLUE"}  
*The name of the configuration.*
- static constexpr int8\_t `ODOMETRY_HEADING_PORT` {17}  
*The port for the odometry heading sensor.*
- static constexpr double `ODOMETRY_HEADING_TUNING_CONSTANT` {1.014}  
*The tuning constant for the odometry heading sensor.*
- static constexpr int8\_t `ODOMETRY_LINEAR_PORT` {15}  
*The port for the odometry linear distance tracking sensor.*
- static constexpr double `ODOMETRY_LINEAR_RADIUS` {1.22}  
*The radius of the odometry linear distance tracking wheel.*
- static constexpr double `ODOMETRY_LINEAR_OFFSET` {-3.35}  
*The offset of the odometry linear distance tracking wheel.*
- static constexpr int8\_t `ODOMETRY_STRAFE_PORT` {16}  
*The port for the odometry strafe distance tracking sensor.*
- static constexpr double `ODOMETRY_STRAFE_RADIUS` {-1.22}  
*The radius of the odometry strafe distance tracking wheel.*
- static constexpr double `ODOMETRY_STRAFE_OFFSET` {4.6}  
*The offset of the odometry strafe distance tracking wheel.*
- static constexpr int8\_t `RESETTER_DISTANCE_PORT` {14}  
*The port for the resetter distance sensor.*
- static constexpr double `RESETTER_DISTANCE_CONSTANT` {1.0}  
*The tuning constant for the resetter distance sensor.*
- static constexpr double `RESETTER_DISTANCE_OFFSET` {}  
*The tuning offset for the resetter distance sensor.*
- static constexpr double `RESETTER_OFFSET_X` {}  
*The x-offset of the resetter.*
- static constexpr double `RESETTER_OFFSET_Y` {}  
*The y-offset of the resetter.*
- static constexpr double `RESETTER_OFFSET_THETA` {}  
*The angle-offset of the resetter.*
- static constexpr bool `DRIVE_KINEMATIC` {false}  
*Whether to use the kinematic drive model or not.*

- static constexpr double `DRIVE_VELOCITY_PROFILE_JERK_RATE` {20.0}  
*The jerk rate of the drive velocity profile.*
- static constexpr double `DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION` {5.0}  
*The maximum acceleration of the drive velocity profile.*
- static constexpr int8\_t `DRIVE_LEFT_MOTOR_1_PORT` {-1}  
*The first left drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_LEFT_MOTOR_1_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The first left drive motor gearset.*
- static constexpr int8\_t `DRIVE_LEFT_MOTOR_2_PORT` {-2}  
*The second left drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_LEFT_MOTOR_2_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The second left drive motor gearset.*
- static constexpr int8\_t `DRIVE_LEFT_MOTOR_3_PORT` {3}  
*The third left drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_LEFT_MOTOR_3_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The third left drive motor gearset.*
- static constexpr int8\_t `DRIVE_LEFT_MOTOR_4_PORT` {4}  
*The fourth left drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_LEFT_MOTOR_4_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The fourth left drive motor gearset.*
- static constexpr int8\_t `DRIVE_RIGHT_MOTOR_1_PORT` {-7}  
*The first right drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_RIGHT_MOTOR_1_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The first right drive motor gearset.*
- static constexpr int8\_t `DRIVE_RIGHT_MOTOR_2_PORT` {-8}  
*The second right drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_RIGHT_MOTOR_2_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The second right drive motor gearset.*
- static constexpr int8\_t `DRIVE_RIGHT_MOTOR_3_PORT` {9}  
*The third right drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_RIGHT_MOTOR_3_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The third right drive motor gearset.*
- static constexpr int8\_t `DRIVE_RIGHT_MOTOR_4_PORT` {10}  
*The fourth right drive motor port.*
- static constexpr pros::v5::MotorGears `DRIVE_RIGHT_MOTOR_4_GEARSET` {pros::E\_MOTOR\_GEARSET\_06}  
*The fourth right drive motor gearset.*
- static constexpr double `DRIVE_VELOCITY_TO_VOLTAGE` {12.0 / 60.0}  
*The conversion from velocity to voltage on the drive Current calculation = 12 volts to 16 inches per second.*
- static constexpr double `DRIVE_MASS` {9.89}  
*The mass of the drive.*
- static constexpr double `DRIVE_RADIUS` {6.0625}  
*The radius of the drive.*
- static constexpr double `DRIVE_MOMENT_OF_INERTIA` {19.887 \* `DRIVE_RADIUS` \* `DRIVE_MASS`}  
*The moment of inertia of the drive.*

- static constexpr double **DRIVE\_GEAR\_RATIO** {600.0 / 331.4}  
*The gear ratio of the drive.*
- static constexpr double **DRIVE\_WHEEL\_RADIUS** {3.25 \* 2.54 / 100}  
*The wheel radius of the drive.*
- static constexpr double **INTAKE\_KP** {1.0}  
*The KP for the intake PID.*
- static constexpr double **INTAKE\_KI** {0.0}  
*The KI for the intake PID.*
- static constexpr double **INTAKE\_KD** {0.0}  
*The KD for the intake PID.*
- static constexpr int8\_t **INTAKE\_MOTOR\_1\_PORT** {-12}  
*The first intake motor port.*
- static constexpr pros::v5::MotorGears **INTAKE\_MOTOR\_1\_GEARSET** {pros::E\_MOTOR\_GEARSET\_06}  
*The first intake motor gearset.*
- static constexpr int8\_t **INTAKE\_MOTOR\_2\_PORT** {19}  
*The second intake motor port.*
- static constexpr pros::v5::MotorGears **INTAKE\_MOTOR\_2\_GEARSET** {pros::E\_MOTOR\_GEARSET\_06}  
*The second intake motor gearset.*
- static constexpr double **INTAKE\_ROLLER\_RADIUS** {1}  
*The radius of the intake roller.*
- static constexpr int8\_t **BALL\_DETECTOR\_DISTANCE\_PORT** {5}  
*The port for the ball detector distance sensor.*
- static constexpr double **BALL\_DETECTOR\_DISTANCE\_CONSTANT** {1.0}  
*The tuning constant for the ball detector distance sensor.*
- static constexpr double **BALL\_DETECTOR\_DISTANCE\_OFFSET** {}  
*The tuning offset for the ball detector distance sensor.*
- static constexpr double **BOOMERANG\_LINEAR\_KP** {11.0}  
*The proportional constant for the boomerang linear pid controller.*
- static constexpr double **BOOMERANG\_LINEAR\_KI** {}  
*The integral constant for the boomerang linear pid controller.*
- static constexpr double **BOOMERANG\_LINEAR\_KD** {640.0}  
*The derivative constant for the boomerang linear pid controller.*
- static constexpr double **BOOMERANG\_ROTATIONAL\_KP** {320.0}  
*The proportional constant for the boomerang rotational pid controller.*
- static constexpr double **BOOMERANG\_ROTATIONAL\_KI** {}  
*The integral constant for the boomerang rotational pid controller.*
- static constexpr double **BOOMERANG\_ROTATIONAL\_KD** {4800.0}  
*The derivative constant for the boomerang rotational pid controller.*
- static constexpr double **BOOMERANG\_LEAD** {0.12}  
*The lead ratio for the boomerang controller.*
- static constexpr double **BOOMERANG\_TARGET\_TOLERANCE** {1.0}  
*The target tolerance for the boomerang controller.*
- static constexpr double **ELEVATOR\_KP** {6.0}  
*The KP for the elevator PID.*
- static constexpr double **ELEVATOR\_KI** {0.0}  
*The KI for the elevator PID.*
- static constexpr double **ELEVATOR\_KD** {0.0}  
*The KD for the elevator PID.*
- static constexpr int8\_t **ELEVATOR\_MOTOR\_1\_PORT** {-11}  
*The first elevator motor port.*
- static constexpr pros::v5::MotorGears **ELEVATOR\_MOTOR\_1\_GEARSET** {pros::E\_MOTOR\_GEARSET\_18}

- static constexpr int8\_t **ELEVATOR\_MOTOR\_2\_PORT** {20}
 

*The first elevator motor gearset.*
- static constexpr pros::v5::MotorGears **ELEVATOR\_MOTOR\_2\_GEARSET** {pros::E\_MOTOR\_GEARSET\_18}
 

*The second elevator motor port.*
- static constexpr int8\_t **ELEVATOR\_ROTATION\_SENSOR\_PORT** {}
 

*The second elevator motor gearset.*
- static constexpr int8\_t **ELEVATOR\_DISTANCE\_PORT** {}
 

*The elevator rotation sensor port.*
- static constexpr double **ELEVATOR\_INCHES\_PER\_RADIAN** {17.0 / (0.8 \* 2 \* M\_PI)}
 

*The number of inches moved per radian on the elevator.*
- static constexpr int8\_t **ELEVATOR\_DISTANCE\_CONSTANT** {}
 

*The elevator distance sensor port.*
- static constexpr double **ELEVATOR\_DISTANCE\_OFFSET** {}
 

*The elevator distance sensor tuning constant.*
- static constexpr char **HANG\_CLAW\_PISTON\_1\_PORT** {}
 

*The first hang claw piston port.*
- static constexpr bool **HANG\_CLAW\_PISTON\_1\_EXTENDED\_STATE** {}
 

*The first hang claw piston's extended state.*
- static constexpr bool **HANG\_CLAW\_CLOSED\_STATE** {}
 

*The hang claw piston state when the claw is closed.*
- static constexpr char **HANG\_ARM\_PISTON\_1\_PORT** {}
 

*The first hang arm piston port.*
- static constexpr bool **HANG\_ARM\_PISTON\_1\_EXTENDED\_STATE** {}
 

*The first hang arm piston's extended state.*
- static constexpr bool **HANG\_ARM\_UP\_STATE** {}
 

*The hang arm piston state when the arm is up.*
- static constexpr char **HANG\_WINCH\_PISTON\_1\_PORT** {}
 

*The first hang winch piston port.*
- static constexpr bool **HANG\_WINCH\_PISTON\_1\_EXTENDED\_STATE** {}
 

*The first hang winch piston's extended state.*
- static constexpr bool **HANG\_WINCH\_ENGAGED\_STATE** {}
 

*The hang winch piston state when the winch is engaged.*
- static constexpr double **LOADER\_KP** {6.0}
 

*The KP for the loader PID.*
- static constexpr double **LOADER\_KI** {0.0}
 

*The KI for the loader PID.*
- static constexpr double **LOADER\_KD** {0.0}
 

*The KD for the loader PID.*
- static constexpr int8\_t **LOADER\_MOTOR\_1\_PORT** {-11}
 

*The first loader motor port.*
- static constexpr pros::v5::MotorGears **LOADER\_MOTOR\_1\_GEARSET** {pros::E\_MOTOR\_GEARSET\_36}
 

*The first loader motor gearset.*
- static constexpr double **LOADER\_LOADED\_POSITION** {5 \* M\_PI / 6}
 

*The loader position when loaded.*
- static constexpr double **LOADER\_READY\_POSITION** {0}
 

*The loader position when ready.*
- static constexpr double **LOADER\_POSITION\_TOLERANCE** {M\_PI / 18}
 

*The loader position tolerance.*
- static constexpr double **TURN\_KP** {160.0}
 

*The proportional constant for the turn pid controller.*

- static constexpr double `TURN_KI` {}
 

*The integral constant for the turn pid controller.*
- static constexpr double `TURN_KD` {12000.0}
 

*The derivative constant for the turn pid controller.*
- static constexpr double `TURN_TARGET_TOLERANCE` { $1.0 * M\_PI / 180$ }
 

*The target tolerance for the turn controller.*
- static constexpr double `TURN_TARGET_VELOCITY` { $1.0 * M\_PI / 180$ }
 

*The target velocity for the turn controller.*
- static constexpr int8\_t `UMBRELLA_PI��_1_PORT` {}
 

*The first umbrella piston port.*
- static constexpr bool `UMBRELLA_PI��_1_EXTENDED_STATE` {}
 

*The first umbrella piston's extended state.*
- static constexpr bool `UMBRELLA_OUT_STATE` {}
 

*The umbrella piston state when the umbrella is out.*
- static constexpr char `LEFT_WING_PI��_1_PORT` {}
 

*The first left wing piston port.*
- static constexpr bool `LEFT_WING_PI��_1_EXTENDED_STATE` {}
 

*The first left wing piston's extended state.*
- static constexpr char `RIGHT_WING_PI��_1_PORT` {}
 

*The first right wing piston port.*
- static constexpr bool `RIGHT_WING_PI��_1_EXTENDED_STATE` {}
 

*The first right wing piston's extended state.*
- static constexpr bool `WINGS_OUT_STATE` {}
 

*The wing piston state when the wings are out.*

### 5.21.1 Detailed Description

The hardware configuration of the blue robot.

#### Author

Nathan Sandvig

Definition at line 84 of file `BlueConfiguration.hpp`.

### 5.21.2 Member Function Documentation

#### 5.21.2.1 `getName()`

```
std::string wisco::configs::BlueConfiguration::getName ( ) [override], [virtual]
```

Get the name of the configuration.

#### Returns

`std::string` The name of the configuration

Implements `wisco::IConfiguration`.

Definition at line 7 of file `BlueConfiguration.cpp`.

```
00008 {
00009     return CONFIGURATION_NAME;
00010 }
```

### 5.21.2.2 buildControlSystem()

```
std::shared_ptr< control::ControlSystem > wisco::configs::BlueConfiguration::buildControlSystem ( ) [override], [virtual]
```

Build a control system using this configuration.

#### Returns

`std::shared_ptr<control::ControlSystem>` The control system built by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 12 of file [BlueConfiguration.cpp](#).

```
00013 {
00014     std::shared_ptr<control::ControlSystem>
00015         control_system{std::make_shared<control::ControlSystem>()};
00016     std::unique_ptr<rtos::IClock> pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00017     std::unique_ptr<rtos::IDelay> pros_delayer{std::make_unique<pros_adapters::ProsDelay>()};
00018
00019     wisco::control::boomerang::PIDBoomerangBuilder pid_boomerang_builder{};
00020     std::unique_ptr<rtos::IMutex> boomerang_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00021     std::unique_ptr<rtos::ITask> boomerang_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00022     wisco::control::PID boomerang_linear_pid{pros_clock, BOOMERANG_LINEAR_KP, BOOMERANG_LINEAR_KI,
00023         BOOMERANG_LINEAR_KD};
00023     wisco::control::PID boomerang_rotational_pid{pros_clock, BOOMERANG_ROTATIONAL_KP,
00024         BOOMERANG_ROTATIONAL_KI, BOOMERANG_ROTATIONAL_KD};
00025     std::unique_ptr<wisco::control::boomerang::IBoomerang> pid_boomerang
00026     {
00027         pid_boomerang_builder.
00028         withDelayer(pros_delayer)->
00029         withMutex(boomerang_pros_mutex)->
00030         withTask(boomerang_pros_task)->
00031         withLinearPID(boomerang_linear_pid)->
00032         withRotationalPID(boomerang_rotational_pid)->
00033         withLead(BOOMERANG_LEAD)->
00034         withTargetTolerance(BOOMERANG_TARGET_TOLERANCE)->
00035         build()
00036     };
00037     std::unique_ptr<wisco::control::AControl>
00038         boomerang_control{std::make_unique<wisco::control::boomerang::BoomerangControl>(pid_boomerang)};
00039     control_system->addControl(boomerang_control);
00040
00041     wisco::control::motion::PITurnBuilder pid_turn_builder{};
00042     std::unique_ptr<rtos::IMutex> turn_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00043     std::unique_ptr<rtos::ITask> turn_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00044     wisco::control::PID turn_pid{pros_clock, TURN_KP, TURN_KI, TURN_KD};
00045     std::unique_ptr<wisco::control::motion::ITurn> pid_turn
00046     {
00047         pid_turn_builder.
00048         withDelayer(pros_delayer)->
00049         withMutex(turn_pros_mutex)->
00050         withTask(turn_pros_task)->
00051         withPID(turn_pid)->
00052         withTargetTolerance(TURN_TARGET_TOLERANCE)->
00053         withTargetVelocity(TURN_TARGET_VELOCITY)->
00054         build()
00055     };
00056     std::unique_ptr<wisco::control::AControl>
00057         motion_control{std::make_unique<wisco::control::motion::MotionControl>(pid_turn)};
00058     control_system->addControl(motion_control);
00059
00060     return control_system;
00061 }
```

### 5.21.2.3 buildController()

```
std::shared_ptr< user::IController > wisco::configs::BlueConfiguration::buildController ( )
[override], [virtual]
```

Build a controller using this configuration.

**Returns**

`std::shared_ptr<user::IController>` The controller build by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 60 of file [BlueConfiguration.cpp](#).

```
00061 {
00062     std::unique_ptr<pros::Controller>
00063         pros_controller{std::make_unique<pros::Controller>(pros::E_CONTROLLER_MASTER)};
00064     std::shared_ptr<user::IController>
00065         pros_controller_controller{std::make_shared<pros_adapters::ProsController>(pros_controller)};
00066     return pros_controller_controller;
00067 }
```

**5.21.2.4 buildRobot()**

```
std::shared_ptr< robot::Robot > wisco::configs::BlueConfiguration::buildRobot () [override],  
[virtual]
```

Build a robot using this configuration.

**Returns**

`robot::Robot` The robot built by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 67 of file [BlueConfiguration.cpp](#).

```
00068 {
00069     std::shared_ptr<robot::Robot> robot{std::make_shared<robot::Robot>()};
00070
00071     // Odometry creation
00072     robot::subsystems::position::InertialOdometryBuilder inertial_odometry_builder{};
00073     std::unique_ptr<wisco::rtos::IClock>
00074         odometry_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00075     std::unique_ptr<wisco::rtos::IDelayLayer>
00076         odometry_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00077     std::unique_ptr<wisco::rtos::IMutex>
00078         odometry_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00079     std::unique_ptr<wisco::rtos::ITask>
00080         odometry_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00081     std::unique_ptr<pros::Imu>
00082         odometry_pros_heading{std::make_unique<pros::Imu>(ODOMETRY_HEADING_PORT)};
00083     std::unique_ptr<wisco::io::IHeadingSensor>
00084         odometry_pros_heading_sensor{std::make_unique<pros_adapters::ProsHeading>(odometry_pros_heading,
00085             ODOMETRY_HEADING_TUNING_CONSTANT)};
00086     std::unique_ptr<pros::Rotation>
00087         odometry_pros_linear_rotation{std::make_unique<pros::Rotation>(ODOMETRY_LINEAR_PORT)};
00088     std::unique_ptr<wisco::io::IRotationSensor>
00089         odometry_pros_linear_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(odometry_pros_linear_rotation)};
00090     std::unique_ptr<wisco::io::IDistanceTrackingSensor>
00091         odometry_linear_tracking_wheel{std::make_unique<wisco::hal::TrackingWheel>(odometry_pros_linear_rotation_sensor,
00092             ODOMETRY_LINEAR_RADIUS)};
00093     std::unique_ptr<pros::Rotation>
00094         odometry_pros_strafe_rotation{std::make_unique<pros::Rotation>(ODOMETRY_STRAFE_PORT)};
00095     std::unique_ptr<wisco::io::IRotationSensor>
00096         odometry_pros_strafe_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(odometry_pros_strafe_rotation)};
00097     std::unique_ptr<wisco::io::IDistanceTrackingSensor>
00098         odometry_strafe_tracking_wheel{std::make_unique<wisco::hal::TrackingWheel>(odometry_pros_strafe_rotation_sensor,
00099             ODOMETRY_STRAFE_RADIUS)};
00100     std::unique_ptr<wisco::robot::subsystems::position::IPositionTracker> inertial_odometry
00101     {
00102         inertial_odometry_builder.
00103         withClock(odometry_pros_clock)->
00104         withDelayLayer(odometry_pros_delayer)->
00105         withMutex(odometry_pros_mutex)->
00106         withTask(odometry_pros_task)->
00107         withHeadingSensor(odometry_pros_heading_sensor)->
00108         withLinearDistanceTrackingSensor(odometry_linear_tracking_wheel)->
00109         withLinearDistanceTrackingOffset(ODOMETRY_LINEAR_OFFSET)->
00110         withStrafeDistanceTrackingSensor(odometry_strafe_tracking_wheel)->
00111         withStrafeDistanceTrackingOffset(ODOMETRY_STRAFE_OFFSET)->
```

```

00097     build()
00098   };
00099   robot::subsystems::position::DistancePositionResetterBuilder distance_position_resetter_builder{};
00100   if (RESETTER_DISTANCE_PORT)
00101   {
00102     std::unique_ptr<pros::Distance>
00103       resetter_pros_distance{std::make_unique<pros::Distance>(RESETTER_DISTANCE_PORT)};
00104     std::unique_ptr<wisco::io::IDistanceSensor>
00105       resetter_pros_distance_sensor{std::make_unique<pros_adapters::ProsDistance>(resetter_pros_distance,
00106         RESETTER_DISTANCE_CONSTANT, RESETTER_DISTANCE_OFFSET)};
00107     distance_position_resetter_builder.withDistanceSensor(resetter_pros_distance_sensor);
00108   }
00109   std::unique_ptr<wisco::robot::subsystems::position::IPositionResetter> distance_position_resetter
00110   {
00111     distance_position_resetter_builder.
00112       withLocalX(RESETTER_OFFSET_X)-
00113       withLocally(RESETTER_OFFSET_Y)-
00114       withLocalTheta(RESETTER_OFFSET_THETA)-
00115       build()
00116   }
00117   std::unique_ptr<wisco::robot::ASubsystem>
00118   odometry_subsystem{std::make_unique<wisco::robot::subsystems::position::PositionSubsystem>(inertial_odometry,
00119     distance_position_resetter)};
00120   robot->addSubsystem(odometry_subsystem);
00121   // Drive creation
00122   if (DRIVE_KINEMATIC)
00123   {
00124     wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder
00125       kinematic_differential_drive_builder{};
00126     std::unique_ptr<wisco::rtos::IDelayer>
00127       drive_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00128     std::unique_ptr<wisco::rtos::IMutex>
00129       drive_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00130     std::unique_ptr<wisco::rtos::ITask>
00131       drive_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00132     std::unique_ptr<wisco::rtos::IClock>
00133       drive_left_velocity_profile_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00134     std::unique_ptr<wisco::robot::subsystems::drive::IVelocityProfile>
00135       drive_left_velocity_profile{std::make_unique<wisco::robot::subsystems::drive::CurveVelocityProfile>(drive_left_velocity,
00136         DRIVE_VELOCITY_PROFILE_JERK_RATE, DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION)};
00137     std::unique_ptr<wisco::rtos::IClock>
00138       drive_right_velocity_profile_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00139     std::unique_ptr<wisco::robot::subsystems::drive::IVelocityProfile>
00140       drive_right_velocity_profile{std::make_unique<wisco::robot::subsystems::drive::CurveVelocityProfile>(drive_right_velocity,
00141         DRIVE_VELOCITY_PROFILE_JERK_RATE, DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION)};
00142     std::unique_ptr<pros::Motor>
00143       drive_pros_left_motor_1{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_1_PORT,
00144         DRIVE_LEFT_MOTOR_1_GEARSET)};
00145     std::unique_ptr<wisco::io::IMotor>
00146       drive_pros_left_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_1)};
00147     std::unique_ptr<pros::Motor>
00148       drive_pros_left_motor_2{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_2_PORT,
00149         DRIVE_LEFT_MOTOR_2_GEARSET)};
00150     std::unique_ptr<wisco::io::IMotor>
00151       drive_pros_left_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_2)};
00152     std::unique_ptr<pros::Motor>
00153       drive_pros_left_motor_3{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_3_PORT,
00154         DRIVE_LEFT_MOTOR_3_GEARSET)};
00155     std::unique_ptr<wisco::io::IMotor>
00156       drive_pros_left_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_3)};
00157     std::unique_ptr<pros::Motor>
00158       drive_pros_left_motor_4{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_4_PORT,
00159         DRIVE_LEFT_MOTOR_4_GEARSET)};
00160     std::unique_ptr<wisco::io::IMotor>
00161       drive_pros_left_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_4)};
00162     std::unique_ptr<pros::Motor>
00163       drive_pros_right_motor_1{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_1_PORT,
00164         DRIVE_RIGHT_MOTOR_1_GEARSET)};
00165     std::unique_ptr<wisco::io::IMotor>
00166       drive_pros_right_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_1)};
00167     std::unique_ptr<pros::Motor>
00168       drive_pros_right_motor_2{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_2_PORT,
00169         DRIVE_RIGHT_MOTOR_2_GEARSET)};
00170     std::unique_ptr<wisco::io::IMotor>
00171       drive_pros_right_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_2)};
00172     std::unique_ptr<pros::Motor>
00173       drive_pros_right_motor_3{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_3_PORT,
00174         DRIVE_RIGHT_MOTOR_3_GEARSET)};
00175     std::unique_ptr<wisco::io::IMotor>
00176       drive_pros_right_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_3)};
00177     std::unique_ptr<pros::Motor>
00178       drive_pros_right_motor_4{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_4_PORT,
00179         DRIVE_RIGHT_MOTOR_4_GEARSET)};
00180     std::unique_ptr<wisco::io::IMotor>
00181       drive_pros_right_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_4)};
00182     std::unique_ptr<wisco::robot::subsystems::drive::IDifferentialDrive> differential_drive

```

```

00145      {
00146          kinematic_differential_drive_builder.
00147          withDelayLayer(drive_pros_delayLayer)->
00148          withMutex(drive_pros_mutex)->
00149          withTask(drive_pros_task)->
00150          withLeftVelocityProfile(drive_left_velocity_profile)->
00151          withRightVelocityProfile(drive_right_velocity_profile)->
00152          withLeftMotor(drive_pros_left_motor_1_motor)->
00153          withLeftMotor(drive_pros_left_motor_2_motor)->
00154          withLeftMotor(drive_pros_left_motor_3_motor)->
00155          withLeftMotor(drive_pros_left_motor_4_motor)->
00156          withRightMotor(drive_pros_right_motor_1_motor)->
00157          withRightMotor(drive_pros_right_motor_2_motor)->
00158          withRightMotor(drive_pros_right_motor_3_motor)->
00159          withRightMotor(drive_pros_right_motor_4_motor)->
00160          withMass(DRIVE_MASS)->
00161          withRadius(DRIVE_RADIUS)->
00162          withMomentOfInertia(DRIVE_MOMENT_OF_INERTIA)->
00163          withGearRatio(DRIVE_GEAR_RATIO)->
00164          withWheelRadius(DRIVE_WHEEL_RADIUS)->
00165          build()
00166      };
00167      std::unique_ptr<wisco::robot::ASubsystem>
00168      drive_subsystem(std::make_unique<wisco::robot::subsystems::drive::DifferentialDriveSubsystem>(differential_drive));
00169      robot->addSubsystem(drive_subsystem);
00170  }
00171  {
00172      wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder
00173      direct_differential_drive_builder();
00174      std::unique_ptr<pros::Motor>
00175      drive_pros_left_motor_1(std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_1_PORT,
00176      DRIVE_LEFT_MOTOR_1_GEARSET));
00177      std::unique_ptr<wisco::io::IMotor>
00178      drive_pros_left_motor_1_motor(std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_1));
00179      std::unique_ptr<pros::Motor>
00180      drive_pros_left_motor_2(std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_2_PORT,
00181      DRIVE_LEFT_MOTOR_2_GEARSET));
00182      std::unique_ptr<wisco::io::IMotor>
00183      drive_pros_left_motor_2_motor(std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_2));
00184      std::unique_ptr<pros::Motor>
00185      drive_pros_left_motor_3(std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_3_PORT,
00186      DRIVE_LEFT_MOTOR_3_GEARSET));
00187      std::unique_ptr<pros::Motor>
00188      drive_pros_left_motor_4(std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_4_PORT,
00189      DRIVE_LEFT_MOTOR_4_GEARSET));
00190      std::unique_ptr<wisco::io::IMotor>
00191      drive_pros_left_motor_4_motor(std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_4));
00192      std::unique_ptr<wisco::robot::subsystems::drive::IDifferentialDrive> differential_drive
00193      {
00194          direct_differential_drive_builder.
00195          withLeftMotor(drive_pros_left_motor_1_motor)->
00196          withLeftMotor(drive_pros_left_motor_2_motor)->
00197          withLeftMotor(drive_pros_left_motor_3_motor)->
00198          withLeftMotor(drive_pros_left_motor_4_motor)->
00199          withRightMotor(drive_pros_right_motor_1_motor)->
00200          withRightMotor(drive_pros_right_motor_2_motor)->
00201          withRightMotor(drive_pros_right_motor_3_motor)->
00202          withRightMotor(drive_pros_right_motor_4_motor)->
00203          withVelocityToVoltage(DRIVE_VELOCITY_TO_VOLTAGE)->
00204          withGearRatio(DRIVE_GEAR_RATIO)->
00205          withWheelRadius(DRIVE_WHEEL_RADIUS)->
00206          withRadius(DRIVE_RADIUS)->
00207          build()
00208      };

```

```

00206     std::unique_ptr<wisco::robot::ASubsystem>
00207     drive_subsystem{std::make_unique<wisco::robot::subsystems::drive::DifferentialDriveSubsystem>(differential_drive)};
00208   }
00209
00210   // Intake creation
00211   wisco::robot::subsystems::intake::PIDIntakeBuilder pid_intake_builder{};
00212   std::unique_ptr<wisco::rtos::IClock>
00213     intake_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00214   std::unique_ptr<wisco::rtos::IDelayLayer>
00215     intake_pros_delayer{std::make_unique<pros_adapters::ProsDelayLayer>()};
00216   std::unique_ptr<wisco::rtos::IMutex>
00217     intake_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00218   std::unique_ptr<wisco::rtos::ITask> intake_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00219   wisco::control::PID intake_pid{intake_pros_clock, INTAKE_KP, INTAKE_KI, INTAKE_KD};
00220   std::unique_ptr<pros::Motor>
00221     intake_pros_motor_1{std::make_unique<pros::Motor>(INTAKE_MOTOR_1_PORT, INTAKE_MOTOR_1_GEARSET)};
00222   std::unique_ptr<wisco::io::IMotor>
00223     intake_pros_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(intake_pros_motor_1)};
00224   std::unique_ptr<pros::Motor>
00225     intake_pros_motor_2{std::make_unique<pros::Motor>(INTAKE_MOTOR_2_PORT, INTAKE_MOTOR_2_GEARSET)};
00226   std::unique_ptr<wisco::io::IMotor>
00227     intake_pros_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(intake_pros_motor_2)};
00228   std::unique_ptr<wisco::robot::subsystems::intake::IIIntake> pid_intake
00229   {
00230     pid_intake_builder.
00231       withClock(intake_pros_clock)->
00232       withDelayLayer(intake_pros_delayer)->
00233       withMutex(intake_pros_mutex)->
00234       withTask(intake_pros_task)->
00235       withPID(intake_pid)->
00236       withMotor(intake_pros_motor_1_motor)->
00237       withMotor(intake_pros_motor_2_motor)->
00238       withRollerRadius(INTAKE_ROLLER_RADIUS)->
00239       build();
00240   };
00241   wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder
00242     distance_vision_ball_detector_builder{};
00243   if (BALL_DETECTOR_DISTANCE_PORT)
00244   {
00245     std::unique_ptr<pros::Distance>
00246       ball_detector_pros_distance{std::make_unique<pros::Distance>(BALL_DETECTOR_DISTANCE_PORT)};
00247     std::unique_ptr<wisco::io::IDistanceSensor>
00248       ball_detector_pros_distance_sensor{std::make_unique<pros_adapters::ProsDistance>(ball_detector_pros_distance,
00249         BALL_DETECTOR_DISTANCE_CONSTANT, BALL_DETECTOR_DISTANCE_OFFSET)};
00250     distance_vision_ball_detector_builder.withDistanceSensor(ball_detector_pros_distance_sensor);
00251   }
00252   std::unique_ptr<wisco::robot::subsystems::intake::IBallDetector> distance_vision_ball_detector
00253   {
00254     distance_vision_ball_detector_builder.
00255     build();
00256   };
00257   std::unique_ptr<wisco::robot::ASubsystem>
00258     intake_subsystem{std::make_unique<wisco::robot::subsystems::intake::IntakeSubsystem>(pid_intake,
00259       distance_vision_ball_detector)};
00260   robot->addSubsystem(intake_subsystem);
00261
00262   // Elevator creation
00263   wisco::robot::subsystems::elevator::PIDElevatorBuilder pid_elevator_builder{};
00264   std::unique_ptr<wisco::rtos::IClock>
00265     elevator_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00266   std::unique_ptr<wisco::rtos::IDelayLayer>
00267     elevator_pros_delayer{std::make_unique<pros_adapters::ProsDelayLayer>()};
00268   std::unique_ptr<wisco::rtos::IMutex>
00269     elevator_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00270   std::unique_ptr<wisco::rtos::ITask>
00271     elevator_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00272   wisco::control::PID elevator_pid{elevator_pros_clock, ELEVATOR_KP, ELEVATOR_KI, ELEVATOR_KD};
00273   std::unique_ptr<pros::Motor>
00274     elevator_pros_motor_1{std::make_unique<pros::Motor>(ELEVATOR_MOTOR_1_PORT, ELEVATOR_MOTOR_1_GEARSET)};
00275   std::unique_ptr<wisco::io::IMotor>
00276     elevator_pros_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(elevator_pros_motor_1)};
00277   std::unique_ptr<pros::Motor>
00278     elevator_pros_motor_2{std::make_unique<pros::Motor>(ELEVATOR_MOTOR_2_PORT, ELEVATOR_MOTOR_2_GEARSET)};
00279   std::unique_ptr<wisco::io::IMotor>
00280     elevator_pros_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(elevator_pros_motor_2)};
00281   if (ELEVATOR_ROTATION_SENSOR_PORT)
00282   {
00283     std::unique_ptr<pros::Rotation>
00284       elevator_pros_rotation{std::make_unique<pros::Rotation>(ELEVATOR_ROTATION_SENSOR_PORT)};
00285     std::unique_ptr<wisco::io::IRotationSensor>
00286       elevator_pros_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(elevator_pros_rotation)};
00287     pid_elevator_builder.withRotationSensor(elevator_pros_rotation_sensor);
00288   }
00289   std::unique_ptr<wisco::robot::subsystems::elevator::IElevator> pid_elevator
00290   {
00291     pid_elevator_builder.

```

```

00269     withClock(elevator_pros_clock)->
00270     withDelayee(elevator_pros_delayer)->
00271     withMutex(elevator_pros_mutex)->
00272     withTask(elevator_pros_task)->
00273     withPID(elevator_pid)->
00274     withMotor(elevator_pros_motor_1_motor)->
00275     withMotor(elevator_pros_motor_2_motor)->
00276     withInchesPerRadian(ELEVATOR_INCHES_PER_RADIAN))->
00277     build()
00278   };
00279   std::unique_ptr<pros::Distance>
00280     elevator_pros_distance{std::make_unique<pros::Distance>(ELEVATOR_DISTANCE_PORT)};
00281   std::unique_ptr<wisco::io::IDistanceSensor>
00282     elevator_pros_distance_sensor{std::make_unique<pros_adapters::ProsDistance>(elevator_pros_distance,
00283       ELEVATOR_DISTANCE_CONSTANT, ELEVATOR_DISTANCE_OFFSET)};
00284   std::unique_ptr<wisco::robot::ASubsystem>
00285     elevator_subsystem{std::make_unique<wisco::robot::subsystems::elevator::ElevatorSubsystem>(pid_elevator,
00286       elevator_pros_distance_sensor)};
00287   robot->addSubsystem(elevator_subsystem);
00288
00289 // Hang creation
00290 wisco::robot::subsystems::hang::PistonClawBuilder piston_claw_builder{};
00291 std::unique_ptr<pros::adi::DigitalOut>
00292   claw_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(HANG_CLAW_PISTON_1_PORT)};
00293 std::unique_ptr<wisco::io::IPiston>
00294   claw_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(claw_pros_piston_1,
00295     HANG_CLAW_PISTON_1_EXTENDED_STATE)};
00296 std::unique_ptr<wisco::robot::subsystems::hang::IClaw> piston_claw
00297 {
00298   piston_claw_builder.
00299   withPiston(claw_pros_piston_1_piston)->
00300   withClosedState(HANG_CLAW_CLOSED_STATE)->
00301   build()
00302 };
00303 wisco::robot::subsystems::hang::PistonToggleArmBuilder piston_toggle_arm_builder{};
00304 std::unique_ptr<pros::adi::DigitalOut>
00305   arm_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(HANG_ARM_PISTON_1_PORT)};
00306 std::unique_ptr<wisco::io::IPiston>
00307   arm_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(arm_pros_piston_1,
00308     HANG_ARM_PISTON_1_EXTENDED_STATE)};
00309 std::unique_ptr<wisco::robot::subsystems::hang::IToggleArm> piston_arm
00310 {
00311   piston_toggle_arm_builder.
00312   withPiston(arm_pros_piston_1_piston)->
00313   withUpState(HANG_ARM_UP_STATE)->
00314   build()
00315 };
00316 wisco::robot::subsystems::hang::PistonWinchBuilder piston_winch_builder{};
00317 std::unique_ptr<pros::adi::DigitalOut>
00318   winch_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(HANG_WINCH_PISTON_1_PORT)};
00319 std::unique_ptr<wisco::io::IPiston>
00320   winch_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(winch_pros_piston_1,
00321     HANG_WINCH_PISTON_1_EXTENDED_STATE)};
00322 std::unique_ptr<wisco::robot::subsystems::hang::IWinch> piston_winch
00323 {
00324   piston_winch_builder.
00325   withPiston(winch_pros_piston_1_piston)->
00326   withEngagedState(HANG_WINCH_ENGAGED_STATE)->
00327   build()
00328 };
00329 std::unique_ptr<wisco::robot::ASubsystem>
00330   hang_subsystem{std::make_unique<wisco::robot::subsystems::hang::HangSubsystem>(piston_claw,
00331     piston_arm, piston_winch)};
00332 robot->addSubsystem(hang_subsystem);
00333
00334 // Loader creation
00335 wisco::robot::subsystems::loader::PIDLoaderBuilder pid_loader_builder{};
00336 std::unique_ptr<wisco::rtos::IClock>
00337   loader_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00338 std::unique_ptr<wisco::rtos::IDelay>
00339   loader_pros_delayer{std::make_unique<pros_adapters::ProsDelay>()};
00340 std::unique_ptr<wisco::rtos::IMutex>
00341   loader_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00342 std::unique_ptr<wisco::rtos::ITask> loader_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00343 wisco::control::PID loader_pid{loader_pros_clock, LOADER_KP, LOADER_KI, LOADER_KD};
00344 std::unique_ptr<pros::Motor>
00345   loader_pros_motor_1{std::make_unique<pros::Motor>(LOADER_MOTOR_1_PORT, LOADER_MOTOR_1_GEARSET)};
00346 std::unique_ptr<wisco::io::IMotor>
00347   loader_pros_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(loader_pros_motor_1)};
00348 std::unique_ptr<wisco::robot::subsystems::loader::ILoader> pid_loader
00349 {
00350   pid_loader_builder.
00351   withClock(loader_pros_clock)->
00352   withDelayee(loader_pros_delayer)->
00353   withMutex(loader_pros_mutex)->
00354   withTask(loader_pros_task)->
00355   withPID(loader_pid)->

```

```

00335     withMotor(loader_pros_motor_1_motor)->
00336     withMatchLoadPosition(LOADER_LOADED_POSITION)->
00337     withReadyPosition(LOADER_READY_POSITION)->
00338     withPositionTolerance(LOADER_POSITION_TOLERANCE)->
00339     build()
00340   };
00341   std::unique_ptr<wisco::robot::ASubsystem>
00342     loader_subsystem{std::make_unique<wisco::robot::subsystems::loader::LoaderSubsystem>(pid_loader)};
00343   robot->addSubsystem(loader_subsystem);
00344 
00345   // Umbrella creation
00346   wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder piston_umbrella_builder{};
00347   std::unique_ptr<pros::adi::DigitalOut>
00348     umbrella_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(UMBRELLA_PISTON_1_PORT)};
00349   std::unique_ptr<wisco::io::IPiston>
00350     umbrella_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(umbrella_pros_piston_1,
00351     UMBRELLA_PISTON_1_EXTENDED_STATE)};
00352   std::unique_ptr<wisco::robot::subsystems::umbrella::IUmbrella> piston_umbrella
00353   {
00354     piston_umbrella_builder.
00355     withPiston(umbrella_pros_piston_1_piston)->
00356     withOutState(UMBRELLA_OUT_STATE)->
00357     build()
00358   };
00359   std::unique_ptr<wisco::robot::ASubsystem>
00360     umbrella_subsystem{std::make_unique<wisco::robot::subsystems::umbrella::UmbrellaSubsystem>(piston_umbrella)};
00361   robot->addSubsystem(umbrella_subsystem);
00362 
00363   // Wings creation
00364   wisco::robot::subsystems::wings::PistonWingsBuilder piston_wings_builder{};
00365   std::unique_ptr<pros::adi::DigitalOut>
00366     left_wing_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(LEFT_WING_PISTON_1_PORT)};
00367   std::unique_ptr<wisco::io::IPiston>
00368     left_wing_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(left_wing_pros_piston_1,
00369     LEFT_WING_PISTON_1_EXTENDED_STATE)};
00370   std::unique_ptr<pros::adi::DigitalOut>
00371     right_wing_pros_piston_1{std::make_unique<pros::adi::DigitalOut>(RIGHT_WING_PISTON_1_PORT)};
00372   std::unique_ptr<wisco::io::IPiston>
00373     right_wing_pros_piston_1_piston{std::make_unique<pros_adapters::ProsPiston>(right_wing_pros_piston_1,
00374     RIGHT_WING_PISTON_1_EXTENDED_STATE)};
00375   std::unique_ptr<wisco::robot::subsystems::wings::IWings> piston_wings
00376   {
00377     piston_wings_builder.
00378     withLeftPiston(left_wing_pros_piston_1_piston)->
00379     withRightPiston(right_wing_pros_piston_1_piston)->
00380     withOutState(WINGS_OUT_STATE)->
00381     build()
00382   };
00383   std::unique_ptr<wisco::robot::ASubsystem>
00384     wings_subsystem{std::make_unique<wisco::robot::subsystems::wings::WingsSubsystem>(piston_wings)};
00385   robot->addSubsystem(wings_subsystem);
00386 
00387   return robot;
00388 }

```

### 5.21.3 Member Data Documentation

#### 5.21.3.1 CONFIGURATION\_NAME

```
constexpr char wisco::configs::BlueConfiguration::CONFIGURATION_NAME[] {"BLUE"} [static],
[constexpr], [private]
```

The name of the configuration.

Definition at line 91 of file [BlueConfiguration.hpp](#).  
00091 {"BLUE"};

#### 5.21.3.2 ODOMETRY\_HEADING\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ODOMETRY_HEADING_PORT {17} [static],
[constexpr], [private]
```

The port for the odometry heading sensor.

Definition at line 97 of file [BlueConfiguration.hpp](#).  
00097 {17};

### 5.21.3.3 ODOMETRY\_HEADING\_TUNING\_CONSTANT

```
constexpr double wisco::configs::BlueConfiguration::ODOMETRY_HEADING_TUNING_CONSTANT {1.014}  
[static], [constexpr], [private]
```

The tuning constant for the odometry heading sensor.

Definition at line 103 of file [BlueConfiguration.hpp](#).

```
00103 {1.014};
```

### 5.21.3.4 ODOMETRY\_LINEAR\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ODOMETRY_LINEAR_PORT {15} [static], [constexpr],  
[private]
```

The port for the odometry linear distance tracking sensor.

Definition at line 109 of file [BlueConfiguration.hpp](#).

```
00109 {15};
```

### 5.21.3.5 ODOMETRY\_LINEAR\_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::ODOMETRY_LINEAR_RADIUS {1.22} [static],  
[constexpr], [private]
```

The radius of the odometry linear distance tracking wheel.

Definition at line 115 of file [BlueConfiguration.hpp](#).

```
00115 {1.22};
```

### 5.21.3.6 ODOMETRY\_LINEAR\_OFFSET

```
constexpr double wisco::configs::BlueConfiguration::ODOMETRY_LINEAR_OFFSET {-3.35} [static],  
[constexpr], [private]
```

The offset of the odometry linear distance tracking wheel.

Definition at line 121 of file [BlueConfiguration.hpp](#).

```
00121 {-3.35};
```

### 5.21.3.7 ODOMETRY\_STRAFE\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ODOMETRY_STRAFE_PORT {16} [static], [constexpr],  
[private]
```

The port for the odometry strafe distance tracking sensor.

Definition at line 127 of file [BlueConfiguration.hpp](#).

```
00127 {16};
```

### 5.21.3.8 ODOMETRY\_STRafe\_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::ODOMETRY_STRafe_RADIUS {-1.22} [static],  
[constexpr], [private]
```

The radius of the odometry strafe distance tracking wheel.

Definition at line 133 of file [BlueConfiguration.hpp](#).

```
00133 {-1.22};
```

### 5.21.3.9 ODOMETRY\_STRafe\_OFFSET

```
constexpr double wisco::configs::BlueConfiguration::ODOMETRY_STRafe_OFFSET {4.6} [static],  
[constexpr], [private]
```

The offset of the odometry strafe distance tracking wheel.

Definition at line 139 of file [BlueConfiguration.hpp](#).

```
00139 {4.6};
```

### 5.21.3.10 RESETTER\_DISTANCE\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::RESETTER_DISTANCE_PORT {14} [static],  
[constexpr], [private]
```

The port for the resetter distance sensor.

Definition at line 145 of file [BlueConfiguration.hpp](#).

```
00145 {14};
```

### 5.21.3.11 RESETTER\_DISTANCE\_CONSTANT

```
constexpr double wisco::configs::BlueConfiguration::RESETTER_DISTANCE_CONSTANT {1.0} [static],  
[constexpr], [private]
```

The tuning constant for the resetter distance sensor.

Definition at line 151 of file [BlueConfiguration.hpp](#).

```
00151 {1.0};
```

### 5.21.3.12 RESETTER\_DISTANCE\_OFFSET

```
constexpr double wisco::configs::BlueConfiguration::RESETTER_DISTANCE_OFFSET {} [static],  
[constexpr], [private]
```

The tuning offset for the resetter distance sensor.

Definition at line 157 of file [BlueConfiguration.hpp](#).

```
00157 {};
```

### 5.21.3.13 RESETTER\_OFFSET\_X

```
constexpr double wisco::configs::BlueConfiguration::RESETTER_OFFSET_X {} [static], [constexpr], [private]
```

The x-offset of the resetter.

Definition at line 163 of file [BlueConfiguration.hpp](#).

```
00163 {};
```

### 5.21.3.14 RESETTER\_OFFSET\_Y

```
constexpr double wisco::configs::BlueConfiguration::RESETTER_OFFSET_Y {} [static], [constexpr], [private]
```

The y-offset of the resetter.

Definition at line 169 of file [BlueConfiguration.hpp](#).

```
00169 {};
```

### 5.21.3.15 RESETTER\_OFFSET\_THETA

```
constexpr double wisco::configs::BlueConfiguration::RESETTER_OFFSET_THETA {} [static], [constexpr], [private]
```

The angle-offset of the resetter.

Definition at line 175 of file [BlueConfiguration.hpp](#).

```
00175 {};
```

### 5.21.3.16 DRIVE\_KINEMATIC

```
constexpr bool wisco::configs::BlueConfiguration::DRIVE_KINEMATIC {false} [static], [constexpr], [private]
```

Whether to use the kinematic drive model or not.

Definition at line 181 of file [BlueConfiguration.hpp](#).

```
00181 {false};
```

### 5.21.3.17 DRIVE\_VELOCITY\_PROFILE\_JERK\_RATE

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_VELOCITY_PROFILE_JERK_RATE {20.0} [static], [constexpr], [private]
```

The jerk rate of the drive velocity profile.

Definition at line 187 of file [BlueConfiguration.hpp](#).

```
00187 {20.0};
```

### 5.21.3.18 DRIVE\_VELOCITY\_PROFILE\_MAX\_ACCELERATION

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION  
{5.0} [static], [constexpr], [private]
```

The maximum acceleration of the drive velocity profile.

Definition at line 193 of file [BlueConfiguration.hpp](#).

```
00193 {5.0};
```

### 5.21.3.19 DRIVE\_LEFT\_MOTOR\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_1_PORT {-1} [static],  
[constexpr], [private]
```

The first left drive motor port.

Definition at line 199 of file [BlueConfiguration.hpp](#).

```
00199 {-1};
```

### 5.21.3.20 DRIVE\_LEFT\_MOTOR\_1\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_1_GEARSET  
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The first left drive motor gearset.

Definition at line 205 of file [BlueConfiguration.hpp](#).

```
00205 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.21 DRIVE\_LEFT\_MOTOR\_2\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_2_PORT {-2} [static],  
[constexpr], [private]
```

The second left drive motor port.

Definition at line 211 of file [BlueConfiguration.hpp](#).

```
00211 {-2};
```

### 5.21.3.22 DRIVE\_LEFT\_MOTOR\_2\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_2_GEARSET  
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The second left drive motor gearset.

Definition at line 217 of file [BlueConfiguration.hpp](#).

```
00217 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.23 DRIVE\_LEFT\_MOTOR\_3\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_3_PORT {3} [static],  
[constexpr], [private]
```

The third left drive motor port.

Definition at line 223 of file [BlueConfiguration.hpp](#).

```
00223 {3};
```

### 5.21.3.24 DRIVE\_LEFT\_MOTOR\_3\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_3_GEARSET  
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The third left drive motor gearset.

Definition at line 229 of file [BlueConfiguration.hpp](#).

```
00229 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.25 DRIVE\_LEFT\_MOTOR\_4\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_4_PORT {4} [static],  
[constexpr], [private]
```

The fourth left drive motor port.

Definition at line 235 of file [BlueConfiguration.hpp](#).

```
00235 {4};
```

### 5.21.3.26 DRIVE\_LEFT\_MOTOR\_4\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_4_GEARSET  
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The fourth left drive motor gearset.

Definition at line 241 of file [BlueConfiguration.hpp](#).

```
00241 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.27 DRIVE\_RIGHT\_MOTOR\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_1_PORT {-7} [static],  
[constexpr], [private]
```

The first right drive motor port.

Definition at line 247 of file [BlueConfiguration.hpp](#).

```
00247 {-7};
```

### 5.21.3.28 DRIVE\_RIGHT\_MOTOR\_1\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_1_GEARSET
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The first right drive motor gearset.

Definition at line 253 of file [BlueConfiguration.hpp](#).

```
00253 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.29 DRIVE\_RIGHT\_MOTOR\_2\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_2_PORT {-8} [static],
[constexpr], [private]
```

The second right drive motor port.

Definition at line 259 of file [BlueConfiguration.hpp](#).

```
00259 {-8};
```

### 5.21.3.30 DRIVE\_RIGHT\_MOTOR\_2\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_2_GEARSET
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The second right drive motor gearset.

Definition at line 265 of file [BlueConfiguration.hpp](#).

```
00265 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.31 DRIVE\_RIGHT\_MOTOR\_3\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_3_PORT {9} [static],
[constexpr], [private]
```

The third right drive motor port.

Definition at line 271 of file [BlueConfiguration.hpp](#).

```
00271 {9};
```

### 5.21.3.32 DRIVE\_RIGHT\_MOTOR\_3\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_3_GEARSET
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The third right drive motor gearset.

Definition at line 277 of file [BlueConfiguration.hpp](#).

```
00277 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.33 DRIVE\_RIGHT\_MOTOR\_4\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_4_PORT {10} [static],  
[constexpr], [private]
```

The fourth right drive motor port.

Definition at line 283 of file [BlueConfiguration.hpp](#).

```
00283 {10};
```

### 5.21.3.34 DRIVE\_RIGHT\_MOTOR\_4\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_4_GEARSET  
{pros::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The fourth right drive motor gearset.

Definition at line 289 of file [BlueConfiguration.hpp](#).

```
00289 {pros::E_MOTOR_GEARSET_06};
```

### 5.21.3.35 DRIVE\_VELOCITY\_TO\_VOLTAGE

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_VELOCITY_TO_VOLTAGE {12.0 / 60.0}  
[static], [constexpr], [private]
```

The conversion from velocity to voltage on the drive Current calculation = 12 volts to 16 inches per second.

Definition at line 296 of file [BlueConfiguration.hpp](#).

```
00296 {12.0 / 60.0};
```

### 5.21.3.36 DRIVE\_MASS

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_MASS {9.89} [static], [constexpr],  
[private]
```

The mass of the drive.

Definition at line 302 of file [BlueConfiguration.hpp](#).

```
00302 {9.89};
```

### 5.21.3.37 DRIVE\_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_RADIUS {6.0625} [static], [constexpr],  
[private]
```

The radius of the drive.

Definition at line 308 of file [BlueConfiguration.hpp](#).

```
00308 {6.0625}; // {6.5 * 2.54 / 100};
```

### 5.21.3.38 DRIVE\_MOMENT\_OF\_INERTIA

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_MOMENT_OF_INERTIA {19.887 * DRIVE_RADIUS  
* DRIVE_MASS} [static], [constexpr], [private]
```

The moment of inertia of the drive.

Definition at line 314 of file [BlueConfiguration.hpp](#).  
00314 {19.887 \* DRIVE\_RADIUS \* DRIVE\_MASS};

### 5.21.3.39 DRIVE\_GEAR\_RATIO

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_GEAR_RATIO {600.0 / 331.4} [static],  
[constexpr], [private]
```

The gear ratio of the drive.

Definition at line 320 of file [BlueConfiguration.hpp](#).  
00320 {600.0 / 331.4};

### 5.21.3.40 DRIVE\_WHEEL\_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_WHEEL_RADIUS {3.25 * 2.54 / 100}  
[static], [constexpr], [private]
```

The wheel radius of the drive.

Definition at line 326 of file [BlueConfiguration.hpp](#).  
00326 {3.25 \* 2.54 / 100};

### 5.21.3.41 INTAKE\_KP

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KP {1.0} [static], [constexpr],  
[private]
```

The KP for the intake PID.

Definition at line 332 of file [BlueConfiguration.hpp](#).  
00332 {1.0};

### 5.21.3.42 INTAKE\_KI

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KI {0.0} [static], [constexpr],  
[private]
```

The KI for the intake PID.

Definition at line 338 of file [BlueConfiguration.hpp](#).  
00338 {0.0};

### 5.21.3.43 INTAKE\_KD

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KD {0.0} [static], [constexpr],  
[private]
```

The KD for the intake PID.

Definition at line 344 of file [BlueConfiguration.hpp](#).  
00344 {0.0};

### 5.21.3.44 INTAKE\_MOTOR\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::INTAKE_MOTOR_1_PORT {-12} [static], [constexpr],  
[private]
```

The first intake motor port.

Definition at line 350 of file [BlueConfiguration.hpp](#).  
00350 {-12};

### 5.21.3.45 INTAKE\_MOTOR\_1\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::INTAKE_MOTOR_1_GEARSET {pros←  
::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The first intake motor gearset.

Definition at line 356 of file [BlueConfiguration.hpp](#).  
00356 {pros::E\_MOTOR\_GEARSET\_06};

### 5.21.3.46 INTAKE\_MOTOR\_2\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::INTAKE_MOTOR_2_PORT {19} [static], [constexpr],  
[private]
```

The second intake motor port.

Definition at line 362 of file [BlueConfiguration.hpp](#).  
00362 {19};

### 5.21.3.47 INTAKE\_MOTOR\_2\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::INTAKE_MOTOR_2_GEARSET {pros←  
::E_MOTOR_GEARSET_06} [static], [constexpr], [private]
```

The second intake motor gearset.

Definition at line 368 of file [BlueConfiguration.hpp](#).  
00368 {pros::E\_MOTOR\_GEARSET\_06};

### 5.21.3.48 INTAKE\_ROLLER\_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_ROLLER_RADIUS {1} [static], [constexpr], [private]
```

The radius of the intake roller.

Definition at line 374 of file [BlueConfiguration.hpp](#).

```
00374 {1};
```

### 5.21.3.49 BALL\_DETECTOR\_DISTANCE\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::BALL_DETECTOR_DISTANCE_PORT {5} [static], [constexpr], [private]
```

The port for the ball detector distance sensor.

Definition at line 380 of file [BlueConfiguration.hpp](#).

```
00380 {5};
```

### 5.21.3.50 BALL\_DETECTOR\_DISTANCE\_CONSTANT

```
constexpr double wisco::configs::BlueConfiguration::BALL_DETECTOR_DISTANCE_CONSTANT {1.0} [static], [constexpr], [private]
```

The tuning constant for the ball detector distance sensor.

Definition at line 386 of file [BlueConfiguration.hpp](#).

```
00386 {1.0};
```

### 5.21.3.51 BALL\_DETECTOR\_DISTANCE\_OFFSET

```
constexpr double wisco::configs::BlueConfiguration::BALL_DETECTOR_DISTANCE_OFFSET {} [static], [constexpr], [private]
```

The tuning offset for the ball detector distance sensor.

Definition at line 392 of file [BlueConfiguration.hpp](#).

```
00392 {};
```

### 5.21.3.52 BOOMERANG\_LINEAR\_KP

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_LINEAR_KP {11.0} [static], [constexpr], [private]
```

The proportional constant for the boomerang linear pid controller.

Definition at line 398 of file [BlueConfiguration.hpp](#).

```
00398 {11.0};
```

### 5.21.3.53 BOOMERANG\_LINEAR\_KI

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_LINEAR_KI {} [static], [constexpr],  
[private]
```

The integral constant for the boomerang linear pid controller.

Definition at line 404 of file [BlueConfiguration.hpp](#).

```
00404 {};
```

### 5.21.3.54 BOOMERANG\_LINEAR\_KD

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_LINEAR_KD {640.0} [static],  
[constexpr], [private]
```

The derivative constant for the boomerang linear pid controller.

Definition at line 410 of file [BlueConfiguration.hpp](#).

```
00410 {640.0};
```

### 5.21.3.55 BOOMERANG\_ROTATIONAL\_KP

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_ROTATIONAL_KP {320.0} [static],  
[constexpr], [private]
```

The proportional constant for the boomerang rotational pid controller.

Definition at line 416 of file [BlueConfiguration.hpp](#).

```
00416 {320.0};
```

### 5.21.3.56 BOOMERANG\_ROTATIONAL\_KI

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_ROTATIONAL_KI {} [static],  
[constexpr], [private]
```

The integral constant for the boomerang rotational pid controller.

Definition at line 422 of file [BlueConfiguration.hpp](#).

```
00422 {};
```

### 5.21.3.57 BOOMERANG\_ROTATIONAL\_KD

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_ROTATIONAL_KD {4800.0} [static],  
[constexpr], [private]
```

The derivative constant for the boomerang rotational pid controller.

Definition at line 428 of file [BlueConfiguration.hpp](#).

```
00428 {4800.0};
```

### 5.21.3.58 BOOMERANG\_LEAD

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_LEAD {0.12} [static], [constexpr],  
[private]
```

The lead ratio for the boomerang controller.

Definition at line 434 of file [BlueConfiguration.hpp](#).

```
00434 {0.12};
```

### 5.21.3.59 BOOMERANG\_TARGET\_TOLERANCE

```
constexpr double wisco::configs::BlueConfiguration::BOOMERANG_TARGET_TOLERANCE {1.0} [static],  
[constexpr], [private]
```

The target tolerance for the boomerang controller.

Definition at line 440 of file [BlueConfiguration.hpp](#).

```
00440 {1.0};
```

### 5.21.3.60 ELEVATOR\_KP

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KP {6.0} [static], [constexpr],  
[private]
```

The KP for the elevator PID.

Definition at line 446 of file [BlueConfiguration.hpp](#).

```
00446 {6.0};
```

### 5.21.3.61 ELEVATOR\_KI

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KI {0.0} [static], [constexpr],  
[private]
```

The KI for the elevator PID.

Definition at line 452 of file [BlueConfiguration.hpp](#).

```
00452 {0.0};
```

### 5.21.3.62 ELEVATOR\_KD

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KD {0.0} [static], [constexpr],  
[private]
```

The KD for the elevator PID.

Definition at line 458 of file [BlueConfiguration.hpp](#).

```
00458 {0.0};
```

### 5.21.3.63 ELEVATOR\_MOTOR\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_1_PORT {-11} [static],  
[constexpr], [private]
```

The first elevator motor port.

Definition at line 464 of file [BlueConfiguration.hpp](#).

```
00464 {-11};
```

### 5.21.3.64 ELEVATOR\_MOTOR\_1\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_1_GEARSET  
{pros::E_MOTOR_GEARSET_18} [static], [constexpr], [private]
```

The first elevator motor gearset.

Definition at line 470 of file [BlueConfiguration.hpp](#).

```
00470 {pros::E_MOTOR_GEARSET_18};
```

### 5.21.3.65 ELEVATOR\_MOTOR\_2\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_2_PORT {20} [static],  
[constexpr], [private]
```

The second elevator motor port.

Definition at line 476 of file [BlueConfiguration.hpp](#).

```
00476 {20};
```

### 5.21.3.66 ELEVATOR\_MOTOR\_2\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_2_GEARSET  
{pros::E_MOTOR_GEARSET_18} [static], [constexpr], [private]
```

The second elevator motor gearset.

Definition at line 482 of file [BlueConfiguration.hpp](#).

```
00482 {pros::E_MOTOR_GEARSET_18};
```

### 5.21.3.67 ELEVATOR\_ROTATION\_SENSOR\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_ROTATION_SENSOR_PORT {} [static],  
[constexpr], [private]
```

The elevator rotation sensor port.

Definition at line 488 of file [BlueConfiguration.hpp](#).

```
00488 {};
```

### 5.21.3.68 ELEVATOR\_INCHES\_PER\_RADIAN

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_INCHES_PER_RADIAN {17.0 / (0.8 *  
2 * M_PI)} [static], [constexpr], [private]
```

The number of inches moved per radian on the elevator.

Definition at line 494 of file [BlueConfiguration.hpp](#).

```
00494 {17.0 / (0.8 * 2 * M_PI)};
```

### 5.21.3.69 ELEVATOR\_DISTANCE\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_DISTANCE_PORT {} [static], [constexpr],  
[private]
```

The elevator distance sensor port.

Definition at line 500 of file [BlueConfiguration.hpp](#).

```
00500 {};
```

### 5.21.3.70 ELEVATOR\_DISTANCE\_CONSTANT

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_DISTANCE_CONSTANT {} [static],  
[constexpr], [private]
```

The elevator distance sensor tuning constant.

Definition at line 506 of file [BlueConfiguration.hpp](#).

```
00506 {};
```

### 5.21.3.71 ELEVATOR\_DISTANCE\_OFFSET

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_DISTANCE_OFFSET {} [static],  
[constexpr], [private]
```

The elevator distance sensor tuning offset.

Definition at line 512 of file [BlueConfiguration.hpp](#).

```
00512 {};
```

### 5.21.3.72 HANG\_CLAW\_PISTON\_1\_PORT

```
constexpr char wisco::configs::BlueConfiguration::HANG_CLAW_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first hang claw piston port.

Definition at line 518 of file [BlueConfiguration.hpp](#).

```
00518 {};
```

### 5.21.3.73 HANG\_CLAW\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_CLAW_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first hang claw piston's extended state.

Definition at line 524 of file [BlueConfiguration.hpp](#).

```
00524 {};
```

### 5.21.3.74 HANG\_CLAW\_CLOSED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_CLAW_CLOSED_STATE {} [static], [constexpr],  
[private]
```

The hang claw piston state when the claw is closed.

Definition at line 530 of file [BlueConfiguration.hpp](#).

```
00530 {};
```

### 5.21.3.75 HANG\_ARM\_PISTON\_1\_PORT

```
constexpr char wisco::configs::BlueConfiguration::HANG_ARM_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first hang arm piston port.

Definition at line 536 of file [BlueConfiguration.hpp](#).

```
00536 {};
```

### 5.21.3.76 HANG\_ARM\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_ARM_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first hang arm piston's extended state.

Definition at line 542 of file [BlueConfiguration.hpp](#).

```
00542 {};
```

### 5.21.3.77 HANG\_ARM\_UP\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_ARM_UP_STATE {} [static], [constexpr],  
[private]
```

The hang arm piston state when the arm is up.

Definition at line 548 of file [BlueConfiguration.hpp](#).

```
00548 {};
```

### 5.21.3.78 HANG\_WINCH\_PISTON\_1\_PORT

```
constexpr char wisco::configs::BlueConfiguration::HANG_WINCH_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first hang winch piston port.

Definition at line 554 of file [BlueConfiguration.hpp](#).

```
00554 {};
```

### 5.21.3.79 HANG\_WINCH\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_WINCH_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first hang winch piston's extended state.

Definition at line 560 of file [BlueConfiguration.hpp](#).

```
00560 {};
```

### 5.21.3.80 HANG\_WINCH\_ENGAGED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::HANG_WINCH_ENGAGED_STATE {} [static], [constexpr],  
[private]
```

The hang winch piston state when the winch is engaged.

Definition at line 566 of file [BlueConfiguration.hpp](#).

```
00566 {};
```

### 5.21.3.81 LOADER\_KP

```
constexpr double wisco::configs::BlueConfiguration::LOADER_KP {6.0} [static], [constexpr],  
[private]
```

The KP for the loader PID.

Definition at line 572 of file [BlueConfiguration.hpp](#).

```
00572 {6.0};
```

### 5.21.3.82 LOADER\_KI

```
constexpr double wisco::configs::BlueConfiguration::LOADER_KI {0.0} [static], [constexpr],  
[private]
```

The KI for the loader PID.

Definition at line 578 of file [BlueConfiguration.hpp](#).

```
00578 {0.0};
```

### 5.21.3.83 LOADER\_KD

```
constexpr double wisco::configs::BlueConfiguration::LOADER_KD {0.0} [static], [constexpr],  
[private]
```

The KD for the loader PID.

Definition at line 584 of file [BlueConfiguration.hpp](#).

```
00584 {0.0};
```

### 5.21.3.84 LOADER\_MOTOR\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::LOADER_MOTOR_1_PORT {-11} [static], [constexpr],  
[private]
```

The first loader motor port.

Definition at line 590 of file [BlueConfiguration.hpp](#).

```
00590 {-11};
```

### 5.21.3.85 LOADER\_MOTOR\_1\_GEARSET

```
constexpr pros::v5::MotorGears wisco::configs::BlueConfiguration::LOADER_MOTOR_1_GEARSET {pros←  
::E_MOTOR_GEARSET_36} [static], [constexpr], [private]
```

The first loader motor gearset.

Definition at line 596 of file [BlueConfiguration.hpp](#).

```
00596 {pros::E_MOTOR_GEARSET_36};
```

### 5.21.3.86 LOADER\_LOADED\_POSITION

```
constexpr double wisco::configs::BlueConfiguration::LOADER_LOADED_POSITION {5 * M_PI / 6}  
[static], [constexpr], [private]
```

The loader position when loaded.

Definition at line 602 of file [BlueConfiguration.hpp](#).

```
00602 {5 * M_PI / 6};
```

### 5.21.3.87 LOADER\_READY\_POSITION

```
constexpr double wisco::configs::BlueConfiguration::LOADER_READY_POSITION {0} [static], [constexpr],  
[private]
```

The loader position when ready.

Definition at line 608 of file [BlueConfiguration.hpp](#).

```
00608 {0};
```

### 5.21.3.88 LOADER\_POSITION\_TOLERANCE

```
constexpr double wisco::configs::BlueConfiguration::LOADER_POSITION_TOLERANCE {M_PI / 18}  
[static], [constexpr], [private]
```

The loader position tolerance.

Definition at line 614 of file [BlueConfiguration.hpp](#).

```
00614 {M_PI / 18};
```

### 5.21.3.89 TURN\_KP

```
constexpr double wisco::configs::BlueConfiguration::TURN_KP {160.0} [static], [constexpr],  
[private]
```

The proportional constant for the turn pid controller.

Definition at line 620 of file [BlueConfiguration.hpp](#).

```
00620 {160.0};
```

### 5.21.3.90 TURN\_KI

```
constexpr double wisco::configs::BlueConfiguration::TURN_KI {} [static], [constexpr], [private]
```

The integral constant for the turn pid controller.

Definition at line 626 of file [BlueConfiguration.hpp](#).

```
00626 {};
```

### 5.21.3.91 TURN\_KD

```
constexpr double wisco::configs::BlueConfiguration::TURN_KD {12000.0} [static], [constexpr],  
[private]
```

The derivative constant for the turn pid controller.

Definition at line 632 of file [BlueConfiguration.hpp](#).

```
00632 {12000.0};
```

### 5.21.3.92 TURN\_TARGET\_TOLERANCE

```
constexpr double wisco::configs::BlueConfiguration::TURN_TARGET_TOLERANCE {1.0 * M_PI / 180}  
[static], [constexpr], [private]
```

The target tolerance for the turn controller.

Definition at line 638 of file [BlueConfiguration.hpp](#).

```
00638 {1.0 * M_PI / 180};
```

### 5.21.3.93 TURN\_TARGET\_VELOCITY

```
constexpr double wisco::configs::BlueConfiguration::TURN_TARGET_VELOCITY {1.0 * M_PI / 180}  
[static], [constexpr], [private]
```

The target velocity for the turn controller.

Definition at line 644 of file [BlueConfiguration.hpp](#).  
00644 {1.0 \* M\_PI / 180};

### 5.21.3.94 UMBRELLA\_PISTON\_1\_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::UMBRELLA_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first umbrella piston port.

Definition at line 650 of file [BlueConfiguration.hpp](#).  
00650 {};

### 5.21.3.95 UMBRELLA\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::UMBRELLA_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first umbrella piston's extended state.

Definition at line 656 of file [BlueConfiguration.hpp](#).  
00656 {};

### 5.21.3.96 UMBRELLA\_OUT\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::UMBRELLA_OUT_STATE {} [static], [constexpr],  
[private]
```

The umbrella piston state when the umbrella is out.

Definition at line 662 of file [BlueConfiguration.hpp](#).  
00662 {};

### 5.21.3.97 LEFT\_WING\_PISTON\_1\_PORT

```
constexpr char wisco::configs::BlueConfiguration::LEFT_WING_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first left wing piston port.

Definition at line 668 of file [BlueConfiguration.hpp](#).  
00668 {};

### 5.21.3.98 LEFT\_WING\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::LEFT_WING_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first left wing piston's extended state.

Definition at line 674 of file [BlueConfiguration.hpp](#).

```
00674 {};
```

### 5.21.3.99 RIGHT\_WING\_PISTON\_1\_PORT

```
constexpr char wisco::configs::BlueConfiguration::RIGHT_WING_PISTON_1_PORT {} [static], [constexpr],  
[private]
```

The first right wing piston port.

Definition at line 680 of file [BlueConfiguration.hpp](#).

```
00680 {};
```

### 5.21.3.100 RIGHT\_WING\_PISTON\_1\_EXTENDED\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::RIGHT_WING_PISTON_1_EXTENDED_STATE {} [static],  
[constexpr], [private]
```

The first right wing piston's extended state.

Definition at line 686 of file [BlueConfiguration.hpp](#).

```
00686 {};
```

### 5.21.3.101 WINGS\_OUT\_STATE

```
constexpr bool wisco::configs::BlueConfiguration::WINGS_OUT_STATE {} [static], [constexpr],  
[private]
```

The wing piston state when the wings are out.

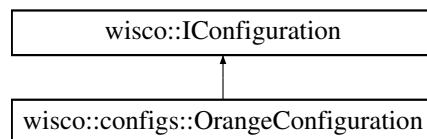
Definition at line 692 of file [BlueConfiguration.hpp](#).

```
00692 {};
```

## 5.22 wisco::configs::OrangeConfiguration Class Reference

The hardware configuration of the orange robot.

Inheritance diagram for wisco::configs::OrangeConfiguration:



## Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the configuration.*
- std::shared\_ptr<[control::ControlSystem](#)> [buildControlSystem \(\)](#) override  
*Build a control system using this configuration.*
- std::shared\_ptr<[user::IController](#)> [buildController \(\)](#) override  
*Build a controller using this configuration.*
- std::shared\_ptr<[robot::Robot](#)> [buildRobot \(\)](#) override  
*Build a robot using this configuration.*

## Public Member Functions inherited from [wisco::IConfiguration](#)

- virtual ~[IConfiguration \(\)](#)=default  
*Destroy the [IConfiguration](#) object.*

## Static Private Attributes

- static constexpr char [CONFIGURATION\\_NAME \[\]](#) {"ORANGE"}  
*The name of the configuration.*

### 5.22.1 Detailed Description

The hardware configuration of the orange robot.

#### Author

Nathan Sandvig

Definition at line 26 of file [OrangeConfiguration.hpp](#).

### 5.22.2 Member Function Documentation

#### 5.22.2.1 [getName\(\)](#)

```
std::string wisco::configs::OrangeConfiguration::getName ( ) [override], [virtual]
```

Get the name of the configuration.

#### Returns

std::string The name of the configuration

Implements [wisco::IConfiguration](#).

Definition at line 7 of file [OrangeConfiguration.cpp](#).

```
00008 {
00009     return CONFIGURATION_NAME;
0010 }
```

### 5.22.2.2 buildControlSystem()

```
std::shared_ptr< control::ControlSystem > wisco::configs::OrangeConfiguration::buildControlSystem ( ) [override], [virtual]
```

Build a control system using this configuration.

#### Returns

`std::shared_ptr<control::ControlSystem>` The control system built by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 12 of file [OrangeConfiguration.cpp](#).

```
00013 {  
00014     std::shared_ptr<control::ControlSystem>  
00015         control_system{std::make_shared<control::ControlSystem>()};  
00016     return control_system;  
00017 }
```

### 5.22.2.3 buildController()

```
std::shared_ptr< user::IController > wisco::configs::OrangeConfiguration::buildController ( ) [override], [virtual]
```

Build a controller using this configuration.

#### Returns

`std::shared_ptr<user::IController>` The controller build by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 19 of file [OrangeConfiguration.cpp](#).

```
00020 {  
00021     return std::shared_ptr<user::IController>{};  
00022 }
```

### 5.22.2.4 buildRobot()

```
std::shared_ptr< robot::Robot > wisco::configs::OrangeConfiguration::buildRobot ( ) [override], [virtual]
```

Build a robot using this configuration.

#### Returns

`robot::Robot` The robot built by this configuration

Implements [wisco::IConfiguration](#).

Definition at line 24 of file [OrangeConfiguration.cpp](#).

```
00025 {  
00026     return std::shared_ptr<robot::Robot>{};  
00027 }
```

### 5.22.3 Member Data Documentation

#### 5.22.3.1 CONFIGURATION\_NAME

```
constexpr char wisco::configs::OrangeConfiguration::CONFIGURATION_NAME[] {"ORANGE"} [static],  
[constexpr], [private]
```

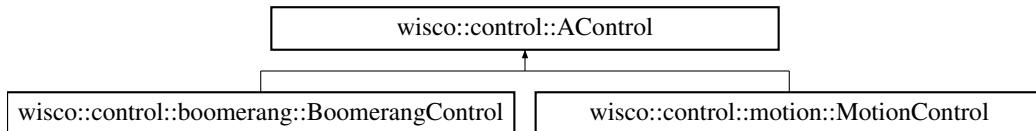
The name of the configuration.

Definition at line 33 of file [OrangeConfiguration.hpp](#).  
00033 {"ORANGE"};

## 5.23 wisco::control::AControl Class Reference

An abstract class for control algorithms.

Inheritance diagram for wisco::control::AControl:



#### Public Member Functions

- **AControl ()=default**  
*Construct a new [AControl](#) object.*
- **AControl (const AControl &other)=default**  
*Construct a new [AControl](#) object.*
- **AControl (AControl &&other)=default**  
*Construct a new [AControl](#) object.*
- **AControl (std::string name)**  
*Construct a new [AControl](#) object.*
- **virtual ~AControl ()=default**  
*Destroy the [AControl](#) object.*
- **const std::string & getName () const**  
*Get the name of the control.*
- **virtual void initialize ()=0**  
*Initializes the control.*
- **virtual void run ()=0**  
*Runs the control.*
- **virtual void command (std::string command\_name, va\_list &args)=0**  
*Runs a command for the control.*
- **virtual void \* state (std::string state\_name)=0**  
*Gets a state of the control.*
- **AControl & operator= (const AControl &rhs)=default**  
*Copy assignment operator for [AControl](#).*
- **AControl & operator= (AControl &&rhs)=default**  
*Move assignment operator for [AControl](#).*

**Private Attributes**

- std::string [m\\_name](#) {}  
*The name of the control.*

### 5.23.1 Detailed Description

An abstract class for control algorithms.

**Author**

Nathan Sandvig

Definition at line [27](#) of file [AControl.hpp](#).

### 5.23.2 Constructor & Destructor Documentation

#### 5.23.2.1 AControl() [1/3]

```
wisco::control::AControl::AControl (
    const AControl & other ) [default]
```

Construct a new [AControl](#) object.

**Parameters**

<i>other</i>	The <a href="#">AControl</a> object being copied
--------------	--

#### 5.23.2.2 AControl() [2/3]

```
wisco::control::AControl::AControl (
    AControl && other ) [default]
```

Construct a new [AControl](#) object.

**Parameters**

<i>other</i>	The <a href="#">AControl</a> object being moved
--------------	---

#### 5.23.2.3 AControl() [3/3]

```
wisco::control::AControl::AControl (
    std::string name ) [inline]
```

Construct a new [AControl](#) object.

**Parameters**

<i>name</i>	The name of the control
-------------	-------------------------

Definition at line 62 of file [AControl.hpp](#).

```
00062 : m_name{name} {}
```

### 5.23.3 Member Function Documentation

#### 5.23.3.1 getName()

```
const std::string & wisco::control::AControl::getName ( ) const [inline]
```

Get the name of the control.

**Returns**

```
const std::string& The name of the control
```

Definition at line 75 of file [AControl.hpp](#).

```
00076 {
00077     return m_name;
00078 }
```

#### 5.23.3.2 initialize()

```
virtual void wisco::control::AControl::initialize ( ) [pure virtual]
```

Initializes the control.

Implemented in [wisco::control::boomerang::BoomerangControl](#), and [wisco::control::motion::MotionControl](#).

#### 5.23.3.3 run()

```
virtual void wisco::control::AControl::run ( ) [pure virtual]
```

Runs the control.

Implemented in [wisco::control::boomerang::BoomerangControl](#), and [wisco::control::motion::MotionControl](#).

#### 5.23.3.4 command()

```
virtual void wisco::control::AControl::command (
    std::string command_name,
    va_list & args ) [pure virtual]
```

Runs a command for the control.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implemented in [wisco::control::boomerang::BoomerangControl](#), and [wisco::control::motion::MotionControl](#).

**5.23.3.5 state()**

```
virtual void* wisco::control::AControl::state (
    std::string state_name ) [pure virtual]
```

Gets a state of the control.

**Parameters**

<i>state_name</i>	The name of the state to get
-------------------	------------------------------

**Returns**

`void*` The current value of that state

Implemented in [wisco::control::boomerang::BoomerangControl](#), and [wisco::control::motion::MotionControl](#).

**5.23.3.6 operator=() [1/2]**

```
AControl & wisco::control::AControl::operator= (
    const AControl & rhs ) [default]
```

Copy assignment operator for [AControl](#).

**Parameters**

<i>rhs</i>	The <a href="#">AControl</a> on the right hand side of the operator
------------	---

**Returns**

`AControl&` This [AControl](#) with the copied values

**5.23.3.7 operator=() [2/2]**

```
AControl & wisco::control::AControl::operator= (
    AControl && rhs ) [default]
```

Move assignment operator for [AControl](#).

**Parameters**

<i>rhs</i>	The <a href="#">AControl</a> value on the right hand side of the operator
------------	---

**Returns**

[AControl&](#) This [AControl](#) with the moved values

## 5.23.4 Member Data Documentation

### 5.23.4.1 m\_name

```
std::string wisco::control::AControl::m_name {} [private]
```

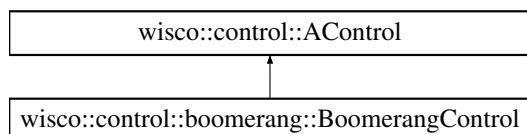
The name of the control.

Definition at line 34 of file [AControl.hpp](#).  
00034 {};

## 5.24 wisco::control::boomerang::BoomerangControl Class Reference

Control adapter for the boomerang controller.

Inheritance diagram for wisco::control::boomerang::BoomerangControl:



### Public Member Functions

- **BoomerangControl** (std::unique\_ptr< [IBoomerang](#) > &boomerang)  
*Construct a new Boomerang Control object.*
- void **initialize** () override  
*Initializes the control.*
- void **run** () override  
*Runs the control.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the control.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the control.*

## Public Member Functions inherited from [wisco::control::AControl](#)

- **AControl ()=default**  
*Construct a new [AControl](#) object.*
- **AControl (const [AControl](#) &other)=default**  
*Construct a new [AControl](#) object.*
- **AControl ([AControl](#) &&other)=default**  
*Construct a new [AControl](#) object.*
- **AControl (std::string name)**  
*Construct a new [AControl](#) object.*
- **virtual ~AControl ()=default**  
*Destroy the [AControl](#) object.*
- **const std::string & getName () const**  
*Get the name of the control.*
- **AControl & operator= (const [AControl](#) &rhs)=default**  
*Copy assignment operator for [AControl](#).*
- **AControl & operator= ([AControl](#) &&rhs)=default**  
*Move assignment operator for [AControl](#).*

## Private Attributes

- **std::unique\_ptr< [IBoomerang](#) > m\_boomerang {}**  
*The boomerang controller being adapted.*

## Static Private Attributes

- **static constexpr char CONTROL\_NAME [] {"BOOMERANG"}**  
*The name of the control.*
- **static constexpr char GO\_TO\_POSITION\_COMMAND\_NAME [] {"GO TO POSITION"}**  
*The name of the go to position command.*
- **static constexpr char PAUSE\_COMMAND\_NAME [] {"PAUSE"}**  
*The name of the pause command.*
- **static constexpr char RESUME\_COMMAND\_NAME [] {"RESUME"}**  
*The name of the resume command.*
- **static constexpr char TARGET\_REACHED\_STATE\_NAME [] {"TARGET REACHED"}**  
*The name of the target reached state.*

## 5.24.1 Detailed Description

Control adapter for the boomerang controller.

### Author

Nathan Sandvig

Definition at line 38 of file [BoomerangControl.hpp](#).

## 5.24.2 Constructor & Destructor Documentation

### 5.24.2.1 BoomerangControl()

```
wisco::control::boomerang::BoomerangControl::BoomerangControl (
    std::unique_ptr< IBoomerang > & boomerang )
```

Construct a new Boomerang Control object.

**Parameters**

<i>boomerang</i>	The boomerang controller being adapted
------------------	--

Definition at line 9 of file [BoomerangControl.cpp](#).

```
00010     : AControl{CONTROL_NAME}, m_boomerang{std::move(boomerang)}
00011 {
00012
00013 }
```

### 5.24.3 Member Function Documentation

#### 5.24.3.1 initialize()

```
void wisco::control::boomerang::BoomerangControl::initialize () [override], [virtual]
```

Initializes the control.

Implements [wisco::control::AControl](#).

Definition at line 15 of file [BoomerangControl.cpp](#).

```
00016 {
00017     if (m_boomerang)
00018         m_boomerang->initialize();
00019 }
```

#### 5.24.3.2 run()

```
void wisco::control::boomerang::BoomerangControl::run () [override], [virtual]
```

Runs the control.

Implements [wisco::control::AControl](#).

Definition at line 21 of file [BoomerangControl.cpp](#).

```
00022 {
00023     if (m_boomerang)
00024         m_boomerang->run();
00025 }
```

#### 5.24.3.3 command()

```
void wisco::control::boomerang::BoomerangControl::command (
    std::string command_name,
    va_list & args) [override], [virtual]
```

Runs a command for the control.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::control::AControl](#).

Definition at line 27 of file [BoomerangControl.cpp](#).

```
00028 {
00029     if (command_name == GO_TO_POSITION_COMMAND_NAME)
00030     {
00031         void* robot_ptr{va_arg(args, void*)};
00032         std::shared_ptr<robot::Robot> robot{*static_cast<std::shared_ptr<robot::Robot>>(robot_ptr)};
00033         double velocity{va_arg(args, double)};
00034         double x{va_arg(args, double)};
00035         double y{va_arg(args, double)};
00036         double theta{va_arg(args, double)};
00037         m_boomerang->goToPosition(robot, velocity, x, y, theta);
00038     }
00039     else if (command_name == PAUSE_COMMAND_NAME)
00040     {
00041         m_boomerang->pause();
00042     }
00043     else if (command_name == RESUME_COMMAND_NAME)
00044     {
00045         m_boomerang->resume();
00046     }
00047 }
```

#### 5.24.3.4 state()

```
void * wisco::control::boomerang::BoomerangControl::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the control.

**Parameters**

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

**Returns**

```
void* The current value of that state
```

Implements [wisco::control::AControl](#).

Definition at line 49 of file [BoomerangControl.cpp](#).

```
00050 {
00051     void* result{nullptr};
00052
00053     if (state_name == TARGET_REACHED_STATE_NAME)
00054     {
00055         bool* velocity{new bool{m_boomerang->targetReached()}};
00056         result = velocity;
00057     }
00058
00059     return result;
00060 }
```

### 5.24.4 Member Data Documentation

#### 5.24.4.1 CONTROL\_NAME

```
constexpr char wisco::control::boomerang::BoomerangControl::CONTROL_NAME[ ] {"BOOMERANG"} [static],
[constexpr], [private]
```

The name of the control.

Definition at line 45 of file [BoomerangControl.hpp](#).

```
00045 {"BOOMERANG"};
```

#### 5.24.4.2 GO\_TO\_POSITION\_COMMAND\_NAME

```
constexpr char wisco::control::boomerang::BoomerangControl::GO_TO_POSITION_COMMAND_NAME[ ] {"GO TO POSITION"} [static], [constexpr], [private]
```

The name of the go to position command.

Definition at line 51 of file [BoomerangControl.hpp](#).

```
00051 {"GO TO POSITION"};
```

#### 5.24.4.3 PAUSE\_COMMAND\_NAME

```
constexpr char wisco::control::boomerang::BoomerangControl::PAUSE_COMMAND_NAME[ ] {"PAUSE"} [static], [constexpr], [private]
```

The name of the pause command.

Definition at line 57 of file [BoomerangControl.hpp](#).

```
00057 {"PAUSE"};
```

#### 5.24.4.4 RESUME\_COMMAND\_NAME

```
constexpr char wisco::control::boomerang::BoomerangControl::RESUME_COMMAND_NAME[ ] {"RESUME"} [static], [constexpr], [private]
```

The name of the resume command.

Definition at line 63 of file [BoomerangControl.hpp](#).

```
00063 {"RESUME"};
```

#### 5.24.4.5 TARGET\_REACHED\_STATE\_NAME

```
constexpr char wisco::control::boomerang::BoomerangControl::TARGET_REACHED_STATE_NAME[ ] {"TARGET REACHED"} [static], [constexpr], [private]
```

The name of the target reached state.

Definition at line 69 of file [BoomerangControl.hpp](#).

```
00069 {"TARGET REACHED"};
```

#### 5.24.4.6 m\_boomerang

```
std::unique_ptr<IBoomerang> wisco::control::boomerang::BoomerangControl::m_boomerang {} [private]
```

The boomerang controller being adapted.

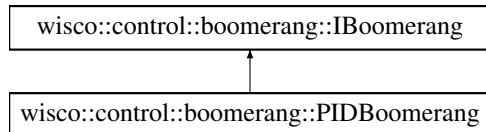
Definition at line 75 of file [BoomerangControl.hpp](#).

```
00075 {};
```

## 5.25 wisco::control::boomerang::IBoomerang Class Reference

Interface for boomerang controllers.

Inheritance diagram for wisco::control::boomerang::IBoomerang:



### Public Member Functions

- virtual ~**IBoomerang** ()=default  
*Destroy the [IBoomerang](#) object.*
- virtual void **initialize** ()=0  
*Initializes the boomerang controller.*
- virtual void **run** ()=0  
*Runs the boomerang controller.*
- virtual void **goToPosition** (const std::shared\_ptr< robot::Robot > &robot, double velocity, double x, double y, double theta)=0  
*Moves to the target position.*
- virtual void **pause** ()=0  
*Pauses the current motion.*
- virtual void **resume** ()=0  
*Resumes the current motion.*
- virtual bool **targetReached** ()=0  
*Checks if the target has been reached.*

### 5.25.1 Detailed Description

Interface for boomerang controllers.

#### Author

Nathan Sandvig

Definition at line 37 of file [IBoomerang.hpp](#).

### 5.25.2 Member Function Documentation

#### 5.25.2.1 initialize()

```
virtual void wisco::control::boomerang::IBoomerang::initialize ( ) [pure virtual]
```

Initializes the boomerang controller.

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

### 5.25.2.2 run()

```
virtual void wisco::control::boomerang::IBoomerang::run ( ) [pure virtual]
```

Runs the boomerang controller.

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

### 5.25.2.3 goToPosition()

```
virtual void wisco::control::boomerang::IBoomerang::goToPosition (
    const std::shared_ptr< robot::Robot > & robot,
    double velocity,
    double x,
    double y,
    double theta ) [pure virtual]
```

Moves to the target position.

#### Parameters

<i>robot</i>	The robot
<i>velocity</i>	The motion velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>theta</i>	The target angle

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

### 5.25.2.4 pause()

```
virtual void wisco::control::boomerang::IBoomerang::pause ( ) [pure virtual]
```

Pauses the current motion.

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

### 5.25.2.5 resume()

```
virtual void wisco::control::boomerang::IBoomerang::resume ( ) [pure virtual]
```

Resumes the current motion.

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

### 5.25.2.6 targetReached()

```
virtual bool wisco::control::boomerang::IBoomerang::targetReached() [pure virtual]
```

Checks if the target has been reached.

#### Returns

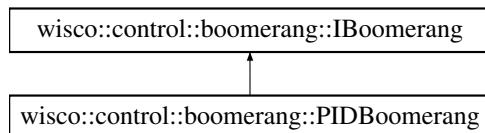
- true The target has been reached
- false The target has not been reached

Implemented in [wisco::control::boomerang::PIDBoomerang](#).

## 5.26 wisco::control::boomerang::PIDBoomerang Class Reference

Boomerang controller for the robot drivetrain.

Inheritance diagram for wisco::control::boomerang::PIDBoomerang:



### Public Member Functions

- void [initialize](#) () override  
*Initializes the boomerang controller.*
- void [run](#) () override  
*Runs the boomerang controller.*
- void [goToPosition](#) (const std::shared\_ptr<[robot::Robot](#)> &robot, double velocity, double x, double y, double theta) override  
*Moves to the target position.*
- void [pause](#) () override  
*Pauses the current motion.*
- void [resume](#) () override  
*Resumes the current motion.*
- bool [targetReached](#) () override  
*Checks if the target has been reached.*
- void [setDelayer](#) (const std::unique\_ptr<[rtos::IDelayer](#)> &delayer)  
*Sets the rtos delayer.*
- void [setMutex](#) (std::unique\_ptr<[rtos::IMutex](#)> &mutex)  
*Sets the rtos mutex.*
- void [setTask](#) (std::unique\_ptr<[rtos::ITask](#)> &task)  
*Sets the rtos task.*
- void [setLinearPID](#) ([PID](#) linear\_pid)  
*Sets the linear [PID](#) controller.*
- void [setRotationalPID](#) ([PID](#) rotational\_pid)  
*Sets the rotational [PID](#) controller.*
- void [setLead](#) (double lead)  
*Sets the lead ratio.*
- void [setTargetTolerance](#) (double target\_tolerance)  
*Sets the target tolerance.*

## Public Member Functions inherited from [wisco::control::boomerang::IBoomerang](#)

- virtual ~[IBoomerang](#) ()=default

*Destroy the [IBoomerang](#) object.*

## Private Member Functions

- void [taskUpdate](#) ()
 

*Runs all the object-specific updates in the task loop.*
- void [setDriveVelocity](#) ([robot::subsystems::drive::Velocity](#) velocity)
 

*Set the velocity for the drive.*
- [robot::subsystems::position::Position](#) [getOdometryPosition](#) ()
 

*Get the position from the odometry.*
- double [calculateDistanceToTarget](#) ([wisco::robot::subsystems::position::Position](#) position)
 

*Calculates the distance to the target position.*
- [path::Point](#) [calculateCarrotPoint](#) (double distance)
 

*Calculates the carrot point.*
- void [updateVelocity](#) ([robot::subsystems::position::Position](#) position, [path::Point](#) carrot\_point)
 

*Updates the drive velocity.*

## Static Private Member Functions

- static void [taskLoop](#) (void \*params)
 

*The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr<[rtos::IDelay](#)> [m\\_delayer](#) {}
 

*The rtos delayer.*
- std::unique\_ptr<[rtos::IMutex](#)> [m\\_mutex](#) {}
 

*The rtos mutex.*
- std::unique\_ptr<[rtos::ITask](#)> [m\\_task](#) {}
 

*The rtos task.*
- [PID](#) [m\\_linear\\_pid](#) {}
 

*The linear PID controller.*
- [PID](#) [m\\_rotational\\_pid](#) {}
 

*The rotational PID controller.*
- double [m\\_lead](#) {}
 

*The lead ratio for the carrot point.*
- double [m\\_target\\_tolerance](#) {}
 

*The acceptable tolerance to reach the target.*
- std::shared\_ptr<[robot::Robot](#)> [control\\_robot](#) {}
 

*The robot being controlled.*
- double [motion\\_velocity](#) {}
 

*The motion velocity in inches per second.*
- double [target\\_x](#) {}
 

*The target x-coordinate.*
- double [target\\_y](#) {}
 

*The target y-coordinate.*
- double [target\\_theta](#) {}
 

*The target angle.*
- bool [paused](#) {}
 

*Whether or not the motion has been paused.*
- bool [target\\_reached](#) {}
 

*Whether or not the target point has been reached.*

## Static Private Attributes

- static constexpr uint8\_t **TASK\_DELAY** {10}  
*The loop delay on the task.*
- static constexpr char **DRIVE\_SUBSYSTEM\_NAME** [] {"DIFFERENTIAL DRIVE"}  
*The name of the drive subsystem.*
- static constexpr char **ODOMETRY\_SUBSYSTEM\_NAME** [] {"POSITION TRACKER"}  
*The name of the odometry subsystem.*
- static constexpr char **DRIVE\_SET\_VELOCITY\_COMMAND\_NAME** [] {"SET VELOCITY"}  
*The name of the drive set velocity command.*
- static constexpr char **ODOMETRY\_GET\_POSITION\_STATE\_NAME** [] {"GET POSITION"}  
*The name of the odometry get position command.*

## 5.26.1 Detailed Description

Boomerang controller for the robot drivetrain.

### Author

Nathan Sandvig

Definition at line 50 of file [PIDBoomerang.hpp](#).

## 5.26.2 Member Function Documentation

### 5.26.2.1 taskLoop()

```
void wisco::control::boomerang::PIDBoomerang::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

#### Parameters

<i>params</i>	The task parameters
---------------	---------------------

Definition at line 9 of file [PIDBoomerang.cpp](#).

```
00010 {
00011     void** parameters{static_cast<void**>(params)};
00012     PIDBoomerang* instance{static_cast<PIDBoomerang*>(parameters[0])};
00013
00014     while (true)
00015     {
00016         instance->taskUpdate();
00017     }
00018 }
```

### 5.26.2.2 taskUpdate()

```
void wisco::control::boomerang::PIDBoomerang::taskUpdate ( ) [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 20 of file PIDBoomerang.cpp.

```

00021 {
00022     if (m_mutex)
00023         m_mutex->take();
00024
00025     if (!paused && !target_reached)
00026     {
00027         auto position{getOdometryPosition()};
00028         double distance{calculateDistanceToTarget(position)};
00029         if (distance < m_target_tolerance)
00030         {
00031             target_reached = true;
00032             robot::subsystems::drive::Velocity stop{0, 0};
00033             setDriveVelocity(stop);
00034         }
00035     else
00036     {
00037         path::Point carrot_point{calculateCarrotPoint(distance)};
00038         updateVelocity(position, carrot_point);
00039     }
00040 }
00041
00042 if (m_mutex)
00043     m_mutex->give();
00044
00045 m_delayer->delay(TASK_DELAY);
00046 }
```

### 5.26.2.3 setDriveVelocity()

```
void wisco::control::boomerang::PIDBoomerang::setDriveVelocity (
    robot::subsystems::drive::Velocity velocity) [private]
```

Set the velocity for the drive.

#### Parameters

<i>velocity</i>	The velocity for the drive
-----------------	----------------------------

Definition at line 48 of file PIDBoomerang.cpp.

```

00049 {
00050     if (control_robot)
00051         control_robot->sendCommand(DRIVE_SUBSYSTEM_NAME, DRIVE_SET_VELOCITY_COMMAND_NAME, velocity);
00052 }
```

### 5.26.2.4 getOdometryPosition()

```
robot::subsystems::position::Position wisco::control::boomerang::getOdometry<-
Position () [private]
```

Get the position from the odometry.

#### Returns

robot::subsystems::position::Position The position from the odometry

Definition at line 54 of file PIDBoomerang.cpp.

```

00055 {
00056     robot::subsystems::position::Position position{};
00057
00058     if (control_robot)
00059     {
```

```

00060     robot::subsystems::position::Position*
00061     result{static_cast<robot::subsystems::position::Position*>(control_robot->getState(ODOMETRY_SUBSYSTEM_NAME,
00062     ODOMETRY_GET_POSITION_STATE_NAME))};
00063     if (result)
00064     {
00065         position = *result;
00066         delete result;
00067     }
00068     return position;
00069 }
```

### 5.26.2.5 calculateDistanceToTarget()

```
double wisco::control::boomerang::PIDBoomerang::calculateDistanceToTarget (
    wisco::robot::subsystems::position::Position position) [private]
```

Calculates the distance to the target position.

#### Parameters

<i>position</i>	The current position
-----------------	----------------------

#### Returns

`double` The distance to the target position

Definition at line 71 of file [PIDBoomerang.cpp](#).

```

00072 {
00073     double x2{std::pow(target_x - position.x, 2)};
00074     double y2{std::pow(target_y - position.y, 2)};
00075     double distance{std::sqrt(x2 + y2)};
00076     return distance;
00077 }
```

### 5.26.2.6 calculateCarrotPoint()

```
path::Point wisco::control::boomerang::PIDBoomerang::calculateCarrotPoint (
    double distance) [private]
```

Calculates the carrot point.

#### Parameters

<i>distance</i>	The distance to the target point
-----------------	----------------------------------

#### Returns

`path::Point` The carrot point

Definition at line 79 of file [PIDBoomerang.cpp](#).

```

00080 {
00081     double carrot_x{target_x - (distance * std::cos(target_theta)) * m_lead};
00082     double carrot_y{target_y - (distance * std::sin(target_theta)) * m_lead};
00083     return path::Point(carrot_x, carrot_y);
00084 }
```

### 5.26.2.7 updateVelocity()

```
void wisco::control::boomerang::PIDBoomerang::updateVelocity (
    robot::subsystems::position::Position position,
    path::Point carrot_point ) [private]
```

Updates the drive velocity.

#### Parameters

<i>position</i>	The current position
<i>carrot_point</i>	The carrot point

Definition at line 86 of file [PIDBoomerang.cpp](#).

```
00087 {
00088     double x_error{carrot_point.getX() - position.x};
00089     double y_error{carrot_point.getY() - position.y};
00090
00091     double rotational_error{bindRadians(std::atan2(y_error, x_error) - position.theta)};
00092     double linear_error{std::cos(rotational_error) * std::sqrt(std::pow(x_error, 2) +
00093         std::pow(y_error, 2))};
00094
00095     double linear_control{m_linear_pid.getControlValue(0, linear_error)};
00096     double rotational_control{m_rotational_pid.getControlValue(0, rotational_error)};
00097     if (std::abs(rotational_control) > std::abs(linear_control))
00098         rotational_control *= std::abs(linear_control / rotational_control);
00099
00100     double velocity_constant{std::abs(motion_velocity / linear_control)};
00101     double left_velocity{linear_control - rotational_control};
00102     double right_velocity{linear_control + rotational_control};
00103     if (std::abs((left_velocity + right_velocity) / 2) > motion_velocity)
00104     {
00105         left_velocity *= velocity_constant;
00106         right_velocity *= velocity_constant;
00107     }
00108     robot::subsystems::drive::Velocity velocity{left_velocity, right_velocity};
00109     setDriveVelocity(velocity);
```

### 5.26.2.8 initialize()

```
void wisco::control::boomerang::PIDBoomerang::initialize () [override], [virtual]
```

Initializes the boomerang controller.

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 111 of file [PIDBoomerang.cpp](#).

```
00112 {
00113     m_linear_pid.reset();
00114     m_rotational_pid.reset();
00115 }
```

### 5.26.2.9 run()

```
void wisco::control::boomerang::PIDBoomerang::run () [override], [virtual]
```

Runs the boomerang controller.

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 117 of file [PIDBoomerang.cpp](#).

```
00118 {
00119     if (m_task)
00120     {
00121         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};
00122         params[0] = this;
00123         m_task->start(&PIDBoomerang::taskLoop, params);
00124     }
00125 }
```

### 5.26.2.10 goToPosition()

```
void wisco::control::boomerang::PIDBoomerang::goToPosition (
    const std::shared_ptr< robot::Robot > & robot,
    double velocity,
    double x,
    double y,
    double theta ) [override], [virtual]
```

Moves to the target position.

#### Parameters

<i>robot</i>	The robot
<i>velocity</i>	The motion velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>theta</i>	The target angle

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 127 of file [PIDBoomerang.cpp](#).

```
00128 {
00129     if (m_mutex)
00130         m_mutex->take();
00131
00132     control_robot = robot;
00133     motion_velocity = velocity;
00134     target_x = x;
00135     target_y = y;
00136     target_theta = theta;
00137     target_reached = false;
00138
00139     if (m_mutex)
00140         m_mutex->give();
00141 }
```

### 5.26.2.11 pause()

```
void wisco::control::boomerang::PIDBoomerang::pause () [override], [virtual]
```

Pauses the current motion.

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 143 of file [PIDBoomerang.cpp](#).

```
00144 {
00145     if (m_mutex)
00146         m_mutex->take();
00147
00148     paused = true;
00149     robot::subsystems::drive::Velocity stop{0, 0};
00150     setDriveVelocity(stop);
00151
00152     if (m_mutex)
00153         m_mutex->give();
00154 }
```

### 5.26.2.12 resume()

```
void wisco::control::boomerang::PIDBoomerang::resume ( ) [override], [virtual]
```

Resumes the current motion.

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 156 of file [PIDBoomerang.cpp](#).

```
00157 {
00158     if (m_mutex)
00159         m_mutex->take();
00160
00161     paused = false;
00162
00163     if (m_mutex)
00164         m_mutex->give();
00165 }
```

### 5.26.2.13 targetReached()

```
bool wisco::control::boomerang::PIDBoomerang::targetReached ( ) [override], [virtual]
```

Checks if the target has been reached.

#### Returns

true The target has been reached

false The target has not been reached

Implements [wisco::control::boomerang::IBoomerang](#).

Definition at line 167 of file [PIDBoomerang.cpp](#).

```
00168 {
00169     return target_reached;
00170 }
```

### 5.26.2.14 setDelayer()

```
void wisco::control::boomerang::PIDBoomerang::setDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Sets the rtos delayer.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 172 of file [PIDBoomerang.cpp](#).

```
00173 {
00174     m_delayer = delayer->clone();
00175 }
```

### 5.26.2.15 setMutex()

```
void wisco::control::boomerang::PIDBoomerang::setMutex (
```

```
std::unique_ptr< rtos::IMutex > & mutex )
```

Sets the rtos mutex.

#### Parameters

<i>mutex</i>	The rtos mutex
--------------	----------------

Definition at line 177 of file [PIDBoomerang.cpp](#).

```
00178 {
00179     m_mutex = std::move(mutex);
00180 }
```

### 5.26.2.16 setTask()

```
void wisco::control::boomerang::PIDBoomerang::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Sets the rtos task.

#### Parameters

<i>task</i>	The rtos task
-------------	---------------

Definition at line 182 of file [PIDBoomerang.cpp](#).

```
00183 {
00184     m_task = std::move(task);
00185 }
```

### 5.26.2.17 setLinearPID()

```
void wisco::control::boomerang::PIDBoomerang::setLinearPID (
    PID linear_pid )
```

Sets the linear [PID](#) controller.

#### Parameters

<i>linear_pid</i>	The linear <a href="#">PID</a> controller
-------------------	---

Definition at line 187 of file [PIDBoomerang.cpp](#).

```
00188 {
00189     m_linear_pid = linear_pid;
00190 }
```

### 5.26.2.18 setRotationalPID()

```
void wisco::control::boomerang::PIDBoomerang::setRotationalPID (
    PID rotational_pid )
```

Sets the rotational [PID](#) controller.

**Parameters**

<i>rotational_pid</i>	The rotational PID controller
-----------------------	-------------------------------

Definition at line 192 of file [PIDBoomerang.cpp](#).

```
00193 {  
00194     m_rotational_pid = rotational_pid;  
00195 }
```

**5.26.2.19 setLead()**

```
void wisco::control::boomerang::PIDBoomerang::setLead (  
    double lead )
```

Sets the lead ratio.

**Parameters**

<i>lead</i>	The lead ratio
-------------	----------------

Definition at line 197 of file [PIDBoomerang.cpp](#).

```
00198 {  
00199     m_lead = lead;  
00200 }
```

**5.26.2.20 setTargetTolerance()**

```
void wisco::control::boomerang::PIDBoomerang::setTargetTolerance (  
    double target_tolerance )
```

Sets the target tolerance.

**Parameters**

<i>target_tolerance</i>	The motion target tolerance
-------------------------	-----------------------------

Definition at line 202 of file [PIDBoomerang.cpp](#).

```
00203 {  
00204     m_target_tolerance = target_tolerance;  
00205 }
```

**5.26.3 Member Data Documentation****5.26.3.1 TASK\_DELAY**

```
constexpr uint8_t wisco::control::boomerang::PIDBoomerang::TASK_DELAY {10} [static], [constexpr],  
[private]
```

The loop delay on the task.

Definition at line 57 of file [PIDBoomerang.hpp](#).

```
00057 {10};
```

### 5.26.3.2 DRIVE\_SUBSYSTEM\_NAME

```
constexpr char wisco::control::boomerang::PIDBoomerang::DRIVE_SUBSYSTEM_NAME[ ] {"DIFFERENTIAL  
DRIVE"} [static], [constexpr], [private]
```

The name of the drive subsystem.

Definition at line 63 of file [PIDBoomerang.hpp](#).  
00063 {"DIFFERENTIAL DRIVE"};

### 5.26.3.3 ODOMETRY\_SUBSYSTEM\_NAME

```
constexpr char wisco::control::boomerang::PIDBoomerang::ODOMETRY_SUBSYSTEM_NAME[ ] {"POSITION  
TRACKER"} [static], [constexpr], [private]
```

The name of the odometry subsystem.

Definition at line 69 of file [PIDBoomerang.hpp](#).  
00069 {"POSITION TRACKER"};

### 5.26.3.4 DRIVE\_SET\_VELOCITY\_COMMAND\_NAME

```
constexpr char wisco::control::boomerang::PIDBoomerang::DRIVE_SET_VELOCITY_COMMAND_NAME[ ]  
{"SET VELOCITY"} [static], [constexpr], [private]
```

The name of the drive set velocity command.

Definition at line 75 of file [PIDBoomerang.hpp](#).  
00075 {"SET VELOCITY"};

### 5.26.3.5 ODOMETRY\_GET\_POSITION\_STATE\_NAME

```
constexpr char wisco::control::boomerang::PIDBoomerang::ODOMETRY_GET_POSITION_STATE_NAME[ ]  
{"GET POSITION"} [static], [constexpr], [private]
```

The name of the odometry get position command.

Definition at line 81 of file [PIDBoomerang.hpp](#).  
00081 {"GET POSITION"};

### 5.26.3.6 m\_delayer

```
std::unique_ptr<rtos::IDelay> wisco::control::boomerang::PIDBoomerang::m_delayer {} [private]
```

The rtos delayer.

Definition at line 94 of file [PIDBoomerang.hpp](#).  
00094 {};

### 5.26.3.7 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::control::boomerang::PIDBoomerang::m_mutex {} [private]
```

The rtos mutex.

Definition at line 100 of file [PIDBoomerang.hpp](#).

```
00100 {};
```

### 5.26.3.8 m\_task

```
std::unique_ptr<rtos::ITask> wisco::control::boomerang::PIDBoomerang::m_task {} [private]
```

The rtos task.

Definition at line 106 of file [PIDBoomerang.hpp](#).

```
00106 {};
```

### 5.26.3.9 m\_linear\_pid

```
PID wisco::control::boomerang::PIDBoomerang::m_linear_pid {} [private]
```

The linear PID controller.

Definition at line 112 of file [PIDBoomerang.hpp](#).

```
00112 {};
```

### 5.26.3.10 m\_rotational\_pid

```
PID wisco::control::boomerang::PIDBoomerang::m_rotational_pid {} [private]
```

The rotational PID controller.

Definition at line 118 of file [PIDBoomerang.hpp](#).

```
00118 {};
```

### 5.26.3.11 m\_lead

```
double wisco::control::boomerang::PIDBoomerang::m_lead {} [private]
```

The lead ratio for the carrot point.

Definition at line 124 of file [PIDBoomerang.hpp](#).

```
00124 {};
```

### 5.26.3.12 m\_target\_tolerance

```
double wisco::control::boomerang::PIDBoomerang::m_target_tolerance {} [private]
```

The acceptable tolerance to reach the target.

Definition at line 130 of file [PIDBoomerang.hpp](#).

```
00130 {};
```

### 5.26.3.13 control\_robot

```
std::shared_ptr<robot::Robot> wisco::control::boomerang::PIDBoomerang::control_robot {} [private]
```

The robot being controlled.

Definition at line 136 of file [PIDBoomerang.hpp](#).

```
00136 {};
```

### 5.26.3.14 motion\_velocity

```
double wisco::control::boomerang::PIDBoomerang::motion_velocity {} [private]
```

The motion velocity in inches per second.

Definition at line 142 of file [PIDBoomerang.hpp](#).

```
00142 {};
```

### 5.26.3.15 target\_x

```
double wisco::control::boomerang::PIDBoomerang::target_x {} [private]
```

The target x-coordinate.

Definition at line 148 of file [PIDBoomerang.hpp](#).

```
00148 {};
```

### 5.26.3.16 target\_y

```
double wisco::control::boomerang::PIDBoomerang::target_y {} [private]
```

The target y-coordinate.

Definition at line 154 of file [PIDBoomerang.hpp](#).

```
00154 {};
```

### 5.26.3.17 target\_theta

```
double wisco::control::boomerang::PIDBoomerang::target_theta {} [private]
```

The target angle.

Definition at line 160 of file [PIDBoomerang.hpp](#).

```
00160 {};
```

### 5.26.3.18 paused

```
bool wisco::control::boomerang::PIDBoomerang::paused {} [private]
```

Whether or not the motion has been paused.

Definition at line 166 of file [PIDBoomerang.hpp](#).

```
00166 {};
```

### 5.26.3.19 target\_reached

```
bool wisco::control::boomerang::PIDBoomerang::target_reached {} [private]
```

Whether or not the target point has been reached.

Definition at line 172 of file [PIDBoomerang.hpp](#).  
00172 {};

## 5.27 wisco::control::boomerang::PIDBoomerangBuilder Class Reference

Builder class for [PID](#) boomerang controllers.

### Public Member Functions

- [PIDBoomerangBuilder \\* withDelayer](#) (const std::unique\_ptr< [rtos::IDelay](#) > &delayer)  
*Adds an rtos delayer to the build.*
- [PIDBoomerangBuilder \\* withMutex](#) (std::unique\_ptr< [rtos::IMutex](#) > &mutex)  
*Adds an rtos mutex to the build.*
- [PIDBoomerangBuilder \\* withTask](#) (std::unique\_ptr< [rtos::ITask](#) > &task)  
*Adds an rtos task to the build.*
- [PIDBoomerangBuilder \\* withLinearPID](#) ([PID](#) linear\_pid)  
*Adds a linear [PID](#) controller to the build.*
- [PIDBoomerangBuilder \\* withRotationalPID](#) ([PID](#) rotational\_pid)  
*Adds a rotational [PID](#) controller to the build.*
- [PIDBoomerangBuilder \\* withLead](#) (double lead)  
*Adds a lead ratio to the build.*
- [PIDBoomerangBuilder \\* withTargetTolerance](#) (double target\_tolerance)  
*Adds a target tolerance to the build.*
- std::unique\_ptr< [IBoomerang](#) > [build](#) ()  
*Builds the [PID](#) boomerang controller.*

### Private Attributes

- std::unique\_ptr< [rtos::IDelay](#) > [m\\_delayer](#) {}  
*The rtos delayer.*
- std::unique\_ptr< [rtos::IMutex](#) > [m\\_mutex](#) {}  
*The rtos mutex.*
- std::unique\_ptr< [rtos::ITask](#) > [m\\_task](#) {}  
*The rtos task.*
- [PID](#) [m\\_linear\\_pid](#) {}  
*The linear [PID](#) controller.*
- [PID](#) [m\\_rotational\\_pid](#) {}  
*The rotational [PID](#) controller.*
- double [m\\_lead](#) {}  
*The lead ratio for the carrot point.*
- double [m\\_target\\_tolerance](#) {}  
*The acceptable tolerance to reach the target.*

### 5.27.1 Detailed Description

Builder class for PID boomerang controllers.

#### Author

Nathan Sandvig

Definition at line 35 of file [PIDBoomerangBuilder.hpp](#).

### 5.27.2 Member Function Documentation

#### 5.27.2.1 withDelayer()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withDelayer ( const std::unique_ptr< rtos::IDelayer > & delayer )
```

Adds an rtos delayer to the build.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

#### Returns

PIDBoomerangBuilder\* This object for build chaining

Definition at line 9 of file [PIDBoomerangBuilder.cpp](#).

```
00010 {  
00011     m_delayer = delayer->clone();  
00012     return this;  
00013 }
```

#### 5.27.2.2 withMutex()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withMutex ( std::unique_ptr< rtos::IMutex > & mutex )
```

Adds an rtos mutex to the build.

#### Parameters

<i>mutex</i>	The rtos mutex
--------------	----------------

#### Returns

PIDBoomerangBuilder\* This object for build chaining

Definition at line 15 of file [PIDBoomerangBuilder.cpp](#).

```
00016 {
00017     m_mutex = std::move(mutex);
00018     return this;
00019 }
```

### 5.27.2.3 withTask()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withTask (
    std::unique_ptr< rtos::ITask > & task )
```

Adds an rtos task to the build.

#### Parameters

<i>task</i>	The rtos task
-------------	---------------

#### Returns

PIDBoomerangBuilder\* This object for build chaining

Definition at line 21 of file [PIDBoomerangBuilder.cpp](#).

```
00022 {
00023     m_task = std::move(task);
00024     return this;
00025 }
```

### 5.27.2.4 withLinearPID()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withLinearPID (
    PID linear_pid )
```

Adds a linear PID controller to the build.

#### Parameters

<i>linear_pid</i>	The linear PID controller
-------------------	---------------------------

#### Returns

PIDBoomerangBuilder\* This object for build chaining

Definition at line 27 of file [PIDBoomerangBuilder.cpp](#).

```
00028 {
00029     m_linear_pid = linear_pid;
00030     return this;
00031 }
```

### 5.27.2.5 withRotationalPID()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withRotationalPID (
    PID rotational_pid )
```

Adds a rotational PID controller to the build.

**Parameters**

<i>rotational_pid</i>	The rotational PID controller
-----------------------	-------------------------------

**Returns**

PIDBoomerangBuilder\* This object for build chaining

Definition at line 33 of file [PIDBoomerangBuilder.cpp](#).

```
00034 {  
00035     m_rotational_pid = rotational_pid;  
00036     return this;  
00037 }
```

### 5.27.2.6 withLead()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withLead (  
    double lead )
```

Adds a lead ratio to the build.

**Parameters**

<i>lead</i>	The lead ratio
-------------	----------------

**Returns**

PIDBoomerangBuilder\* This object for build chaining

Definition at line 39 of file [PIDBoomerangBuilder.cpp](#).

```
00040 {  
00041     m_lead = lead;  
00042     return this;  
00043 }
```

### 5.27.2.7 withTargetTolerance()

```
PIDBoomerangBuilder * wisco::control::boomerang::PIDBoomerangBuilder::withTargetTolerance (  
    double target_tolerance )
```

Adds a target tolerance to the build.

**Parameters**

<i>target_tolerance</i>	The motion target tolerance
-------------------------	-----------------------------

**Returns**

PIDBoomerangBuilder\* This object for build chaining

Definition at line 45 of file [PIDBoomerangBuilder.cpp](#).

```
00046 {
00047     m_target_tolerance = target_tolerance;
00048     return this;
00049 }
```

### 5.27.2.8 build()

```
std::unique_ptr<IBoomerang> wisco::control::boomerang::PIDBoomerangBuilder::build()
```

Builds the **PID** boomerang controller.

#### Returns

`std::unique_ptr<IBoomerang>` The **PID** boomerang controller as a boomerang interface object

Definition at line 51 of file [PIDBoomerangBuilder.cpp](#).

```
00052 {
00053     std::unique_ptr<PIDBoomerang> pid_boomerang{std::make_unique<PIDBoomerang>()};
00054     pid_boomerang->setDelayer(m_delayer);
00055     pid_boomerang->setMutex(m_mutex);
00056     pid_boomerang->setTask(m_task);
00057     pid_boomerang->setLinearPID(m_linear_pid);
00058     pid_boomerang->setRotationalPID(m_rotational_pid);
00059     pid_boomerang->setLead(m_lead);
00060     pid_boomerang->setTargetTolerance(m_target_tolerance);
00061     return pid_boomerang;
00062 }
```

## 5.27.3 Member Data Documentation

### 5.27.3.1 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::control::boomerang::PIDBoomerangBuilder::m_delayer {}  
[private]
```

The rtos delayer.

Definition at line 41 of file [PIDBoomerangBuilder.hpp](#).

```
00041 {};
```

### 5.27.3.2 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::control::boomerang::PIDBoomerangBuilder::m_mutex {}  
[private]
```

The rtos mutex.

Definition at line 47 of file [PIDBoomerangBuilder.hpp](#).

```
00047 {};
```

### 5.27.3.3 m\_task

```
std::unique_ptr<rtos::ITask> wisco::control::boomerang::PIDBoomerangBuilder::m_task {} [private]
```

The rtos task.

Definition at line 53 of file [PIDBoomerangBuilder.hpp](#).

```
00053 {};
```

### 5.27.3.4 m\_linear\_pid

```
PID wisco::control::boomerang::PIDBoomerangBuilder::m_linear_pid {} [private]
```

The linear PID controller.

Definition at line 59 of file [PIDBoomerangBuilder.hpp](#).  
00059 {};

### 5.27.3.5 m\_rotational\_pid

```
PID wisco::control::boomerang::PIDBoomerangBuilder::m_rotational_pid {} [private]
```

The rotational PID controller.

Definition at line 65 of file [PIDBoomerangBuilder.hpp](#).  
00065 {};

### 5.27.3.6 m\_lead

```
double wisco::control::boomerang::PIDBoomerangBuilder::m_lead {} [private]
```

The lead ratio for the carrot point.

Definition at line 71 of file [PIDBoomerangBuilder.hpp](#).  
00071 {};

### 5.27.3.7 m\_target\_tolerance

```
double wisco::control::boomerang::PIDBoomerangBuilder::m_target_tolerance {} [private]
```

The acceptable tolerance to reach the target.

Definition at line 77 of file [PIDBoomerangBuilder.hpp](#).  
00077 {};

## 5.28 wisco::control::ControlSystem Class Reference

A container class for controls.

### Public Member Functions

- void [addControl](#) (std::unique\_ptr< AControl > &control)  
*Adds a control to the robot.*
- bool [removeControl](#) (std::string control\_name)  
*Removes a control from the robot.*
- void [initialize](#) ()  
*Initializes all controls in the robot.*
- void [sendCommand](#) (std::string control\_name, std::string command\_name,...)  
*Sends a command to a control.*
- void \* [getState](#) (std::string control\_name, std::string state\_name)  
*Gets a state of a control.*

### Private Attributes

- std::vector< std::unique\_ptr< AControl > > **controls** {}  
*The controls contained in the robot.*

## 5.28.1 Detailed Description

A container class for controls.

### Author

Nathan Sandvig

Definition at line 31 of file [ControlSystem.hpp](#).

## 5.28.2 Member Function Documentation

### 5.28.2.1 addControl()

```
void wisco::control::ControlSystem::addControl ( 
    std::unique_ptr< AControl > & control )
```

Adds a control to the robot.

#### Parameters

<i>control</i>	The control being added to the robot
----------------	--------------------------------------

Definition at line 7 of file [ControlSystem.cpp](#).

```
00008 { 
00009     controls.push_back(std::move(control));
00010 }
```

### 5.28.2.2 removeControl()

```
bool wisco::control::ControlSystem::removeControl ( 
    std::string control_name )
```

Removes a control from the robot.

#### Parameters

<i>control_name</i>	The name of the control to remove from the robot
---------------------	--

#### Returns

true The control was removed  
false The control was not contained in the robot

Definition at line 12 of file [ControlSystem.cpp](#).

```
00013 {
00014     bool removed{false};
00015     for (auto it{controls.begin()}; it != controls.end(); ++it)
00016     {
00017         if ((*it)->getName() == control_name)
00018         {
00019             controls.erase(it);
00020             removed = true;
00021             break;
00022         }
00023     }
00024     return removed;
00025 }
```

### 5.28.2.3 initialize()

```
void wisco::control::ControlSystem::initialize ( )
```

Initializes all controls in the robot.

Definition at line 27 of file [ControlSystem.cpp](#).

```
00028 {
00029     for (auto& control : controls)
00030         control->initialize();
00031     for (auto& control : controls)
00032         control->run();
00033 }
```

### 5.28.2.4 sendCommand()

```
void wisco::control::ControlSystem::sendCommand (
    std::string control_name,
    std::string command_name,
    ... )
```

Sends a command to a control.

#### Parameters

<i>control_name</i>	The name of the control
<i>command_name</i>	The name of the command
...	The parameters for the command

Definition at line 35 of file [ControlSystem.cpp](#).

```
00036 {
00037     va_list args;
00038     va_start(args, command_name);
00039     for (auto& control : controls)
00040     {
00041         if (control->getName() == control_name)
00042         {
00043             control->command(command_name, args);
00044             break;
00045         }
00046     }
00047     va_end(args);
00048 }
```

### 5.28.2.5 getState()

```
void * wisco::control::ControlSystem::getState (
    std::string control_name,
    std::string state_name )
```

Gets a state of a control.

#### Parameters

<i>control_name</i>	The name of the control
<i>state_name</i>	The name of the state

#### Returns

`void*` The current value of that state

Definition at line 50 of file [ControlSystem.cpp](#).

```
00051 {
00052     void* state{nullptr};
00053     for (auto& control : controls)
00054     {
00055         if (control->getName() == control_name)
00056         {
00057             state = control->state(state_name);
00058             break;
00059         }
00060     }
00061     return state;
00062 }
```

## 5.28.3 Member Data Documentation

### 5.28.3.1 controls

`std::vector<std::unique_ptr<AControl>>` `wisco::control::ControlSystem::controls {} [private]`

The controls contained in the robot.

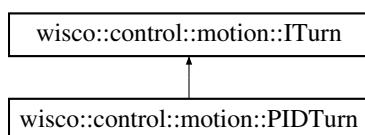
Definition at line 38 of file [ControlSystem.hpp](#).

```
00038 {};
```

## 5.29 wisco::control::motion::ITurn Class Reference

Interface for turn motion controls.

Inheritance diagram for `wisco::control::motion::ITurn`:



**Public Member Functions**

- virtual ~**ITurn** ()=default  
*Destroy the **ITurn** object.*
- virtual void **initialize** ()=0  
*Initializes the turn.*
- virtual void **run** ()=0  
*Runs the turn.*
- virtual void **turnToAngle** (const std::shared\_ptr< **robot::Robot** > &**robot**, double **velocity**, double **theta**, **ETurnDirection** **direction**=**ETurnDirection::AUTO**)=0  
*Turns to the target angle.*
- virtual void **turnToPoint** (const std::shared\_ptr< **robot::Robot** > &**robot**, double **velocity**, double **x**, double **y**, **ETurnDirection** **direction**=**ETurnDirection::AUTO**)=0  
*Turns to the target point.*
- virtual void **pause** ()=0  
*Pauses the current motion.*
- virtual void **resume** ()=0  
*Resumes the current motion.*
- virtual bool **targetReached** ()=0  
*Checks if the target has been reached.*

**5.29.1 Detailed Description**

Interface for turn motion controls.

**Author**

Nathan Sandvig

Definition at line 39 of file [ITurn.hpp](#).

**5.29.2 Member Function Documentation****5.29.2.1 initialize()**

```
virtual void wisco::control::motion::ITurn::initialize ( ) [pure virtual]
```

Initializes the turn.

Implemented in [wisco::control::motion::PIDTurn](#).

**5.29.2.2 run()**

```
virtual void wisco::control::motion::ITurn::run ( ) [pure virtual]
```

Runs the turn.

Implemented in [wisco::control::motion::PIDTurn](#).

**5.29.2.3 turnToAngle()**

```
virtual void wisco::control::motion::ITurn::turnToAngle (
    const std::shared_ptr< robot::Robot > &robot,
    double velocity,
    double theta,
    ETurnDirection direction = ETurnDirection::AUTO ) [pure virtual]
```

Turns to the target angle.

**Parameters**

<i>robot</i>	The robot
<i>velocity</i>	The angular velocity
<i>theta</i>	The target angle
<i>direction</i>	The turn direction (default AUTO)

Implemented in [wisco::control::motion::PIDTurn](#).

**5.29.2.4 turnToPoint()**

```
virtual void wisco::control::motion::ITurn::turnToPoint (
    const std::shared_ptr< robot::Robot > & robot,
    double velocity,
    double x,
    double y,
    ETurnDirection direction = ETurnDirection::AUTO ) [pure virtual]
```

Turns to the target point.

**Parameters**

<i>robot</i>	The robot
<i>velocity</i>	The angular velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>direction</i>	The turn direction (default AUTO)

Implemented in [wisco::control::motion::PIDTurn](#).

**5.29.2.5 pause()**

```
virtual void wisco::control::motion::ITurn::pause ( ) [pure virtual]
```

Pauses the current motion.

Implemented in [wisco::control::motion::PIDTurn](#).

**5.29.2.6 resume()**

```
virtual void wisco::control::motion::ITurn::resume ( ) [pure virtual]
```

Resumes the current motion.

Implemented in [wisco::control::motion::PIDTurn](#).

### 5.29.2.7 targetReached()

```
virtual bool wisco::control::motion::ITurn::targetReached ( ) [pure virtual]
```

Checks if the target has been reached.

#### Returns

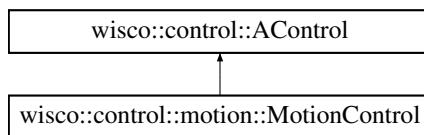
- true The target has been reached
- false The target has not been reached

Implemented in [wisco::control::motion::PIDTurn](#).

## 5.30 wisco::control::motion::MotionControl Class Reference

Control adapter for the motion controller.

Inheritance diagram for wisco::control::motion::MotionControl:



### Public Member Functions

- **MotionControl** (std::unique\_ptr<[ITurn](#)> &turn)  
*Construct a new Motion Control object.*
- void **initialize** () override  
*Initializes the control.*
- void **run** () override  
*Runs the control.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the control.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the control.*

### Public Member Functions inherited from [wisco::control::AControl](#)

- **AControl** ()=default  
*Construct a new AControl object.*
- **AControl** (const [AControl](#) &other)=default  
*Construct a new AControl object.*
- **AControl** ([AControl](#) &&other)=default  
*Construct a new AControl object.*
- **AControl** (std::string name)  
*Construct a new AControl object.*
- virtual ~**AControl** ()=default  
*Destroy the AControl object.*
- const std::string & **getName** () const  
*Get the name of the control.*
- **AControl** & **operator=** (const [AControl](#) &rhs)=default  
*Copy assignment operator for AControl.*
- **AControl** & **operator=** ([AControl](#) &&rhs)=default  
*Move assignment operator for AControl.*

## Private Attributes

- std::unique\_ptr< ITurn > m\_turn {}  
*The turn controller being adapted.*

## Static Private Attributes

- static constexpr char CONTROL\_NAME [] {"MOTION"}  
*The name of the control.*
- static constexpr char TURN\_TO\_ANGLE\_COMMAND\_NAME [] {"TURN TO ANGLE"}  
*The name of the turn to angle command.*
- static constexpr char TURN\_TO\_POINT\_COMMAND\_NAME [] {"TURN TO POINT"}  
*The name of the turn to point command.*
- static constexpr char PAUSE\_TURN\_COMMAND\_NAME [] {"PAUSE TURN"}  
*The name of the pause turn command.*
- static constexpr char RESUME\_TURN\_COMMAND\_NAME [] {"RESUME TURN"}  
*The name of the resume turn command.*
- static constexpr char TURN\_TARGET\_REACHED\_STATE\_NAME [] {"TURN TARGET REACHED"}  
*The name of the turn target reached state.*

### 5.30.1 Detailed Description

Control adapter for the motion controller.

#### Author

Nathan Sandvig

Definition at line 38 of file MotionControl.hpp.

### 5.30.2 Constructor & Destructor Documentation

#### 5.30.2.1 MotionControl()

```
wisco::control::motion::MotionControl::MotionControl (
    std::unique_ptr< ITurn > & turn )
```

Construct a new Motion Control object.

#### Parameters

<i>turn</i>	The turn controller being adapted
-------------	-----------------------------------

Definition at line 9 of file MotionControl.cpp.

```
00010     : AControl(CONTROL_NAME), m_turn{std::move(turn) }
00011 {
00012
00013 }
```

### 5.30.3 Member Function Documentation

#### 5.30.3.1 initialize()

```
void wisco::control::motion::MotionControl::initialize ( ) [override], [virtual]
```

Initializes the control.

Implements [wisco::control::AControl](#).

Definition at line 15 of file [MotionControl.cpp](#).

```
00016 {
00017     if (m_turn)
00018         m_turn->initialize();
00019 }
```

#### 5.30.3.2 run()

```
void wisco::control::motion::MotionControl::run ( ) [override], [virtual]
```

Runs the control.

Implements [wisco::control::AControl](#).

Definition at line 21 of file [MotionControl.cpp](#).

```
00022 {
00023     if (m_turn)
00024         m_turn->run();
00025 }
```

#### 5.30.3.3 command()

```
void wisco::control::motion::MotionControl::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the control.

##### Parameters

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::control::AControl](#).

Definition at line 27 of file [MotionControl.cpp](#).

```
00028 {
00029     if (command_name == TURN_TO_ANGLE_COMMAND_NAME)
00030     {
00031         void* robot_ptr{va_arg(args, void*)};
00032         std::shared_ptr<robot::Robot> robot{*static_cast<std::shared_ptr<robot::Robot>>(robot_ptr)};
00033         double velocity{va_arg(args, double)};
00034         double theta{va_arg(args, double)};
00035         ETurnDirection direction{va_arg(args, ETurnDirection)};
00036         m_turn->turnToAngle(robot, velocity, theta, direction);
00037     }
00038     else if (command_name == TURN_TO_POINT_COMMAND_NAME)
00039     {
```

```

00040     void* robot_ptr{va_arg(args, void*)};
00041     std::shared_ptr<robot::Robot> robot{*static_cast<std::shared_ptr<robot::Robot>>(robot_ptr)};
00042     double velocity{va_arg(args, double)};
00043     double x{va_arg(args, double)};
00044     double y{va_arg(args, double)};
00045     ETurnDirection direction{va_arg(args, ETurnDirection)};
00046     m_turn->turnToPoint(robot, velocity, x, y, direction);
00047 }
00048 else if (command_name == PAUSE_TURN_COMMAND_NAME)
00049 {
00050     m_turn->pause();
00051 }
00052 else if (command_name == RESUME_TURN_COMMAND_NAME)
00053 {
00054     m_turn->resume();
00055 }
00056 }
```

### 5.30.3.4 state()

```
void * wisco::control::motion::MotionControl::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the control.

#### Parameters

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::control::AControl](#).

Definition at line 58 of file [MotionControl.cpp](#).

```

00059 {
00060     void* result=nullptr;
00061
00062     if (state_name == TURN_TARGET_REACHED_STATE_NAME)
00063     {
00064         bool* velocity{new bool{m_turn->targetReached()}};
00065         result = velocity;
00066     }
00067
00068     return result;
00069 }
```

## 5.30.4 Member Data Documentation

### 5.30.4.1 CONTROL\_NAME

```
constexpr char wisco::control::motion::MotionControl::CONTROL_NAME[ ] {"MOTION"} [static],
[constexpr], [private]
```

The name of the control.

Definition at line 45 of file [MotionControl.hpp](#).

```
00045 {"MOTION"};
```

#### 5.30.4.2 TURN\_TO\_ANGLE\_COMMAND\_NAME

```
constexpr char wisco::control::motion::MotionControl::TURN_TO_ANGLE_COMMAND_NAME[ ] { "TURN TO  
ANGLE" } [static], [constexpr], [private]
```

The name of the turn to angle command.

Definition at line 51 of file [MotionControl.hpp](#).

```
00051 {"TURN TO ANGLE"};
```

#### 5.30.4.3 TURN\_TO\_POINT\_COMMAND\_NAME

```
constexpr char wisco::control::motion::MotionControl::TURN_TO_POINT_COMMAND_NAME[ ] { "TURN TO  
POINT" } [static], [constexpr], [private]
```

The name of the turn to point command.

Definition at line 57 of file [MotionControl.hpp](#).

```
00057 {"TURN TO POINT"};
```

#### 5.30.4.4 PAUSE\_TURN\_COMMAND\_NAME

```
constexpr char wisco::control::motion::MotionControl::PAUSE_TURN_COMMAND_NAME[ ] { "PAUSE TURN" }  
[static], [constexpr], [private]
```

The name of the pause turn command.

Definition at line 63 of file [MotionControl.hpp](#).

```
00063 {"PAUSE TURN"};
```

#### 5.30.4.5 RESUME\_TURN\_COMMAND\_NAME

```
constexpr char wisco::control::motion::MotionControl::RESUME_TURN_COMMAND_NAME[ ] { "RESUME  
TURN" } [static], [constexpr], [private]
```

The name of the resume turn command.

Definition at line 69 of file [MotionControl.hpp](#).

```
00069 {"RESUME TURN"};
```

#### 5.30.4.6 TURN\_TARGET\_REACHED\_STATE\_NAME

```
constexpr char wisco::control::motion::MotionControl::TURN_TARGET_REACHED_STATE_NAME[ ] { "TURN  
TARGET REACHED" } [static], [constexpr], [private]
```

The name of the turn target reached state.

Definition at line 75 of file [MotionControl.hpp](#).

```
00075 {"TURN TARGET REACHED"};
```

### 5.30.4.7 m\_turn

```
std::unique_ptr<ITurn> wisco::control::motion::MotionControl::m_turn {} [private]
```

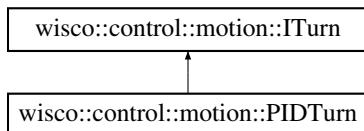
The turn controller being adapted.

Definition at line 81 of file [MotionControl.hpp](#).  
00081 {};

## 5.31 wisco::control::motion::PIDTurn Class Reference

Interface for turn motion controls.

Inheritance diagram for wisco::control::motion::PIDTurn:



### Public Member Functions

- void [initialize \(\)](#) override  
*Initializes the turn.*
- void [run \(\)](#) override  
*Runs the turn.*
- void [turnToAngle \(const std::shared\\_ptr< robot::Robot > &robot, double velocity, double theta, ETurnDirection direction=ETurnDirection::AUTO\)](#) override  
*Turns to the target angle.*
- void [turnToPoint \(const std::shared\\_ptr< robot::Robot > &robot, double velocity, double x, double y, ETurnDirection direction=ETurnDirection::AUTO\)](#) override  
*Turns to the target point.*
- void [pause \(\)](#) override  
*Pauses the current motion.*
- void [resume \(\)](#) override  
*Resumes the current motion.*
- bool [targetReached \(\)](#) override  
*Checks if the target has been reached.*
- void [setDelayer \(const std::unique\\_ptr< rtos::IDelayer > &delayer\)](#)  
*Sets the rtos delayer.*
- void [setMutex \(std::unique\\_ptr< rtos::IMutex > &mutex\)](#)  
*Sets the rtos mutex.*
- void [setTask \(std::unique\\_ptr< rtos::ITask > &task\)](#)  
*Sets the rtos task.*
- void [setPID \(PID pid\)](#)  
*Sets the PID controller.*
- void [setTargetTolerance \(double target\\_tolerance\)](#)  
*Sets the target tolerance.*
- void [setTargetVelocity \(double target\\_velocity\)](#)  
*Sets the target velocity.*

## Public Member Functions inherited from [wisco::control::motion::ITurn](#)

- virtual ~[ITurn](#) ()=default  
*Destroy the [ITurn](#) object.*

## Private Member Functions

- void [taskUpdate](#) ()  
*Runs all the object-specific updates in the task loop.*
- void [setDriveVelocity](#) ([robot::subsystems::drive::Velocity](#) velocity)  
*Set the velocity for the drive.*
- double [getDriveRadius](#) ()  
*Get the radius of the drive.*
- [robot::subsystems::position::Position](#) [getOdometryPosition](#) ()  
*Get the position from the odometry.*
- double [calculateAngleToTarget](#) ([wisco::robot::subsystems::position::Position](#) position)  
*Calculates the angle to the target position.*
- [robot::subsystems::drive::Velocity](#) [calculateDriveVelocity](#) (double robot\_angle, double target\_angle)  
*Calculates the velocity of the drive.*

## Static Private Member Functions

- static void [taskLoop](#) (void \*params)  
*The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr< [rtos::IDelayer](#) > [m\\_delayer](#) {}  
*The rtos delayer.*
- std::unique\_ptr< [rtos::IMutex](#) > [m\\_mutex](#) {}  
*The rtos mutex.*
- std::unique\_ptr< [rtos::ITask](#) > [m\\_task](#) {}  
*The rtos task.*
- [PID](#) [m\\_pid](#) {}  
*The PID controller.*
- double [m\\_target\\_tolerance](#) {}  
*The acceptable tolerance to reach the target.*
- double [m\\_target\\_velocity](#) {}  
*The acceptable velocity to reach the target.*
- std::shared\_ptr< [robot::Robot](#) > [control\\_robot](#) {}  
*The robot being controlled.*
- [ETurnDirection](#) [turn\\_direction](#) {}  
*The turn direction.*
- double [turn\\_velocity](#) {}  
*The turn velocity in radians per second.*
- double [target\\_x](#) {}  
*The target x-coordinate.*
- double [target\\_y](#) {}  
*The target y-coordinate.*
- bool [paused](#) {}  
*Whether or not the motion has been paused.*
- bool [target\\_reached](#) {}  
*Whether or not the target point has been reached.*
- bool [forced\\_direction\\_reached](#) {}  
*Whether or not the forced direction has been reached.*

## Static Private Attributes

- static constexpr uint8\_t **TASK\_DELAY** {10}  
*The loop delay on the task.*
- static constexpr char **DRIVE\_SUBSYSTEM\_NAME** [] {"DIFFERENTIAL DRIVE"}  
*The name of the drive subsystem.*
- static constexpr char **ODOMETRY\_SUBSYSTEM\_NAME** [] {"POSITION TRACKER"}  
*The name of the odometry subsystem.*
- static constexpr char **DRIVE\_SET\_VELOCITY\_COMMAND\_NAME** [] {"SET VELOCITY"}  
*The name of the drive set velocity command.*
- static constexpr char **DRIVE\_GET\_RADIUS\_STATE\_NAME** [] {"GET RADIUS"}  
*The name of the drive get radius command.*
- static constexpr char **ODOMETRY\_GET\_POSITION\_STATE\_NAME** [] {"GET POSITION"}  
*The name of the odometry get position command.*
- static constexpr double **TURN\_TO\_ANGLE\_DISTANCE** {120000}  
*The distance to calculate a point at for turn to angle.*

### 5.31.1 Detailed Description

Interface for turn motion controls.

#### Author

Nathan Sandvig

Definition at line 48 of file [PIDTurn.hpp](#).

### 5.31.2 Member Function Documentation

#### 5.31.2.1 taskLoop()

```
void wisco::control::motion::PIDTurn::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

#### Parameters

<i>params</i>	The task parameters
---------------	---------------------

Definition at line 11 of file [PIDTurn.cpp](#).

```
00012 {
00013     void** parameters{static_cast<void**>(params)};
00014     PIDTurn* instance{static_cast<PIDTurn*>(parameters[0])};
00015
00016     while (true)
00017     {
00018         instance->taskUpdate();
00019     }
00020 }
```

### 5.31.2.2 taskUpdate()

```
void wisco::control::motion::PIDTurn::taskUpdate ( ) [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 22 of file [PIDTurn.cpp](#).

```
00023 {
00024     if (m_mutex)
00025         m_mutex->take();
00026
00027     if (!paused && !target_reached)
00028     {
00029         auto position{getOdometryPosition()};
00030         double target_angle{calculateAngleToTarget(position)};
00031         double error{bindRadians(target_angle - position.theta)};
00032         if (std::abs(error) < m_target_tolerance && std::abs(position.thetaV) < m_target_velocity)
00033         {
00034             target_reached = true;
00035             robot::subsystems::drive::Velocity stop{0, 0};
00036             setDriveVelocity(stop);
00037         }
00038     else
00039     {
00040         robot::subsystems::drive::Velocity velocity{calculateDriveVelocity(position.theta,
00041             target_angle)};
00042         setDriveVelocity(velocity);
00043     }
00044
00045     if (m_mutex)
00046         m_mutex->give();
00047
00048     m_delayer->delay(TASK_DELAY);
00049 }
```

### 5.31.2.3 setDriveVelocity()

```
void wisco::control::motion::PIDTurn::setDriveVelocity (
    robot::subsystems::drive::Velocity velocity) [private]
```

Set the velocity for the drive.

#### Parameters

<i>velocity</i>	The velocity for the drive
-----------------	----------------------------

Definition at line 51 of file [PIDTurn.cpp](#).

```
00052 {
00053     if (control_robot)
00054         control_robot->sendCommand(DRIVE_SUBSYSTEM_NAME, DRIVE_SET_VELOCITY_COMMAND_NAME, velocity);
00055 }
```

### 5.31.2.4 getDriveRadius()

```
double wisco::control::motion::PIDTurn::getDriveRadius ( ) [private]
```

Get the radius of the drive.

**Returns**

`double` The radius of the drive

Definition at line 57 of file [PIDTurn.cpp](#).

```
00058 {
00059     double radius{};
00060
00061     if (control_robot)
00062     {
00063         double* result{static_cast<double*>(control_robot->getState(DRIVE_SUBSYSTEM_NAME,
00064                                         DRIVE_GET_RADIUS_STATE_NAME))};
00065         if (result)
00066         {
00067             radius = *result;
00068             delete result;
00069         }
00070     }
00071     return radius;
00072 }
```

**5.31.2.5 getOdometryPosition()**

```
robot::subsystems::position::Position wisco::control::motion::PIDTurn::getOdometryPosition ( )
[private]
```

Get the position from the odometry.

**Returns**

`robot::subsystems::position::Position` The position from the odometry

Definition at line 74 of file [PIDTurn.cpp](#).

```
00075 {
00076     robot::subsystems::position::Position position{};
00077
00078     if (control_robot)
00079     {
00080         robot::subsystems::position::Position*
00081         result{static_cast<robot::subsystems::position::Position*>(control_robot->getState(ODOMETRY_SUBSYSTEM_NAME,
00082                                         ODOMETRY_GET_POSITION_STATE_NAME))};
00083         if (result)
00084         {
00085             position = *result;
00086             delete result;
00087         }
00088     }
00089 }
```

**5.31.2.6 calculateAngleToTarget()**

```
double wisco::control::motion::PIDTurn::calculateAngleToTarget (
    wisco::robot::subsystems::position::Position position ) [private]
```

Calculates the angle to the target position.

**Parameters**

<code>position</code>	The current position
-----------------------	----------------------

**Returns**

`double` The angle to the target position

Definition at line 91 of file [PIDTurn.cpp](#).

```
00092 {
00093     double x_error{target_x - position.x};
00094     double y_error{target_y - position.y};
00095     double angle{std::atan2(y_error, x_error)};
00096     return angle;
00097 }
```

**5.31.2.7 calculateDriveVelocity()**

```
robot::subsystems::drive::Velocity wisco::control::motion::PIDTurn::calculateDriveVelocity (
    double robot_angle,
    double target_angle) [private]
```

Calculates the velocity of the drive.

**Parameters**

<code>robot_angle</code>	The robot angle
<code>target_angle</code>	The angle to the target point

**Returns**

`robot::subsystems::drive::Velocity` The drive velocity

Definition at line 99 of file [PIDTurn.cpp](#).

```
00100 {
00101     double error{bindRadians(target_angle - robot_angle)};
00102     if (!forced_direction_reached)
00103     {
00104         if (turn_direction == ETurnDirection::CLOCKWISE && error > 0)
00105             error -= 2 * M_PI;
00106         else if (turn_direction == ETurnDirection::COUNTERCLOCKWISE && error < 0)
00107             error += 2 * M_PI;
00108         if (std::abs(error) < M_PI)
00109             forced_direction_reached = true;
00110     }
00111     double control_value{m_pid.getControlValue(0, error)};
00112     if (std::abs(control_value) > turn_velocity)
00113         control_value *= turn_velocity / std::abs(control_value);
00114     robot::subsystems::drive::Velocity velocity{-control_value, control_value};
00115     return velocity;
00116 }
00117 }
```

**5.31.2.8 initialize()**

```
void wisco::control::motion::PIDTurn::initialize () [override], [virtual]
```

Initializes the turn.

Implements [wisco::control::motion::ITurn](#).

Definition at line 119 of file [PIDTurn.cpp](#).

```
00120 {
00121     m_pid.reset();
00122 }
```

### 5.31.2.9 run()

```
void wisco::control::motion::PIDTurn::run ( ) [override], [virtual]
```

Runs the turn.

Implements [wisco::control::motion::ITurn](#).

Definition at line 124 of file [PIDTurn.cpp](#).

```
00125 {
00126     if (m_task)
00127     {
00128         void** params{static_cast<void**>(malloc(l * sizeof(void*)))};
00129         params[0] = this;
00130         m_task->start(&PIDTurn::taskLoop, params);
00131     }
00132 }
```

### 5.31.2.10 turnToAngle()

```
void wisco::control::motion::PIDTurn::turnToAngle (
    const std::shared_ptr< robot::Robot > & robot,
    double velocity,
    double theta,
    ETurnDirection direction = ETurnDirection::AUTO ) [override], [virtual]
```

Turns to the target angle.

#### Parameters

<i>robot</i>	The robot
<i>velocity</i>	The angular velocity
<i>theta</i>	The target angle
<i>direction</i>	The turn direction (default AUTO)

Implements [wisco::control::motion::ITurn](#).

Definition at line 134 of file [PIDTurn.cpp](#).

```
00135 {
00136     if (m_mutex)
00137         m_mutex->take();
00138
00139     m_pid.reset();
00140
00141     control_robot = robot;
00142     double drive_radius{getDriveRadius()};
00143     auto position{getOdometryPosition()};
00144
00145     turn_direction = direction;
00146     turn_velocity = velocity * drive_radius;
00147     target_x = position.x + (TURN_TO_ANGLE_DISTANCE * std::cos(theta));
00148     target_y = position.y + (TURN_TO_ANGLE_DISTANCE * std::sin(theta));
00149     target_reached = false;
00150     forced_direction_reached = false;
00151
00152     if (m_mutex)
00153         m_mutex->give();
00154 }
```

### 5.31.2.11 turnToPoint()

```
void wisco::control::motion::PIDTurn::turnToPoint (
    const std::shared_ptr< robot::Robot > & robot,
```

```
    double velocity,
    double x,
    double y,
    ETurnDirection direction = ETurnDirection::AUTO ) [override], [virtual]
```

Turns to the target point.

#### Parameters

<i>robot</i>	The robot
<i>velocity</i>	The angular velocity
<i>x</i>	The target x-coordinate
<i>y</i>	The target y-coordinate
<i>direction</i>	The turn direction (default AUTO)

Implements [wisco::control::motion::ITurn](#).

Definition at line 156 of file [PIDTurn.cpp](#).

```
00157 {
00158     if (m_mutex)
00159         m_mutex->take();
00160
00161     m_pid.reset();
00162
00163     control_robot = robot;
00164     double drive_radius{getDriveRadius()};
00165
00166     turn_direction = direction;
00167     turn_velocity = velocity * drive_radius;
00168     target_x = x;
00169     target_y = y;
00170     target_reached = false;
00171     forced_direction_reached = false;
00172
00173     if (m_mutex)
00174         m_mutex->give();
00175 }
```

#### 5.31.2.12 pause()

```
void wisco::control::motion::PIDTurn::pause () [override], [virtual]
```

Pauses the current motion.

Implements [wisco::control::motion::ITurn](#).

Definition at line 177 of file [PIDTurn.cpp](#).

```
00178 {
00179     if (m_mutex)
00180         m_mutex->take();
00181
00182     paused = true;
00183     robot::subsystems::drive::Velocity stop{0, 0};
00184     setDriveVelocity(stop);
00185
00186     if (m_mutex)
00187         m_mutex->give();
00188 }
```

### 5.31.2.13 resume()

```
void wisco::control::motion::PIDTurn::resume () [override], [virtual]
```

Resumes the current motion.

Implements [wisco::control::motion::ITurn](#).

Definition at line 190 of file [PIDTurn.cpp](#).

```
00191 {
00192     if (m_mutex)
00193         m_mutex->take();
00194
00195     paused = false;
00196
00197     if (m_mutex)
00198         m_mutex->give();
00199 }
```

### 5.31.2.14 targetReached()

```
bool wisco::control::motion::PIDTurn::targetReached () [override], [virtual]
```

Checks if the target has been reached.

#### Returns

- true The target has been reached
- false The target has not been reached

Implements [wisco::control::motion::ITurn](#).

Definition at line 201 of file [PIDTurn.cpp](#).

```
00202 {
00203     return target_reached;
00204 }
```

### 5.31.2.15 setDelayer()

```
void wisco::control::motion::PIDTurn::setDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Sets the rtos delayer.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 206 of file [PIDTurn.cpp](#).

```
00207 {
00208     m_delayer = delayer->clone();
00209 }
```

### 5.31.2.16 setMutex()

```
void wisco::control::motion::PIDTurn::setMutex (
```

```
std::unique_ptr< rtos::IMutex > & mutex )
```

Sets the rtos mutex.

**Parameters**

<i>mutex</i>	The rtos mutex
--------------	----------------

Definition at line 211 of file [PIDTurn.cpp](#).

```
00212 {
00213     m_mutex = std::move(mutex);
00214 }
```

### 5.31.2.17 setTask()

```
void wisco::control::motion::PIDTurn::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Sets the rtos task.

**Parameters**

<i>task</i>	The rtos task
-------------	---------------

Definition at line 216 of file [PIDTurn.cpp](#).

```
00217 {
00218     m_task = std::move(task);
00219 }
```

### 5.31.2.18 setPID()

```
void wisco::control::motion::PIDTurn::setPID (
    PID pid )
```

Sets the [PID](#) controller.

**Parameters**

<i>pid</i>	The <a href="#">PID</a> controller
------------	------------------------------------

Definition at line 221 of file [PIDTurn.cpp](#).

```
00222 {
00223     m_pid = pid;
00224 }
```

### 5.31.2.19 setTargetTolerance()

```
void wisco::control::motion::PIDTurn::setTargetTolerance (
    double target_tolerance )
```

Sets the target tolerance.

**Parameters**

<i>target_tolerance</i>	The motion target tolerance
-------------------------	-----------------------------

Definition at line 226 of file [PIDTurn.cpp](#).

```
00227 {  
00228     m_target_tolerance = target_tolerance;  
00229 }
```

### 5.31.2.20 setTargetVelocity()

```
void wisco::control::motion::PIDTurn::setTargetVelocity (  
    double target_velocity )
```

Sets the target velocity.

**Parameters**

<i>target_velocity</i>	The motion target velocity
------------------------	----------------------------

Definition at line 231 of file [PIDTurn.cpp](#).

```
00232 {  
00233     m_target_velocity = target_velocity;  
00234 }
```

## 5.31.3 Member Data Documentation

### 5.31.3.1 TASK\_DELAY

```
constexpr uint8_t wisco::control::motion::PIDTurn::TASK_DELAY {10} [static], [constexpr],  
[private]
```

The loop delay on the task.

Definition at line 55 of file [PIDTurn.hpp](#).

```
00055 {10};
```

### 5.31.3.2 DRIVE\_SUBSYSTEM\_NAME

```
constexpr char wisco::control::motion::PIDTurn::DRIVE_SUBSYSTEM_NAME[] {"DIFFERENTIAL DRIVE"}  
[static], [constexpr], [private]
```

The name of the drive subsystem.

Definition at line 61 of file [PIDTurn.hpp](#).

```
00061 {"DIFFERENTIAL DRIVE"};
```

### 5.31.3.3 ODOMETRY\_SUBSYSTEM\_NAME

```
constexpr char wisco::control::motion::PIDTurn::ODOMETRY_SUBSYSTEM_NAME[ ] {"POSITION TRACKER"}  
[static], [constexpr], [private]
```

The name of the odometry subsystem.

Definition at line 67 of file [PIDTurn.hpp](#).

```
00067 {"POSITION TRACKER"};
```

### 5.31.3.4 DRIVE\_SET\_VELOCITY\_COMMAND\_NAME

```
constexpr char wisco::control::motion::PIDTurn::DRIVE_SET_VELOCITY_COMMAND_NAME[ ] {"SET VELOCITY"}  
[static], [constexpr], [private]
```

The name of the drive set velocity command.

Definition at line 73 of file [PIDTurn.hpp](#).

```
00073 {"SET VELOCITY"};
```

### 5.31.3.5 DRIVE\_GET\_RADIUS\_STATE\_NAME

```
constexpr char wisco::control::motion::PIDTurn::DRIVE_GET_RADIUS_STATE_NAME[ ] {"GET RADIUS"}  
[static], [constexpr], [private]
```

The name of the drive get radius command.

Definition at line 79 of file [PIDTurn.hpp](#).

```
00079 {"GET RADIUS"};
```

### 5.31.3.6 ODOMETRY\_GET\_POSITION\_STATE\_NAME

```
constexpr char wisco::control::motion::PIDTurn::ODOMETRY_GET_POSITION_STATE_NAME[ ] {"GET POSITION"}  
[static], [constexpr], [private]
```

The name of the odometry get position command.

Definition at line 85 of file [PIDTurn.hpp](#).

```
00085 {"GET POSITION"};
```

### 5.31.3.7 TURN\_TO\_ANGLE\_DISTANCE

```
constexpr double wisco::control::motion::PIDTurn::TURN_TO_ANGLE_DISTANCE {120000} [static],  
[constexpr], [private]
```

The distance to calculate a point at for turn to angle.

Definition at line 91 of file [PIDTurn.hpp](#).

```
00091 {120000};
```

### 5.31.3.8 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::control::motion::PIDTurn::m_delayer {} [private]
```

The rtos delayer.

Definition at line 104 of file [PIDTurn.hpp](#).

```
00104 {};
```

### 5.31.3.9 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::control::motion::PIDTurn::m_mutex {} [private]
```

The rtos mutex.

Definition at line 110 of file [PIDTurn.hpp](#).

```
00110 {};
```

### 5.31.3.10 m\_task

```
std::unique_ptr<rtos::ITask> wisco::control::motion::PIDTurn::m_task {} [private]
```

The rtos task.

Definition at line 116 of file [PIDTurn.hpp](#).

```
00116 {};
```

### 5.31.3.11 m\_pid

```
PID wisco::control::motion::PIDTurn::m_pid {} [private]
```

The PID controller.

Definition at line 122 of file [PIDTurn.hpp](#).

```
00122 {};
```

### 5.31.3.12 m\_target\_tolerance

```
double wisco::control::motion::PIDTurn::m_target_tolerance {} [private]
```

The acceptable tolerance to reach the target.

Definition at line 128 of file [PIDTurn.hpp](#).

```
00128 {};
```

### 5.31.3.13 m\_target\_velocity

```
double wisco::control::motion::PIDTurn::m_target_velocity {} [private]
```

The acceptable velocity to reach the target.

Definition at line 134 of file [PIDTurn.hpp](#).

```
00134 {};
```

### 5.31.3.14 control\_robot

```
std::shared_ptr<robot::Robot> wisco::control::motion::PIDTurn::control_robot {} [private]
```

The robot being controlled.

Definition at line 140 of file [PIDTurn.hpp](#).

```
00140 {};
```

### 5.31.3.15 turn\_direction

```
ETurnDirection wisco::control::motion::PIDTurn::turn_direction {} [private]
```

The turn direction.

Definition at line 146 of file [PIDTurn.hpp](#).

```
00146 {};
```

### 5.31.3.16 turn\_velocity

```
double wisco::control::motion::PIDTurn::turn_velocity {} [private]
```

The turn velocity in radians per second.

Definition at line 152 of file [PIDTurn.hpp](#).

```
00152 {};
```

### 5.31.3.17 target\_x

```
double wisco::control::motion::PIDTurn::target_x {} [private]
```

The target x-coordinate.

Definition at line 158 of file [PIDTurn.hpp](#).

```
00158 {};
```

### 5.31.3.18 target\_y

```
double wisco::control::motion::PIDTurn::target_y {} [private]
```

The target y-coordinate.

Definition at line 164 of file [PIDTurn.hpp](#).

```
00164 {};
```

### 5.31.3.19 paused

```
bool wisco::control::motion::PIDTurn::paused {} [private]
```

Whether or not the motion has been paused.

Definition at line 170 of file [PIDTurn.hpp](#).

```
00170 {};
```

### 5.31.3.20 target\_reached

```
bool wisco::control::motion::PIDTurn::target_reached {} [private]
```

Whether or not the target point has been reached.

Definition at line 176 of file [PIDTurn.hpp](#).  
00176 {};

### 5.31.3.21 forced\_direction\_reached

```
bool wisco::control::motion::PIDTurn::forced_direction_reached {} [private]
```

Whether or not the forced direction has been reached.

Definition at line 182 of file [PIDTurn.hpp](#).  
00182 {};

## 5.32 wisco::control::motion::PIDTurnBuilder Class Reference

Builder class for [PID](#) turn controllers.

### Public Member Functions

- [PIDTurnBuilder \\* withDelayer](#) (const std::unique\_ptr<[rtos::IDelay](#)> &delayer)  
*Adds an rtos delayer to the build.*
- [PIDTurnBuilder \\* withMutex](#) (std::unique\_ptr<[rtos::IMutex](#)> &mutex)  
*Adds an rtos mutex to the build.*
- [PIDTurnBuilder \\* withTask](#) (std::unique\_ptr<[rtos::ITask](#)> &task)  
*Adds an rtos task to the build.*
- [PIDTurnBuilder \\* withPID](#) (PID pid)  
*Adds a PID controller to the build.*
- [PIDTurnBuilder \\* withTargetTolerance](#) (double target\_tolerance)  
*Adds a target tolerance to the build.*
- [PIDTurnBuilder \\* withTargetVelocity](#) (double target\_velocity)  
*Adds a target velocity to the build.*
- std::unique\_ptr<[ITurn](#)> [build](#) ()  
*Builds the PID turn controller.*

### Private Attributes

- std::unique\_ptr<[rtos::IDelay](#)> [m\\_delayer](#) {}  
*The rtos delayer.*
- std::unique\_ptr<[rtos::IMutex](#)> [m\\_mutex](#) {}  
*The rtos mutex.*
- std::unique\_ptr<[rtos::ITask](#)> [m\\_task](#) {}  
*The rtos task.*
- PID [m\\_pid](#) {}  
*The PID controller.*
- double [m\\_target\\_tolerance](#) {}  
*The acceptable tolerance to reach the target.*
- double [m\\_target\\_velocity](#) {}  
*The acceptable velocity to reach the target.*

### 5.32.1 Detailed Description

Builder class for PID turn controllers.

#### Author

Nathan Sandvig

Definition at line 35 of file [PIDTurnBuilder.hpp](#).

### 5.32.2 Member Function Documentation

#### 5.32.2.1 withDelayer()

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Adds an rtos delayer to the build.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

#### Returns

PIDTurnBuilder\* This object for build chaining

Definition at line 9 of file [PIDTurnBuilder.cpp](#).

```
00010 {
00011     m_delayer = delayer->clone();
00012     return this;
00013 }
```

#### 5.32.2.2 withMutex()

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Adds an rtos mutex to the build.

#### Parameters

<i>mutex</i>	The rtos mutex
--------------	----------------

#### Returns

PIDTurnBuilder\* This object for build chaining

Definition at line 15 of file [PIDTurnBuilder.cpp](#).

```
00016 {
00017     m_mutex = std::move(mutex);
00018     return this;
00019 }
```

### 5.32.2.3 withTask()

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withTask (
    std::unique_ptr< rtos::ITask > & task )
```

Adds an rtos task to the build.

#### Parameters

<i>task</i>	The rtos task
-------------	---------------

#### Returns

PIDTurnBuilder\* This object for build chaining

Definition at line 21 of file [PIDTurnBuilder.cpp](#).

```
00022 {
00023     m_task = std::move(task);
00024     return this;
00025 }
```

### 5.32.2.4 withPID()

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withPID (
    PID pid )
```

Adds a [PID](#) controller to the build.

#### Parameters

<i>pid</i>	The <a href="#">PID</a> controller
------------	------------------------------------

#### Returns

PIDTurnBuilder\* This object for build chaining

Definition at line 27 of file [PIDTurnBuilder.cpp](#).

```
00028 {
00029     m_pid = pid;
00030     return this;
00031 }
```

### 5.32.2.5 withTargetTolerance()

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withTargetTolerance (
    double target_tolerance )
```

Adds a target tolerance to the build.

#### Parameters

<i>target_tolerance</i>	The motion target tolerance
-------------------------	-----------------------------

**Returns**

PIDTurnBuilder\* This object for build chaining

Definition at line 33 of file [PIDTurnBuilder.cpp](#).

```
00034 {
00035     m_target_tolerance = target_tolerance;
00036     return this;
00037 }
```

**5.32.2.6 withTargetVelocity()**

```
PIDTurnBuilder * wisco::control::motion::PIDTurnBuilder::withTargetVelocity (
    double target_velocity )
```

Adds a target velocity to the build.

**Parameters**

<i>target_velocity</i>	The motion target velocity
------------------------	----------------------------

**Returns**

PIDTurnBuilder\* This object for build chaining

Definition at line 39 of file [PIDTurnBuilder.cpp](#).

```
00040 {
00041     m_target_velocity = target_velocity;
00042     return this;
00043 }
```

**5.32.2.7 build()**

```
std::unique_ptr< ITurn > wisco::control::motion::PIDTurnBuilder::build ( )
```

Builds the **PID** turn controller.

**Returns**

std::unique\_ptr<ITurn> The **PID** turn controller as a turn interface object

Definition at line 45 of file [PIDTurnBuilder.cpp](#).

```
00046 {
00047     std::unique_ptr<PIDTurn> pid_turn{std::make_unique<PIDTurn>()};
00048     pid_turn->setDelayer(m_delayer);
00049     pid_turn->setMutex(m_mutex);
00050     pid_turn->setTask(m_task);
00051     pid_turn->setPID(m_pid);
00052     pid_turn->setTargetTolerance(m_target_tolerance);
00053     pid_turn->setTargetVelocity(m_target_velocity);
00054     return pid_turn;
00055 }
```

### 5.32.3 Member Data Documentation

#### 5.32.3.1 m\_delayer

```
std::unique_ptr<rtos::IDelay> wisco::control::motion::PIDTurnBuilder::m_delayer {} [private]
```

The rtos delayer.

Definition at line 41 of file [PIDTurnBuilder.hpp](#).  
00041 {};

#### 5.32.3.2 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::control::motion::PIDTurnBuilder::m_mutex {} [private]
```

The rtos mutex.

Definition at line 47 of file [PIDTurnBuilder.hpp](#).  
00047 {};

#### 5.32.3.3 m\_task

```
std::unique_ptr<rtos::ITask> wisco::control::motion::PIDTurnBuilder::m_task {} [private]
```

The rtos task.

Definition at line 53 of file [PIDTurnBuilder.hpp](#).  
00053 {};

#### 5.32.3.4 m\_pid

```
PID wisco::control::motion::PIDTurnBuilder::m_pid {} [private]
```

The PID controller.

Definition at line 59 of file [PIDTurnBuilder.hpp](#).  
00059 {};

#### 5.32.3.5 m\_target\_tolerance

```
double wisco::control::motion::PIDTurnBuilder::m_target_tolerance {} [private]
```

The acceptable tolerance to reach the target.

Definition at line 65 of file [PIDTurnBuilder.hpp](#).  
00065 {};

### 5.32.3.6 m\_target\_velocity

```
double wisco::control::motion::PIDTurnBuilder::m_target_velocity {} [private]
```

The acceptable velocity to reach the target.

Definition at line 71 of file [PIDTurnBuilder.hpp](#).  
00071 {};

## 5.33 wisco::control::path::Point Class Reference

Class for a point with an x and y coordinate.

### Public Member Functions

- **Point ()=default**  
*Construct a new Point object.*
- **Point (const Point &copy)=default**  
*Construct a new Point object.*
- **Point (Point &&move)=default**  
*Construct a new Point object.*
- **Point (double x, double y)**  
*Construct a new Point object.*
- **~Point ()=default**  
*Destroy the Point object.*
- **void setX (double x)**  
*Sets the x-coordinate of the point.*
- **void setY (double y)**  
*Sets the y-coordinate of the point.*
- **double getX () const**  
*Gets the x-coordinate of the point.*
- **double getY () const**  
*Gets the y-coordinate of the point.*
- **Point operator+ (const Point &rhs)**  
*Addition operator overload for point.*
- **Point operator- (const Point &rhs)**  
*Subtraction operator overload for point.*
- **Point operator\* (double rhs)**  
*Multiplication operator overload for point.*
- **Point operator/ (double rhs)**  
*Division operator overload for point.*
- **Point & operator+= (const Point &rhs)**  
*Addition assignment operator overload for point.*
- **Point & operator-= (const Point &rhs)**  
*Subtraction assignment operator overload for point.*
- **Point & operator\*= (double rhs)**  
*Multiplication assignment operator overload for point.*
- **Point & operator/= (double rhs)**  
*Division assignment operator overload for point.*
- **Point & operator= (const Point &copy)=default**  
*Copy assignment operator for point.*
- **Point & operator= (Point &&move)=default**  
*Move assignment operator for point.*

### Private Attributes

- double `m_x` {}  
*The x-coordinate of the point.*
- double `m_y` {}  
*The y-coordinate of the point.*

### 5.33.1 Detailed Description

Class for a point with an x and y coordinate.

#### Author

Nathan Sandvig

Definition at line 33 of file [Point.hpp](#).

### 5.33.2 Constructor & Destructor Documentation

#### 5.33.2.1 Point() [1/3]

```
wisco::control::path::Point::Point (
    const Point & copy ) [default]
```

Construct a new [Point](#) object.

##### Parameters

<code>copy</code>	The point object being copied
-------------------	-------------------------------

#### 5.33.2.2 Point() [2/3]

```
wisco::control::path::Point::Point (
    Point && move ) [default]
```

Construct a new [Point](#) object.

##### Parameters

<code>move</code>	The point object being moved
-------------------	------------------------------

#### 5.33.2.3 Point() [3/3]

```
wisco::control::path::Point::Point (
    double x,
    double y )
```

Construct a new [Point](#) object.

#### Parameters

<code>x</code>	The x-coordinate of the point
<code>y</code>	The y-coordinate of the point

Definition at line 9 of file [Point.cpp](#).

```
00009 : m_x{x}, m_y{y}
00010 {
00011
00012 }
```

### 5.33.3 Member Function Documentation

#### 5.33.3.1 setX()

```
void wisco::control::path::Point::setX (
    double x )
```

Sets the x-coordinate of the point.

#### Parameters

<code>x</code>	The x-coordinate of the point
----------------	-------------------------------

Definition at line 14 of file [Point.cpp](#).

```
00015 {
00016     m_x = x;
00017 }
```

#### 5.33.3.2 setY()

```
void wisco::control::path::Point::setY (
    double y )
```

Sets the y-coordinate of the point.

#### Parameters

<code>y</code>	The y-coordinate of the point
----------------	-------------------------------

Definition at line 19 of file [Point.cpp](#).

```
00020 {
00021     m_y = y;
00022 }
```

#### 5.33.3.3 getX()

```
double wisco::control::path::Point::getX ( ) const
```

Gets the x-coordinate of the point.

**Returns**

double The x-coordinate of the point

Definition at line 24 of file [Point.cpp](#).

```
00025 {
00026     return m_x;
00027 }
```

**5.33.3.4 getY()**

```
double wisco::control::path::Point::getY ( ) const
```

Gets the y-coordinate of the point.

**Returns**

double The y-coordinate of the point

Definition at line 29 of file [Point.cpp](#).

```
00030 {
00031     return m_y;
00032 }
```

**5.33.3.5 operator+()**

```
Point wisco::control::path::Point::operator+ (
    const Point & rhs )
```

Addition operator overload for point.

**Parameters**

<i>rhs</i>	The point on the right hand side of the operator
------------	--

**Returns**

[Point](#) A new point with the sum of the coordinates of both points

Definition at line 34 of file [Point.cpp](#).

```
00035 {
00036     return Point{m_x + rhs.m_x, m_y + rhs.m_y};
00037 }
```

**5.33.3.6 operator-()**

```
Point wisco::control::path::Point::operator- (
    const Point & rhs )
```

Subtraction operator overload for point.

**Parameters**

<i>rhs</i>	The point on the right hand side of the operator
------------	--

**Returns**

[Point](#) A new point with the difference of the coordinates of both points

Definition at line 39 of file [Point.cpp](#).

```
00040 {
00041     return Point{m_x - rhs.m_x, m_y - rhs.m_y};
00042 }
```

**5.33.3.7 operator\*()**

```
Point wisco::control::path::Point::operator* (
    double rhs )
```

Multiplication operator overload for point.

**Parameters**

<i>rhs</i>	The multiplier on the right hand side of the operator
------------	---

**Returns**

[Point](#) A new point with the coordinates of this point multiplied by the multiplier

Definition at line 44 of file [Point.cpp](#).

```
00045 {
00046     return Point{m_x * rhs, m_y * rhs};
00047 }
```

**5.33.3.8 operator/()**

```
Point wisco::control::path::Point::operator/ (
    double rhs )
```

Division operator overload for point.

**Parameters**

<i>rhs</i>	The divisor on the right hand side of the operator
------------	--

**Returns**

[Point](#) A new point with the coordinates of this point divided by the divisor

Definition at line 49 of file [Point.cpp](#).

```
00050 {
00051     return Point{m_x / rhs, m_y / rhs};
00052 }
```

### 5.33.3.9 operator+=()

```
Point & wisco::control::path::Point::operator+= (
    const Point & rhs )
```

Addition assignment operator overload for point.

#### Parameters

<i>rhs</i>	The point on the right hand side of the operator
------------	--

#### Returns

`Point&` This point with the sum of the coordinates of both points

Definition at line 54 of file [Point.cpp](#).

```
00055 {
00056     m_x += rhs.m_x;
00057     m_y += rhs.m_y;
00058     return *this;
00059 }
```

### 5.33.3.10 operator-=(\*)

```
Point & wisco::control::path::Point::operator-= (
    const Point & rhs )
```

Subtraction assignment operator overload for point.

#### Parameters

<i>rhs</i>	The point on the right hand side of the operator
------------	--

#### Returns

`Point&` This point with the difference of the coordinates of both points

Definition at line 61 of file [Point.cpp](#).

```
00062 {
00063     m_x -= rhs.m_x;
00064     m_y -= rhs.m_y;
00065     return *this;
00066 }
```

### 5.33.3.11 operator\*=(\*)

```
Point & wisco::control::path::Point::operator*=(
    double rhs )
```

Multiplication assignment operator overload for point.

**Parameters**

<i>rhs</i>	The multiplier on the right hand side of the operator
------------	---

**Returns**

`Point&` This point with the coordinates of this point multiplied by the multiplier

Definition at line 68 of file `Point.cpp`.

```
00069 {
00070     m_x *= rhs;
00071     m_y *= rhs;
00072     return *this;
00073 }
```

**5.33.3.12 operator/()**

```
Point & wisco::control::path::Point::operator/= (
    double rhs )
```

Division assignment operator overload for point.

**Parameters**

<i>rhs</i>	The divisor on the right hand side of the operator
------------	--

**Returns**

`Point&` This point with the coordinates of this point divided by the divisor

Definition at line 75 of file `Point.cpp`.

```
00076 {
00077     m_x /= rhs;
00078     m_y /= rhs;
00079     return *this;
00080 }
```

**5.33.3.13 operator=() [1/2]**

```
Point & wisco::control::path::Point::operator= (
    const Point & copy ) [default]
```

Copy assignment operator for point.

**Parameters**

<i>copy</i>	The point on the right hand side of the operator
-------------	--

**Returns**

`Point&` This point with the coordinates of the other point

### 5.33.3.14 operator=( ) [2/2]

```
Point & wisco::control::path::Point::operator= (
    Point && move ) [default]
```

Move assignment operator for point.

#### Parameters

<code>move</code>	The point value on the right hand side of the operator
-------------------	--

#### Returns

`Point&` This point with the coordinates of the point value

## 5.33.4 Member Data Documentation

### 5.33.4.1 m\_x

```
double wisco::control::path::Point::m_x {} [private]
```

The x-coordinate of the point.

Definition at line 40 of file [Point.hpp](#).  
00040 {};

### 5.33.4.2 m\_y

```
double wisco::control::path::Point::m_y {} [private]
```

The y-coordinate of the point.

Definition at line 46 of file [Point.hpp](#).  
00046 {};

## 5.34 wisco::control::path::QuinticBezier Class Reference

A quintic bezier with a point function.

#### Public Member Functions

- **QuinticBezier ()=default**  
*Construct a new Quintic Bezier object.*
- **QuinticBezier (const QuinticBezier &copy)=default**  
*Construct a new Quintic Bezier object.*
- **QuinticBezier (QuinticBezier &&move)=default**  
*Construct a new Quintic Bezier object.*
- **QuinticBezier (Point c1, Point c2, Point c3, Point c4, Point c5, Point c6)**  
*Construct a new Quintic Bezier object.*
- **Point calculatePoint (double t)**  
*Gets the point at f(t)*
- **QuinticBezier & operator= (const QuinticBezier &rhs)=default**  
*Copy assignment operator for QuinticBezier.*
- **QuinticBezier & operator= (QuinticBezier &&rhs)=default**  
*Move assignment operator for QuinticBezier.*

## Public Attributes

- **Point k0 {}**  
*The first control point.*
- **Point c0 {}**  
*The second control point.*
- **Point c1 {}**  
*The third control point.*
- **Point c2 {}**  
*The fourth control point.*
- **Point c3 {}**  
*The fifth control point.*
- **Point k1 {}**  
*The sixth control point.*

### 5.34.1 Detailed Description

A quintic bezier with a point function.

#### Author

Nathan Sandvig

Definition at line 37 of file [QuinticBezier.hpp](#).

### 5.34.2 Constructor & Destructor Documentation

#### 5.34.2.1 QuinticBezier() [1/3]

```
wisco::control::path::QuinticBezier::QuinticBezier (
    const QuinticBezier & copy ) [default]
```

Construct a new Quintic Bezier object.

#### Parameters

<i>copy</i>	The <a href="#">QuinticBezier</a> object being copied
-------------	---

#### 5.34.2.2 QuinticBezier() [2/3]

```
wisco::control::path::QuinticBezier::QuinticBezier (
    QuinticBezier && move ) [default]
```

Construct a new Quintic Bezier object.

#### Parameters

<i>move</i>	The <a href="#">QuinticBezier</a> object being moved
-------------	--

### 5.34.2.3 QuinticBezier() [3/3]

```
wisco::control::path::QuinticBezier::QuinticBezier (
    Point c1,
    Point c2,
    Point c3,
    Point c4,
    Point c5,
    Point c6 )
```

Construct a new Quintic Bezier object.

#### Parameters

<i>c1</i>	The first control point
<i>c2</i>	The second control point
<i>c3</i>	The third control point
<i>c4</i>	The fourth control point
<i>c5</i>	The fifth control point
<i>c6</i>	The sixth control point

Definition at line 9 of file [QuinticBezier.cpp](#).

```
00010     : k0{c1}, c0{c2}, c1{c3}, c2{c4}, c3{c5}, k1{c6}
00011 {
00012
00013 }
```

## 5.34.3 Member Function Documentation

### 5.34.3.1 calculatePoint()

```
Point wisco::control::path::QuinticBezier::calculatePoint (
    double t )
```

Gets the point at f(t)

#### Parameters

<i>t</i>	The input value
----------	-----------------

#### Returns

**Point** The point at that input value

Definition at line 15 of file [QuinticBezier.cpp](#).

```
00016 {
00017     return std::pow(1 - t, 5) * k0
00018         + 5 * std::pow(1 - t, 4) * t * c0
00019         + 10 * std::pow(1 - t, 3) * std::pow(t, 2) * c1
00020         + 10 * std::pow(1 - t, 2) * std::pow(t, 3) * c2
00021         + 5 * (1 - t) * std::pow(t, 4) * c3
00022         + std::pow(t, 5) * k1;
00023 }
```

### 5.34.3.2 operator=() [1/2]

```
QuinticBezier & wisco::control::path::QuinticBezier::operator= (
    const QuinticBezier & rhs ) [default]
```

Copy assignment operator for [QuinticBezier](#).

#### Parameters

<i>rhs</i>	The <a href="#">QuinticBezier</a> on the right hand side of the operator
------------	--

#### Returns

[QuinticBezier](#)& This [QuinticBezier](#) with the control points of the other [QuinticBezier](#)

### 5.34.3.3 operator=() [2/2]

```
QuinticBezier & wisco::control::path::QuinticBezier::operator= (
    QuinticBezier && rhs ) [default]
```

Move assignment operator for [QuinticBezier](#).

#### Parameters

<i>rhs</i>	The <a href="#">QuinticBezier</a> value on the right hand side of the operator
------------	--

#### Returns

[QuinticBezier](#)& This [QuinticBezier](#) with the control points of the [QuinticBezier](#) value

## 5.34.4 Member Data Documentation

### 5.34.4.1 k0

```
Point wisco::control::path::QuinticBezier::k0 {}
```

The first control point.

Definition at line 44 of file [QuinticBezier.hpp](#).  
00044 {};

### 5.34.4.2 c0

```
Point wisco::control::path::QuinticBezier::c0 {}
```

The second control point.

Definition at line 50 of file [QuinticBezier.hpp](#).  
00050 {};

#### 5.34.4.3 c1

```
Point wisco::control::path::QuinticBezier::c1 {}
```

The third control point.

Definition at line 56 of file [QuinticBezier.hpp](#).

```
00056 {};
```

#### 5.34.4.4 c2

```
Point wisco::control::path::QuinticBezier::c2 {}
```

The fourth control point.

Definition at line 62 of file [QuinticBezier.hpp](#).

```
00062 {};
```

#### 5.34.4.5 c3

```
Point wisco::control::path::QuinticBezier::c3 {}
```

The fifth control point.

Definition at line 68 of file [QuinticBezier.hpp](#).

```
00068 {};
```

#### 5.34.4.6 k1

```
Point wisco::control::path::QuinticBezier::k1 {}
```

The sixth control point.

Definition at line 74 of file [QuinticBezier.hpp](#).

```
00074 {};
```

### 5.35 wisco::control::path::QuinticBezierSpline Class Reference

A spline of quintic beziers.

#### Static Public Member Functions

- static std::vector< [Point](#) > [calculate](#) (std::vector< [Point](#) > &control\_points)

*Calculates a smooth spline using quintic beziers.*

### 5.35.1 Detailed Description

A spline of quintic beziers.

#### Author

Nathan Sandvig

Definition at line 37 of file [QuinticBezierSpline.hpp](#).

### 5.35.2 Member Function Documentation

#### 5.35.2.1 calculate()

```
std::vector< Point > wisco::control::path::QuinticBezierSpline::calculate (
    std::vector< Point > & control_points ) [static]
```

Calculates a smooth spline using quintic beziers.

#### Parameters

<i>control_points</i>	The control points for the spline
-----------------------	-----------------------------------

#### Returns

`std::vector<Point>` The points in the spline

Definition at line 9 of file [QuinticBezierSpline.cpp](#).

```
00010 {
00011     if (control_points.size() == 0 || (control_points.size() - 1) % 3 != 0)
00012         return std::vector<Point>{};
00013
00014     // Create the beziers
00015     std::vector<QuinticBezier> beziers{};
00016     for (int i{}; i < control_points.size() - 1; i += 3)
00017     {
00018         beziers.push_back(QuinticBezier{control_points[i], Point{}, control_points[i + 1],
00019             control_points[i + 2], Point{}, control_points[i + 3]} );
00020     }
00021     // Calculate the smoothing points
00022     beziers[0].c0 = (beziers[0].k0 + beziers[0].c1) / 2.0;
00023
00024     for (int i{1}; i < beziers.size(); ++i)
00025         beziers[i].c0 = (4.0 * beziers[i].k0 - beziers[i - 1].c2 + beziers[i].c1) / 4.0;
00026
00027     for (int i{}; i < beziers.size() - 1; ++i)
00028         beziers[i].c3 = 2.0 * beziers[i].k1 - beziers[i + 1].c0;
00029
00030     beziers[beziers.size() - 1].c3 = (beziers[beziers.size() - 1].k1 + beziers[beziers.size() - 1].c2)
00031     / 2.0;
00032
00033     // Calculate the spline
00034     std::vector<Point> spline{};
00035     for (auto& bezier : beziers)
00036     {
00037         for (double t{0.0}; t < 1.0; t += 0.02)
00038         {
00039             spline.push_back(bezier.calculatePoint(t));
00040         }
00041     }
00042 }
```

## 5.36 wisco::control::PID Class Reference

A general-purpose PID controller.

### Public Member Functions

- **PID ()=default**  
*Construct a new PID object.*
- **PID (const std::unique\_ptr< rtos::IClock > &clock, double kp, double ki, double kd)**  
*Construct a new PID object.*
- **PID (const PID &copy)**  
*Construct a new PID object.*
- **PID (PID &&move)=default**  
*Construct a new PID object.*
- **double getControlValue (double current, double target)**  
*Get the control value output of the PID controller.*
- **void reset ()**  
*Resets the PID controller.*
- **PID & operator= (const PID &rhs)**  
*Copy assignment operator.*
- **PID & operator= (PID &&rhs)=default**  
*Move assignment operator.*

### Private Attributes

- **std::unique\_ptr< rtos::IClock > m\_clock {}**  
*The system clock.*
- **double m\_kp {}**  
*The proportional constant.*
- **double m\_ki {}**  
*The integral constant.*
- **double m\_kd {}**  
*The derivative constant.*
- **double accumulated\_error {}**  
*The accumulated error.*
- **double last\_error {}**  
*The error during the last timestep.*
- **double last\_time {}**  
*The system clock time during the last timestep.*

### 5.36.1 Detailed Description

A general-purpose PID controller.

#### Author

Nathan Sandvig

Definition at line 29 of file [PID.hpp](#).

## 5.36.2 Constructor & Destructor Documentation

### 5.36.2.1 PID() [1/3]

```
wisco::control::PID::PID (
    const std::unique_ptr< rtos::IClock > & clock,
    double kp,
    double ki,
    double kd )
```

Construct a new [PID](#) object.

#### Parameters

<i>clock</i>	The system clock
<i>kp</i>	The proportional constant
<i>ki</i>	The integral constant
<i>kd</i>	The derivative constant

Definition at line 7 of file [PID.cpp](#).

```
00008     : m_clock{clock->clone()}, m_kp{kp}, m_ki{ki}, m_kd{kd}
00009 {
00010
00011 }
```

### 5.36.2.2 PID() [2/3]

```
wisco::control::PID::PID (
    const PID & copy )
```

Construct a new [PID](#) object.

#### Parameters

<i>copy</i>	The <a href="#">PID</a> object being copied
-------------	---

Definition at line 13 of file [PID.cpp](#).

```
00014     : m_clock{copy.m_clock->clone()},
00015     m_kp{copy.m_kp},
00016     m_ki{copy.m_ki},
00017     m_kd{copy.m_kd},
00018     accumulated_error{copy.accumulated_error},
00019     last_error{copy.last_error},
00020     last_time{copy.last_time}
00021 {
00022
00023 }
```

### 5.36.2.3 PID() [3/3]

```
wisco::control::PID::PID (
    PID && move ) [default]
```

Construct a new [PID](#) object.

**Parameters**

<i>move</i>	The PID object being moved
-------------	----------------------------

**5.36.3 Member Function Documentation****5.36.3.1 getControlValue()**

```
double wisco::control::PID::getControlValue (
    double current,
    double target )
```

Get the control value output of the PID controller.

**Parameters**

<i>current</i>	The current system value
<i>target</i>	The target system value

**Returns**

double The output system value

Definition at line 25 of file [PID.cpp](#).

```
00026 {
00027     double time_change{};
00028     if (m_clock)
00029     {
00030         double current_time = m_clock->getTime();
00031         time_change = current_time - last_time;
00032         last_time = current_time;
00033     }
00034     double error{target - current};
00035     accumulated_error += (error * time_change);
00036     double error_change{(error - last_error) / time_change};
00037     last_error = error;
00038
00039     return (m_kp * error) + (m_ki * accumulated_error) + (m_kd * error_change);
00040 }
00041 }
```

**5.36.3.2 reset()**

```
void wisco::control::PID::reset ( )
```

Resets the PID controller.

Definition at line 43 of file [PID.cpp](#).

```
00044 {
00045     if (m_clock)
00046         last_time = m_clock->getTime();
00047     accumulated_error = 0;
00048     last_error = 0;
00049 }
```

**5.36.3.3 operator=() [1/2]**

```
PID & wisco::control::PID::operator= (
    const PID & rhs )
```

Copy assignment operator.

**Parameters**

<i>rhs</i>	The <a href="#">PID</a> object being copied
------------	---

**Returns**

[PID&](#) This [PID](#) object with the assigned values

Definition at line 51 of file [PID.cpp](#).

```
00052 {
00053     m_clock = rhs.m_clock->clone();
00054     m_kp = rhs.m_kp;
00055     m_ki = rhs.m_ki;
00056     m_kd = rhs.m_kd;
00057     accumulated_error = rhs.accumulated_error;
00058     last_error = rhs.last_error;
00059     last_time = rhs.last_time;
00060     return *this;
00061 }
```

**5.36.3.4 operator=() [2/2]**

```
PID & wisco::control::PID::operator= (
    PID && rhs ) [default]
```

Move assignment operator.

**Parameters**

<i>rhs</i>	The <a href="#">PID</a> object being moved
------------	--

**Returns**

[PID&](#) This [PID](#) object with the assigned values

**5.36.4 Member Data Documentation****5.36.4.1 m\_clock**

```
std::unique_ptr<rtos::IClock> wisco::control::PID::m_clock {} [private]
```

The system clock.

Definition at line 36 of file [PID.hpp](#).

```
00036 {};
```

**5.36.4.2 m\_kp**

```
double wisco::control::PID::m_kp {} [private]
```

The proportional constant.

Definition at line 42 of file [PID.hpp](#).

```
00042 {};
```

#### 5.36.4.3 m\_ki

```
double wisco::control::PID::m_ki {} [private]
```

The integral constant.

Definition at line 48 of file [PID.hpp](#).

```
00048 {};
```

#### 5.36.4.4 m\_kd

```
double wisco::control::PID::m_kd {} [private]
```

The derivative constant.

Definition at line 54 of file [PID.hpp](#).

```
00054 {};
```

#### 5.36.4.5 accumulated\_error

```
double wisco::control::PID::accumulated_error {} [private]
```

The accumulated error.

Definition at line 60 of file [PID.hpp](#).

```
00060 {};
```

#### 5.36.4.6 last\_error

```
double wisco::control::PID::last_error {} [private]
```

The error during the last timestep.

Definition at line 66 of file [PID.hpp](#).

```
00066 {};
```

#### 5.36.4.7 last\_time

```
double wisco::control::PID::last_time {} [private]
```

The system clock time during the last timestep.

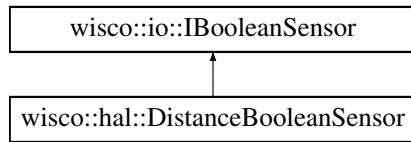
Definition at line 72 of file [PID.hpp](#).

```
00072 {};
```

## 5.37 wisco::hal::DistanceBooleanSensor Class Reference

A distance sensor used to create boolean outputs.

Inheritance diagram for wisco::hal::DistanceBooleanSensor:



### Public Member Functions

- `DistanceBooleanSensor (std::unique_ptr< io::IDistanceSensor > &distance_sensor, DistanceBooleanMode mode, double lower_threshold)`  
*Construct a new Distance Boolean Sensor object.*
- `DistanceBooleanSensor (std::unique_ptr< io::IDistanceSensor > &distance_sensor, DistanceBooleanMode mode, double lower_threshold, double upper_threshold)`  
*Construct a new Distance Boolean Sensor object.*
- `virtual void initialize ()=0`  
*Initializes the sensor.*
- `virtual void reset ()=0`  
*Resets the sensor.*
- `virtual bool getValue ()=0`  
*Get the boolean value of the sensor.*

### Public Member Functions inherited from [wisco::io::IBooleanSensor](#)

- `virtual ~IBooleanSensor ()=default`  
*Destroy the `IBooleanSensor` object.*

### Private Attributes

- `std::unique_ptr< io::IDistanceSensor > m_distance_sensor {}`  
*The distance sensor.*
- `DistanceBooleanMode m_mode {}`  
*The mode of the sensor.*
- `double m_lower_threshold {}`  
*The lower threshold of the sensor.*
- `double m_upper_threshold {}`  
*The upper threshold of the sensor.*
- `bool value {}`  
*The current boolean value.*

### 5.37.1 Detailed Description

A distance sensor used to create boolean outputs.

#### Author

Nathan Sandvig

Definition at line 32 of file [DistanceBooleanSensor.hpp](#).

### 5.37.2 Constructor & Destructor Documentation

#### 5.37.2.1 DistanceBooleanSensor() [1/2]

```
wisco::hal::DistanceBooleanSensor::DistanceBooleanSensor (
    std::unique_ptr< io::IDistanceSensor > & distance_sensor,
    DistanceBooleanMode mode,
    double lower_threshold )
```

Construct a new Distance Boolean Sensor object.

#### Parameters

<i>distance_sensor</i>	The distance sensor
<i>mode</i>	The mode of the sensor
<i>lower_threshold</i>	The lower threshold of the sensor

Definition at line 7 of file [DistanceBooleanSensor.cpp](#).

```
00010     : m_distance_sensor{std::move(distance_sensor)}, m_mode{mode}, m_lower_threshold{lower_threshold},
00011     m_upper_threshold{lower_threshold}
00012 {
00013 }
```

#### 5.37.2.2 DistanceBooleanSensor() [2/2]

```
wisco::hal::DistanceBooleanSensor::DistanceBooleanSensor (
    std::unique_ptr< io::IDistanceSensor > & distance_sensor,
    DistanceBooleanMode mode,
    double lower_threshold,
    double upper_threshold )
```

Construct a new Distance Boolean Sensor object.

#### Parameters

<i>distance_sensor</i>	The distance sensor
<i>mode</i>	The mode of the sensor
<i>lower_threshold</i>	The lower threshold of the sensor
<i>upper_threshold</i>	The upper threshold of the sensor

Definition at line 15 of file [DistanceBooleanSensor.cpp](#).

```
00019     : m_distance_sensor{std::move(distance_sensor)}, m_mode{mode}, m_lower_threshold{lower_threshold},  
00020     m_upper_threshold{upper_threshold}  
00021 {  
00022 }
```

### 5.37.3 Member Function Documentation

#### 5.37.3.1 initialize()

```
void wisco::hal::DistanceBooleanSensor::initialize () [pure virtual]
```

Initializes the sensor.

Implements [wisco::io::IBooleanSensor](#).

Definition at line 24 of file [DistanceBooleanSensor.cpp](#).

```
00025 {  
00026     reset();  
00027 }
```

#### 5.37.3.2 reset()

```
void wisco::hal::DistanceBooleanSensor::reset () [pure virtual]
```

Resets the sensor.

Implements [wisco::io::IBooleanSensor](#).

Definition at line 29 of file [DistanceBooleanSensor.cpp](#).

```
00030 {  
00031     if (!m_distance_sensor)  
00032         return;  
00033  
00034     double distance{m_distance_sensor->getDistance ()};  
00035     switch (m_mode)  
00036     {  
00037         case DistanceBooleanMode::ABOVE_THRESHOLD:  
00038             value = distance > m_upper_threshold;  
00039             break;  
00040         case DistanceBooleanMode::BELOW_THRESHOLD:  
00041             value = distance < m_lower_threshold;  
00042             break;  
00043         case DistanceBooleanMode::BETWEEN_THRESHOLD:  
00044             value = (distance > m_lower_threshold &&  
00045                         distance < m_upper_threshold);  
00046             break;  
00047     }  
00048 }
```

#### 5.37.3.3 getValue()

```
bool wisco::hal::DistanceBooleanSensor::getValue () [pure virtual]
```

Get the boolean value of the sensor.

**Returns**

bool The value of the sensor

Implements [wisco::io::IBooleanSensor](#).

Definition at line 50 of file [DistanceBooleanSensor.cpp](#).

```
00051 {
00052     double distance{};
00053     if (m_distance_sensor)
00054         distance = m_distance_sensor->getDistance();
00055
00056     switch (m_mode)
00057     {
00058         case DistanceBooleanMode::ABOVE_THRESHOLD:
00059             if (value)
00060                 value = distance > m_lower_threshold;
00061             else
00062                 value = distance > m_upper_threshold;
00063             break;
00064         case DistanceBooleanMode::BELOW_THRESHOLD:
00065             if (value)
00066                 value = distance < m_upper_threshold;
00067             else
00068                 value = distance < m_lower_threshold;
00069             break;
00070         case DistanceBooleanMode::BETWEEN_THRESHOLD:
00071             value = (distance > m_lower_threshold &&
00072                     distance < m_upper_threshold);
00073             break;
00074     }
00075
00076     return value;
00077 }
```

## 5.37.4 Member Data Documentation

### 5.37.4.1 m\_distance\_sensor

```
std::unique_ptr<io::IDistanceSensor> wisco::hal::DistanceBooleanSensor::m_distance_sensor {}  
[private]
```

The distance sensor.

Definition at line 39 of file [DistanceBooleanSensor.hpp](#).

```
00039 {};
```

### 5.37.4.2 m\_mode

```
DistanceBooleanMode wisco::hal::DistanceBooleanSensor::m_mode {} [private]
```

The mode of the sensor.

Definition at line 45 of file [DistanceBooleanSensor.hpp](#).

```
00045 {};
```

### 5.37.4.3 m\_lower\_threshold

```
double wisco::hal::DistanceBooleanSensor::m_lower_threshold {} [private]
```

The lower threshold of the sensor.

Definition at line 51 of file [DistanceBooleanSensor.hpp](#).

```
00051 {};
```

#### 5.37.4.4 m\_upper\_threshold

```
double wisco::hal::DistanceBooleanSensor::m_upper_threshold {} [private]
```

The upper threshold of the sensor.

Definition at line 57 of file [DistanceBooleanSensor.hpp](#).

```
00057 {};
```

#### 5.37.4.5 value

```
bool wisco::hal::DistanceBooleanSensor::value {} [private]
```

The current boolean value.

Definition at line 63 of file [DistanceBooleanSensor.hpp](#).

```
00063 {};
```

## 5.38 wisco::hal::MotorGroup Class Reference

A group of motors on the same connected output SHOULD ONLY BE USED WITH IDENTICAL MOTORS.

### Public Member Functions

- void [addMotor](#) (std::unique\_ptr<io::IMotor> &motor)  
*Adds a motor to the motor group.*
- void [initialize](#) ()  
*Initializes the motors.*
- double [getTorqueConstant](#) ()  
*Get the torque constant of the motors.*
- double [getResistance](#) ()  
*Get the resistance of the motors.*
- double [getAngularVelocityConstant](#) ()  
*Get the angular velocity constant of the motors.*
- double [getGearRatio](#) ()  
*Get the gear ratio of the motors (1 if n/a)*
- double [getAngularVelocity](#) ()  
*Get the angular velocity of the motors in radians/second.*
- double [getPosition](#) ()  
*Get the average position of the motors in the group.*
- void [setVoltage](#) (double volts)  
*Set the voltage input to the motors in Volts.*
- MotorGroup & [operator=](#) (MotorGroup &rhs)  
*Override for the assignment operator for MotorGroup.*

### Private Attributes

- std::vector< std::unique\_ptr<io::IMotor> > [motors](#) {}  
*The motors in the group.*

### 5.38.1 Detailed Description

A group of motors on the same connected output SHOULD ONLY BE USED WITH IDENTICAL MOTORS.

#### Author

Nathan Sandvig

Definition at line 31 of file [MotorGroup.hpp](#).

### 5.38.2 Member Function Documentation

#### 5.38.2.1 addMotor()

```
void wisco::hal::MotorGroup::addMotor (
    std::unique_ptr< io::IMotor > & motor )
```

Adds a motor to the motor group.

##### Parameters

<i>motor</i>	The motor being added to the group
--------------	------------------------------------

Definition at line 7 of file [MotorGroup.cpp](#).

```
00008 {
00009     motors.push_back(std::move(motor));
00010 }
```

#### 5.38.2.2 initialize()

```
void wisco::hal::MotorGroup::initialize ( )
```

Initializes the motors.

Definition at line 12 of file [MotorGroup.cpp](#).

```
00013 {
00014     for (auto& motor : motors)
00015         if (motor)
00016             motor->initialize();
00017 }
```

#### 5.38.2.3 getTorqueConstant()

```
double wisco::hal::MotorGroup::getTorqueConstant ( )
```

Get the torque constant of the motors.

##### Returns

double The torque constant of the motors

Definition at line 19 of file [MotorGroup.cpp](#).

```
00020 {
00021     double sum_constant{};
00022     for (auto& motor : motors)
00023         if (motor)
00024             sum_constant += motor->getTorqueConstant();
00025
00026     return sum_constant;
00027 }
```

### 5.38.2.4 getResistance()

```
double wisco::hal::MotorGroup::getResistance ( )
```

Get the resistance of the motors.

#### Returns

double The resistance of the motors

Definition at line 29 of file [MotorGroup.cpp](#).

```
00030 {
00031     double average_resistance{};
00032     if (!motors.empty())
00033     {
00034         for (auto& motor : motors)
00035             if (motor)
00036                 average_resistance += motor->getResistance();
00037         average_resistance /= motors.size();
00038     }
00039
00040     return average_resistance;
00041 }
```

### 5.38.2.5 getAngularVelocityConstant()

```
double wisco::hal::MotorGroup::getAngularVelocityConstant ( )
```

Get the angular velocity constant of the motors.

#### Returns

double The angular velocity constant of the motors

Definition at line 43 of file [MotorGroup.cpp](#).

```
00044 {
00045     double average_constant{};
00046     if (!motors.empty())
00047     {
00048         for (auto& motor : motors)
00049             if (motor)
00050                 average_constant += motor->getAngularVelocityConstant();
00051         average_constant /= motors.size();
00052     }
00053
00054     return average_constant;
00055 }
```

### 5.38.2.6 getGearRatio()

```
double wisco::hal::MotorGroup::getGearRatio ( )
```

Get the gear ratio of the motors (1 if n/a)

#### Returns

double The gear ratio of the motors

Definition at line 57 of file [MotorGroup.cpp](#).

```
00058 {
00059     double gear_ratio{};
00060     if (!motors.empty() && motors.front())
00061         gear_ratio = motors.front()->getGearRatio();
00062     return gear_ratio;
00063 }
```

### 5.38.2.7 getAngularVelocity()

```
double wisco::hal::MotorGroup::getAngularVelocity ( )
```

Get the angular velocity of the motors in radians/second.

#### Returns

`double` The angular velocity of the motors in radians/second

Definition at line 65 of file [MotorGroup.cpp](#).

```
00066 {
00067     double average_velocity{};
00068     if (!motors.empty())
00069     {
00070         for (auto& motor : motors)
00071             if (motor)
00072                 average_velocity += motor->getAngularVelocity\(\);
00073         average_velocity /= motors.size();
00074     }
00075
00076     return average_velocity;
00077 }
```

### 5.38.2.8 getPosition()

```
double wisco::hal::MotorGroup::getPosition ( )
```

Get the average position of the motors in the group.

#### Returns

`double` The average position of the motors in the group

Definition at line 79 of file [MotorGroup.cpp](#).

```
00080 {
00081     double average_position{};
00082     if (!motors.empty())
00083     {
00084         for (auto& motor : motors)
00085             if (motor)
00086                 average_position += motor->getPosition\(\);
00087         average_position /= motors.size();
00088     }
00089
00090     return average_position;
00091 }
```

### 5.38.2.9 setVoltage()

```
void wisco::hal::MotorGroup::setVoltage (
    double volts )
```

Set the voltage input to the motors in Volts.

#### Parameters

<code>volts</code>	The voltage input in Volts
--------------------	----------------------------

Definition at line 93 of file [MotorGroup.cpp](#).

```
00094 {
00095     for (auto& motor : motors)
00096         if (motor)
00097             motor->setVoltage(volts);
00098 }
```

### 5.38.2.10 operator=([\)](#)

```
MotorGroup & wisco::hal::MotorGroup::operator= (
    MotorGroup & rhs )
```

Override for the assignment operator for [MotorGroup](#).

#### Parameters

<i>rhs</i>	The <a href="#">MotorGroup</a> object on the right hand side of the operator
------------	--

#### Returns

[MotorGroup](#)& This [MotorGroup](#) object with the assigned values

Definition at line 100 of file [MotorGroup.cpp](#).

```
00101 {
00102     motors.clear();
00103     for (uint8_t i{0}; i < rhs.motors.size(); ++i)
00104         motors.push_back(std::move(rhs.motors.at(i)));
00105     rhs.motors.clear();
00106     return *this;
00107 }
```

## 5.38.3 Member Data Documentation

### 5.38.3.1 motors

```
std::vector<std::unique_ptr<io::IMotor>> wisco::hal::MotorGroup::motors {} [private]
```

The motors in the group.

Definition at line 38 of file [MotorGroup.hpp](#).

```
00038 {};
```

## 5.39 wisco::hal::PistonGroup Class Reference

A group of pistons on the same connected output.

#### Public Member Functions

- void [addPiston](#) (std::unique\_ptr<[io::IPiston](#)> &piston)  
*Adds a piston to the piston group.*
- void [setState](#) (bool state)  
*Sets the state of the pistons.*
- bool [getState](#) ()  
*Gets the state of the pistons.*
- [PistonGroup](#) & [operator=](#) ([PistonGroup](#) &rhs)  
*Override for the assignment operator for [PistonGroup](#).*

## Private Attributes

- std::vector< std::unique\_ptr< io::IPiston > > **pistons** {}  
*The pistons in the group.*
- bool **m\_state** {}  
*The state of the piston group.*

### 5.39.1 Detailed Description

A group of pistons on the same connected output.

#### Author

Nathan Sandvig

Definition at line 30 of file [PistonGroup.hpp](#).

### 5.39.2 Member Function Documentation

#### 5.39.2.1 addPiston()

```
void wisco::hal::PistonGroup::addPiston (
    std::unique_ptr< io::IPiston > & piston )
```

Adds a piston to the piston group.

#### Parameters

<i>piston</i>	The piston being added to the group
---------------	-------------------------------------

Definition at line 7 of file [PistonGroup.cpp](#).

```
00008 {
00009     pistons.push_back(std::move(piston));
00010 }
```

#### 5.39.2.2 setState()

```
void wisco::hal::PistonGroup::setState (
    bool state )
```

Sets the state of the pistons.

#### Parameters

<i>state</i>	The state of the pistons
--------------	--------------------------

Definition at line 12 of file [PistonGroup.cpp](#).

```
00013 {
```

```

00014     if (m_state != state)
00015     {
00016         m_state = state;
00017         for (auto& piston : pistons)
00018             if (piston)
00019                 piston->toggle();
00020     }
00021 }
```

### 5.39.2.3 getState()

```
bool wisco::hal::PistonGroup::getState ( )
```

Gets the state of the pistons.

#### Returns

- true The pistons are active
- false The pistons are not active

Definition at line 23 of file [PistonGroup.cpp](#).

```

00024 {
00025     return m_state;
00026 }
```

### 5.39.2.4 operator=( )

```
PistonGroup & wisco::hal::PistonGroup::operator= (
    PistonGroup & rhs )
```

Override for the assignment operator for [PistonGroup](#).

#### Parameters

<code>rhs</code>	The <a href="#">PistonGroup</a> object on the right hand side of the operator
------------------	---

#### Returns

- [PistonGroup&](#) This [PistonGroup](#) object with the assigned values

Definition at line 28 of file [PistonGroup.cpp](#).

```

00029 {
00030     pistons.clear();
00031     for (uint8_t i{0}; i < rhs.pistons.size(); ++i)
00032         pistons.push_back(std::move(rhs.pistons.at(i)));
00033     rhs.pistons.clear();
00034     return *this;
00035 }
```

## 5.39.3 Member Data Documentation

### 5.39.3.1 pistons

```
std::vector<std::unique_ptr<io::IPiston>> wisco::hal::PistonGroup::pistons {} [private]
```

The pistons in the group.

Definition at line 37 of file [PistonGroup.hpp](#).

```
00037 {};
```

### 5.39.3.2 m\_state

```
bool wisco::hal::PistonGroup::m_state {} [private]
```

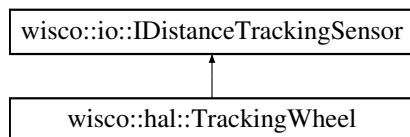
The state of the piston group.

Definition at line 43 of file [PistonGroup.hpp](#).  
00043 {};

## 5.40 wisco::hal::TrackingWheel Class Reference

A tracking wheel sensor.

Inheritance diagram for wisco::hal::TrackingWheel:



### Public Member Functions

- **TrackingWheel** (std::unique\_ptr<[io::IRotationSensor](#)> &sensor, double wheel\_radius)  
*Construct a new Tracking Wheel object.*
- void **initialize** () override  
*Initializes the sensor.*
- void **reset** () override  
*Resets the sensor.*
- double **getDistance** () override  
*Get the distance tracked by the sensor in inches.*
- void **setDistance** (double distance) override  
*Set the distance tracked by the sensor in inches.*

### Public Member Functions inherited from [wisco::io::IDistanceTrackingSensor](#)

- virtual ~[IDistanceTrackingSensor](#) ()=default  
*Destroy the [IDistanceTrackingSensor](#) object.*

### Private Attributes

- std::unique\_ptr<[io::IRotationSensor](#)> **m\_sensor** {}  
*The sensor on the tracking wheel.*
- double **m\_wheel\_radius** {}  
*The radius of the wheel in inches.*

### 5.40.1 Detailed Description

A tracking wheel sensor.

#### Author

Nathan Sandvig

Definition at line 28 of file [TrackingWheel.hpp](#).

### 5.40.2 Constructor & Destructor Documentation

#### 5.40.2.1 TrackingWheel()

```
wisco::hal::TrackingWheel::TrackingWheel (
    std::unique_ptr< io::IRotationSensor > & sensor,
    double wheel_radius )
```

Construct a new Tracking Wheel object.

#### Parameters

<i>sensor</i>	The rotation sensor on the tracking wheel
<i>wheel_radius</i>	The radius of the tracking wheel in inches

Definition at line 7 of file [TrackingWheel.cpp](#).

```
00007
00008     m_sensor{std::move(sensor)}, m_wheel_radius{wheel_radius}
00009 {
00010
00011 }
```

### 5.40.3 Member Function Documentation

#### 5.40.3.1 initialize()

```
void wisco::hal::TrackingWheel::initialize () [override], [virtual]
```

Initializes the sensor.

Implements [wisco::io::IDistanceTrackingSensor](#).

Definition at line 13 of file [TrackingWheel.cpp](#).

```
00014 {
00015     if (m_sensor)
00016         m_sensor->initialize();
00017 }
```

### 5.40.3.2 reset()

```
void wisco::hal::TrackingWheel::reset ( ) [override], [virtual]
```

Resets the sensor.

Implements [wisco::io::IDistanceTrackingSensor](#).

Definition at line 19 of file [TrackingWheel.cpp](#).

```
00020 {
00021     if (m_sensor)
00022         m_sensor->reset ();
00023 }
```

### 5.40.3.3 getDistance()

```
double wisco::hal::TrackingWheel::getDistance ( ) [override], [virtual]
```

Get the distance tracked by the sensor in inches.

#### Returns

`double` The distance tracked by the sensor

Implements [wisco::io::IDistanceTrackingSensor](#).

Definition at line 25 of file [TrackingWheel.cpp](#).

```
00026 {
00027     return m_sensor->getRotation() * m_wheel_radius;
00028 }
```

### 5.40.3.4 setDistance()

```
void wisco::hal::TrackingWheel::setDistance (
    double distance ) [override], [virtual]
```

Set the distance tracked by the sensor in inches.

#### Parameters

<code>distance</code>	The new distance tracked value
-----------------------	--------------------------------

Implements [wisco::io::IDistanceTrackingSensor](#).

Definition at line 30 of file [TrackingWheel.cpp](#).

```
00031 {
00032     m_sensor->setRotation(distance / m_wheel_radius);
00033 }
```

## 5.40.4 Member Data Documentation

### 5.40.4.1 m\_sensor

```
std::unique_ptr<io::IRotationSensor> wisco::hal::TrackingWheel::m_sensor {} [private]
```

The sensor on the tracking wheel.

Definition at line 35 of file [TrackingWheel.hpp](#).  
00035 {};

#### 5.40.4.2 m\_wheel\_radius

```
double wisco::hal::TrackingWheel::m_wheel_radius {} [private]
```

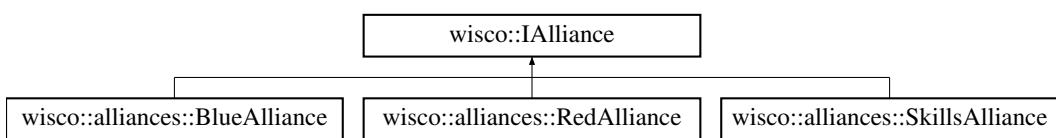
The radius of the wheel in inches.

Definition at line 41 of file [TrackingWheel.hpp](#).  
00041 {};

## 5.41 wisco::IAlliance Class Reference

Interface for the alliances for the robot.

Inheritance diagram for wisco::IAlliance:



### Public Member Functions

- virtual ~**IAlliance** ()=default  
*Destroy the **IAlliance** object.*
- virtual std::string **getName** ()=0  
*Get the name of the alliance.*

### 5.41.1 Detailed Description

Interface for the alliances for the robot.

#### Author

Nathan Sandvig

Definition at line 19 of file [IAlliance.hpp](#).

## 5.41.2 Member Function Documentation

### 5.41.2.1 getName()

```
virtual std::string wisco::IAlliance::getName () [pure virtual]
```

Get the name of the alliance.

#### Returns

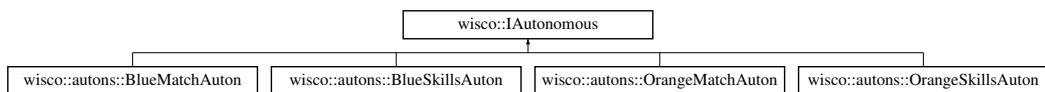
`std::string` The name of the alliance

Implemented in [wisco::alliances::BlueAlliance](#), [wisco::alliances::RedAlliance](#), and [wisco::alliances::SkillsAlliance](#).

## 5.42 wisco::IAutonomous Class Reference

Interface for the autonomous routines in the system.

Inheritance diagram for wisco::IAutonomous:



### Public Member Functions

- virtual ~**IAutonomous** ()=default  
*Destroy the `IAutonomous` object.*
- virtual `std::string getName ()=0`  
*Get the name of the autonomous.*
- virtual `void initialize (std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot)=0`  
*Initialize the autonomous.*
- virtual `void run (std::shared_ptr< rtos::IClock > clock, std::unique_ptr< rtos::IDelay > &delayer, std::shared_ptr< control::ControlSystem > control_system, std::shared_ptr< robot::Robot > robot)=0`  
*Run the autonomous.*

### 5.42.1 Detailed Description

Interface for the autonomous routines in the system.

#### Author

Nathan Sandvig

Definition at line 25 of file [IAutonomous.hpp](#).

## 5.42.2 Member Function Documentation

### 5.42.2.1 getName()

```
virtual std::string wisco::IAutonomous::getName () [pure virtual]
```

Get the name of the autonomous.

#### Returns

`std::string` The name of the autonomous

Implemented in [wisco::autons::BlueMatchAuton](#), [wisco::autons::BlueSkillsAuton](#), [wisco::autons::OrangeMatchAuton](#), and [wisco::autons::OrangeSkillsAuton](#).

### 5.42.2.2 initialize()

```
virtual void wisco::IAutonomous::initialize (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [pure virtual]
```

Initialize the autonomous.

#### Parameters

<code>control_system</code>	The control system
<code>robot</code>	The robot

Implemented in [wisco::autons::BlueMatchAuton](#), [wisco::autons::BlueSkillsAuton](#), [wisco::autons::OrangeMatchAuton](#), and [wisco::autons::OrangeSkillsAuton](#).

### 5.42.2.3 run()

```
virtual void wisco::IAutonomous::run (
    std::shared_ptr< rtos::IClock > clock,
    std::unique_ptr< rtos::IDelay > & delayer,
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< robot::Robot > robot ) [pure virtual]
```

Run the autonomous.

#### Parameters

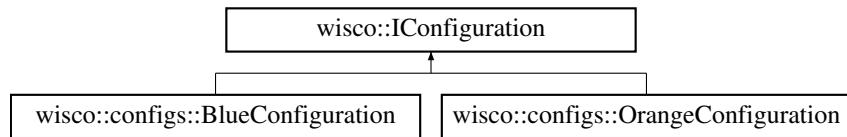
<code>clock</code>	The rtos clock
<code>delayer</code>	The rtos delayer
<code>control_system</code>	The control system
<code>robot</code>	The robot

Implemented in [wisco::autons::BlueMatchAuton](#), [wisco::autons::BlueSkillsAuton](#), [wisco::autons::OrangeMatchAuton](#), and [wisco::autons::OrangeSkillsAuton](#).

## 5.43 wisco::IConfiguration Class Reference

Interface for the configurations in the system.

Inheritance diagram for wisco::IConfiguration:



### Public Member Functions

- virtual ~**IConfiguration** ()=default  
*Destroy the **IConfiguration** object.*
- virtual std::string **getName** ()=0  
*Get the name of the configuration.*
- virtual std::shared\_ptr<[control::ControlSystem](#)> **buildControlSystem** ()=0  
*Build a control system using this configuration.*
- virtual std::shared\_ptr<[user::IController](#)> **buildController** ()=0  
*Build a controller using this configuration.*
- virtual std::shared\_ptr<[robot::Robot](#)> **buildRobot** ()=0  
*Build a robot using this configuration.*

### 5.43.1 Detailed Description

Interface for the configurations in the system.

#### Author

Nathan Sandvig

Definition at line [24](#) of file [IConfiguration.hpp](#).

### 5.43.2 Member Function Documentation

#### 5.43.2.1 getName()

```
virtual std::string wisco::IConfiguration::getName ( ) [pure virtual]
```

Get the name of the configuration.

#### Returns

std::string The name of the configuration

Implemented in [wisco::configs::BlueConfiguration](#), and [wisco::configs::OrangeConfiguration](#).

### 5.43.2.2 buildControlSystem()

```
virtual std::shared_ptr< control::ControlSystem > wisco::IConfiguration::buildControlSystem ( ) [pure virtual]
```

Build a control system using this configuration.

#### Returns

`std::shared_ptr<control::ControlSystem>` The control system built by this configuration

Implemented in [wisco::configs::BlueConfiguration](#), and [wisco::configs::OrangeConfiguration](#).

### 5.43.2.3 buildController()

```
virtual std::shared_ptr< user::IController > wisco::IConfiguration::buildController ( ) [pure virtual]
```

Build a controller using this configuration.

#### Returns

`std::shared_ptr<user::IController>` The controller build by this configuration

Implemented in [wisco::configs::BlueConfiguration](#), and [wisco::configs::OrangeConfiguration](#).

### 5.43.2.4 buildRobot()

```
virtual std::shared_ptr< robot::Robot > wisco::IConfiguration::buildRobot ( ) [pure virtual]
```

Build a robot using this configuration.

#### Returns

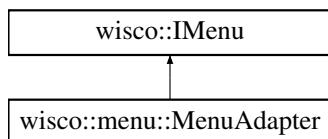
`robot::Robot` The robot built by this configuration

Implemented in [wisco::configs::BlueConfiguration](#), and [wisco::configs::OrangeConfiguration](#).

## 5.44 wisco::IMenu Class Reference

Interface for the menu system.

Inheritance diagram for wisco::IMenu:



## Public Member Functions

- virtual ~**IMenu** ()=default  
*Destroy the **IMenu** object.*
- virtual void **addAlliance** (std::unique\_ptr< **IAlliance** > &alliance)=0  
*Adds an alliance to the menu system.*
- virtual void **addAutonomous** (std::unique\_ptr< **IAutonomous** > &autonomous)=0  
*Adds an autonomous routine to the menu system.*
- virtual void **addConfiguration** (std::unique\_ptr< **IConfiguration** > &configuration)=0  
*Adds a hardware configuration to the menu system.*
- virtual void **addProfile** (std::unique\_ptr< **IProfile** > &profile)=0  
*Adds a driver profile to the menu system.*
- virtual void **display** ()=0  
*Display the menu.*
- virtual bool **isStarted** ()=0  
*Check if the system has been started.*
- virtual **SystemConfiguration** **getSystemConfiguration** ()=0  
*Get the system configuration information.*

### 5.44.1 Detailed Description

Interface for the menu system.

#### Author

Nathan Sandvig

Definition at line 19 of file **IMenu.hpp**.

### 5.44.2 Member Function Documentation

#### 5.44.2.1 addAlliance()

```
virtual void wisco::IMenu::addAlliance (
    std::unique_ptr< IAlliance > & alliance ) [pure virtual]
```

Adds an alliance to the menu system.

#### Parameters

<i>alliance</i>	The new alliance
-----------------	------------------

Implemented in **wisco::menu::MenuAdapter**.

#### 5.44.2.2 addAutonomous()

```
virtual void wisco::IMenu::addAutonomous (
    std::unique_ptr< IAutonomous > & autonomous ) [pure virtual]
```

Adds an autonomous routine to the menu system.

**Parameters**

<code>autonomous</code>	The new autonomous routine
-------------------------	----------------------------

Implemented in [wisco::menu::MenuAdapter](#).

#### 5.44.2.3 addConfiguration()

```
virtual void wisco::IMenu::addConfiguration (
    std::unique_ptr< IConfiguration > & configuration ) [pure virtual]
```

Adds a hardware configuration to the menu system.

**Parameters**

<code>configuration</code>	The new hardware configuration
----------------------------	--------------------------------

Implemented in [wisco::menu::MenuAdapter](#).

#### 5.44.2.4 addProfile()

```
virtual void wisco::IMenu::addProfile (
    std::unique_ptr< IProfile > & profile ) [pure virtual]
```

Adds a driver profile to the menu system.

**Parameters**

<code>profile</code>	The new driver profile
----------------------	------------------------

Implemented in [wisco::menu::MenuAdapter](#).

#### 5.44.2.5 display()

```
virtual void wisco::IMenu::display () [pure virtual]
```

Display the menu.

Implemented in [wisco::menu::MenuAdapter](#).

#### 5.44.2.6 isStarted()

```
virtual bool wisco::IMenu::isStarted () [pure virtual]
```

Check if the system has been started.

**Returns**

true The system has been started  
 false The system has not been started

Implemented in [wisco::menu::MenuAdapter](#).

**5.44.2.7 getSystemConfiguration()**

```
virtual SystemConfiguration wisco::IMenu::getSystemConfiguration () [pure virtual]
```

Get the system configuration information.

**Returns**

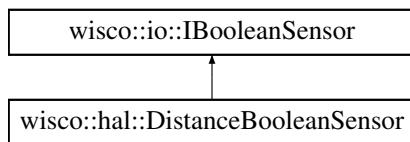
[SystemConfiguration](#) The system configuration information

Implemented in [wisco::menu::MenuAdapter](#).

**5.45 wisco::io::IBooleanSensor Class Reference**

Interface for sensors that generate a boolean value.

Inheritance diagram for wisco::io::IBooleanSensor:

**Public Member Functions**

- virtual ~[IBooleanSensor](#) ()=default  
*Destroy the [IBooleanSensor](#) object.*
- virtual void [initialize](#) ()=0  
*Initializes the sensor.*
- virtual void [reset](#) ()=0  
*Resets the sensor.*
- virtual bool [getValue](#) ()=0  
*Get the boolean value of the sensor.*

**5.45.1 Detailed Description**

Interface for sensors that generate a boolean value.

**Author**

Nathan Sandvig

Definition at line 25 of file [IBooleanSensor.hpp](#).

## 5.45.2 Member Function Documentation

### 5.45.2.1 initialize()

```
virtual void wisco::io::IBooleanSensor::initialize () [pure virtual]
```

Initializes the sensor.

Implemented in [wisco::hal::DistanceBooleanSensor](#).

### 5.45.2.2 reset()

```
virtual void wisco::io::IBooleanSensor::reset () [pure virtual]
```

Resets the sensor.

Implemented in [wisco::hal::DistanceBooleanSensor](#).

### 5.45.2.3 getValue()

```
virtual bool wisco::io::IBooleanSensor::getValue () [pure virtual]
```

Get the boolean value of the sensor.

Returns

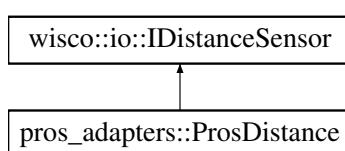
`bool` The value of the sensor

Implemented in [wisco::hal::DistanceBooleanSensor](#).

## 5.46 wisco::io::IDistanceSensor Class Reference

Interface for distance tracking sensors.

Inheritance diagram for wisco::io::IDistanceSensor:



### Public Member Functions

- virtual ~**IDistanceSensor** ()=default  
*Destroy the `IDistanceSensor` object.*
- virtual void **initialize** ()=0  
*Initializes the sensor.*
- virtual void **reset** ()=0  
*Resets the sensor.*
- virtual double **getDistance** ()=0  
*Get the distance detected by the sensor in inches.*

### 5.46.1 Detailed Description

Interface for distance tracking sensors.

#### Author

Nathan Sandvig

Definition at line 25 of file [IDistanceSensor.hpp](#).

### 5.46.2 Member Function Documentation

#### 5.46.2.1 initialize()

```
virtual void wisco::io::IDistanceSensor::initialize () [pure virtual]
```

Initializes the sensor.

Implemented in [pros\\_adapters::ProsDistance](#).

#### 5.46.2.2 reset()

```
virtual void wisco::io::IDistanceSensor::reset () [pure virtual]
```

Resets the sensor.

Implemented in [pros\\_adapters::ProsDistance](#).

#### 5.46.2.3 getDistance()

```
virtual double wisco::io::IDistanceSensor::getDistance () [pure virtual]
```

Get the distance detected by the sensor in inches.

#### Returns

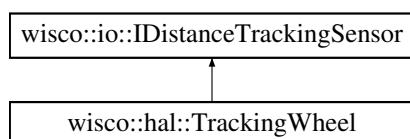
double The distance detected by the sensor

Implemented in [pros\\_adapters::ProsDistance](#).

## 5.47 wisco::io::IDistanceTrackingSensor Class Reference

Interface for distance tracking sensors.

Inheritance diagram for wisco::io::IDistanceTrackingSensor:



## Public Member Functions

- virtual ~**IDistanceTrackingSensor** ()=default  
*Destroy the [IDistanceTrackingSensor](#) object.*
- virtual void **initialize** ()=0  
*Initializes the sensor.*
- virtual void **reset** ()=0  
*Resets the sensor.*
- virtual double **getDistance** ()=0  
*Get the distance tracked by the sensor in inches.*
- virtual void **setDistance** (double distance)=0  
*Set the distance tracked by the sensor in inches.*

### 5.47.1 Detailed Description

Interface for distance tracking sensors.

#### Author

Nathan Sandvig

Definition at line 23 of file [IDistanceTrackingSensor.hpp](#).

### 5.47.2 Member Function Documentation

#### 5.47.2.1 initialize()

```
virtual void wisco::io::IDistanceTrackingSensor::initialize ( ) [pure virtual]
```

Initializes the sensor.

Implemented in [wisco::hal::TrackingWheel](#).

#### 5.47.2.2 reset()

```
virtual void wisco::io::IDistanceTrackingSensor::reset ( ) [pure virtual]
```

Resets the sensor.

Implemented in [wisco::hal::TrackingWheel](#).

#### 5.47.2.3 getDistance()

```
virtual double wisco::io::IDistanceTrackingSensor::getDistance ( ) [pure virtual]
```

Get the distance tracked by the sensor in inches.

#### Returns

double The distance tracked by the sensor

Implemented in [wisco::hal::TrackingWheel](#).

#### 5.47.2.4 setDistance()

```
virtual void wisco::io::IDistanceTrackingSensor::setDistance ( double distance ) [pure virtual]
```

Set the distance tracked by the sensor in inches.

**Parameters**

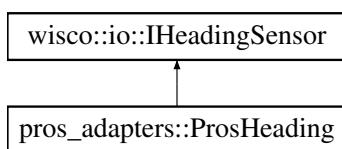
<i>distance</i>	The new distance tracked value
-----------------	--------------------------------

Implemented in [wisco::hal::TrackingWheel](#).

## 5.48 wisco::io::IHeadingSensor Class Reference

Interface for heading sensors.

Inheritance diagram for wisco::io::IHeadingSensor:



### Public Member Functions

- virtual ~**IHeadingSensor** ()=default  
*Destroy the **IHeadingSensor** object.*
- virtual void **Initialize** ()=0  
*Initializes the sensor.*
- virtual void **reset** ()=0  
*Resets the sensor.*
- virtual double **getHeading** ()=0  
*Get the heading of the sensor in radians.*
- virtual void **setHeading** (double heading)=0  
*Set the heading of the sensor in radians.*
- virtual double **getRotation** ()=0  
*Get the rotation of the sensor in radians.*
- virtual void **setRotation** (double rotation)=0  
*Set the rotation of the sensor in radians.*

### 5.48.1 Detailed Description

Interface for heading sensors.

#### Author

Nathan Sandvig

Definition at line 23 of file [IHeadingSensor.hpp](#).

## 5.48.2 Member Function Documentation

### 5.48.2.1 initialize()

```
virtual void wisco::io::IHeadingSensor::initialize () [pure virtual]
```

Initializes the sensor.

Implemented in [pros\\_adapters::ProsHeading](#).

### 5.48.2.2 reset()

```
virtual void wisco::io::IHeadingSensor::reset () [pure virtual]
```

Resets the sensor.

Implemented in [pros\\_adapters::ProsHeading](#).

### 5.48.2.3 getHeading()

```
virtual double wisco::io::IHeadingSensor::getHeading () [pure virtual]
```

Get the heading of the sensor in radians.

#### Returns

double The heading in radians

Implemented in [pros\\_adapters::ProsHeading](#).

### 5.48.2.4 setHeading()

```
virtual void wisco::io::IHeadingSensor::setHeading (
    double heading) [pure virtual]
```

Set the heading of the sensor in radians.

#### Parameters

<i>heading</i>	The heading in radians
----------------	------------------------

Implemented in [pros\\_adapters::ProsHeading](#).

### 5.48.2.5 getRotation()

```
virtual double wisco::io::IHeadingSensor::getRotation () [pure virtual]
```

Get the rotation of the sensor in radians.

**Returns**

double The rotation in radians

Implemented in [pros\\_adapters::ProsHeading](#).

**5.48.2.6 setRotation()**

```
virtual void wisco::io::IHeadingSensor::setRotation (
    double rotation ) [pure virtual]
```

Set the rotation of the sensor in radians.

**Parameters**

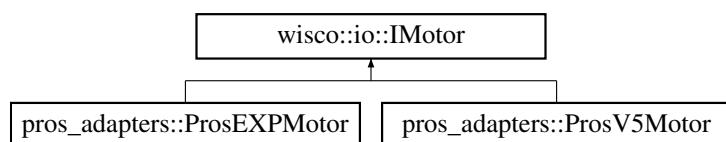
<i>rotation</i>	The rotation in radians
-----------------	-------------------------

Implemented in [pros\\_adapters::ProsHeading](#).

**5.49 wisco::io::IMotor Class Reference**

Interface for electric motors controlled by voltage.

Inheritance diagram for wisco::io::IMotor:

**Public Member Functions**

- virtual ~**IMotor** ()=default  
*Destroy the [IMotor](#) object.*
- virtual void **initialize** ()=0  
*Initializes the motor.*
- virtual double **getTorqueConstant** ()=0  
*Get the torque constant of the motor.*
- virtual double **getResistance** ()=0  
*Get the resistance of the motor.*
- virtual double **getAngularVelocityConstant** ()=0  
*Get the angular velocity constant of the motor.*
- virtual double **getGearRatio** ()=0  
*Get the gear ratio of the motor (1 if n/a)*
- virtual double **getAngularVelocity** ()=0  
*Get the angular velocity of the motor in radians/second.*
- virtual double **getPosition** ()=0  
*Get the position of the motor in total radians.*
- virtual void **setVoltage** (double volts)=0  
*Set the voltage input to the motor in Volts.*

## 5.49.1 Detailed Description

Interface for electric motors controlled by voltage.

### Author

Nathan Sandvig

Definition at line 24 of file [IMotor.hpp](#).

## 5.49.2 Member Function Documentation

### 5.49.2.1 initialize()

```
virtual void wisco::io::IMotor::initialize () [pure virtual]
```

Initializes the motor.

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

### 5.49.2.2 getTorqueConstant()

```
virtual double wisco::io::IMotor::getTorqueConstant () [pure virtual]
```

Get the torque constant of the motor.

#### Returns

double The torque constant of the motor

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

### 5.49.2.3 getResistance()

```
virtual double wisco::io::IMotor::getResistance () [pure virtual]
```

Get the resistance of the motor.

#### Returns

double The resistance of the motor

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

#### 5.49.2.4 getAngularVelocityConstant()

```
virtual double wisco::io::IMotor::getAngularVelocityConstant() [pure virtual]
```

Get the angular velocity constant of the motor.

##### Returns

double The angular velocity constant of the motor

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

#### 5.49.2.5 getGearRatio()

```
virtual double wisco::io::IMotor::getGearRatio() [pure virtual]
```

Get the gear ratio of the motor (1 if n/a)

##### Returns

double The gear ratio of the motor

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

#### 5.49.2.6 getAngularVelocity()

```
virtual double wisco::io::IMotor::getAngularVelocity() [pure virtual]
```

Get the angular velocity of the motor in radians/second.

##### Returns

double The angular velocity of the motor in radians/second

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

#### 5.49.2.7 getPosition()

```
virtual double wisco::io::IMotor::getPosition() [pure virtual]
```

Get the position of the motor in total radians.

##### Returns

double The total number of radians moved since last reset

Implemented in [pros\\_adapters::ProsV5Motor](#).

#### 5.49.2.8 setVoltage()

```
virtual void wisco::io::IMotor::setVoltage( double volts ) [pure virtual]
```

Set the voltage input to the motor in Volts.

**Parameters**

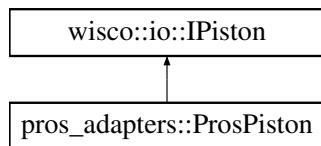
<code>volts</code>	The voltage input in Volts
--------------------	----------------------------

Implemented in [pros\\_adapters::ProsEXPMotor](#), and [pros\\_adapters::ProsV5Motor](#).

## 5.50 wisco::io::IPiston Class Reference

Interface for a discrete state-based piston.

Inheritance diagram for wisco::io::IPiston:



### Public Member Functions

- virtual ~**IPiston** ()=default  
*Destroy the `IPiston` object.*
- virtual void **extend** ()=0  
*Extends the piston.*
- virtual void **retract** ()=0  
*Retracts the piston.*
- virtual void **toggle** ()=0  
*Toggles the piston state.*
- virtual bool **isExtended** ()=0  
*Checks if the piston is extended.*
- virtual bool **isRetracted** ()=0  
*Checks if the piston is retracted.*

### 5.50.1 Detailed Description

Interface for a discrete state-based piston.

**Author**

Nathan Sandvig

Definition at line 25 of file [IPiston.hpp](#).

## 5.50.2 Member Function Documentation

### 5.50.2.1 extend()

```
virtual void wisco::io::IPiston::extend () [pure virtual]
```

Extends the piston.

Implemented in [pros\\_adapters::ProsPiston](#).

### 5.50.2.2 retract()

```
virtual void wisco::io::IPiston::retract () [pure virtual]
```

Retracts the piston.

Implemented in [pros\\_adapters::ProsPiston](#).

### 5.50.2.3 toggle()

```
virtual void wisco::io::IPiston::toggle () [pure virtual]
```

Toggles the piston state.

Implemented in [pros\\_adapters::ProsPiston](#).

### 5.50.2.4 isExtended()

```
virtual bool wisco::io::IPiston::isExtended () [pure virtual]
```

Checks if the piston is extended.

#### Returns

true The piston is extended

false The piston is not extended

Implemented in [pros\\_adapters::ProsPiston](#).

### 5.50.2.5 isRetracted()

```
virtual bool wisco::io::IPiston::isRetracted () [pure virtual]
```

Checks if the piston is retracted.

#### Returns

true The piston is retracted

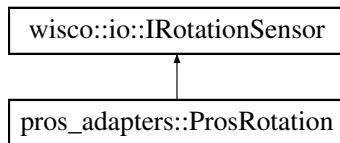
false The piston is not retracted

Implemented in [pros\\_adapters::ProsPiston](#).

## 5.51 wisco::io::IRotationSensor Class Reference

Interface for rotation sensors.

Inheritance diagram for wisco::io::IRotationSensor:



### Public Member Functions

- virtual ~**IRotationSensor** ()=default  
*Destroy the [IRotationSensor](#) object.*
- virtual void **initialize** ()=0  
*Initializes the rotation sensor.*
- virtual void **reset** ()=0  
*Resets the rotation sensor.*
- virtual double **getRotation** ()=0  
*Get the rotation of the sensor in radians.*
- virtual void **setRotation** (double rotation)=0  
*Set the rotation of the sensor in radians.*
- virtual double **getAngle** ()=0  
*Get the angle of the sensor in radians.*

### 5.51.1 Detailed Description

Interface for rotation sensors.

#### Author

Nathan Sandvig

Definition at line 23 of file [IRotationSensor.hpp](#).

### 5.51.2 Member Function Documentation

#### 5.51.2.1 initialize()

```
virtual void wisco::io::IRotationSensor::initialize ( ) [pure virtual]
```

Initializes the rotation sensor.

Implemented in [pros\\_adapters::ProsRotation](#).

### 5.51.2.2 **reset()**

```
virtual void wisco::io::IRotationSensor::reset ( ) [pure virtual]
```

Resets the rotation sensor.

Implemented in [pros\\_adapters::ProsRotation](#).

### 5.51.2.3 **getRotation()**

```
virtual double wisco::io::IRotationSensor::getRotation ( ) [pure virtual]
```

Get the rotation of the sensor in radians.

Returns

double The number of radians of rotation

Implemented in [pros\\_adapters::ProsRotation](#).

### 5.51.2.4 **setRotation()**

```
virtual void wisco::io::IRotationSensor::setRotation ( double rotation ) [pure virtual]
```

Set the rotation of the sensor in radians.

Parameters

<i>rotation</i>	The number of radians of rotation
-----------------	-----------------------------------

Implemented in [pros\\_adapters::ProsRotation](#).

### 5.51.2.5 **getAngle()**

```
virtual double wisco::io::IRotationSensor::getAngle ( ) [pure virtual]
```

Get the angle of the sensor in radians.

Returns

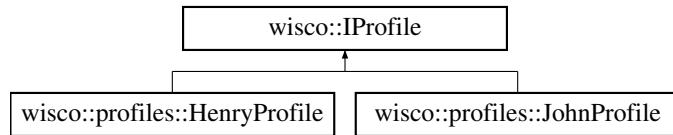
double The angle in radians

Implemented in [pros\\_adapters::ProsRotation](#).

## 5.52 wisco::IProfile Class Reference

Interface for the profiles in the system.

Inheritance diagram for wisco::IProfile:



### Public Member Functions

- virtual ~**IProfile** ()=default  
*Destroy the [IProfile](#) object.*
- virtual std::string **getName** ()=0  
*Get the name of the profile.*
- virtual int **getControlMode** ([user::EControlType](#) control\_type) const =0  
*Get the control mode for a specific control type.*
- virtual void **setControlMode** ([user::EControlType](#) control\_type, int control\_mode)=0  
*Set the control mode for a specific control type.*
- virtual [user::EControllerAnalog](#) **getAnalogControlMapping** ([user::EControl](#) control) const =0  
*Get the mapping of a control to analog inputs.*
- virtual [user::EControllerDigital](#) **getDigitalControlMapping** ([user::EControl](#) control) const =0  
*Get the mapping of a control to digital inputs.*

### 5.52.1 Detailed Description

Interface for the profiles in the system.

#### Author

Nathan Sandvig

Definition at line 24 of file [IProfile.hpp](#).

### 5.52.2 Member Function Documentation

#### 5.52.2.1 getName()

```
virtual std::string wisco::IProfile::getName ( ) [pure virtual]
```

Get the name of the profile.

#### Returns

std::string The name of the profile

Implemented in [wisco::profiles::HenryProfile](#), and [wisco::profiles::JohnProfile](#).

#### 5.52.2.2 getControlMode()

```
virtual int wisco::IProfile::getControlMode (
    user::EControlType control_type ) const [pure virtual]
```

Get the control mode for a specific control type.

**Parameters**

<i>control_type</i>	The control type
---------------------	------------------

**Returns**

int The control mode

Implemented in [wisco::profiles::HenryProfile](#), and [wisco::profiles::JohnProfile](#).

**5.52.2.3 setControlMode()**

```
virtual void wisco::IProfile::setControlMode (
    user::EControlType control_type,
    int control_mode ) [pure virtual]
```

Set the control mode for a specific control type.

**Parameters**

<i>control_type</i>	The control type
<i>control_mode</i>	The control mode

Implemented in [wisco::profiles::HenryProfile](#), and [wisco::profiles::JohnProfile](#).

**5.52.2.4 getAnalogControlMapping()**

```
virtual user::EControllerAnalog wisco::IProfile::getAnalogControlMapping (
    user::EControl control ) const [pure virtual]
```

Get the mapping of a control to analog inputs.

**Parameters**

<i>control</i>	The control
----------------	-------------

**Returns**

[user::EControllerAnalog](#) The mapping of this control to a analog input

Implemented in [wisco::profiles::HenryProfile](#), and [wisco::profiles::JohnProfile](#).

**5.52.2.5 getDigitalControlMapping()**

```
virtual user::EControllerDigital wisco::IProfile::getDigitalControlMapping (
    user::EControl control ) const [pure virtual]
```

Get the mapping of a control to digital inputs.

**Parameters**

<code>control</code>	The control
----------------------	-------------

**Returns**

`user::EControllerDigital` The mapping of this control to a digital input

Implemented in [wisco::profiles::HenryProfile](#), and [wisco::profiles::JohnProfile](#).

## 5.53 wisco::MatchController Class Reference

Handles the field controller inputs during a match.

**Public Member Functions**

- `MatchController (std::unique_ptr< IMenu > &menu, const std::shared_ptr< rtos::IClock > &clock, std::unique_ptr< rtos::IDelay > &delayer)`  
*Construct a new Match Controller object.*
- `void initialize ()`  
*Runs the robot initialization code.*
- `void disabled ()`  
*Runs the robot disablement code.*
- `void competitionInitialize ()`  
*Runs the robot competition initialization code.*
- `void autonomous ()`  
*Runs the robot autonomous code.*
- `void operatorControl ()`  
*Runs the robot operator control code.*

**Private Attributes**

- `std::unique_ptr< IMenu > m_menu {}`  
*The menu system.*
- `std::shared_ptr< rtos::IClock > m_clock {}`  
*The rtos clock.*
- `std::unique_ptr< rtos::IDelay > m_delayer {}`  
*The rtos delayer.*
- `AutonomousManager autonomous_manager {m_clock, m_delayer}`  
*The autonomous management object.*
- `OPControlManager opcontrol_manager {m_clock, m_delayer}`  
*The opcontrol management object.*
- `std::shared_ptr< control::ControlSystem > control_system {}`  
*The control system.*
- `std::shared_ptr< user::IController > controller {}`  
*The user input controller.*
- `std::shared_ptr< robot::Robot > robot {}`  
*The robot being controlled.*

## Static Private Attributes

- static constexpr uint32\_t **MENU\_DELAY** {10}

*The number of milliseconds to wait to check the menu.*

### 5.53.1 Detailed Description

Handles the field controller inputs during a match.

#### Author

Nathan Sandvig

Definition at line 26 of file [MatchController.hpp](#).

### 5.53.2 Constructor & Destructor Documentation

#### 5.53.2.1 MatchController()

```
wisco::MatchController::MatchController (
    std::unique_ptr< IMenu > & menu,
    const std::shared_ptr< rtos::IClock > & clock,
    std::unique_ptr< rtos::IDelayer > & delayer )
```

Construct a new Match Controller object.

#### Parameters

<i>menu</i>	The menu to use in the match controller
<i>clock</i>	The rtos clock to use in the match controller
<i>delayer</i>	The rtos delayer to use in the match controller

Definition at line 5 of file [MatchController.cpp](#).

```
00005 :
00006     m_menu{std::move(menu)}, m_clock{clock}, m_delayer{std::move(delayer)},
00007     autonomous_manager{m_clock, m_delayer}, opcontrol_manager{m_clock, m_delayer}
00008 {
00009
00010 }
```

### 5.53.3 Member Function Documentation

#### 5.53.3.1 initialize()

```
void wisco::MatchController::initialize ( )
```

Runs the robot initialization code.

Definition at line 12 of file [MatchController.cpp](#).

```
00013 {
```

```

00014     if (m_menu)
00015     {
00016         m_menu->display();
00017         while (m_delayer && !m_menu->isStarted())
00018             m_delayer->delay(MENU_DELAY);
00019     }
00020
00021     SystemConfiguration system_configuration{};
00022     if (m_menu)
00023         system_configuration = m_menu->getSystemConfiguration();
00024     autonomous_manager.setAutonomous(system_configuration.autonomous);
00025     opcontrol_manager.setProfile(system_configuration.profile);
00026     control_system = system_configuration.configuration->buildControlSystem();
00027     controller = system_configuration.configuration->buildController();
00028     robot = system_configuration.configuration->buildRobot();
00029
00030     if (robot)
00031         robot->initialize();
00032     if (control_system)
00033         control_system->initialize();
00034     if (controller)
00035     {
00036         controller->initialize();
00037         controller->run();
00038     }
00039
00040     autonomous_manager.initializeAutonomous(control_system, robot);
00041     opcontrol_manager.initializeOpcontrol(control_system, controller, robot);
00042 }
```

### 5.53.3.2 disabled()

```
void wisco::MatchController::disabled ( )
```

Runs the robot disablement code.

Definition at line 44 of file [MatchController.cpp](#).

```
00045 {
00046
00047 }
```

### 5.53.3.3 competitionInitialize()

```
void wisco::MatchController::competitionInitialize ( )
```

Runs the robot competition initialization code.

Definition at line 49 of file [MatchController.cpp](#).

```
00050 {
00051
00052 }
```

### 5.53.3.4 autonomous()

```
void wisco::MatchController::autonomous ( )
```

Runs the robot autonomous code.

Definition at line 54 of file [MatchController.cpp](#).

```
00055 {
00056     autonomous_manager.runAutonomous(control_system, robot);
00057 }
```

### 5.53.3.5 operatorControl()

```
void wisco::MatchController::operatorControl ()
```

Runs the robot operator control code.

Definition at line 59 of file [MatchController.cpp](#).

```
00060 {
00061     opcontrol_manager.runOpcontrol(control_system, controller, robot);
00062 }
```

## 5.53.4 Member Data Documentation

### 5.53.4.1 MENU\_DELAY

```
constexpr uint32_t wisco::MatchController::MENU_DELAY {10} [static], [constexpr], [private]
```

The number of milliseconds to wait to check the menu.

Definition at line 33 of file [MatchController.hpp](#).

```
00033 {10};
```

### 5.53.4.2 m\_menu

```
std::unique_ptr<IMenu> wisco::MatchController::m_menu {} [private]
```

The menu system.

Definition at line 39 of file [MatchController.hpp](#).

```
00039 {};
```

### 5.53.4.3 m\_clock

```
std::shared_ptr<rtos::IClock> wisco::MatchController::m_clock {} [private]
```

The rtos clock.

Definition at line 45 of file [MatchController.hpp](#).

```
00045 {};
```

### 5.53.4.4 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::MatchController::m_delayer {} [private]
```

The rtos delayer.

Definition at line 51 of file [MatchController.hpp](#).

```
00051 {};
```

#### 5.53.4.5 autonomous\_manager

```
AutonomousManager wisco::MatchController::autonomous_manager {m_clock, m_delayer} [private]
```

The autonomous management object.

Definition at line [57](#) of file [MatchController.hpp](#).

```
00057 {m_clock, m_delayer};
```

#### 5.53.4.6 opcontrol\_manager

```
OPControlManager wisco::MatchController::opcontrol_manager {m_clock, m_delayer} [private]
```

The opcontrol management object.

Definition at line [63](#) of file [MatchController.hpp](#).

```
00063 {m_clock, m_delayer};
```

#### 5.53.4.7 control\_system

```
std::shared_ptr<control::ControlSystem> wisco::MatchController::control_system {} [private]
```

The control system.

Definition at line [69](#) of file [MatchController.hpp](#).

```
00069 {};
```

#### 5.53.4.8 controller

```
std::shared_ptr<user::IController> wisco::MatchController::controller {} [private]
```

The user input controller.

Definition at line [75](#) of file [MatchController.hpp](#).

```
00075 {};
```

#### 5.53.4.9 robot

```
std::shared_ptr<robot::Robot> wisco::MatchController::robot {} [private]
```

The robot being controlled.

Definition at line [81](#) of file [MatchController.hpp](#).

```
00081 {};
```

## 5.54 wisco::menu::LvglMenu Class Reference

Controls an lvgl-based menu selection system.

## Public Member Functions

- void `addOption` (`Option` option)
 

*Adds an option to the menu system.*
- void `removeOption` (const std::string &option\_name)
 

*Removes an option from the menu system.*
- void `drawMainMenu` ()
 

*Draws the main menu screen.*
- void `drawSettingsMenu` ()
 

*Draws the settings menu screen.*
- void `setComplete` ()
 

*Set the menu selection to complete.*
- void `readConfiguration` ()
 

*Reads the configuration of the menu.*
- void `writeConfiguration` ()
 

*Writes the configuration of the menu.*
- void `displayMenu` ()
 

*Displays the menu.*
- bool `selectionComplete` ()
 

*Checks if the selection process is complete.*
- std::string `getSelection` (const std::string &option\_name)
 

*Get the selection for an option.*

## Static Private Member Functions

- static void `initializeStyles` ()
 

*Initializes the styles for the class.*

## Private Attributes

- std::vector< `Option` > `options` {}
 

*The options available in the menu.*
- bool `complete` {false}
 

*Whether or not selection is complete.*

## Static Private Attributes

- static constexpr char `CONFIGURATION_FILE` [] {"/usr/system/menu\_data.txt"}
 

*The path of the file that stores the configuration.*
- static constexpr int `COLUMN_WIDTH` {16}
 

*The width of a column on the status page.*
- static constexpr int `BUTTONS_PER_LINE` {2}
 

*The number of buttons to display on a line.*
- static lv\_style\_t `button_default_style`

*The default style for a button.*
- static lv\_style\_t `button_pressed_style`

*The pressed style for a button.*
- static lv\_style\_t `container_default_style`

*The default style for a container.*

- static lv\_style\_t [container\\_pressed\\_style](#)  
*The pressed style for a container.*
- static lv\_style\_t [button\\_matrix\\_main\\_style](#)  
*The background style for a button matrix.*
- static lv\_style\_t [button\\_matrix\\_items\\_style](#)  
*The button style for a button matrix.*
- static bool [styles\\_initialized](#) = false  
*Whether or not the styles have been initialized.*

## 5.54.1 Detailed Description

Controls an lvgl-based menu selection system.

### Author

Nathan Sandvig

Definition at line [60](#) of file [LvglMenu.hpp](#).

## 5.54.2 Member Function Documentation

### 5.54.2.1 initializeStyles()

```
void wisco::menu::LvglMenu::initializeStyles () [static], [private]
```

Initializes the styles for the class.

Definition at line [62](#) of file [LvglMenu.cpp](#).

```
00063 {
00064     if (styles\_initialized)
00065         return;
00066
00067     // Create the default button style
00068     lv_style_init(&button\_default\_style);
00069     lv_style_set_radius(&button\_default\_style, 5);
00070     lv_style_set_bg_opa(&button\_default\_style, LV_OPA_100);
00071     lv_style_set_bg_color(&button\_default\_style, lv_color_make(192, 192, 192));
00072     lv_style_set_bg_grad_color(&button\_default\_style, lv_color_darken(lv_color_make(192, 192, 192),
00073     8));
00074     lv_style_set_border_opa(&button\_default\_style, LV_OPA_100);
00075     lv_style_set_border_width(&button\_default\_style, 2);
00076     lv_style_set_border_color(&button\_default\_style, lv_color_black());
00077     lv_style_set_text_color(&button\_default\_style, lv_color_black());
00078     lv_style_set_text_font(&button\_default\_style, &lv_font_montserrat_20);
00079
00080     // Create the pressed button style
00081     lv_style_init(&button\_pressed\_style);
00082     lv_style_set_radius(&button\_pressed\_style, 5);
00083     lv_style_set_bg_opa(&button\_pressed\_style, LV_OPA_100);
00084     lv_style_set_translate_y(&button\_pressed\_style, 3);
00085     lv_style_set_shadow_ofs_y(&button\_pressed\_style, 3);
00086     lv_style_set_bg_color(&button\_pressed\_style, lv_color_darken(lv_color_make(192, 192, 192), 16));
00087     lv_style_set_bg_grad_color(&button\_pressed\_style, lv_color_darken(lv_color_make(192, 192, 192),
00088     24));
00089     lv_style_set_border_opa(&button\_pressed\_style, LV_OPA_100);
00090     lv_style_set_border_width(&button\_pressed\_style, 2);
00091     lv_style_set_border_color(&button\_pressed\_style, lv_color_black());
00092     lv_style_set_text_color(&button\_pressed\_style, lv_color_black());
00093     lv_style_set_text_font(&button\_pressed\_style, &lv_font_montserrat_20);
00094
00095     // Create the default container style
00096     lv_style_init(&container\_default\_style);
00097     lv_style_set_radius(&container\_default\_style, 0);
00098     lv_style_set_bg_opa(&container\_default\_style, LV_OPA_100);
```

```

00097     lv_style_set_bg_color(&container_default_style, lv_color_make(0, 104, 179));
00098     lv_style_set_border_width(&container_pressed_style, 0);
00099     lv_style_set_text_color(&container_default_style, lv_color_white());
00100     lv_style_set_text_align(&container_default_style, LV_TEXT_ALIGN_CENTER);
00101     lv_style_set_pad_ver(&container_default_style, 10);
00102
00103     // Create the pressed container style
00104     lv_style_init(&container_pressed_style);
00105     lv_style_set_radius(&container_pressed_style, 0);
00106     lv_style_set_bg_opa(&container_pressed_style, LV_OPA_100);
00107     lv_style_set_bg_color(&container_pressed_style, lv_color_make(244, 115, 33));
00108     lv_style_set_border_width(&container_pressed_style, 0);
00109     lv_style_set_text_color(&container_pressed_style, lv_color_black());
00110     lv_style_set_text_align(&container_pressed_style, LV_TEXT_ALIGN_CENTER);
00111     lv_style_set_pad_ver(&container_default_style, 10);
00112
00113     // Create the button matrix main style
00114     lv_style_init(&button_matrix_main_style);
00115     lv_style_set_bg_color(&button_matrix_main_style, lv_color_make(173, 205, 234));
00116     lv_style_set_border_width(&button_matrix_main_style, 0);
00117
00118     // Create the button matrix items style
00119     lv_style_init(&button_matrix_items_style);
00120
00121     // Set the style initialization flag
00122     styles_initialized = true;
00123 }
```

### 5.54.2.2 addOption()

```
void wisco::menu::LvglMenu::addOption (
    Option option )
```

Adds an option to the menu system.

#### Parameters

<i>option</i>	The option being added
---------------	------------------------

Definition at line 125 of file [LvglMenu.cpp](#).

```
00126 {
00127     options.push_back(option);
00128 }
```

### 5.54.2.3 removeOption()

```
void wisco::menu::LvglMenu::removeOption (
    const std::string & option_name )
```

Removes an option from the menu system.

#### Parameters

<i>option_name</i>	The name of the option to remove
--------------------	----------------------------------

Definition at line 130 of file [LvglMenu.cpp](#).

```
00131 {
00132     for (auto it{options.begin()}; it != options.end(); ++it)
00133     {
00134         if (option_name == it->name)
00135         {
00136             options.erase(it);
00137             break;
00138         }
00139     }
00140 }
```

```
00139     }
00140 }
```

#### 5.54.2.4 drawMainMenu()

```
void wisco::menu::LvglMenu::drawMainMenu ( )
```

Draws the main menu screen.

Definition at line 142 of file [LvglMenu.cpp](#).

```
00143 {
00144     // Set the background color to light blue
00145     lv_obj_set_style_bg_color(lv_scr_act(), lv_color_make(173, 205, 234), 0);
00146     lv_obj_refresh_style(lv_scr_act(), LV_PART_MAIN, LV_STYLE_BG_COLOR);
00147
00148     // Create the big line at the bottom
00149     static lv_point_t big_line_points[] = { {0, 205}, {480, 205} };
00150     static lv_style_t big_line_style;
00151     lv_style_init(&big_line_style);
00152     lv_style_set_line_width(&big_line_style, 55);
00153     lv_style_set_line_color(&big_line_style, lv_color_make(0, 104, 179));
00154     lv_style_set_line_rounded(&big_line_style, false);
00155     lv_obj_t* big_line = lv_line_create(lv_scr_act());
00156     lv_line_set_points(big_line, big_line_points, 2);
00157     lv_obj_add_style(big_line, &big_line_style, 0);
00158
00159     // Create the stripe on the line at the bottom
00160     static lv_point_t stripe_line_points[] = { {0, 220}, {480, 220} };
00161     static lv_style_t stripe_line_style;
00162     lv_style_init(&stripe_line_style);
00163     lv_style_set_line_width(&stripe_line_style, 13);
00164     lv_style_set_line_color(&stripe_line_style, lv_color_make(244, 115, 33));
00165     lv_style_set_line_rounded(&stripe_line_style, false);
00166     lv_obj_t* stripe_line = lv_line_create(lv_scr_act());
00167     lv_line_set_points(stripe_line, stripe_line_points, 2);
00168     lv_obj_add_style(stripe_line, &stripe_line_style, 0);
00169
00170     // Create the left diagonal line
00171     static lv_point_t left_diagonal_line_points[] = { {320, 190}, {510, 0} };
00172     static lv_style_t left_diagonal_line_style;
00173     lv_style_init(&left_diagonal_line_style);
00174     lv_style_set_line_width(&left_diagonal_line_style, 23);
00175     lv_style_set_line_color(&left_diagonal_line_style, lv_color_make(0, 104, 179));
00176     lv_style_set_line_rounded(&left_diagonal_line_style, false);
00177     lv_obj_t* left_diagonal_line = lv_line_create(lv_scr_act());
00178     lv_line_set_points(left_diagonal_line, left_diagonal_line_points, 2);
00179     lv_obj_add_style(left_diagonal_line, &left_diagonal_line_style, 0);
00180
00181     // Create the right diagonal line
00182     static lv_point_t right_diagonal_line_points[] = { {370, 190}, {560, 0} };
00183     static lv_style_t right_diagonal_line_style;
00184     lv_style_init(&right_diagonal_line_style);
00185     lv_style_set_line_width(&right_diagonal_line_style, 23);
00186     lv_style_set_line_color(&right_diagonal_line_style, lv_color_make(0, 104, 179));
00187     lv_style_set_line_rounded(&right_diagonal_line_style, false);
00188     lv_obj_t* right_diagonal_line = lv_line_create(lv_scr_act());
00189     lv_line_set_points(right_diagonal_line, right_diagonal_line_points, 2);
00190     lv_obj_add_style(right_diagonal_line, &right_diagonal_line_style, 0);
00191
00192     // Add the WISCOBOTS text
00193     static lv_style_t team_name_label_style;
00194     lv_style_init(&team_name_label_style);
00195     lv_style_set_text_font(&team_name_label_style, &pros_font_dejavu_mono_30);
00196     lv_style_set_text_color(&team_name_label_style, lv_color_make(244, 115, 33));
00197     lv_obj_t* team_name_label = lv_label_create(lv_scr_act());
00198     lv_obj_add_style(team_name_label, &team_name_label_style, 0);
00199     lv_label_set_text(team_name_label, "wiscobots");
00200     lv_obj_align(team_name_label, LV_ALIGN_BOTTOM_MID, 0, -26);
00201
00202     // Add the status label
00203     static lv_style_t status_label_style;
00204     lv_style_init(&status_label_style);
00205     lv_style_set_border_width(&status_label_style, 2);
00206     lv_style_set_pad_all(&status_label_style, 3);
00207     lv_style_set_border_color(&status_label_style, lv_color_make(0, 104, 179));
00208     lv_style_set_text_color(&status_label_style, lv_color_black());
00209     lv_obj_t* status_label = lv_label_create(lv_scr_act());
00210     lv_obj_add_style(status_label, &status_label_style, 0);
00211     std::string status_text{};
00212     for (Option& option : options)
00213     {
```

```

00214     if (option.name != options.front().name)
00215         status_text += '\n';
00216     status_text += option.name;
00217     status_text += ":";
00218     for (uint8_t i{0}; i < COLUMN_WIDTH - option.name.length() - 1; ++i)
00219         status_text += " ";
00220     status_text += option.choices[option.selected];
00221 }
00222 lv_label_set_text_fmt(status_label, "%s", status_text.c_str());
00223 lv_obj_align(status_label, LV_ALIGN_TOP_LEFT, 20, 100);
00224
00225 // Add the start button
00226 lv_obj_t* start_button = lv_btn_create(lv_scr_act());
00227 lv_obj_remove_style_all(start_button);
00228 lv_obj_add_style(start_button, &button_default_style, 0);
00229 lv_obj_add_style(start_button, &button_pressed_style, LV_STATE_PRESSED);
00230 lv_obj_set_size(start_button, 160, 70);
00231 lv_obj_align(start_button, LV_ALIGN_TOP_LEFT, 20, 15);
00232 static void* start_user_data[] { this };
00233 lv_obj_add_event_cb(start_button, startButtonEventHandler, LV_EVENT_CLICKED, start_user_data);
00234 lv_obj_t * start_button_label = lv_label_create(start_button);
00235 lv_label_set_text(start_button_label, "START");
00236 lv_obj_center(start_button_label);
00237
00238 // Add the settings button
00239 lv_obj_t * settings_button = lv_btn_create(lv_scr_act());
00240 lv_obj_remove_style_all(settings_button);
00241 lv_obj_add_style(settings_button, &button_default_style, 0);
00242 lv_obj_add_style(settings_button, &button_pressed_style, LV_STATE_PRESSED);
00243 lv_obj_set_size(settings_button, 70, 70);
00244 lv_obj_align(settings_button, LV_ALIGN_TOP_LEFT, 190, 15);
00245 static void* settings_user_data[] { this };
00246 lv_obj_add_event_cb(settings_button, settingsButtonEventHandler, LV_EVENT_CLICKED,
    settings_user_data);
00247 lv_obj_t * settings_button_label = lv_label_create(settings_button);
00248 lv_label_set_text(settings_button_label, LV_SYMBOL_SETTINGS);
00249 lv_obj_center(settings_button_label);
00250 }

```

### 5.54.2.5 drawSettingsMenu()

```
void wisco::menu::LvglMenu::drawSettingsMenu ()
```

Draws the settings menu screen.

Definition at line 252 of file LvglMenu.cpp.

```

00253 {
00254     // Create the menu
00255     lv_obj_t* menu{lv_menu_create(lv_scr_act())};
00256     lv_menu_set_mode_root_back_btn(menu, LV_MENU_ROOT_BACK_BTN_ENABLED);
00257     lv_obj_set_style_bg_color(menu, lv_color_make(0, 104, 179), 0);
00258     lv_obj_set_size(menu, lv_disp_get_hor_res(NULL), lv_disp_get_ver_res(NULL));
00259     lv_obj_center(menu);
00260
00261     // Create a root page
00262     lv_obj_t* root_page{lv_menu_page_create(menu, NULL)};
00263     lv_obj_set_style_pad_hor(root_page, lv_obj_get_style_pad_left(lv_menu_get_main_header(menu), 0),
00264     0);
00265     lv_obj_t* section{lv_menu_section_create(root_page)};
00266     lv_menu_set_sidebar_page(menu, root_page);
00267
00268     // Create the back button
00269     lv_obj_t* back_btn{lv_menu_get_sidebar_header_back_btn(menu)};
00270     lv_obj_remove_style_all(back_btn);
00271     lv_obj_add_style(back_btn, &button_default_style, 0);
00272     lv_obj_add_style(back_btn, &button_pressed_style, LV_STATE_PRESSED);
00273     lv_obj_set_style_text_font(back_btn, &lv_font_montserrat_14, 0);
00274     lv_obj_set_style_text_font(back_btn, &lv_font_montserrat_14, LV_STATE_PRESSED);
00275     lv_obj_set_style_pad_all(back_btn, 3, 0);
00276     lv_obj_set_style_pad_all(back_btn, 3, LV_STATE_PRESSED);
00277     lv_obj_set_style_translate_y(back_btn, 0, LV_STATE_PRESSED);
00278     lv_obj_set_style_shadow_ofs_y(back_btn, 0, LV_STATE_PRESSED);
00279     lv_obj_t* back_btn_label{lv_label_create(back_btn)};
00280     lv_label_set_text(back_btn_label, " Back");
00281     static void* back_user_data[] { nullptr, nullptr };
00282     back_user_data[0] = menu;
00283     back_user_data[1] = this;
00284     lv_obj_add_event_cb(menu, settingsBackButtonEventHandler, LV_EVENT_CLICKED, back_user_data);
00285
00286     static std::vector<std::shared_ptr<std::vector<const char*>>> option_button_matrix_maps{};
00287     static std::vector<void*> option_button_matrix_user_data{};

```

```

00287     for (Option& option : options)
00288     {
00289         lv_obj_t* option_page(lv_menu_page_create(menu, NULL));
00290         lv_obj_set_style_pad_hor(option_page, lv_obj_get_style_pad_left(lv_menu_get_main_header(menu),
00291         0), 0);
00292         lv_menu_separator_create(option_page);
00293         std::shared_ptr<std::vector<const char*>>
00294             option_button_matrix_map{std::make_shared<std::vector<const char*>>()};
00295         uint8_t line_counter{};
00296         for (std::string& choice : option.choices)
00297         {
00298             if (line_counter >= BUTTONS_PER_LINE)
00299             {
00300                 option_button_matrix_map->push_back("\n");
00301                 line_counter = 0;
00302             }
00303             option_button_matrix_map->push_back(choice.c_str());
00304             ++line_counter;
00305         }
00306         option_button_matrix_map->push_back("");
00307         option_button_matrix_maps.push_back(option_button_matrix_map);
00308
00309         lv_obj_t* option_button_matrix{lv_btnmatrix_create(option_page)};
00310         lv_btnmatrix_set_map(option_button_matrix, option_button_matrix_map->data());
00311         lv_btnmatrix_set_one_checked(option_button_matrix, true);
00312         lv_btnmatrix_set_btn_ctrl_all(option_button_matrix, LV_BTNMATRIX_CTRL_CHECKABLE);
00313         lv_btnmatrix_set_btn_ctrl(option_button_matrix, option.selected, LV_BTNMATRIX_CTRL_CHECKED);
00314         lv_obj_add_style(option_button_matrix, &button_matrix_main_style, LV_PART_MAIN);
00315         lv_obj_set_size(option_button_matrix, 300, 220);
00316
00317         void* option_user_data[] { nullptr };
00318         option_user_data[0] = &option;
00319         option_button_matrix_user_data.push_back(option_user_data);
00320         lv_obj_add_event_cb(option_button_matrix, settingsButtonMatrixEventHandler,
00321             LV_EVENT_VALUE_CHANGED, &option);
00322
00323         lv_obj_t* option_menu_container{lv_menu_cont_create(section)};
00324         lv_obj_remove_style_all(option_menu_container);
00325         lv_obj_add_style(option_menu_container, &container_default_style, 0);
00326         lv_obj_add_style(option_menu_container, &container_pressed_style, LV_STATE_CHECKED);
00327         lv_obj_t* option_menu_container_label{lv_label_create(option_menu_container)};
00328         lv_label_set_text(option_menu_container_label, option.name.c_str());
00329         lv_menu_set_load_page_event(menu, option_menu_container, option_page);
00330
00331     }
00332
00333     lv_event_send(lv_obj_get_child(lv_obj_get_child(lv_menu_get_cur_sidebar_page(menu), 0), 0),
00334     LV_EVENT_CLICKED, NULL);
00335 }
```

### 5.54.2.6 setComplete()

```
void wisco::menu::LvglMenu::setComplete ( )
```

Set the menu selection to complete.

Definition at line 333 of file [LvglMenu.cpp](#).

```
00334 {
00335     complete = true;
00336 }
```

### 5.54.2.7 readConfiguration()

```
void wisco::menu::LvglMenu::readConfiguration ( )
```

Reads the configuration of the menu.

Definition at line 338 of file [LvglMenu.cpp](#).

```
00339 {
00340     std::ifstream configuration_file{CONFIGURATION_FILE};
00341     if (configuration_file.fail())
00342         return;
00343
00344     std::string option_name{};
```

```

00345     while (configuration_file >> option_name)
00346     {
00347         std::string option_selection{};
00348         if (configuration_file >> option_selection)
00349         {
00350             for (Option& option : options)
00351             {
00352                 if (option_name == option.name)
00353                 {
00354                     for (uint8_t i{0}; i < option.choices.size(); ++i)
00355                     {
00356                         if (option_selection == option.choices[i])
00357                         {
00358                             option.selected = i;
00359                         }
00360                     }
00361                 }
00362             }
00363         }
00364     }
00365     configuration_file.close();
00366 }
```

### 5.54.2.8 writeConfiguration()

```
void wisco::menu::LvglMenu::writeConfiguration ( )
```

Writes the configuration of the menu.

Definition at line 369 of file [LvglMenu.cpp](#).

```

00370 {
00371     std::ofstream configuration_file{CONFIGURATION_FILE};
00372     if (configuration_file.fail())
00373         return;
00374
00375     for (Option option : options)
00376         configuration_file << option.name << ' ' << option.choices[option.selected] << std::endl;
00377
00378     configuration_file.close();
00379 }
```

### 5.54.2.9 displayMenu()

```
void wisco::menu::LvglMenu::displayMenu ( )
```

Displays the menu.

Definition at line 381 of file [LvglMenu.cpp](#).

```

00382 {
00383     initializeStyles();
00384     readConfiguration();
00385     drawMainMenu();
00386 }
```

### 5.54.2.10 selectionComplete()

```
bool wisco::menu::LvglMenu::selectionComplete ( )
```

Checks if the selection process is complete.

Returns

- true The selection process is complete
- false The selection process is not complete

Definition at line 388 of file [LvglMenu.cpp](#).

```

00389 {
00390     return complete;
00391 }
```

### 5.54.2.11 getSelection()

```
std::string wisco::menu::LvglMenu::getSelection (
    const std::string & option_name )
```

Get the selection for an option.

#### Parameters

<i>option_name</i>	The name of the option
--------------------	------------------------

#### Returns

std::string The selection for that option

Definition at line 393 of file [LvglMenu.cpp](#).

```
00394 {
00395     std::string selection{};
00396     for (Option& option : options)
00397         if (option_name == option.name)
00398             selection = option.choices[option.selected];
00399     return selection;
00400 }
```

## 5.54.3 Member Data Documentation

### 5.54.3.1 CONFIGURATION\_FILE

```
constexpr char wisco::menu::LvglMenu::CONFIGURATION_FILE[] {"/usd/system/menu_data.txt"} [static],
[constexpr], [private]
```

The path of the file that stores the configuration.

Definition at line 67 of file [LvglMenu.hpp](#).

```
00067 {"/usd/system/menu_data.txt"};
```

### 5.54.3.2 COLUMN\_WIDTH

```
constexpr int wisco::menu::LvglMenu::COLUMN_WIDTH {16} [static], [constexpr], [private]
```

The width of a column on the status page.

Definition at line 73 of file [LvglMenu.hpp](#).

```
00073 {16};
```

### 5.54.3.3 BUTTONS\_PER\_LINE

```
constexpr int wisco::menu::LvglMenu::BUTTONS_PER_LINE {2} [static], [constexpr], [private]
```

The number of buttons to display on a line.

Definition at line 79 of file [LvglMenu.hpp](#).

```
00079 {2};
```

#### 5.54.3.4 button\_default\_style

```
lv_style_t wisco::menu::LvglMenu::button_default_style [static], [private]
```

The default style for a button.

Definition at line 85 of file [LvglMenu.hpp](#).

#### 5.54.3.5 button\_pressed\_style

```
lv_style_t wisco::menu::LvglMenu::button_pressed_style [static], [private]
```

The pressed style for a button.

Definition at line 91 of file [LvglMenu.hpp](#).

#### 5.54.3.6 container\_default\_style

```
lv_style_t wisco::menu::LvglMenu::container_default_style [static], [private]
```

The default style for a container.

Definition at line 97 of file [LvglMenu.hpp](#).

#### 5.54.3.7 container\_pressed\_style

```
lv_style_t wisco::menu::LvglMenu::container_pressed_style [static], [private]
```

The pressed style for a container.

Definition at line 103 of file [LvglMenu.hpp](#).

#### 5.54.3.8 button\_matrix\_main\_style

```
lv_style_t wisco::menu::LvglMenu::button_matrix_main_style [static], [private]
```

The background style for a button matrix.

Definition at line 109 of file [LvglMenu.hpp](#).

#### 5.54.3.9 button\_matrix\_items\_style

```
lv_style_t wisco::menu::LvglMenu::button_matrix_items_style [static], [private]
```

The button style for a button matrix.

Definition at line 115 of file [LvglMenu.hpp](#).

### 5.54.3.10 styles\_initialized

```
bool wisco::menu::LvglMenu::styles_initialized = false [static], [private]
```

Whether or not the styles have been initialized.

Definition at line 121 of file [LvglMenu.hpp](#).

### 5.54.3.11 options

```
std::vector<Option> wisco::menu::LvglMenu::options {} [private]
```

The options available in the menu.

Definition at line 127 of file [LvglMenu.hpp](#).  
00127 {};

### 5.54.3.12 complete

```
bool wisco::menu::LvglMenu::complete {false} [private]
```

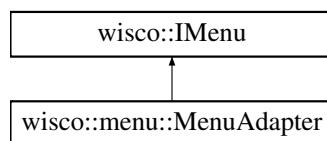
Whether or not selection is complete.

Definition at line 133 of file [LvglMenu.hpp](#).  
00133 {false};

## 5.55 wisco::menu::MenuAdapter Class Reference

This class adapts the menu system to the [IMenu](#) interface.

Inheritance diagram for wisco::menu::MenuAdapter:



### Public Member Functions

- void [addAlliance](#) (std::unique\_ptr<[IAlliance](#)> &alliance) override  
*Adds an alliance to the menu system.*
- void [addAutonomous](#) (std::unique\_ptr<[IAutonomous](#)> &autonomous) override  
*Adds an autonomous routine to the menu system.*
- void [addConfiguration](#) (std::unique\_ptr<[IConfiguration](#)> &configuration) override  
*Adds a hardware configuration to the menu system.*
- void [addProfile](#) (std::unique\_ptr<[IProfile](#)> &profile) override  
*Adds a driver profile to the menu system.*
- void [display](#) () override  
*Displays the menu.*
- bool [isStarted](#) () override  
*Checks if the system is started.*
- [SystemConfiguration getSystemConfiguration](#) () override  
*Get the System Configuration settings.*

## Public Member Functions inherited from [wisco::IMenu](#)

- virtual ~[IMenu](#) ()=default

*Destroy the [IMenu](#) object.*

## Private Attributes

- std::vector< std::unique\_ptr< [IAlliance](#) > > [alliances](#) {}  
*The alliances available in the menu system.*
- std::vector< std::unique\_ptr< [IAutonomous](#) > > [autonomous\\_routines](#) {}  
*The autonomous routines available in the menu system.*
- std::vector< std::unique\_ptr<  [IConfiguration](#) > > [hardware\\_configurations](#) {}  
*The hardware configurations available in the menu system.*
- std::vector< std::unique\_ptr<  [IProfile](#) > > [driver\\_profiles](#) {}  
*The driver profiles available in the menu system.*
- [LvglMenu lvgl\\_menu](#) {}  
*The lvgl menu being adapted.*

## Static Private Attributes

- static constexpr char [ALLIANCE\\_OPTION\\_NAME](#) [] {"ALLIANCE"}  
*The alliance option name.*
- static constexpr char [AUTONOMOUS\\_OPTION\\_NAME](#) [] {"AUTON"}  
*The autonomous option name.*
- static constexpr char [CONFIGURATION\\_OPTION\\_NAME](#) [] {"CONFIG"}  
*The configuration option name.*
- static constexpr char [PROFILE\\_OPTION\\_NAME](#) [] {"PROFILE"}  
*The profile option name.*

### 5.55.1 Detailed Description

This class adapts the menu system to the [IMenu](#) interface.

#### Author

Nathan Sandvig

Definition at line 29 of file [MenuAdapter.hpp](#).

### 5.55.2 Member Function Documentation

#### 5.55.2.1 [addAlliance\(\)](#)

```
void wisco::menu::MenuAdapter::addAlliance (
    std::unique_ptr< IAlliance > & alliance ) [override], [virtual]
```

Adds an alliance to the menu system.

**Parameters**

<i>alliance</i>	The new alliance
-----------------	------------------

Implements [wisco::IMenu](#).

Definition at line 8 of file [MenuAdapter.cpp](#).

```
00009 {
00010     bool unique{true};
00011     for (std::unique_ptr<IAlliance>& existing_alliance : alliances)
00012         if (existing_alliance->getName() == alliance->getName())
00013             unique = false;
00014     if (unique)
00015         alliances.push_back(std::move(alliance));
00016 }
```

**5.55.2.2 addAutonomous()**

```
void wisco::menu::MenuAdapter::addAutonomous (
    std::unique_ptr< IAutonomous > & autonomous ) [override], [virtual]
```

Adds an autonomous routine to the menu system.

**Parameters**

<i>autonomous</i>	The new autonomous routine
-------------------	----------------------------

Implements [wisco::IMenu](#).

Definition at line 18 of file [MenuAdapter.cpp](#).

```
00019 {
00020     bool unique{true};
00021     for (std::unique_ptr<IAutonomous>& existing_autonomous : autonomous_routines)
00022         if (existing_autonomous->getName() == autonomous->getName())
00023             unique = false;
00024     if (unique)
00025         autonomous_routines.push_back(std::move(autonomous));
00026 }
```

**5.55.2.3 addConfiguration()**

```
void wisco::menu::MenuAdapter::addConfiguration (
    std::unique_ptr< IConfiguration > & configuration ) [override], [virtual]
```

Adds a hardware configuration to the menu system.

**Parameters**

<i>configuration</i>	The new hardware configuration
----------------------	--------------------------------

Implements [wisco::IMenu](#).

Definition at line 28 of file [MenuAdapter.cpp](#).

```
00029 {
00030     bool unique{true};
00031     for (std::unique_ptr< IConfiguration >& existing_configuration : hardware_configurations)
00032         if (existing_configuration->getName() == configuration->getName())
```

```

00033     unique = false;
00034     if (unique)
00035         hardware_configurations.push_back(std::move(configuration));
00036 }
```

#### 5.55.2.4 addProfile()

```
void wisco::menu::MenuAdapter::addProfile (
    std::unique_ptr< IProfile > & profile ) [override], [virtual]
```

Adds a driver profile to the menu system.

##### Parameters

<i>profile</i>	The new driver profile
----------------	------------------------

Implements [wisco::IMenu](#).

Definition at line 38 of file [MenuAdapter.cpp](#).

```

00039 {
00040     bool unique{true};
00041     for (std::unique_ptr<IProfile>& existing_profile : driver_profiles)
00042         if (existing_profile->getName() == profile->getName())
00043             unique = false;
00044     if (unique)
00045         driver_profiles.push_back(std::move(profile));
00046 }
```

#### 5.55.2.5 display()

```
void wisco::menu::MenuAdapter::display () [override], [virtual]
```

Displays the menu.

Implements [wisco::IMenu](#).

Definition at line 48 of file [MenuAdapter.cpp](#).

```

00049 {
00050     std::vector<std::string> alliance_options{};
00051     for (std::unique_ptr<IAlliance>& alliance : alliances)
00052         alliance_options.push_back(alliance->getName());
00053     Option alliance_option{ALLIANCE_OPTION_NAME, alliance_options};
00054
00055     std::vector<std::string> autonomous_options{};
00056     for (std::unique_ptr<IAutonomous>& autonomous : autonomous_routines)
00057         autonomous_options.push_back(autonomous->getName());
00058     Option autonomous_option{AUTONOMOUS_OPTION_NAME, autonomous_options};
00059
00060     std::vector<std::string> configuration_options{};
00061     for (std::unique_ptr< IConfiguration>& configuration : hardware_configurations)
00062         configuration_options.push_back(configuration->getName());
00063     Option configuration_option{CONFIGURATION_OPTION_NAME, configuration_options};
00064
00065     std::vector<std::string> profile_options{};
00066     for (std::unique_ptr< IProfile >& profile : driver_profiles)
00067         profile_options.push_back(profile->getName());
00068     Option profile_option{PROFILE_OPTION_NAME, profile_options};
00069
00070     lvgl_menu.addOption(alliance_option);
00071     lvgl_menu.addOption(autonomous_option);
00072     lvgl_menu.addOption(configuration_option);
00073     lvgl_menu.addOption(profile_option);
00074
00075     lvgl_menu.displayMenu();
00076 }
```

### 5.55.2.6 isStarted()

```
bool wisco::menu::MenuAdapter::isStarted ( ) [override], [virtual]
```

Checks if the system is started.

#### Returns

- true The system is started
- false The system is not started

Implements [wisco::IMenu](#).

Definition at line 78 of file [MenuAdapter.cpp](#).

```
00079 {
00080     return lvgl_menu.selectionComplete();
00081 }
```

### 5.55.2.7 getSystemConfiguration()

```
SystemConfiguration wisco::menu::MenuAdapter::getSystemConfiguration ( ) [override], [virtual]
```

Get the System Configuration settings.

#### Returns

- [SystemConfiguration](#) The system configuration settings

Implements [wisco::IMenu](#).

Definition at line 83 of file [MenuAdapter.cpp](#).

```
00084 {
00085     SystemConfiguration system_configuration{};
00086
00087     for (std::unique_ptr<IAlliance>& alliance : alliances)
00088     {
00089         if (lvgl_menu.getSelection(ALLIANCE_OPTION_NAME) == alliance->getName())
00090         {
00091             system_configuration.alliance = std::move(alliance);
00092             break;
00093         }
00094     }
00095
00096     for (std::unique_ptr<IAutonomous>& autonomous : autonomous_routines)
00097     {
00098         if (lvgl_menu.getSelection(AUTONOMOUS_OPTION_NAME) == autonomous->getName())
00099         {
00100             system_configuration.autonomous = std::move(autonomous);
00101             break;
00102         }
00103     }
00104
00105     for (std::unique_ptr<IConfiguration>& configuration : hardware_configurations)
00106     {
00107         if (lvgl_menu.getSelection(CONFIGURATION_OPTION_NAME) == configuration->getName())
00108         {
00109             system_configuration.configuration = std::move(configuration);
00110             break;
00111         }
00112     }
00113
00114     for (std::unique_ptr<IProfile>& profile : driver_profiles)
00115     {
00116         if (lvgl_menu.getSelection(PROFILE_OPTION_NAME) == profile->getName())
00117         {
00118             system_configuration.profile = std::move(profile);
00119             break;
00120         }
00121     }
00122
00123     return system_configuration;
00124 }
```

### 5.55.3 Member Data Documentation

#### 5.55.3.1 ALLIANCE\_OPTION\_NAME

```
constexpr char wisco::menu::MenuAdapter::ALLIANCE_OPTION_NAME[] {"ALLIANCE"} [static], [constexpr],  
[private]
```

The alliance option name.

Definition at line 36 of file [MenuAdapter.hpp](#).  
00036 {"ALLIANCE"};

#### 5.55.3.2 AUTONOMOUS\_OPTION\_NAME

```
constexpr char wisco::menu::MenuAdapter::AUTONOMOUS_OPTION_NAME[] {"AUTON"} [static], [constexpr],  
[private]
```

The autonomous option name.

Definition at line 42 of file [MenuAdapter.hpp](#).  
00042 {"AUTON"};

#### 5.55.3.3 CONFIGURATION\_OPTION\_NAME

```
constexpr char wisco::menu::MenuAdapter::CONFIGURATION_OPTION_NAME[] {"CONFIG"} [static],  
[constexpr], [private]
```

The configuration option name.

Definition at line 48 of file [MenuAdapter.hpp](#).  
00048 {"CONFIG"};

#### 5.55.3.4 PROFILE\_OPTION\_NAME

```
constexpr char wisco::menu::MenuAdapter::PROFILE_OPTION_NAME[] {"PROFILE"} [static], [constexpr],  
[private]
```

The profile option name.

Definition at line 54 of file [MenuAdapter.hpp](#).  
00054 {"PROFILE"};

#### 5.55.3.5 alliances

```
std::vector<std::unique_ptr<IAlliance>> wisco::menu::MenuAdapter::alliances {} [private]
```

The alliances available in the menu system.

Definition at line 60 of file [MenuAdapter.hpp](#).  
00060 {};

### 5.55.3.6 autonomous\_routines

```
std::vector<std::unique_ptr<IAutonomous>> wisco::menu::MenuAdapter::autonomous_routines {}  
[private]
```

The autonomous routines available in the menu system.

Definition at line 66 of file [MenuAdapter.hpp](#).  
00066 {};

### 5.55.3.7 hardware\_configurations

```
std::vector<std::unique_ptr<IConfiguration>> wisco::menu::MenuAdapter::hardware_configurations {} [private]
```

The hardware configurations available in the menu system.

Definition at line 72 of file [MenuAdapter.hpp](#).  
00072 {};

### 5.55.3.8 driver\_profiles

```
std::vector<std::unique_ptr<IProfile>> wisco::menu::MenuAdapter::driver_profiles {} [private]
```

The driver profiles available in the menu system.

Definition at line 78 of file [MenuAdapter.hpp](#).  
00078 {};

### 5.55.3.9 lvgl\_menu

```
LvglMenu wisco::menu::MenuAdapter::lvgl_menu {} [private]
```

The lvgl menu being adapted.

Definition at line 84 of file [MenuAdapter.hpp](#).  
00084 {};

## 5.56 wisco::menu::Option Struct Reference

An option in the menu system.

### Public Attributes

- std::string **name** {}  
*The name of the option.*
- std::vector< std::string > **choices** {}  
*The choices available for that option.*
- int **selected** {}  
*The index of the selected choice.*

### 5.56.1 Detailed Description

An option in the menu system.

#### Author

Nathan Sandvig

Definition at line [26](#) of file [Option.hpp](#).

### 5.56.2 Member Data Documentation

#### 5.56.2.1 name

```
std::string wisco::menu::Option::name {}
```

The name of the option.

Definition at line [32](#) of file [Option.hpp](#).  
00032 {};

#### 5.56.2.2 choices

```
std::vector<std::string> wisco::menu::Option::choices {}
```

The choices available for that option.

Definition at line [38](#) of file [Option.hpp](#).  
00038 {};

#### 5.56.2.3 selected

```
int wisco::menu::Option::selected {}
```

The index of the selected choice.

Definition at line [44](#) of file [Option.hpp](#).  
00044 {};

## 5.57 wisco::OPControlManager Class Reference

Manages the execution of the operator control.

## Public Member Functions

- `OPControlManager` (const std::shared\_ptr< `rtos::IClock` > &`clock`, const std::unique\_ptr< `rtos::IDelay` > &`delayer`)  
*Construct a new `OPControlManager` object.*
- `void setProfile` (std::unique\_ptr< `IProfile` > &`profile`)  
*Set the operator profile.*
- `void initializeOpcontrol` (std::shared\_ptr< `control::ControlSystem` > `control_system`, std::shared\_ptr< `user::IController` > `controller`, std::shared\_ptr< `robot::Robot` > `robot`)  
*Initialize the operator control.*
- `void runOpcontrol` (std::shared\_ptr< `control::ControlSystem` > `control_system`, std::shared\_ptr< `user::IController` > `controller`, std::shared\_ptr< `robot::Robot` > `robot`)  
*Run the operator control.*

## Private Attributes

- `std::shared_ptr< rtos::IClock > m_clock {}`  
*The rtos clock for the control loop.*
- `std::unique_ptr< rtos::IDelay > m_delay {}`  
*The rtos delay for the control loop.*
- `std::unique_ptr< IProfile > m_profile {}`  
*The driver profile.*

## Static Private Attributes

- `static constexpr uint32_t CONTROL_DELAY {10}`  
*The loop delay for control inputs.*

### 5.57.1 Detailed Description

Manages the execution of the operator control.

#### Author

Nathan Sandvig

Definition at line 34 of file [OpcontrolManager.hpp](#).

### 5.57.2 Constructor & Destructor Documentation

#### 5.57.2.1 OPControlManager()

```
wisco::OPControlManager::OPControlManager (
    const std::shared_ptr< rtos::IClock > & clock,
    const std::unique_ptr< rtos::IDelay > & delayer )
```

Construct a new `OPControlManager` object.

**Parameters**

<i>clock</i>	The rtos clock
<i>delayer</i>	The rtos delayer

Definition at line 6 of file [OPControlManager.cpp](#).

```
00007     : m_clock{clock}, m_delayer{delayer->clone()}
00008 {
00009
00010 }
```

### 5.57.3 Member Function Documentation

#### 5.57.3.1 setProfile()

```
void wisco::OPControlManager::setProfile (
    std::unique_ptr< IProfile > & profile )
```

Set the operator profile.

**Parameters**

<i>profile</i>	The operator profile
----------------	----------------------

Definition at line 12 of file [OPControlManager.cpp](#).

```
00013 {
00014     m_profile = std::move(profile);
00015 }
```

#### 5.57.3.2 initializeOpcontrol()

```
void wisco::OPControlManager::initializeOpcontrol (
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< user::IController > controller,
    std::shared_ptr< robot::Robot > robot )
```

Initialize the operator control.

**Parameters**

<i>control_system</i>	The control system
<i>controller</i>	The controller for the robot
<i>robot</i>	The robot being controlled

Definition at line 17 of file [OPControlManager.cpp](#).

```
00020 {
00021
00022 }
```

#### 5.57.3.3 runOpcontrol()

```
void wisco::OPControlManager::runOpcontrol (
```

```
    std::shared_ptr< control::ControlSystem > control_system,
    std::shared_ptr< user::IController > controller,
    std::shared_ptr< robot::Robot > robot )
```

Run the operator control.

#### Parameters

<i>control_system</i>	
<i>controller</i>	The controller for the robot
<i>robot</i>	The robot being controlled

Definition at line 24 of file [OPControlManager.cpp](#).

```
00027 {
00028     control_system->sendCommand("BOOMERANG", "PAUSE");
00029
00030     user::drive::DifferentialDriveOperator drive_operator{controller, robot};
00031     user::elevator::ElevatorOperator elevator_operator{controller, robot};
00032     user::hang::HangOperator hang_operator{controller, robot};
00033     user::intake::IntakeOperator intake_operator{controller, robot};
00034     user::loader::LoaderOperator loader_operator{controller, robot};
00035     user::umbrella::UmbrellaOperator umbrella_operator{controller, robot};
00036     user::wings::WingsOperator wings_operator{controller, robot};
00037     uint32_t current_time{};
00038     while (true)
00039     {
00040         current_time = m_clock->getTime();
00041
00042         drive_operator.setDriveVoltage(static_cast<user::drive::EChassisControlMode>(m_profile->getControlMode(user::EControlType));
00043         elevator_operator.setElevatorPosition(m_profile);
00044         hang_operator.setHangState(m_profile);
00045         intake_operator.setIntakeVoltage(m_profile);
00046         loader_operator.setLoaderPosition(m_profile);
00047         umbrella_operator.setUmbrellaPosition(m_profile);
00048         wings_operator.setWingsPosition(m_profile);
00049
00050         m_delayer->delayUntil(current_time + CONTROL_DELAY);
00051     }
00052 }
```

## 5.57.4 Member Data Documentation

### 5.57.4.1 CONTROL\_DELAY

```
constexpr uint32_t wisco::OPControlManager::CONTROL_DELAY {10} [static], [constexpr], [private]
```

The loop delay for control inputs.

Definition at line 41 of file [OpcontrolManager.hpp](#).

```
00041 {10};
```

### 5.57.4.2 m\_clock

```
std::shared_ptr<rtos::IClock> wisco::OPControlManager::m_clock {} [private]
```

The rtos clock for the control loop.

Definition at line 47 of file [OpcontrolManager.hpp](#).

```
00047 {};
```

### 5.57.4.3 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::OPControlManager::m_delayer {} [private]
```

The rtos delayer for the control loop.

Definition at line 53 of file [OpcontrolManager.hpp](#).

```
00053 {};
```

### 5.57.4.4 m\_profile

```
std::unique_ptr<IPProfile> wisco::OPControlManager::m_profile {} [private]
```

The driver profile.

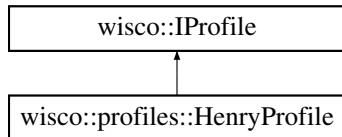
Definition at line 59 of file [OpcontrolManager.hpp](#).

```
00059 {};
```

## 5.58 wisco::profiles::HenryProfile Class Reference

Driver profile for Henry.

Inheritance diagram for wisco::profiles::HenryProfile:



### Public Member Functions

- std::string [getName \(\)](#) override  
*Get the name of the profile.*
- int [getControlMode \(user::EControlType control\\_type\) const](#) override  
*Get the control mode for a specific control type.*
- void [setControlMode \(user::EControlType control\\_type, int control\\_mode\) override](#)  
*Set the control mode for a specific control type.*
- user::EControllerAnalog [getAnalogControlMapping \(user::EControl control\) const](#) override  
*Get the mapping of a control to analog inputs.*
- user::EControllerDigital [getDigitalControlMapping \(user::EControl control\) const](#) override  
*Get the mapping of a control to digital inputs.*

### Public Member Functions inherited from [wisco::IPProfile](#)

- virtual ~[IPProfile \(\)=default](#)  
*Destroy the [IPProfile](#) object.*

### Private Attributes

- std::map< user::EControlType, int > **CONTROL\_MODE\_MAP**  
*The control modes for the profile.*
- const std::map< user::EControl, user::EControllerAnalog > **ANALOG\_CONTROL\_MAP** {}  
*The mapping of the controls to the analog inputs.*
- const std::map< user::EControl, user::EControllerDigital > **DIGITAL\_CONTROL\_MAP**  
*The mapping of the controls to the digital inputs.*

### Static Private Attributes

- static constexpr char **PROFILE\_NAME** [] {"HENRY"}  
*The name of the profile.*

## 5.58.1 Detailed Description

Driver profile for Henry.

### Author

Nathan Sandvig

Definition at line 34 of file [HenryProfile.hpp](#).

## 5.58.2 Member Function Documentation

### 5.58.2.1 getName()

```
std::string wisco::profiles::HenryProfile::getName ( ) [override], [virtual]
```

Get the name of the profile.

### Returns

std::string The name of the profile

Implements [wisco::IProfile](#).

Definition at line 7 of file [HenryProfile.cpp](#).

```
00008 {  
00009     return PROFILE_NAME;  
00010 }
```

### 5.58.2.2 getControlMode()

```
int wisco::profiles::HenryProfile::getControlMode (  
    user::EControlType control_type ) const [override], [virtual]
```

Get the control mode for a specific control type.

**Parameters**

<i>control_type</i>	The control type
---------------------	------------------

**Returns**

int The control mode

Implements [wisco::IProfile](#).

Definition at line 12 of file [HenryProfile.cpp](#).

```
00013 {
00014     int mode{};
00015     if (CONTROL_MODE_MAP.contains(control_type))
00016         mode = CONTROL_MODE_MAP.at(control_type);
00017     return mode;
00018 }
```

**5.58.2.3 setControlMode()**

```
void wisco::profiles::HenryProfile::setControlMode (
    user::EControlType control_type,
    int control_mode ) [override], [virtual]
```

Set the control mode for a specific control type.

**Parameters**

<i>control_type</i>	The control type
<i>control_mode</i>	The control mode

Implements [wisco::IProfile](#).

Definition at line 20 of file [HenryProfile.cpp](#).

```
00021 {
00022     CONTROL_MODE_MAP[control_type] = control_mode;
00023 }
```

**5.58.2.4 getAnalogControlMapping()**

```
user::EControllerAnalog wisco::profiles::HenryProfile::getAnalogControlMapping (
    user::EControl control ) const [override], [virtual]
```

Get the mapping of a control to analog inputs.

**Parameters**

<i>control</i>	The control
----------------	-------------

**Returns**

[user::EControllerAnalog](#) The mapping of this control to a analog input

Implements [wisco::IProfile](#).

Definition at line 25 of file [HenryProfile.cpp](#).

```
00026 {  
00027     user::EControllerAnalog analog{user::EControllerAnalog::NONE};  
00028     if (ANALOG_CONTROL_MAP.contains(control))  
00029         analog = ANALOG_CONTROL_MAP.at(control);  
00030     return analog;  
00031 }
```

### 5.58.2.5 `getDigitalControlMapping()`

```
user::EControllerDigital wisco::profiles::HenryProfile::getDigitalControlMapping (   
    user::EControl control ) const [override], [virtual]
```

Get the mapping of a control to digital inputs.

#### Parameters

<i>control</i>	The control
----------------	-------------

#### Returns

[user::EControllerDigital](#) The mapping of this control to a digital input

Implements [wisco::IProfile](#).

Definition at line 33 of file [HenryProfile.cpp](#).

```
00034 {  
00035     user::EControllerDigital digital{user::EControllerDigital::NONE};  
00036     if (DIGITAL_CONTROL_MAP.contains(control))  
00037         digital = DIGITAL_CONTROL_MAP.at(control);  
00038     return digital;  
00039 }
```

## 5.58.3 Member Data Documentation

### 5.58.3.1 PROFILE\_NAME

```
constexpr char wisco::profiles::HenryProfile::PROFILE_NAME[ ] {"HENRY"} [static], [constexpr],  
[private]
```

The name of the profile.

Definition at line 41 of file [HenryProfile.hpp](#).

```
00041 {"HENRY"};
```

### 5.58.3.2 CONTROL\_MODE\_MAP

```
std::map<user::EControlType, int> wisco::profiles::HenryProfile::CONTROL_MODE_MAP [private]
```

**Initial value:**

```
{
    {user::EControlType::DRIVE, static_cast<int>(user::drive::EChassisControlMode::SPLIT_ARCADE_LEFT)},
    {user::EControlType::ELEVATOR,
     static_cast<int>(user::elevator::EElevatorControlMode::PRESET_TOGGLE_LADDER_INTAKE)},
    {user::EControlType::HANG, static_cast<int>(user::hang::EHangControlMode::PRESET_TOGGLE_LADDER)},
    {user::EControlType::INTAKE, static_cast<int>(user::intake::EIntakeControlMode::SPLIT_HOLD)},
    {user::EControlType::LOADER, static_cast<int>(user::loader::ELoaderControlMode::SINGLE_TOGGLE)},
    {user::EControlType::UMBRELLA,
     static_cast<int>(user::umbrella::EUmbrellaControlMode::SINGLE_TOGGLE)},
    {user::EControlType::WINGS, static_cast<int>(user::wings::EWingsControlMode::DUAL_TOGGLE)}
}
```

The control modes for the profile.

Definition at line 47 of file [HenryProfile.hpp](#).

```
00048 {
00049     {user::EControlType::DRIVE,
00050      static_cast<int>(user::drive::EChassisControlMode::SPLIT_ARCADE_LEFT)},
00051     {user::EControlType::ELEVATOR,
00052      static_cast<int>(user::elevator::EElevatorControlMode::PRESET_TOGGLE_LADDER_INTAKE)},
00053     {user::EControlType::HANG,
00054      static_cast<int>(user::hang::EHangControlMode::PRESET_TOGGLE_LADDER)},
00055     {user::EControlType::INTAKE, static_cast<int>(user::intake::EIntakeControlMode::SPLIT_HOLD)},
00056     {user::EControlType::LOADER,
00057      static_cast<int>(user::loader::ELoaderControlMode::SINGLE_TOGGLE)},
00058     {user::EControlType::UMBRELLA,
00059      static_cast<int>(user::umbrella::EUmbrellaControlMode::SINGLE_TOGGLE)},
00060     {user::EControlType::WINGS, static_cast<int>(user::wings::EWingsControlMode::DUAL_TOGGLE)}
00061 };
```

### 5.58.3.3 ANALOG\_CONTROL\_MAP

```
const std::map<user::EControl, user::EControllerAnalog> wisco::profiles::HenryProfile::←
ANALOG_CONTROL_MAP {} [private]
```

The mapping of the controls to the analog inputs.

Definition at line 62 of file [HenryProfile.hpp](#).

```
00062 {};
```

### 5.58.3.4 DIGITAL\_CONTROL\_MAP

```
const std::map<user::EControl, user::EControllerDigital> wisco::profiles::HenryProfile::←
DIGITAL_CONTROL_MAP [private]
```

**Initial value:**

```
{
    {user::EControl::ELEVATOR_IN, user::EControllerDigital::TRIGGER_RIGHT_BOTTOM},
    {user::EControl::ELEVATOR_OUT, user::EControllerDigital::TRIGGER_RIGHT_TOP},
    {user::EControl::HANG_NEXT, user::EControllerDigital::BUTTON_A},
    {user::EControl::HANG_PREVIOUS, user::EControllerDigital::BUTTON_B},
    {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
    {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM},
    {user::EControl::LOADER_TOGGLE, user::EControllerDigital::DPAD_LEFT},
    {user::EControl::UMBRELLA_TOGGLE, user::EControllerDigital::DPAD_DOWN},
    {user::EControl::WINGS_TOGGLE, user::EControllerDigital::SCUFF_LEFT_REAR}
}
```

The mapping of the controls to the digital inputs.

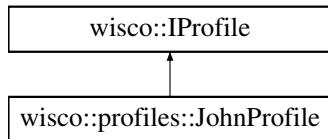
Definition at line 68 of file [HenryProfile.hpp](#).

```
00069 {
00070     {user::EControl::ELEVATOR_IN, user::EControllerDigital::TRIGGER_RIGHT_BOTTOM},
00071     {user::EControl::ELEVATOR_OUT, user::EControllerDigital::TRIGGER_RIGHT_TOP},
00072     {user::EControl::HANG_NEXT, user::EControllerDigital::BUTTON_A},
00073     {user::EControl::HANG_PREVIOUS, user::EControllerDigital::BUTTON_B},
00074     {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
00075     {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM},
00076     {user::EControl::LOADER_TOGGLE, user::EControllerDigital::DPAD_LEFT},
00077     {user::EControl::UMBRELLA_TOGGLE, user::EControllerDigital::DPAD_DOWN},
00078     {user::EControl::WINGS_TOGGLE, user::EControllerDigital::SCUFF_LEFT_REAR}
00079 };
```

## 5.59 wisco::profiles::JohnProfile Class Reference

Driver profile for John.

Inheritance diagram for wisco::profiles::JohnProfile:



### Public Member Functions

- std::string [getName \(\) override](#)  
*Get the name of the profile.*
- int [getControlMode \(user::EControlType control\\_type\) const override](#)  
*Get the control mode for a specific control type.*
- void [setControlMode \(user::EControlType control\\_type, int control\\_mode\) override](#)  
*Set the control mode for a specific control type.*
- user::EControllerAnalog [getAnalogControlMapping \(user::EControl control\) const override](#)  
*Get the mapping of a control to analog inputs.*
- user::EControllerDigital [getDigitalControlMapping \(user::EControl control\) const override](#)  
*Get the mapping of a control to digital inputs.*

### Public Member Functions inherited from [wisco::IProfile](#)

- virtual ~[IProfile \(\)=default](#)  
*Destroy the [IProfile](#) object.*

### Private Attributes

- std::map< [user::EControlType](#), int > [CONTROL\\_MODE\\_MAP](#)  
*The control modes for the profile.*
- const std::map< [user::EControl](#), [user::EControllerAnalog](#) > [ANALOG\\_CONTROL\\_MAP](#) {}  
*The mapping of the controls to the analog inputs.*
- const std::map< [user::EControl](#), [user::EControllerDigital](#) > [DIGITAL\\_CONTROL\\_MAP](#)  
*The mapping of the controls to the digital inputs.*

### Static Private Attributes

- static constexpr char [PROFILE\\_NAME \[\] {"JOHN"}](#)  
*The name of the profile.*

### 5.59.1 Detailed Description

Driver profile for John.

#### Author

Nathan Sandvig

Definition at line 34 of file [JohnProfile.hpp](#).

### 5.59.2 Member Function Documentation

#### 5.59.2.1 getName()

```
std::string wisco::profiles::JohnProfile::getName ( ) [override], [virtual]
```

Get the name of the profile.

#### Returns

`std::string` The name of the profile

Implements [wisco::IProfile](#).

Definition at line 7 of file [JohnProfile.cpp](#).

```
00008 {  
00009     return PROFILE_NAME;  
00010 }
```

#### 5.59.2.2 getControlMode()

```
int wisco::profiles::JohnProfile::getControlMode (  
    user::EControlType control_type ) const [override], [virtual]
```

Get the control mode for a specific control type.

#### Parameters

<code>control_type</code>	The control type
---------------------------	------------------

#### Returns

`int` The control mode

Implements [wisco::IProfile](#).

Definition at line 13 of file [JohnProfile.cpp](#).

```
00014 {  
00015     int mode{};  
00016     if (CONTROL_MODE_MAP.contains(control_type))  
00017         mode = CONTROL_MODE_MAP.at(control_type);  
00018     return mode;  
00019 }
```

### 5.59.2.3 setControlMode()

```
void wisco::profiles::JohnProfile::setControlMode (
    user::EControlType control_type,
    int control_mode ) [override], [virtual]
```

Set the control mode for a specific control type.

#### Parameters

<i>control_type</i>	The control type
<i>control_mode</i>	The control mode

Implements [wisco::IProfile](#).

Definition at line 21 of file [JohnProfile.cpp](#).

```
00022 {
00023     CONTROL_MODE_MAP[control_type] = control_mode;
00024 }
```

### 5.59.2.4 getAnalogControlMapping()

```
user::EControllerAnalog wisco::profiles::JohnProfile::getAnalogControlMapping (
    user::EControl control ) const [override], [virtual]
```

Get the mapping of a control to analog inputs.

#### Parameters

<i>control</i>	The control
----------------	-------------

#### Returns

[user::EControllerAnalog](#) The mapping of this control to a analog input

Implements [wisco::IProfile](#).

Definition at line 26 of file [JohnProfile.cpp](#).

```
00027 {
00028     user::EControllerAnalog analog{user::EControllerAnalog::NONE};
00029     if (ANALOG_CONTROL_MAP.contains(control))
00030         analog = ANALOG_CONTROL_MAP.at(control);
00031     return analog;
00032 }
```

### 5.59.2.5 getDigitalControlMapping()

```
user::EControllerDigital wisco::profiles::JohnProfile::getDigitalControlMapping (
    user::EControl control ) const [override], [virtual]
```

Get the mapping of a control to digital inputs.

**Parameters**

<i>control</i>	The control
----------------	-------------

**Returns**

`user::EControllerDigital` The mapping of this control to a digital input

Implements [wisco::IProfile](#).

Definition at line 34 of file [JohnProfile.cpp](#).

```
00035 {
00036     user::EControllerDigital digital{user::EControllerDigital::NONE};
00037     if (DIGITAL_CONTROL_MAP.contains(control))
00038         digital = DIGITAL_CONTROL_MAP.at(control);
00039     return digital;
00040 }
```

## 5.59.3 Member Data Documentation

### 5.59.3.1 PROFILE\_NAME

```
constexpr char wisco::profiles::JohnProfile::PROFILE_NAME[] {"JOHN"} [static], [constexpr],
[private]
```

The name of the profile.

Definition at line 41 of file [JohnProfile.hpp](#).

```
00041 {"JOHN"};
```

### 5.59.3.2 CONTROL\_MODE\_MAP

```
std::map<user::EControlType, int> wisco::profiles::JohnProfile::CONTROL_MODE_MAP [private]
```

**Initial value:**

```
{
    {user::EControlType::DRIVE, static_cast<int>(user::drive::EChassisControlMode::TANK)},
    {user::EControlType::ELEVATOR,
        static_cast<int>(user::elevator::EElevatorControlMode::PRESET_TOGGLE_LADDER_INTAKE)},
    {user::EControlType::HANG, static_cast<int>(user::hang::EHangControlMode::PRESET_TOGGLE_LADDER)},
    {user::EControlType::INTAKE, static_cast<int>(user::intake::EIntakeControlMode::SPLIT_HOLD)},
    {user::EControlType::LOADER, static_cast<int>(user::loader::ELoaderControlMode::MACRO)},
    {user::EControlType::UMBRELLA,
        static_cast<int>(user::umbrella::EUmbrellaControlMode::SINGLE_TOGGLE)},
    {user::EControlType::WINGS, static_cast<int>(user::wings::EWingsControlMode::DUAL_HOLD)}
}
```

The control modes for the profile.

Definition at line 47 of file [JohnProfile.hpp](#).

```
00048 {
00049     {user::EControlType::DRIVE, static_cast<int>(user::drive::EChassisControlMode::TANK)},
00050     {user::EControlType::ELEVATOR,
        static_cast<int>(user::elevator::EElevatorControlMode::PRESET_TOGGLE_LADDER_INTAKE)},
00051     {user::EControlType::HANG,
        static_cast<int>(user::hang::EHangControlMode::PRESET_TOGGLE_LADDER)},
00052     {user::EControlType::INTAKE, static_cast<int>(user::intake::EIntakeControlMode::SPLIT_HOLD)},
00053     {user::EControlType::LOADER, static_cast<int>(user::loader::ELoaderControlMode::MACRO)},
00054     {user::EControlType::UMBRELLA,
        static_cast<int>(user::umbrella::EUmbrellaControlMode::SINGLE_TOGGLE)},
00055     {user::EControlType::WINGS, static_cast<int>(user::wings::EWingsControlMode::DUAL_HOLD)}
00056 };
```

### 5.59.3.3 ANALOG\_CONTROL\_MAP

```
const std::map<user::EControl, user::EControllerAnalog> wisco::profiles::JohnProfile::ANALOG←
_ANALOG_CONTROL_MAP {} [private]
```

The mapping of the controls to the analog inputs.

Definition at line 62 of file [JohnProfile.hpp](#).

```
00062 {};
```

### 5.59.3.4 DIGITAL\_CONTROL\_MAP

```
const std::map<user::EControl, user::EControllerDigital> wisco::profiles::JohnProfile::←
DIGITAL_CONTROL_MAP [private]
```

**Initial value:**

```
{
    {user::EControl::ELEVATOR_IN, user::EControllerDigital::TRIGGER_RIGHT_BOTTOM},
    {user::EControl::ELEVATOR_OUT, user::EControllerDigital::TRIGGER_RIGHT_TOP},
    {user::EControl::HANG_NEXT, user::EControllerDigital::BUTTON_A},
    {user::EControl::HANG_PREVIOUS, user::EControllerDigital::BUTTON_B},
    {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
    {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM},
    {user::EControl::LOADER_TOGGLE, user::EControllerDigital::DPAD_LEFT},
    {user::EControl::UMBRELLA_TOGGLE, user::EControllerDigital::DPAD_DOWN},
    {user::EControl::WINGS_HOLD, user::EControllerDigital::SCUFF_RIGHT_REAR}
}
```

The mapping of the controls to the digital inputs.

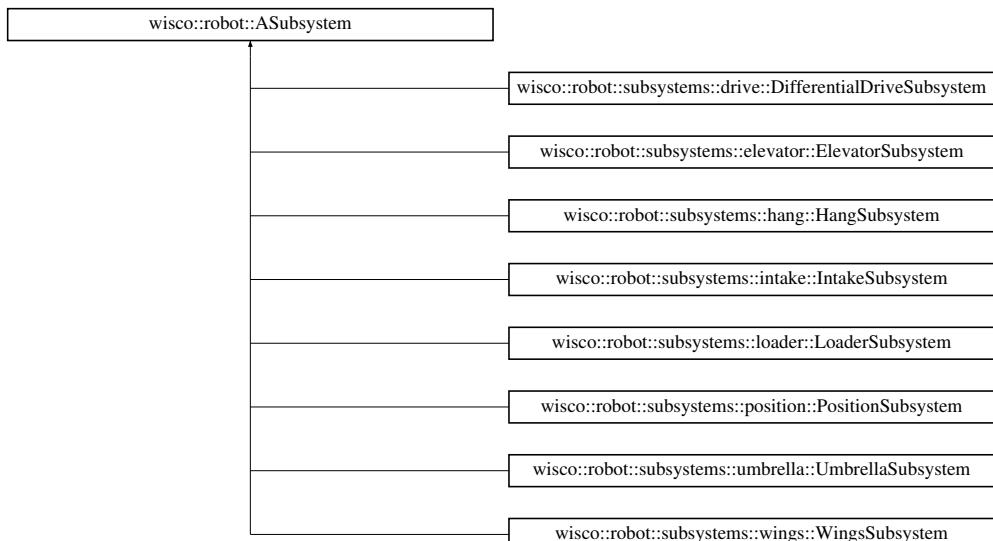
Definition at line 68 of file [JohnProfile.hpp](#).

```
00069 {
00070     {user::EControl::ELEVATOR_IN, user::EControllerDigital::TRIGGER_RIGHT_BOTTOM},
00071     {user::EControl::ELEVATOR_OUT, user::EControllerDigital::TRIGGER_RIGHT_TOP},
00072     {user::EControl::HANG_NEXT, user::EControllerDigital::BUTTON_A},
00073     {user::EControl::HANG_PREVIOUS, user::EControllerDigital::BUTTON_B},
00074     {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
00075     {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM},
00076     {user::EControl::LOADER_TOGGLE, user::EControllerDigital::DPAD_LEFT},
00077     {user::EControl::UMBRELLA_TOGGLE, user::EControllerDigital::DPAD_DOWN},
00078     {user::EControl::WINGS_HOLD, user::EControllerDigital::SCUFF_RIGHT_REAR}
00079};
```

## 5.60 wisco::robot::ASubsystem Class Reference

An abstract class for robot subsystems.

Inheritance diagram for wisco::robot::ASubsystem:



## Public Member Functions

- **ASubsystem ()=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (const ASubsystem &other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (ASubsystem &&other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (std::string name)**  
*Construct a new ASubsystem object.*
- **virtual ~ASubsystem ()=default**  
*Destroy the ASubsystem object.*
- **const std::string & getName () const**  
*Get the name of the subsystem.*
- **virtual void initialize ()=0**  
*Initializes the subsystem.*
- **virtual void run ()=0**  
*Runs the subsystem.*
- **virtual void command (std::string command\_name, va\_list &args)=0**  
*Runs a command for the subsystem.*
- **virtual void \* state (std::string state\_name)=0**  
*Gets a state of the subsystem.*
- **ASubsystem & operator= (const ASubsystem &rhs)=default**  
*Copy assignment operator for ASubsystem.*
- **ASubsystem & operator= (ASubsystem &&rhs)=default**  
*Move assignment operator for ASubsystem.*

## Private Attributes

- **std::string m\_name {}**  
*The name of the subsystem.*

### 5.60.1 Detailed Description

An abstract class for robot subsystems.

#### Author

Nathan Sandvig

Definition at line 28 of file [ASubsystem.hpp](#).

### 5.60.2 Constructor & Destructor Documentation

#### 5.60.2.1 ASubsystem() [1/3]

```
wisco::robot::ASubsystem::ASubsystem (
    const ASubsystem & other ) [default]
```

Construct a new ASubsystem object.

**Parameters**

<i>other</i>	The <a href="#">ASubsystem</a> object being copied
--------------	--

**5.60.2.2 ASubsystem() [2/3]**

```
wisco::robot::ASubsystem::ASubsystem (
    ASubsystem && other )  [default]
```

Construct a new [ASubsystem](#) object.

**Parameters**

<i>other</i>	The <a href="#">ASubsystem</a> object being moved
--------------	---

**5.60.2.3 ASubsystem() [3/3]**

```
wisco::robot::ASubsystem::ASubsystem (
    std::string name )  [inline]
```

Construct a new [ASubsystem](#) object.

**Parameters**

<i>name</i>	The name of the subsystem
-------------	---------------------------

Definition at line 63 of file [ASubsystem.hpp](#).

```
00063 : m_name{name} {}
```

**5.60.3 Member Function Documentation****5.60.3.1 getName()**

```
const std::string & wisco::robot::ASubsystem::getName ( ) const  [inline]
```

Get the name of the subsystem.

**Returns**

```
const std::string& The name of the subsystem
```

Definition at line 76 of file [ASubsystem.hpp](#).

```
00077 {
00078     return m_name;
00079 }
```

### 5.60.3.2 initialize()

```
virtual void wisco::robot::ASubsystem::initialize ( ) [pure virtual]
```

Initializes the subsystem.

Implemented in [wisco::robot::subsystems::drive::DifferentialDriveSubsystem](#), [wisco::robot::subsystems::elevator::ElevatorSubsystem](#), [wisco::robot::subsystems::hang::HangSubsystem](#), [wisco::robot::subsystems::intake::IntakeSubsystem](#), [wisco::robot::subsystems::load::LoadSubsystem](#), [wisco::robot::subsystems::position::PositionSubsystem](#), [wisco::robot::subsystems::umbrella::UmbrellaSubsystem](#), and [wisco::robot::subsystems::wings::WingsSubsystem](#).

### 5.60.3.3 run()

```
virtual void wisco::robot::ASubsystem::run ( ) [pure virtual]
```

Runs the subsystem.

Implemented in [wisco::robot::subsystems::drive::DifferentialDriveSubsystem](#), [wisco::robot::subsystems::elevator::ElevatorSubsystem](#), [wisco::robot::subsystems::hang::HangSubsystem](#), [wisco::robot::subsystems::intake::IntakeSubsystem](#), [wisco::robot::subsystems::load::LoadSubsystem](#), [wisco::robot::subsystems::position::PositionSubsystem](#), [wisco::robot::subsystems::umbrella::UmbrellaSubsystem](#), and [wisco::robot::subsystems::wings::WingsSubsystem](#).

### 5.60.3.4 command()

```
virtual void wisco::robot::ASubsystem::command (
    std::string command_name,
    va_list & args ) [pure virtual]
```

Runs a command for the subsystem.

#### Parameters

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implemented in [wisco::robot::subsystems::drive::DifferentialDriveSubsystem](#), [wisco::robot::subsystems::elevator::ElevatorSubsystem](#), [wisco::robot::subsystems::hang::HangSubsystem](#), [wisco::robot::subsystems::intake::IntakeSubsystem](#), [wisco::robot::subsystems::load::LoadSubsystem](#), [wisco::robot::subsystems::position::PositionSubsystem](#), [wisco::robot::subsystems::umbrella::UmbrellaSubsystem](#), and [wisco::robot::subsystems::wings::WingsSubsystem](#).

### 5.60.3.5 state()

```
virtual void * wisco::robot::ASubsystem::state (
    std::string state_name ) [pure virtual]
```

Gets a state of the subsystem.

#### Parameters

<i>state_name</i>	The name of the state to get
-------------------	------------------------------

**Returns**

void\* The current value of that state

Implemented in [wisco::robot::subsystems::drive::DifferentialDriveSubsystem](#), [wisco::robot::subsystems::elevator::ElevatorSubsystem](#), [wisco::robot::subsystems::hang::HangSubsystem](#), [wisco::robot::subsystems::intake::IntakeSubsystem](#), [wisco::robot::subsystems::umbrella::UmbrellaSubsystem](#), and [wisco::robot::subsystems::wings::WingsSubsystem](#).

**5.60.3.6 operator=() [1/2]**

```
ASubsystem & wisco::robot::ASubsystem::operator= (
    const ASubsystem & rhs ) [default]
```

Copy assignment operator for [ASubsystem](#).

**Parameters**

<i>rhs</i>	The <a href="#">ASubsystem</a> on the right hand side of the operator
------------	---

**Returns**

[ASubsystem](#)& This [ASubsystem](#) with the copied values

**5.60.3.7 operator=() [2/2]**

```
ASubsystem & wisco::robot::ASubsystem::operator= (
    ASubsystem && rhs ) [default]
```

Move assignment operator for [ASubsystem](#).

**Parameters**

<i>rhs</i>	The <a href="#">ASubsystem</a> value on the right hand side of the operator
------------	---

**Returns**

[ASubsystem](#)& This [ASubsystem](#) with the moved values

**5.60.4 Member Data Documentation****5.60.4.1 m\_name**

```
std::string wisco::robot::ASubsystem::m_name {} [private]
```

The name of the subsystem.

Definition at line 35 of file [ASubsystem.hpp](#).  
00035 {};

## 5.61 wisco::robot::Robot Class Reference

A container class for subsystems.

### Public Member Functions

- void `addSubsystem` (`std::unique_ptr< ASubsystem > &subsystem)`  
*Adds a subsystem to the robot.*
- bool `removeSubsystem` (`std::string subsystem_name)`  
*Removes a subsystem from the robot.*
- void `initialize` ()  
*Initializes all subsystems in the robot.*
- void `sendCommand` (`std::string subsystem_name, std::string command_name,...)`  
*Sends a command to a subsystem.*
- void \* `getState` (`std::string subsystem_name, std::string state_name)`  
*Gets a state of a subsystem.*

### Private Attributes

- `std::vector< std::unique_ptr< ASubsystem > > subsystems {}`  
*The subsystems contained in the robot.*

### 5.61.1 Detailed Description

A container class for subsystems.

#### Author

Nathan Sandvig

Definition at line 30 of file [Robot.hpp](#).

### 5.61.2 Member Function Documentation

#### 5.61.2.1 addSubsystem()

```
void wisco::robot::Robot::addSubsystem ( 
    std::unique_ptr< ASubsystem > & subsystem )
```

Adds a subsystem to the robot.

#### Parameters

<code>subsystem</code>	The subsystem being added to the robot
------------------------	--

Definition at line 7 of file [Robot.cpp](#).

```
00008 {
00009     subsystems.push_back(std::move(subsystem));
00010 }
```

### 5.61.2.2 removeSubsystem()

```
bool wisco::robot::Robot::removeSubsystem (
    std::string subsystem_name )
```

Removes a subsystem from the robot.

#### Parameters

<i>subsystem_name</i>	The name of the subsystem to remove from the robot
-----------------------	--

#### Returns

**true** The subsystem was removed  
**false** The subsystem was not contained in the robot

#### Definition at line 12 of file Robot.cpp.

```
00013 {
00014     bool removed{false};
00015     for (auto it{subsystems.begin()}; it != subsystems.end(); ++it)
00016     {
00017         if ((*it)->getName() == subsystem_name)
00018         {
00019             subsystems.erase(it);
00020             removed = true;
00021             break;
00022         }
00023     }
00024     return removed;
00025 }
```

### 5.61.2.3 initialize()

```
void wisco::robot::Robot::initialize ( )
```

Initializes all subsystems in the robot.

#### Definition at line 27 of file Robot.cpp.

```
00028 {
00029     for (auto& subsystem : subsystems)
00030         subsystem->initialize();
00031     for (auto& subsystem : subsystems)
00032         subsystem->run();
00033 }
```

### 5.61.2.4 sendCommand()

```
void wisco::robot::Robot::sendCommand (
    std::string subsystem_name,
    std::string command_name,
    ... )
```

Sends a command to a subsystem.

**Parameters**

<i>subsystem_name</i>	The name of the subsystem
<i>command_name</i>	The name of the command
...	The parameters for the command

Definition at line 35 of file [Robot.cpp](#).

```
00036 {
00037     va_list args;
00038     va_start(args, command_name);
00039     for (auto& subsystem : subsystems)
00040     {
00041         if (subsystem->getName() == subsystem_name)
00042         {
00043             subsystem->command(command_name, args);
00044             break;
00045         }
00046     }
00047     va_end(args);
00048 }
```

### 5.61.2.5 getState()

```
void * wisco::robot::Robot::getState (
    std::string subsystem_name,
    std::string state_name )
```

Gets a state of a subsystem.

**Parameters**

<i>subsystem_name</i>	The name of the subsystem
<i>state_name</i>	The name of the state

**Returns**

`void*` The current value of that state

Definition at line 50 of file [Robot.cpp](#).

```
00051 {
00052     void* state=nullptr;
00053     for (auto& subsystem : subsystems)
00054     {
00055         if (subsystem->getName() == subsystem_name)
00056         {
00057             state = subsystem->state(state_name);
00058             break;
00059         }
00060     }
00061     return state;
00062 }
```

## 5.61.3 Member Data Documentation

### 5.61.3.1 subsystems

```
std::vector<std::unique_ptr<ASubsystem>> wisco::robot::Robot::subsystems {} [private]
```

The subsystems contained in the robot.

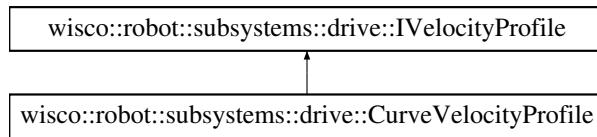
Definition at line 37 of file [Robot.hpp](#).

```
00037 {};
```

## 5.62 wisco::robot::subsystems::drive::CurveVelocityProfile Class Reference

An s-curve velocity profile for the drive.

Inheritance diagram for wisco::robot::subsystems::drive::CurveVelocityProfile:



### Public Member Functions

- `CurveVelocityProfile (std::unique_ptr< rtos::IClock > &clock, double jerk_rate, double max_acceleration)`  
*Construct a new Curve Velocity Profile object.*
- `double getAcceleration (double current_velocity, double target_velocity) override`  
*Get the target acceleration from the profile.*
- `void setAcceleration (double acceleration) override`  
*Set the current acceleration.*

### Public Member Functions inherited from wisco::robot::subsystems::drive::IVelocityProfile

- `virtual ~IVelocityProfile ()=default`  
*Destroy the IVelocityProfile object.*

### Private Attributes

- `std::unique_ptr< rtos::IClock > m_clock {}`  
*The system clock.*
- `double m_jerk_rate {}`  
*The jerk rate of the velocity profile.*
- `double m_max_acceleration {}`  
*The maximum acceleration output of the velocity profile.*
- `double m_current_acceleration {}`  
*The current acceleration rate of the profile.*
- `double last_time`  
*The last timestamp during execution.*

### 5.62.1 Detailed Description

An s-curve velocity profile for the drive.

#### Author

Nathan Sandvig

Definition at line 48 of file [CurveVelocityProfile.hpp](#).

## 5.62.2 Constructor & Destructor Documentation

### 5.62.2.1 CurveVelocityProfile()

```
wisco::robot::subsystems::drive::CurveVelocityProfile::CurveVelocityProfile (
    std::unique_ptr< rtos::IClock > & clock,
    double jerk_rate,
    double max_acceleration )
```

Construct a new Curve Velocity Profile object.

#### Parameters

<i>clock</i>	The system clock
<i>jerk_rate</i>	The jerk rate for the velocity profile
<i>max_acceleration</i>	The maximum acceleration value

Definition at line 13 of file [CurveVelocityProfile.cpp](#).

```
00014     : m_clock{std::move(clock)}, m_jerk_rate{jerk_rate}, m_max_acceleration{max_acceleration}
00015 {
00016
00017 }
```

## 5.62.3 Member Function Documentation

### 5.62.3.1 getAcceleration()

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::getAcceleration (
    double current_velocity,
    double target_velocity ) [override], [virtual]
```

Get the target acceleration from the profile.

#### Parameters

<i>current_velocity</i>	The current velocity
<i>target_velocity</i>	The target velocity

#### Returns

double The acceleration in m/s<sup>2</sup>

Implements [wisco::robot::subsystems::drive::IVelocityProfile](#).

Definition at line 19 of file [CurveVelocityProfile.cpp](#).

```
00020 {
00021     double time_change{};
00022     if (m_clock)
00023     {
00024         double current_time = m_clock->getTime();
00025         time_change = current_time - last_time;
00026         last_time = current_time;
00027     }
00028     // TODO calculate acceleration in a better manner
```

```

00030     double target_acceleration = (target_velocity - current_velocity) / time_change;
00031
00032     if (target_acceleration > m_max_acceleration)
00033         target_acceleration = m_max_acceleration;
00034     else if (target_acceleration < -m_max_acceleration)
00035         target_acceleration = -m_max_acceleration;
00036
00037     if (target_acceleration > (m_current_acceleration + (m_jerk_rate * time_change)))
00038         target_acceleration = (m_current_acceleration + (m_jerk_rate * time_change));
00039     else if (target_acceleration < (m_current_acceleration - (m_jerk_rate * time_change)))
00040         target_acceleration = (m_current_acceleration - (m_jerk_rate * time_change));
00041
00042     m_current_acceleration = target_acceleration;
00043
00044     return target_acceleration;
00045 }
```

### 5.62.3.2 setAcceleration()

```
void wisco::robot::subsystems::drive::CurveVelocityProfile::setAcceleration (
    double acceleration ) [override], [virtual]
```

Set the current acceleration.

#### Parameters

<i>acceleration</i>	The current acceleration
---------------------	--------------------------

Implements [wisco::robot::subsystems::drive::IVelocityProfile](#).

Definition at line 47 of file [CurveVelocityProfile.cpp](#).

```

00048 {
00049     m_current_acceleration = acceleration;
00050     last_time = m_clock->getTime();
00051 }
```

## 5.62.4 Member Data Documentation

### 5.62.4.1 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::drive::CurveVelocityProfile::m_clock
{} [private]
```

The system clock.

Definition at line 55 of file [CurveVelocityProfile.hpp](#).

```
00055 {};
```

### 5.62.4.2 m\_jerk\_rate

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::m_jerk_rate {} [private]
```

The jerk rate of the velocity profile.

Definition at line 61 of file [CurveVelocityProfile.hpp](#).

```
00061 {};
```

#### 5.62.4.3 m\_max\_acceleration

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::m_max_acceleration {} [private]
```

The maximum acceleration output of the velocity profile.

Definition at line 67 of file [CurveVelocityProfile.hpp](#).  
00067 {};

#### 5.62.4.4 m\_current\_acceleration

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::m_current_acceleration {} [private]
```

The current acceleration rate of the profile.

Definition at line 73 of file [CurveVelocityProfile.hpp](#).  
00073 {};

#### 5.62.4.5 last\_time

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::last_time [private]
```

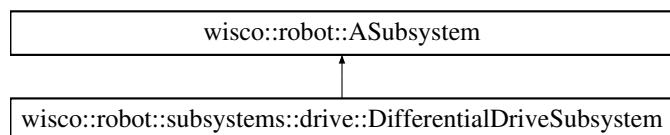
The last timestamp during execution.

Definition at line 79 of file [CurveVelocityProfile.hpp](#).

### 5.63 wisco::robot::subsystems::drive::DifferentialDriveSubsystem Class Reference

The subsystem adapter for differential drives.

Inheritance diagram for wisco::robot::subsystems::drive::DifferentialDriveSubsystem:



#### Public Member Functions

- **DifferentialDriveSubsystem** (std::unique\_ptr< IDifferentialDrive > &differential\_drive)  
*Construct a new Differential Drive Subsystem object.*
- void **initialize** () override  
*Initializes the subsystem.*
- void **run** () override  
*Runs the subsystem.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the subsystem.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the subsystem.*

## Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem ()=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem (const [ASubsystem](#) &other)=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem ([ASubsystem](#) &&other)=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem (std::string name)**  
*Construct a new [ASubsystem](#) object.*
- **virtual ~ASubsystem ()=default**  
*Destroy the [ASubsystem](#) object.*
- **const std::string & getName () const**  
*Get the name of the subsystem.*
- **[ASubsystem](#) & operator= (const [ASubsystem](#) &rhs)=default**  
*Copy assignment operator for [ASubsystem](#).*
- **[ASubsystem](#) & operator= ([ASubsystem](#) &&rhs)=default**  
*Move assignment operator for [ASubsystem](#).*

## Private Attributes

- **std::unique\_ptr< [IDifferentialDrive](#) > m\_differential\_drive {}**  
*The differential drive being adapted.*

## Static Private Attributes

- **static constexpr char SUBSYSTEM\_NAME [] {"DIFFERENTIAL DRIVE"}**  
*The name of the subsystem.*
- **static constexpr char SET\_VELOCITY\_COMMAND\_NAME [] {"SET VELOCITY"}**  
*The name of the set velocity command.*
- **static constexpr char SET\_VOLTAGE\_COMMAND\_NAME [] {"SET VOLTAGE"}**  
*The name of the set voltage command.*
- **static constexpr char GET\_VELOCITY\_STATE\_NAME [] {"GET VELOCITY"}**  
*The name of the get velocity command.*
- **static constexpr char GET\_RADIUS\_STATE\_NAME [] {"GET RADIUS"}**  
*The name of the get radius state.*

## 5.63.1 Detailed Description

The subsystem adapter for differential drives.

### Author

Nathan Sandvig

Definition at line 47 of file [DifferentialDriveSubsystem.hpp](#).

## 5.63.2 Constructor & Destructor Documentation

### 5.63.2.1 DifferentialDriveSubsystem()

```
wisco::robot::subsystems::drive::DifferentialDriveSubsystem::DifferentialDriveSubsystem (
    std::unique_ptr< IDifferentialDrive > & differential_drive )
```

Construct a new Differential Drive Subsystem object.

**Parameters**

<i>differential_drive</i>	The differential drive being adapted
---------------------------	--------------------------------------

Definition at line 11 of file [DifferentialDriveSubsystem.cpp](#).

```
00012     : ASubsystem{SUBSYSTEM_NAME}, m_differential_drive{std::move(differential_drive)}
00013 {
00014
00015 }
```

### 5.63.3 Member Function Documentation

#### 5.63.3.1 initialize()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::initialize () [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [DifferentialDriveSubsystem.cpp](#).

```
00018 {
00019     m_differential_drive->initialize();
00020 }
```

#### 5.63.3.2 run()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::run () [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 22 of file [DifferentialDriveSubsystem.cpp](#).

```
00023 {
00024     m_differential_drive->run();
00025 }
```

#### 5.63.3.3 command()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::command (
    std::string command_name,
    va_list & args) [override], [virtual]
```

Runs a command for the subsystem.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 27 of file [DifferentialDriveSubsystem.cpp](#).

```
00028 {
00029     if (command_name == SET_VELOCITY_COMMAND_NAME)
00030     {
00031         double left_velocity{va_arg(args, double)};
00032         double right_velocity{va_arg(args, double)};
00033         Velocity velocity{left_velocity, right_velocity};
00034         m_differential_drive->setVelocity(velocity);
00035     }
00036     else if (command_name == SET_VOLTAGE_COMMAND_NAME)
00037     {
00038         double left_voltage{va_arg(args, double)};
00039         double right_voltage{va_arg(args, double)};
00040         m_differential_drive->setVoltage(left_voltage, right_voltage);
00041     }
00042 }
```

### 5.63.3.4 state()

```
void * wisco::robot::subsystems::drive::DifferentialDriveSubsystem::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the subsystem.

#### Parameters

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 44 of file [DifferentialDriveSubsystem.cpp](#).

```
00045 {
00046     void* result=nullptr;
00047
00048     if (state_name == GET_VELOCITY_STATE_NAME)
00049     {
00050         Velocity* velocity{new Velocity{m_differential_drive->getVelocity()}};
00051         result = velocity;
00052     }
00053     else if (state_name == GET_RADIUS_STATE_NAME)
00054     {
00055         double* radius{new double{m_differential_drive->getRadius()}};
00056         result = radius;
00057     }
00058
00059     return result;
00060 }
```

## 5.63.4 Member Data Documentation

### 5.63.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::drive::DifferentialDriveSubsystem::SUBSYSTEM_NAME[ ]
{"DIFFERENTIAL DRIVE"} [static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 54 of file [DifferentialDriveSubsystem.hpp](#).

```
00054 {"DIFFERENTIAL DRIVE"};
```

#### 5.63.4.2 SET\_VELOCITY\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::drive::DifferentialDriveSubsystem::SET_VELOCITY_←
COMMAND_NAME[] {"SET VELOCITY"} [static], [constexpr], [private]
```

The name of the set velocity command.

Definition at line 60 of file [DifferentialDriveSubsystem.hpp](#).

```
00060 {"SET VELOCITY"};
```

#### 5.63.4.3 SET\_VOLTAGE\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::drive::DifferentialDriveSubsystem::SET_VOLTAGE_←
COMMAND_NAME[] {"SET VOLTAGE"} [static], [constexpr], [private]
```

The name of the set voltage command.

Definition at line 66 of file [DifferentialDriveSubsystem.hpp](#).

```
00066 {"SET VOLTAGE"};
```

#### 5.63.4.4 GET\_VELOCITY\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::drive::DifferentialDriveSubsystem::GET_VELOCITY_←
STATE_NAME[] {"GET VELOCITY"} [static], [constexpr], [private]
```

The name of the get velocity command.

Definition at line 72 of file [DifferentialDriveSubsystem.hpp](#).

```
00072 {"GET VELOCITY"};
```

#### 5.63.4.5 GET\_RADIUS\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::drive::DifferentialDriveSubsystem::GET_RADIUS_STATE_←
_NAME[] {"GET RADIUS"} [static], [constexpr], [private]
```

The name of the get radius state.

Definition at line 78 of file [DifferentialDriveSubsystem.hpp](#).

```
00078 {"GET RADIUS"};
```

#### 5.63.4.6 m\_differential\_drive

```
std::unique_ptr<IDifferentialDrive> wisco::robot::subsystems::drive::DifferentialDriveSubsystem::m_differential_drive {} [private]
```

The differential drive being adapted.

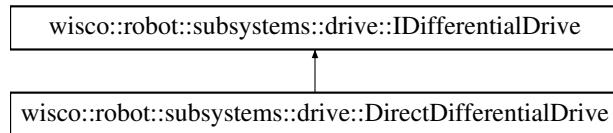
Definition at line 84 of file [DifferentialDriveSubsystem.hpp](#).

```
00084 {};
```

## 5.64 wisco::robot::subsystems::drive::DirectDifferentialDrive Class Reference

A direct drive controller with independent left and right wheelsets.

Inheritance diagram for wisco::robot::subsystems::drive::DirectDifferentialDrive:



### Public Member Functions

- void [initialize \(\)](#) override  
*Initializes the differential drive.*
- void [run \(\)](#) override  
*Runs the differential drive.*
- [Velocity getVelocity \(\)](#) override  
*Get the velocity values of the drive.*
- void [setVelocity \(Velocity velocity\)](#) override  
*Set the velocity values of the drive.*
- void [setVoltage \(double left\\_voltage, double right\\_voltage\)](#) override  
*Set the voltages of the drive directly.*
- double [getRadius \(\)](#) override  
*Gets the radius of the drive.*
- void [setLeftMotors \(hal::MotorGroup &left\\_motors\)](#)  
*Set the left drive motors.*
- void [setRightMotors \(hal::MotorGroup &right\\_motors\)](#)  
*Set the right drive motors.*
- void [setVelocityToVoltage \(double velocity\\_to\\_voltage\)](#)  
*Set the velocity to voltage conversion constant.*
- void [setGearRatio \(double gear\\_ratio\)](#)  
*Set the gear ratio.*
- void [setWheelRadius \(double wheel\\_radius\)](#)  
*Set the wheel radius.*
- void [setRadius \(double radius\)](#)  
*Set the drive radius.*

### Public Member Functions inherited from [wisco::robot::subsystems::drive::IDifferentialDrive](#)

- virtual ~[IDifferentialDrive \(\)](#)=default  
*Destroy the `IDifferentialDrive` object.*

## Private Attributes

- `hal::MotorGroup m_left_motors {}`  
*The left motors on the differential drive.*
- `hal::MotorGroup m_right_motors {}`  
*The right motors on the differential drive.*
- `double m_velocity_to_voltage {1.0}`  
*Converts the input velocity to a voltage to control.*
- `double m_gear_ratio {}`  
*The gear ratio from the motors to the drive (drive gear / motor gear)*
- `double m_wheel_radius {}`  
*The radius of the drive wheels.*
- `double m_radius {}`  
*The radius of the drive.*

## 5.64.1 Detailed Description

A direct drive controller with independent left and right wheelsets.

### Author

Nathan Sandvig

Definition at line 45 of file [DirectDifferentialDrive.hpp](#).

## 5.64.2 Member Function Documentation

### 5.64.2.1 initialize()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::initialize ( ) [override],  
[virtual]
```

Initializes the differential drive.

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 11 of file [DirectDifferentialDrive.cpp](#).

```
00012 {  
00013     m_left_motors.initialize();  
00014     m_right_motors.initialize();  
00015 }
```

### 5.64.2.2 run()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::run ( ) [override], [virtual]
```

Runs the differential drive.

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 17 of file [DirectDifferentialDrive.cpp](#).

```
00018 {  
00019  
00020 }
```

### 5.64.2.3 getVelocity()

```
Velocity wisco::robot::subsystems::drive::DirectDifferentialDrive::getVelocity () [override], [virtual]
```

Get the velocity values of the drive.

#### Returns

double The drive velocity

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 22 of file [DirectDifferentialDrive.cpp](#).

```
00023 {
00024     Velocity velocity
00025     {
00026         m_left_motors.getAngularVelocity() * m_wheel_radius / m_gear_ratio,
00027         m_right_motors.getAngularVelocity() * m_wheel_radius / m_gear_ratio
00028     };
00029     return velocity;
00030 }
```

### 5.64.2.4 setVelocity()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVelocity (
    Velocity velocity ) [override], [virtual]
```

Set the velocity values of the drive.

#### Parameters

<i>velocity</i>	The velocity values for the drive
-----------------	-----------------------------------

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 32 of file [DirectDifferentialDrive.cpp](#).

```
00033 {
00034     m_left_motors.setVoltage(velocity.left_velocity * m_velocity_to_voltage);
00035     m_right_motors.setVoltage(velocity.right_velocity * m_velocity_to_voltage);
00036 }
```

### 5.64.2.5 setVoltage()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVoltage (
    double left_voltage,
    double right_voltage ) [override], [virtual]
```

Set the voltages of the drive directly.

#### Parameters

<i>left_voltage</i>	The voltage for the left side of the drive
<i>right_voltage</i>	The voltage for the right side of the drive

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 38 of file [DirectDifferentialDrive.cpp](#).

```
00039 {
00040     m_left_motors.setVoltage(left_voltage);
00041     m_right_motors.setVoltage(right_voltage);
00042 }
```

### 5.64.2.6 getRadius()

```
double wisco::robot::subsystems::drive::DirectDifferentialDrive::getRadius ( ) [override],  
[virtual]
```

Gets the radius of the drive.

Returns

double The radius of the drive

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 44 of file [DirectDifferentialDrive.cpp](#).

```
00045 {
00046     return m_radius;
00047 }
```

### 5.64.2.7 setLeftMotors()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setLeftMotors (   
    hal::MotorGroup & left_motors )
```

Set the left drive motors.

Parameters

<i>left_motors</i>	The motors on the left side of the drive
--------------------	--

Definition at line 49 of file [DirectDifferentialDrive.cpp](#).

```
00050 {
00051     m_left_motors = left_motors;
00052 }
```

### 5.64.2.8 setRightMotors()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setRightMotors (   
    hal::MotorGroup & right_motors )
```

Set the right drive motors.

Parameters

<i>right_motors</i>	The motors on the right side of the drive
---------------------	---

Definition at line 54 of file [DirectDifferentialDrive.cpp](#).

```
00055 {  
00056     m_right_motors = right_motors;  
00057 }
```

### 5.64.2.9 setVelocityToVoltage()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVelocityToVoltage (  
    double velocity_to_voltage )
```

Set the velocity to voltage conversion constant.

#### Parameters

<i>velocity_to_voltage</i>	The velocity to voltage conversion constant
----------------------------	---

Definition at line 59 of file [DirectDifferentialDrive.cpp](#).

```
00060 {  
00061     m_velocity_to_voltage = velocity_to_voltage;  
00062 }
```

### 5.64.2.10 setGearRatio()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setGearRatio (   
    double gear_ratio )
```

Set the gear ratio.

#### Parameters

<i>gear_ratio</i>	The gear ratio of the drive
-------------------	-----------------------------

Definition at line 64 of file [DirectDifferentialDrive.cpp](#).

```
00065 {  
00066     m_gear_ratio = gear_ratio;  
00067 }
```

### 5.64.2.11 setWheelRadius()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setWheelRadius (   
    double wheel_radius )
```

Set the wheel radius.

#### Parameters

<i>wheel_radius</i>	The wheel radius of the drive
---------------------	-------------------------------

Definition at line 69 of file [DirectDifferentialDrive.cpp](#).

```
00070 {  
00071     m_wheel_radius = wheel_radius;  
00072 }
```

### 5.64.2.12 setRadius()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setRadius (
    double radius )
```

Set the drive radius.

#### Parameters

<i>radius</i>	The drive radius
---------------	------------------

Definition at line 74 of file [DirectDifferentialDrive.cpp](#).

```
00075 {
00076     m_radius = radius;
00077 }
```

## 5.64.3 Member Data Documentation

### 5.64.3.1 m\_left\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::DirectDifferentialDrive::m_left_motors {}
[private]
```

The left motors on the differential drive.

Definition at line 52 of file [DirectDifferentialDrive.hpp](#).

```
00052 {};
```

### 5.64.3.2 m\_right\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::DirectDifferentialDrive::m_right_motors {}
[private]
```

The right motors on the differential drive.

Definition at line 58 of file [DirectDifferentialDrive.hpp](#).

```
00058 {};
```

### 5.64.3.3 m\_velocity\_to\_voltage

```
double wisco::robot::subsystems::drive::DirectDifferentialDrive::m_velocity_to_voltage {1.0}
[private]
```

Converts the input velocity to a voltage to control.

Definition at line 64 of file [DirectDifferentialDrive.hpp](#).

```
00064 {1.0};
```

#### 5.64.3.4 m\_gear\_ratio

```
double wisco::robot::subsystems::drive::DirectDifferentialDrive::m_gear_ratio {} [private]
```

The gear ratio from the motors to the drive (drive gear / motor gear)

Definition at line 70 of file [DirectDifferentialDrive.hpp](#).

```
00070 {};
```

#### 5.64.3.5 m\_wheel\_radius

```
double wisco::robot::subsystems::drive::DirectDifferentialDrive::m_wheel_radius {} [private]
```

The radius of the drive wheels.

Definition at line 76 of file [DirectDifferentialDrive.hpp](#).

```
00076 {};
```

#### 5.64.3.6 m\_radius

```
double wisco::robot::subsystems::drive::DirectDifferentialDrive::m_radius {} [private]
```

The radius of the drive.

Definition at line 82 of file [DirectDifferentialDrive.hpp](#).

```
00082 {};
```

## 5.65 wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder Class Reference

Builder class for the direct differential drive class.

### Public Member Functions

- `DirectDifferentialDriveBuilder * withLeftMotor (std::unique_ptr< io::IMotor > &left_motor)`  
*Add a left drive motor to the build.*
- `DirectDifferentialDriveBuilder * withRightMotor (std::unique_ptr< io::IMotor > &right_motor)`  
*Add a right drive motor to the build.*
- `DirectDifferentialDriveBuilder * withVelocityToVoltage (double velocity_to_voltage)`  
*Add the velocity to voltage conversion constant to the build.*
- `DirectDifferentialDriveBuilder * withGearRatio (double gear_ratio)`  
*Add the gear ratio to the build.*
- `DirectDifferentialDriveBuilder * withWheelRadius (double wheel_radius)`  
*Add the wheel radius to the build.*
- `DirectDifferentialDriveBuilder * withRadius (double radius)`  
*Adds the drive radius to the build.*
- `std::unique_ptr< IDifferentialDrive > build ()`  
*Builds the differential drive system.*

## Private Attributes

- `hal::MotorGroup m_left_motors {}`  
*The left motors on the differential drive.*
- `hal::MotorGroup m_right_motors {}`  
*The right motors on the differential drive.*
- `double m_velocity_to_voltage {1.0}`  
*The conversion constant from velocity to voltage.*
- `double m_gear_ratio {}`  
*The gear ratio from the motors to the drive (drive gear / motor gear)*
- `double m_wheel_radius {}`  
*The radius of the drive wheels.*
- `double m_radius {}`  
*The radius of the drive.*

### 5.65.1 Detailed Description

Builder class for the direct differential drive class.

#### Author

Nathan Sandvig

Definition at line 45 of file [DirectDifferentialDriveBuilder.hpp](#).

### 5.65.2 Member Function Documentation

#### 5.65.2.1 withLeftMotor()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDrive<-->
Builder::withLeftMotor (
    std::unique_ptr< io::IMotor > & left_motor )
```

Add a left drive motor to the build.

#### Parameters

<code>left_motor</code>	The motor on the left side of the drive
-------------------------	---

#### Returns

`DirectDifferentialDriveBuilder*` This object for build chaining

Definition at line 11 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00012 {
00013     m_left_motors.addMotor(left_motor);
00014     return this;
00015 }
```

### 5.65.2.2 withRightMotor()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::withRightMotor (
    std::unique_ptr< io::IMotor > & right_motor )
```

Add a right drive motor to the build.

#### Parameters

<i>right_motor</i>	The motor on the right side of the drive
--------------------	--

#### Returns

DirectDifferentialDriveBuilder\* This object for build chaining

Definition at line 17 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00018 {
00019     m_right_motors.addMotor(right_motor);
00020     return this;
00021 }
```

### 5.65.2.3 withVelocityToVoltage()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::withVelocityToVoltage (
    double velocity_to_voltage )
```

Add the velocity to voltage conversion constant to the build.

#### Parameters

<i>velocity_to_voltage</i>	The velocity to voltage conversion constant of the drive
----------------------------	--

#### Returns

DirectDifferentialDriveBuilder\* This object for build chaining

Definition at line 23 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00024 {
00025     m_velocity_to_voltage = velocity_to_voltage;
00026     return this;
00027 }
```

### 5.65.2.4 withGearRatio()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::withGearRatio (
    double gear_ratio )
```

Add the gear ratio to the build.

**Parameters**

<i>gear_ratio</i>	The gear ratio of the drive
-------------------	-----------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 29 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00030 {
00031     m_gear_ratio = gear_ratio;
00032     return this;
00033 }
```

**5.65.2.5 withWheelRadius()**

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::withWheelRadius (
    double wheel_radius )
```

Add the wheel radius to the build.

**Parameters**

<i>wheel_radius</i>	The wheel radius of the drive
---------------------	-------------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 35 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00036 {
00037     m_wheel_radius = wheel_radius;
00038     return this;
00039 }
```

**5.65.2.6 withRadius()**

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::withRadius (
    double radius )
```

Adds the drive radius to the build.

**Parameters**

<i>radius</i>	The drive radius
---------------	------------------

**Returns**

DirectDifferentialDriveBuilder\* This object for build chaining

Definition at line 41 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00042 {
00043     m_radius = radius;
00044     return this;
00045 }
```

### 5.65.2.7 build()

```
std::unique_ptr< IDifferentialDrive > wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::build ( )
```

Builds the differential drive system.

#### Returns

`std::unique_ptr<IDifferentialDrive>` The differential drive system as a differential drive interface

Definition at line 47 of file [DirectDifferentialDriveBuilder.cpp](#).

```
00048 {
00049     std::unique_ptr<DirectDifferentialDrive>
00050         differential_drive(std::make_unique<DirectDifferentialDrive>());
00051     differential_drive->setLeftMotors(m_left_motors);
00052     differential_drive->setRightMotors(m_right_motors);
00053     differential_drive->setVelocityToVoltage(m_velocity_to_voltage);
00054     differential_drive->setRadius(m_radius);
00055     return differential_drive;
00055 }
```

## 5.65.3 Member Data Documentation

### 5.65.3.1 m\_left\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_left_motors
{} [private]
```

The left motors on the differential drive.

Definition at line 52 of file [DirectDifferentialDriveBuilder.hpp](#).

```
00052 {};
```

### 5.65.3.2 m\_right\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_right_motors
{} [private]
```

The right motors on the differential drive.

Definition at line 58 of file [DirectDifferentialDriveBuilder.hpp](#).

```
00058 {};
```

### 5.65.3.3 m\_velocity\_to\_voltage

```
double wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_velocity_to_voltage
{1.0} [private]
```

The conversion constant from velocity to voltage.

Definition at line 64 of file [DirectDifferentialDriveBuilder.hpp](#).

```
00064 {1.0};
```

#### 5.65.3.4 m\_gear\_ratio

```
double wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_gear_ratio {} [private]
```

The gear ratio from the motors to the drive (drive gear / motor gear)

Definition at line 70 of file [DirectDifferentialDriveBuilder.hpp](#).  
00070 {};

#### 5.65.3.5 m\_wheel\_radius

```
double wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_wheel_radius {} [private]
```

The radius of the drive wheels.

Definition at line 76 of file [DirectDifferentialDriveBuilder.hpp](#).  
00076 {};

#### 5.65.3.6 m\_radius

```
double wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_radius {} [private]
```

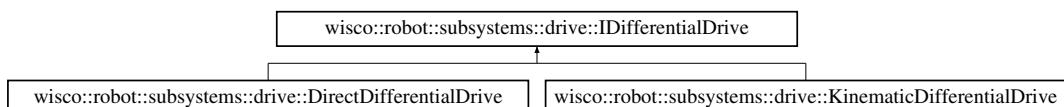
The radius of the drive.

Definition at line 82 of file [DirectDifferentialDriveBuilder.hpp](#).  
00082 {};

## 5.66 wisco::robot::subsystems::drive::IDifferentialDrive Class Reference

Interface for differential drivetrains.

Inheritance diagram for wisco::robot::subsystems::drive::IDifferentialDrive:



### Public Member Functions

- virtual ~**IDifferentialDrive** ()=default  
*Destroy the **IDifferentialDrive** object.*
- virtual void **initialize** ()=0  
*Initializes the differential drive.*
- virtual void **run** ()=0  
*Runs the differential drive.*
- virtual **Velocity getVelocity** ()=0  
*Get the velocity values of the drive.*
- virtual void **setVelocity** (**Velocity** velocity)=0  
*Set the velocity values of the drive.*
- virtual void **setVoltage** (double left\_voltage, double right\_voltage)=0  
*Set the voltages of the drive directly.*
- virtual double **getRadius** ()=0  
*Gets the radius of the drive.*

## 5.66.1 Detailed Description

Interface for differential drivetrains.

### Author

Nathan Sandvig

Definition at line 43 of file [IDifferentialDrive.hpp](#).

## 5.66.2 Member Function Documentation

### 5.66.2.1 initialize()

```
virtual void wisco::robot::subsystems::drive::IDifferentialDrive::initialize () [pure virtual]
```

Initializes the differential drive.

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

### 5.66.2.2 run()

```
virtual void wisco::robot::subsystems::drive::IDifferentialDrive::run () [pure virtual]
```

Runs the differential drive.

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

### 5.66.2.3 getVelocity()

```
virtual Velocity wisco::robot::subsystems::drive::IDifferentialDrive::getVelocity () [pure virtual]
```

Get the velocity values of the drive.

### Returns

double The drive velocity

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

### 5.66.2.4 setVelocity()

```
virtual void wisco::robot::subsystems::drive::IDifferentialDrive::setVelocity (
    Velocity velocity ) [pure virtual]
```

Set the velocity values of the drive.

**Parameters**

<i>velocity</i>	The velocity values for the drive
-----------------	-----------------------------------

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

**5.66.2.5 setVoltage()**

```
virtual void wisco::robot::subsystems::drive::IDifferentialDrive::setVoltage (
    double left_voltage,
    double right_voltage ) [pure virtual]
```

Set the voltages of the drive directly.

**Parameters**

<i>left_voltage</i>	The voltage for the left side of the drive
<i>right_voltage</i>	The voltage for the right side of the drive

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

**5.66.2.6 getRadius()**

```
virtual double wisco::robot::subsystems::drive::IDifferentialDrive::getRadius () [pure virtual]
```

Gets the radius of the drive.

**Returns**

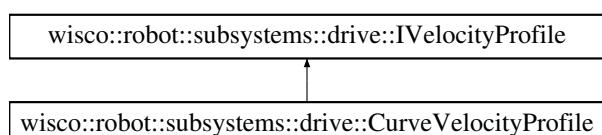
double The radius of the drive

Implemented in [wisco::robot::subsystems::drive::DirectDifferentialDrive](#), and [wisco::robot::subsystems::drive::KinematicDifferentialDrive](#)

**5.67 wisco::robot::subsystems::drive::IVelocityProfile Class Reference**

Interface for drive velocity profiles.

Inheritance diagram for wisco::robot::subsystems::drive::IVelocityProfile:



## Public Member Functions

- virtual ~**IVelocityProfile** ()=default  
*Destroy the `IVelocityProfile` object.*
- virtual double **getAcceleration** (double current\_velocity, double target\_velocity)=0  
*Get the target acceleration from the profile.*
- virtual void **setAcceleration** (double acceleration)=0  
*Set the current acceleration.*

### 5.67.1 Detailed Description

Interface for drive velocity profiles.

#### Author

Nathan Sandvig

Definition at line 41 of file [IVelocityProfile.hpp](#).

### 5.67.2 Member Function Documentation

#### 5.67.2.1 getAcceleration()

```
virtual double wisco::robot::subsystems::drive::IVelocityProfile::getAcceleration (
    double current_velocity,
    double target_velocity ) [pure virtual]
```

Get the target acceleration from the profile.

##### Parameters

<i>current_velocity</i>	The current velocity
<i>target_velocity</i>	The target velocity

##### Returns

double The acceleration in m/s<sup>2</sup>

Implemented in [wisco::robot::subsystems::drive::CurveVelocityProfile](#).

#### 5.67.2.2 setAcceleration()

```
virtual void wisco::robot::subsystems::drive::IVelocityProfile::setAcceleration (
    double acceleration ) [pure virtual]
```

Set the current acceleration.

## Parameters

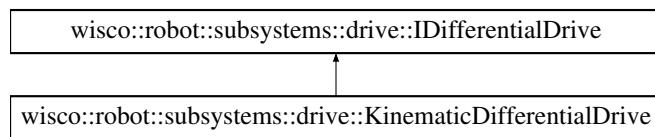
<code>acceleration</code>	The current acceleration
---------------------------	--------------------------

Implemented in [wisco::robot::subsystems::drive::CurveVelocityProfile](#).

## 5.68 wisco::robot::subsystems::drive::KinematicDifferentialDrive Class Reference

A kinematic drive controller with independent left and right wheelsets.

Inheritance diagram for wisco::robot::subsystems::drive::KinematicDifferentialDrive:



### Public Member Functions

- void `initialize ()` override  
*Initializes the differential drive.*
- void `run ()` override  
*Runs the differential drive.*
- `Velocity getVelocity ()` override  
*Get the velocity values of the drive.*
- void `setVelocity (Velocity velocity)` override  
*Set the velocity values of the drive.*
- void `setVoltage (double left_voltage, double right_voltage)` override  
*Set the voltages of the drive directly.*
- double `getRadius ()` override  
*Gets the radius of the drive.*
- void `setDelayer (std::unique_ptr< rtos::IDelay > &delayer)`  
*Set the rtos delayer.*
- void `setMutex (std::unique_ptr< rtos::IMutex > &mutex)`  
*Set the os mutex.*
- void `setTask (std::unique_ptr< rtos::ITask > &task)`  
*Set the rtos task handler.*
- void `setVelocityProfiles (std::unique_ptr< IVelocityProfile > &left_velocity_profile, std::unique_ptr< IVelocityProfile > &right_velocity_profile)`  
*Set the Velocity Profiles.*
- void `setLeftMotors (hal::MotorGroup &left_motors)`  
*Set the left drive motors.*
- void `setRightMotors (hal::MotorGroup &right_motors)`  
*Set the right drive motors.*
- void `setMass (double mass)`  
*Set the mass.*

- void [setRadius](#) (double radius)  
*Set the radius.*
- void [setMomentOfInertia](#) (double moment\_of\_inertia)  
*Set the moment of inertia.*
- void [setGearRatio](#) (double gear\_ratio)  
*Set the gear ratio.*
- void [setWheelRadius](#) (double wheel\_radius)  
*Set the wheel radius.*

## Public Member Functions inherited from wisco::robot::subsystems::drive::IDifferentialDrive

- virtual ~[IDifferentialDrive](#) ()=default  
*Destroy the [IDifferentialDrive](#) object.*

## Private Member Functions

- void [taskUpdate](#) ()  
*Runs all the object-specific updates in the task loop.*
- void [updateAcceleration](#) ()  
*Updates the motor values using the target acceleration values.*

## Static Private Member Functions

- static void [taskLoop](#) (void \*params)  
*The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr<[rtos::IDelayer](#)> [m\\_delayer](#) {}  
*The system delayer.*
- std::unique\_ptr<[rtos::IMutex](#)> [m\\_mutex](#) {}  
*The os mutex.*
- std::unique\_ptr<[rtos::ITask](#)> [m\\_task](#) {}  
*The task handler.*
- std::unique\_ptr<[IVelocityProfile](#)> [m\\_left\\_velocity\\_profile](#) {}  
*The left velocity profile.*
- std::unique\_ptr<[IVelocityProfile](#)> [m\\_right\\_velocity\\_profile](#) {}  
*The right velocity profile.*
- [hal::MotorGroup](#) [m\\_left\\_motors](#) {}  
*The left motors on the differential drive.*
- [hal::MotorGroup](#) [m\\_right\\_motors](#) {}  
*The right motors on the differential drive.*
- double [m\\_mass](#) {}  
*The mass of the robot.*
- double [m\\_radius](#) {}  
*The radius of the drive.*
- double [m\\_moment\\_of\\_inertia](#) {}

- **double m\_gear\_ratio {}**  
*The gear ratio from the motors to the drive (drive gear / motor gear)*
- **double m\_wheel\_radius {}**  
*The radius of the drive wheels.*
- **double c1 {}**  
*The first kinematic constant.*
- **double c2 {}**  
*The second kinematic constant.*
- **double c3 {}**  
*The third kinematic constant.*
- **double c4 {}**  
*The fourth kinematic constant.*
- **double c5 {}**  
*The fifth kinematic constant.*
- **double c6 {}**  
*The sixth kinematic constant.*
- **double c7 {}**  
*The seventh kinematic constant.*
- **Velocity m\_velocity {}**  
*The target velocity for the drive.*

## Static Private Attributes

- **static constexpr uint8\_t TASK\_DELAY {10}**  
*The loop delay on the task.*

### 5.68.1 Detailed Description

A kinematic drive controller with independent left and right wheelsets.

#### Author

Nathan Sandvig

Definition at line 52 of file [KinematicDifferentialDrive.hpp](#).

### 5.68.2 Member Function Documentation

#### 5.68.2.1 taskLoop()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

**Parameters**

<i>params</i>	
---------------	--

Definition at line 11 of file [KinematicDifferentialDrive.cpp](#).

```
00012 {
00013     void** parameters{static_cast<void**>(params)};
00014     KinematicDifferentialDrive* instance{static_cast<KinematicDifferentialDrive*>(parameters[0])};
00015
00016     while (true)
00017     {
00018         instance->taskUpdate();
00019     }
00020 }
```

**5.68.2.2 taskUpdate()**

`void wisco::robot::subsystems::drive::KinematicDifferentialDrive::taskUpdate () [private]`

Runs all the object-specific updates in the task loop.

Definition at line 22 of file [KinematicDifferentialDrive.cpp](#).

```
00023 {
00024     updateAcceleration();
00025     m_delayer->delay(TASK_DELAY);
00026 }
```

**5.68.2.3 updateAcceleration()**

`void wisco::robot::subsystems::drive::KinematicDifferentialDrive::updateAcceleration () [private]`

Updates the motor values using the target acceleration values.

Definition at line 28 of file [KinematicDifferentialDrive.cpp](#).

```
00029 {
00030     if (m_mutex)
00031         m_mutex->take();
00032
00033     Velocity velocity{getVelocity()};
00034
00035     double left_acceleration{m_left_velocity_profile->getAcceleration(velocity.left_velocity,
00036         m_velocity.left_velocity)};
00036     double right_acceleration{m_right_velocity_profile->getAcceleration(velocity.right_velocity,
00037         m_velocity.right_velocity)};
00038
00038     double left_voltage{((c5 * left_acceleration
00039                         - c1 * c7 * velocity.left_velocity
00040                         - c6 * right_acceleration)
00041                         /
00042                         (c2 * c7));
00043     double right_voltage{((c5 * right_acceleration
00044                         - c3 * c7 * velocity.right_velocity
00045                         - c6 * left_acceleration)
00046                         /
00047                         (c4 * c7));
00048
00049     // TODO fix kinematic constants
00050     //m_left_motors.setVoltage(left_voltage);
00051     //m_right_motors.setVoltage(right_voltage);
00052
00053     if (m_mutex)
00054         m_mutex->give();
00055 }
```

### 5.68.2.4 initialize()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::initialize ( ) [override],  
[virtual]
```

Initializes the differential drive.

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 57 of file [KinematicDifferentialDrive.cpp](#).

```
00058 {  
00059     m_left_motors.initialize();  
00060     m_right_motors.initialize();  
00061     m_left_velocity_profile->setAcceleration(0);  
00062     m_right_velocity_profile->setAcceleration(0);  
00063  
00064     c1 = (-1 * std::pow(m_left_motors.getGearRatio() * m_gear_ratio, 2) *  
00065             m_left_motors.getTorqueConstant())  
00066             / (m_left_motors.getAngularVelocityConstant() * m_left_motors.getResistance() *  
00067             std::pow(m_wheel_radius, 2));  
00068  
00069     c2 = (m_left_motors.getGearRatio() * m_gear_ratio * m_left_motors.getTorqueConstant())  
00070             / (m_left_motors.getResistance() * m_wheel_radius);  
00071  
00072     c3 = (-1 * std::pow(m_right_motors.getGearRatio() * m_gear_ratio, 2) *  
00073             m_right_motors.getTorqueConstant())  
00074             / (m_right_motors.getAngularVelocityConstant() * m_right_motors.getResistance() *  
00075             std::pow(m_wheel_radius, 2));  
00076  
00077     c4 = (m_right_motors.getGearRatio() * m_gear_ratio * m_right_motors.getTorqueConstant())  
00078             / (m_right_motors.getResistance() * m_wheel_radius);  
00079  
00080     c5 = (1 / m_mass) + (std::pow(m_radius, 2) / m_moment_of_inertia);  
00081     c6 = (1 / m_mass) - (std::pow(m_radius, 2) / m_moment_of_inertia);  
00082     c7 = std::pow(c5, 2) - std::pow(c6, 2);  
00083 }
```

### 5.68.2.5 run()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::run ( ) [override], [virtual]
```

Runs the differential drive.

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 83 of file [KinematicDifferentialDrive.cpp](#).

```
00084 {  
00085     if (m_task)  
00086     {  
00087         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};  
00088         params[0] = this;  
00089         m_task->start(&KinematicDifferentialDrive::taskLoop, params);  
00090     }  
00091 }
```

### 5.68.2.6 getVelocity()

```
Velocity wisco::robot::subsystems::drive::KinematicDifferentialDrive::getVelocity ( ) [override],  
[virtual]
```

Get the velocity values of the drive.

**Returns**

```
double The drive velocity
```

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 93 of file [KinematicDifferentialDrive.cpp](#).

```
00094 {
00095     Velocity velocity
00096     {
00097         m_left_motors.getAngularVelocity() * m_wheel_radius / m_gear_ratio,
00098         m_right_motors.getAngularVelocity() * m_wheel_radius / m_gear_ratio
00099     };
00100     return velocity;
00101 }
```

**5.68.2.7 setVelocity()**

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setVelocity (
    Velocity velocity) [override], [virtual]
```

Set the velocity values of the drive.

**Parameters**

<i>velocity</i>	The velocity values for the drive
-----------------	-----------------------------------

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 103 of file [KinematicDifferentialDrive.cpp](#).

```
00104 {
00105     if (m_mutex)
00106         m_mutex->take();
00107
00108     m_velocity = velocity;
00109
00110     if (m_mutex)
00111         m_mutex->give();
00112 }
```

**5.68.2.8 setVoltage()**

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setVoltage (
    double left_voltage,
    double right_voltage) [override], [virtual]
```

Set the voltages of the drive directly.

**Parameters**

<i>left_voltage</i>	The voltage for the left side of the drive
<i>right_voltage</i>	The voltage for the right side of the drive

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 114 of file [KinematicDifferentialDrive.cpp](#).

```
00115 {
```

```
00116     m_left_motors.setVoltage(left_voltage);
00117     m_right_motors.setVoltage(right_voltage);
00118 }
```

### 5.68.2.9 getRadius()

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::getRadius () [override],  
[virtual]
```

Gets the radius of the drive.

#### Returns

`double` The radius of the drive

Implements [wisco::robot::subsystems::drive::IDifferentialDrive](#).

Definition at line 120 of file [KinematicDifferentialDrive.cpp](#).

```
00121 {  
00122     return m_radius;  
00123 }
```

### 5.68.2.10 setDelayer()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setDelayer (  
    std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

#### Parameters

<code>delayer</code>	The rtos delayer
----------------------	------------------

Definition at line 125 of file [KinematicDifferentialDrive.cpp](#).

```
00126 {  
00127     m_delayer = std::move(delayer);  
00128 }
```

### 5.68.2.11 setMutex()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMutex (  
    std::unique_ptr< rtos::IMutex > & mutex )
```

Set the os mutex.

#### Parameters

<code>mutex</code>	The os mutex
--------------------	--------------

Definition at line 130 of file [KinematicDifferentialDrive.cpp](#).

```
00131 {  
00132     m_mutex = std::move(mutex);  
00133 }
```

### 5.68.2.12 setTask()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Set the rtos task handler.

#### Parameters

<i>task</i>	The rtos task handler
-------------	-----------------------

Definition at line 135 of file [KinematicDifferentialDrive.cpp](#).

```
00136 {
00137     m_task = std::move(task);
00138 }
```

### 5.68.2.13 setVelocityProfiles()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setVelocityProfiles (
    std::unique_ptr< IVelocityProfile > & left_velocity_profile,
    std::unique_ptr< IVelocityProfile > & right_velocity_profile )
```

Set the [Velocity Profiles](#).

#### Parameters

<i>left_velocity_profile</i>	The velocity profile for the left side
<i>right_velocity_profile</i>	The velocity profile for the right side

Definition at line 140 of file [KinematicDifferentialDrive.cpp](#).

```
00141 {
00142     m_left_velocity_profile = std::move(left_velocity_profile);
00143     m_right_velocity_profile = std::move(right_velocity_profile);
00144 }
```

### 5.68.2.14 setLeftMotors()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setLeftMotors (
    hal::MotorGroup & left_motors )
```

Set the left drive motors.

#### Parameters

<i>left_motors</i>	The motors on the left side of the drive
--------------------	--

Definition at line 146 of file [KinematicDifferentialDrive.cpp](#).

```
00147 {
00148     m_left_motors = left_motors;
00149 }
```

### 5.68.2.15 setRightMotors()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setRightMotors (
    hal::MotorGroup & right_motors )
```

Set the right drive motors.

#### Parameters

<i>right_motors</i>	The motors on the right side of the drive
---------------------	---

Definition at line 151 of file [KinematicDifferentialDrive.cpp](#).

```
00152 {
00153     m_right_motors = right_motors;
00154 }
```

### 5.68.2.16 setMass()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMass (
    double mass )
```

Set the mass.

#### Parameters

<i>mass</i>	The mass of the drive
-------------	-----------------------

Definition at line 156 of file [KinematicDifferentialDrive.cpp](#).

```
00157 {
00158     m_mass = mass;
00159 }
```

### 5.68.2.17 setRadius()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setRadius (
    double radius )
```

Set the radius.

#### Parameters

<i>radius</i>	The radius of the drive
---------------	-------------------------

Definition at line 161 of file [KinematicDifferentialDrive.cpp](#).

```
00162 {
00163     m_radius = radius;
00164 }
```

### 5.68.2.18 setMomentOfInertia()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMomentOfInertia (
    double moment_of_inertia )
```

Set the moment of inertia.

**Parameters**

<i>moment_of_inertia</i>	The moment of inertia of the drive
--------------------------	------------------------------------

Definition at line 166 of file [KinematicDifferentialDrive.cpp](#).

```
00167 {  
00168     m_moment_of_inertia = moment_of_inertia;  
00169 }
```

**5.68.2.19 setGearRatio()**

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setGearRatio (  
    double gear_ratio )
```

Set the gear ratio.

**Parameters**

<i>gear_ratio</i>	The gear ratio of the drive
-------------------	-----------------------------

Definition at line 171 of file [KinematicDifferentialDrive.cpp](#).

```
00172 {  
00173     m_gear_ratio = gear_ratio;  
00174 }
```

**5.68.2.20 setWheelRadius()**

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setWheelRadius (  
    double wheel_radius )
```

Set the wheel radius.

**Parameters**

<i>wheel_radius</i>	The wheel radius of the drive
---------------------	-------------------------------

Definition at line 176 of file [KinematicDifferentialDrive.cpp](#).

```
00177 {  
00178     m_wheel_radius = wheel_radius;  
00179 }
```

**5.68.3 Member Data Documentation****5.68.3.1 TASK\_DELAY**

```
constexpr uint8_t wisco::robot::subsystems::drive::KinematicDifferentialDrive::TASK_DELAY {10}  
[static], [constexpr], [private]
```

The loop delay on the task.

Definition at line 59 of file [KinematicDifferentialDrive.hpp](#).

```
00059 {10};
```

### 5.68.3.2 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_delayer {} [private]
```

The system delayer.

Definition at line 72 of file [KinematicDifferentialDrive.hpp](#).  
00072 {};

### 5.68.3.3 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_mutex {} [private]
```

The os mutex.

Definition at line 78 of file [KinematicDifferentialDrive.hpp](#).  
00078 {};

### 5.68.3.4 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_task {} [private]
```

The task handler.

Definition at line 84 of file [KinematicDifferentialDrive.hpp](#).  
00084 {};

### 5.68.3.5 m\_left\_velocity\_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_left_velocity_profile {} [private]
```

The left velocity profile.

Definition at line 90 of file [KinematicDifferentialDrive.hpp](#).  
00090 {};

### 5.68.3.6 m\_right\_velocity\_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_right_velocity_profile {} [private]
```

The right velocity profile.

Definition at line 96 of file [KinematicDifferentialDrive.hpp](#).  
00096 {};

### 5.68.3.7 m\_left\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_left_motors {}  
[private]
```

The left motors on the differential drive.

Definition at line 102 of file [KinematicDifferentialDrive.hpp](#).  
00102 {};

### 5.68.3.8 m\_right\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_right_motors {}  
[private]
```

The right motors on the differential drive.

Definition at line 108 of file [KinematicDifferentialDrive.hpp](#).  
00108 {};

### 5.68.3.9 m\_mass

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_mass {} [private]
```

The mass of the robot.

Definition at line 114 of file [KinematicDifferentialDrive.hpp](#).  
00114 {};

### 5.68.3.10 m\_radius

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_radius {} [private]
```

The radius of the drive.

Definition at line 120 of file [KinematicDifferentialDrive.hpp](#).  
00120 {};

### 5.68.3.11 m\_moment\_of\_inertia

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_moment_of_inertia {}  
[private]
```

The moment of inertia of the robot.

Definition at line 126 of file [KinematicDifferentialDrive.hpp](#).  
00126 {};

### 5.68.3.12 m\_gear\_ratio

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_gear_ratio {} [private]
```

The gear ratio from the motors to the drive (drive gear / motor gear)

Definition at line 132 of file [KinematicDifferentialDrive.hpp](#).

```
00132 {};
```

### 5.68.3.13 m\_wheel\_radius

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_wheel_radius {} [private]
```

The radius of the drive wheels.

Definition at line 138 of file [KinematicDifferentialDrive.hpp](#).

```
00138 {};
```

### 5.68.3.14 c1

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c1 {} [private]
```

The first kinematic constant.

Definition at line 144 of file [KinematicDifferentialDrive.hpp](#).

```
00144 {};
```

### 5.68.3.15 c2

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c2 {} [private]
```

The second kinematic constant.

Definition at line 150 of file [KinematicDifferentialDrive.hpp](#).

```
00150 {};
```

### 5.68.3.16 c3

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c3 {} [private]
```

The third kinematic constant.

Definition at line 156 of file [KinematicDifferentialDrive.hpp](#).

```
00156 {};
```

### 5.68.3.17 c4

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c4 {} [private]
```

The fourth kinematic constant.

Definition at line 162 of file [KinematicDifferentialDrive.hpp](#).

```
00162 {};
```

### 5.68.3.18 c5

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c5 {} [private]
```

The fifth kinematic constant.

Definition at line 168 of file [KinematicDifferentialDrive.hpp](#).

```
00168 {};
```

### 5.68.3.19 c6

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c6 {} [private]
```

The sixth kinematic constant.

Definition at line 174 of file [KinematicDifferentialDrive.hpp](#).

```
00174 {};
```

### 5.68.3.20 c7

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c7 {} [private]
```

The seventh kinematic constant.

Definition at line 180 of file [KinematicDifferentialDrive.hpp](#).

```
00180 {};
```

### 5.68.3.21 m\_velocity

```
Velocity wisco::robot::subsystems::drive::KinematicDifferentialDrive::m_velocity {} [private]
```

The target velocity for the drive.

Definition at line 186 of file [KinematicDifferentialDrive.hpp](#).

```
00186 {};
```

## 5.69 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder Class Reference

Builder class for the kinematic differential drive class.

## Public Member Functions

- `KinematicDifferentialDriveBuilder * withDelayer (std::unique_ptr< rtos::IDelay > &delayer)`  
*Add an rtos delayer to the build.*
- `KinematicDifferentialDriveBuilder * withMutex (std::unique_ptr< rtos::IMutex > &mutex)`  
*Add an os mutex to the build.*
- `KinematicDifferentialDriveBuilder * withTask (std::unique_ptr< rtos::ITask > &task)`  
*Add an rtos task handler to the build.*
- `KinematicDifferentialDriveBuilder * withLeftVelocityProfile (std::unique_ptr< IVelocityProfile > &left_velocity_profile)`  
*Adds a left velocity profile to the build.*
- `KinematicDifferentialDriveBuilder * withRightVelocityProfile (std::unique_ptr< IVelocityProfile > &right_velocity_profile)`  
*Adds a right velocity profile to the build.*
- `KinematicDifferentialDriveBuilder * withLeftMotor (std::unique_ptr< io::IMotor > &left_motor)`  
*Add a left drive motor to the build.*
- `KinematicDifferentialDriveBuilder * withRightMotor (std::unique_ptr< io::IMotor > &right_motor)`  
*Add a right drive motor to the build.*
- `KinematicDifferentialDriveBuilder * withMass (double mass)`  
*Add the mass to the build.*
- `KinematicDifferentialDriveBuilder * withRadius (double radius)`  
*Add the radius to the build.*
- `KinematicDifferentialDriveBuilder * withMomentOfInertia (double moment_of_inertia)`  
*Add the moment of inertia to the build.*
- `KinematicDifferentialDriveBuilder * withGearRatio (double gear_ratio)`  
*Add the gear ratio to the build.*
- `KinematicDifferentialDriveBuilder * withWheelRadius (double wheel_radius)`  
*Add the wheel radius to the build.*
- `std::unique_ptr< IDifferentialDrive > build ()`  
*Builds the differential drive system.*

## Private Attributes

- `std::unique_ptr< rtos::IDelay > m_delayer {}`  
*The system delayer.*
- `std::unique_ptr< rtos::IMutex > m_mutex {}`  
*The os mutex.*
- `std::unique_ptr< rtos::ITask > m_task {}`  
*The task handler.*
- `std::unique_ptr< IVelocityProfile > m_left_velocity_profile {}`  
*The left velocity profile.*
- `std::unique_ptr< IVelocityProfile > m_right_velocity_profile {}`  
*The right velocity profile.*
- `hal::MotorGroup m_left_motors {}`  
*The left motors on the differential drive.*
- `hal::MotorGroup m_right_motors {}`  
*The right motors on the differential drive.*
- `double m_mass {}`  
*The mass of the robot.*
- `double m_radius {}`  
*The radius of the drive.*

- double `m_moment_of_inertia {}`  
*The moment of inertia of the robot.*
- double `m_gear_ratio {}`  
*The gear ratio from the motors to the drive (drive gear / motor gear)*
- double `m_wheel_radius {}`  
*The radius of the drive wheels.*

## 5.69.1 Detailed Description

Builder class for the kinematic differential drive class.

### Author

Nathan Sandvig

Definition at line 45 of file [KinematicDifferentialDriveBuilder.hpp](#).

## 5.69.2 Member Function Documentation

### 5.69.2.1 withDelayer()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withDelayer (
    std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

#### Parameters

<code>delayer</code>	The rtos delayer
----------------------	------------------

#### Returns

`KinematicDifferentialDriveBuilder*` This object for build chaining

Definition at line 11 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00012 {
00013     m_delayer = std::move(delayer);
00014     return this;
00015 }
```

### 5.69.2.2 withMutex()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Add an os mutex to the build.

**Parameters**

<i>mutex</i>	The os mutex
--------------	--------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 17 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00018 {  
00019     m_mutex = std::move(mutex);  
00020     return this;  
00021 }
```

### 5.69.2.3 withTask()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withTask (  
    std::unique_ptr< rtos::ITask > & task )
```

Add an rtos task handler to the build.

**Parameters**

<i>task</i>	The rtos task handler
-------------	-----------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 23 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00024 {  
00025     m_task = std::move(task);  
00026     return this;  
00027 }
```

### 5.69.2.4 withLeftVelocityProfile()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withLeftVelocityProfile (  
    std::unique_ptr< IVelocityProfile > & left_velocity_profile )
```

Adds a left velocity profile to the build.

**Parameters**

<i>left_velocity_profile</i>	The left velocity profile
------------------------------	---------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 29 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00030 {
00031     m_left_velocity_profile = std::move(left_velocity_profile);
00032     return this;
00033 }
```

### 5.69.2.5 withRightVelocityProfile()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withRightVelocityProfile (
    std::unique_ptr< IVelocityProfile > & right_velocity_profile )
```

Adds a right velocity profile to the build.

#### Parameters

<i>right_velocity_profile</i>	The right velocity profile
-------------------------------	----------------------------

#### Returns

`KinematicDifferentialDriveBuilder*` This object for build chaining

Definition at line 35 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00036 {
00037     m_right_velocity_profile = std::move(right_velocity_profile);
00038     return this;
00039 }
```

### 5.69.2.6 withLeftMotor()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withLeftMotor (
    std::unique_ptr< io::IMotor > & left_motor )
```

Add a left drive motor to the build.

#### Parameters

<i>left_motor</i>	The motor on the left side of the drive
-------------------	---

#### Returns

`KinematicDifferentialDriveBuilder*` This object for build chaining

Definition at line 41 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00042 {
00043     m_left_motors.addMotor(left_motor);
00044     return this;
00045 }
```

### 5.69.2.7 withRightMotor()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withRightMotor (
    std::unique_ptr< io::IMotor > & right_motor )
```

Add a right drive motor to the build.

#### Parameters

<i>right_motor</i>	The motor on the right side of the drive
--------------------	--

#### Returns

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 47 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00048 {  
00049     m_right_motors.addMotor(right_motor);  
00050     return this;  
00051 }
```

### 5.69.2.8 withMass()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withMass (double mass)
```

Add the mass to the build.

#### Parameters

<i>mass</i>	The mass of the drive
-------------	-----------------------

#### Returns

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 53 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00054 {  
00055     m_mass = mass;  
00056     return this;  
00057 }
```

### 5.69.2.9 withRadius()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withRadius (double radius)
```

Add the radius to the build.

#### Parameters

<i>radius</i>	The radius of the drive
---------------	-------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 59 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00060 {
00061     m_radius = radius;
00062     return this;
00063 }
```

**5.69.2.10 withMomentOfInertia()**

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential<-
DriveBuilder::withMomentOfInertia (
    double moment_of_inertia )
```

Add the moment of inertia to the build.

**Parameters**

<i>moment_of_inertia</i>	The moment of inertia of the drive
--------------------------	------------------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 65 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00066 {
00067     m_moment_of_inertia = moment_of_inertia;
00068     return this;
00069 }
```

**5.69.2.11 withGearRatio()**

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential<-
DriveBuilder::withGearRatio (
    double gear_ratio )
```

Add the gear ratio to the build.

**Parameters**

<i>gear_ratio</i>	The gear ratio of the drive
-------------------	-----------------------------

**Returns**

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 71 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00072 {
00073     m_gear_ratio = gear_ratio;
00074     return this;
00075 }
```

### 5.69.2.12 withWheelRadius()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::withWheelRadius (
    double wheel_radius )
```

Add the wheel radius to the build.

#### Parameters

<i>wheel_radius</i>	The wheel radius of the drive
---------------------	-------------------------------

#### Returns

KinematicDifferentialDriveBuilder\* This object for build chaining

Definition at line 77 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00078 {
00079     m_wheel_radius = wheel_radius;
00080     return this;
00081 }
```

### 5.69.2.13 build()

```
std::unique_ptr< IDifferentialDrive > wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::build ()
```

Builds the differential drive system.

#### Returns

std::unique\_ptr<IDifferentialDrive> The differential drive system as a differential drive interface

Definition at line 83 of file [KinematicDifferentialDriveBuilder.cpp](#).

```
00084 {
00085     std::unique_ptr<KinematicDifferentialDrive>
00086         differential_drive{std::make_unique<KinematicDifferentialDrive>()};
00087     differential_drive->setDelayer(m_delayer);
00088     differential_drive->setMutex(m_mutex);
00089     differential_drive->setTask(m_task);
00090     differential_drive->setVelocityProfiles(m_left_velocity_profile, m_right_velocity_profile);
00091     differential_drive->setLeftMotors(m_left_motors);
00092     differential_drive->setRightMotors(m_right_motors);
00093     differential_drive->setMass(m_mass);
00094     differential_drive->setRadius(m_radius);
00095     differential_drive->setMomentOfInertia(m_moment_of_inertia);
00096     differential_drive->setGearRatio(m_gear_ratio);
00097     differential_drive->setWheelRadius(m_wheel_radius);
00098     return differential_drive;
00099 }
```

## 5.69.3 Member Data Documentation

### 5.69.3.1 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_delayer {} [private]
```

The system delayer.

Definition at line 52 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00052 {};
```

### 5.69.3.2 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::drive::KinematicDifferentialDrive<→
Builder::m_mutex {} [private]
```

The os mutex.

Definition at line 58 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00058 {};

### 5.69.3.3 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::drive::KinematicDifferentialDrive<→
Builder::m_task {} [private]
```

The task handler.

Definition at line 64 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00064 {};

### 5.69.3.4 m\_left\_velocity\_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferential<→
DriveBuilder::m_left_velocity_profile {} [private]
```

The left velocity profile.

Definition at line 70 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00070 {};

### 5.69.3.5 m\_right\_velocity\_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferential<→
DriveBuilder::m_right_velocity_profile {} [private]
```

The right velocity profile.

Definition at line 76 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00076 {};

### 5.69.3.6 m\_left\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_left<→
motors {} [private]
```

The left motors on the differential drive.

Definition at line 82 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00082 {};

### 5.69.3.7 m\_right\_motors

```
hal::MotorGroup wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_right_←  
motors {} [private]
```

The right motors on the differential drive.

Definition at line 88 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00088 {};
```

### 5.69.3.8 m\_mass

```
double wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_mass {} [private]
```

The mass of the robot.

Definition at line 94 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00094 {};
```

### 5.69.3.9 m\_radius

```
double wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_radius {} [private]
```

The radius of the drive.

Definition at line 100 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00100 {};
```

### 5.69.3.10 m\_moment\_of\_inertia

```
double wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_moment_of_inertia  
{} [private]
```

The moment of inertia of the robot.

Definition at line 106 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00106 {};
```

### 5.69.3.11 m\_gear\_ratio

```
double wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_gear_ratio {}  
[private]
```

The gear ratio from the motors to the drive (drive gear / motor gear)

Definition at line 112 of file [KinematicDifferentialDriveBuilder.hpp](#).

```
00112 {};
```

### 5.69.3.12 m\_wheel\_radius

```
double wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder::m_wheel_radius {}  
[private]
```

The radius of the drive wheels.

Definition at line 118 of file [KinematicDifferentialDriveBuilder.hpp](#).  
00118 {};

## 5.70 wisco::robot::subsystems::drive::Velocity Struct Reference

Holds the velocity values for the drive.

### Public Attributes

- double [left\\_velocity](#) {}  
*The velocity for the left side of the drive.*
- double [right\\_velocity](#) {}  
*The velocity for the right side of the drive.*

### 5.70.1 Detailed Description

Holds the velocity values for the drive.

#### Author

Nathan Sandvig

Definition at line 41 of file [Velocity.hpp](#).

### 5.70.2 Member Data Documentation

#### 5.70.2.1 left\_velocity

```
double wisco::robot::subsystems::drive::Velocity::left_velocity {}
```

The velocity for the left side of the drive.

Definition at line 47 of file [Velocity.hpp](#).  
00047 {};

#### 5.70.2.2 right\_velocity

```
double wisco::robot::subsystems::drive::Velocity::right_velocity {}
```

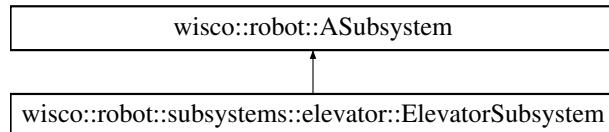
The velocity for the right side of the drive.

Definition at line 53 of file [Velocity.hpp](#).  
00053 {};

## 5.71 wisco::robot::subsystems::elevator::ElevatorSubsystem Class Reference

The subsystem adapter for elevators.

Inheritance diagram for wisco::robot::subsystems::elevator::ElevatorSubsystem:



### Public Member Functions

- **ElevatorSubsystem** (std::unique\_ptr< IElevator > &elevator, std::unique\_ptr< io::IDistanceSensor > &distance\_sensor)
   
*Construct a new ELEVATOR Subsystem object.*
- void **initialize** () override
   
*Initializes the subsystem.*
- void **run** () override
   
*Runs the subsystem.*
- void **command** (std::string command\_name, va\_list &args) override
   
*Runs a command for the subsystem.*
- void \* **state** (std::string state\_name) override
   
*Gets a state of the subsystem.*

### Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem** ()=default
   
*Construct a new ASubsystem object.*
- **ASubsystem** (const [ASubsystem](#) &other)=default
   
*Construct a new ASubsystem object.*
- **ASubsystem** ([ASubsystem](#) &&other)=default
   
*Construct a new ASubsystem object.*
- **ASubsystem** (std::string name)
   
*Construct a new ASubsystem object.*
- virtual ~**ASubsystem** ()=default
   
*Destroy the ASubsystem object.*
- const std::string & **getName** () const
   
*Get the name of the subsystem.*
- **ASubsystem** & **operator=** (const [ASubsystem](#) &rhs)=default
   
*Copy assignment operator for ASubsystem.*
- **ASubsystem** & **operator=** ([ASubsystem](#) &&rhs)=default
   
*Move assignment operator for ASubsystem.*

## Private Attributes

- std::unique\_ptr< [IElevator](#) > m\_elevator {}  
*The elevator being adapted.*
- std::unique\_ptr< [io::IDistanceSensor](#) > m\_distance\_sensor {}  
*The distance sensor being adapted.*

## Static Private Attributes

- static constexpr char [SUBSYSTEM\\_NAME](#) [] {"ELEVATOR"}  
*The name of the subsystem.*
- static constexpr char [SET\\_POSITION\\_COMMAND\\_NAME](#) [] {"SET POSITION"}  
*The name of the set velocity command.*
- static constexpr char [GET\\_POSITION\\_STATE\\_NAME](#) [] {"GET POSITION"}  
*The name of the get velocity command.*
- static constexpr char [CAP\\_DISTANCE\\_STATE\\_NAME](#) [] {"CAP DISTANCE"}  
*The name of the cap distance state.*

### 5.71.1 Detailed Description

The subsystem adapter for elevators.

#### Author

Nathan Sandvig

Definition at line 47 of file [ElevatorSubsystem.hpp](#).

### 5.71.2 Constructor & Destructor Documentation

#### 5.71.2.1 ElevatorSubsystem()

```
wisco::robot::subsystems::elevator::ElevatorSubsystem::ElevatorSubsystem (
    std::unique_ptr< IElevator > & elevator,
    std::unique_ptr< io::IDistanceSensor > & distance_sensor )
```

Construct a new ELEVATOR Subsystem object.

#### Parameters

<i>elevator</i>	The elevator being adapted
<i>distance_sensor</i>	The distance sensor being adapted

Definition at line 11 of file [ElevatorSubsystem.cpp](#).

```
00012 : ASubsystem{SUBSYSTEM\_NAME}, m_elevator{std::move(elevator)},
        m_distance_sensor{std::move(distance_sensor)}
00013 {
00014
00015 }
```

## 5.71.3 Member Function Documentation

### 5.71.3.1 initialize()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::initialize () [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [ElevatorSubsystem.cpp](#).

```
00018 {
00019     if (m_elevator)
00020         m_elevator->initialize();
00021     if (m_distance_sensor)
00022         m_distance_sensor->initialize();
00023 }
```

### 5.71.3.2 run()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::run () [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 25 of file [ElevatorSubsystem.cpp](#).

```
00026 {
00027     if (m_elevator)
00028         m_elevator->run();
00029 }
```

### 5.71.3.3 command()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

#### Parameters

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 31 of file [ElevatorSubsystem.cpp](#).

```
00032 {
00033     if (command_name == SET_POSITION_COMMAND_NAME)
00034     {
00035         double position{va_arg(args, double)};
00036         m_elevator->setPosition(position);
00037     }
00038 }
```

### 5.71.3.4 state()

```
void * wisco::robot::subsystems::elevator::ElevatorSubsystem::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the subsystem.

#### Parameters

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 40 of file [ElevatorSubsystem.cpp](#).

```
00041 {
00042     void* result=nullptr;
00043
00044     if (state_name == GET_POSITION_STATE_NAME)
00045     {
00046         double* position{new double{m_elevator->getPosition()}};
00047         result = position;
00048     }
00049     else if (state_name == CAP_DISTANCE_STATE_NAME)
00050     {
00051         if (m_distance_sensor)
00052         {
00053             double* distance{new double{m_distance_sensor->getDistance()}};
00054             result = distance;
00055         }
00056     }
00057
00058     return result;
00059 }
```

## 5.71.4 Member Data Documentation

### 5.71.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::elevator::ElevatorSubsystem::SUBSYSTEM_NAME[ ] {"ELEVATOR"}  
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 54 of file [ElevatorSubsystem.hpp](#).

```
00054 {"ELEVATOR"};
```

### 5.71.4.2 SET\_POSITION\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::elevator::ElevatorSubsystem::SET_POSITION_COMMAND_NAME[ ] {"SET POSITION"}  
[static], [constexpr], [private]
```

The name of the set velocity command.

Definition at line 60 of file [ElevatorSubsystem.hpp](#).

```
00060 {"SET POSITION"};
```

#### 5.71.4.3 GET\_POSITION\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::elevator::ElevatorSubsystem::GET_POSITION_STATE_←
NAME[ ] {"GET POSITION"} [static], [constexpr], [private]
```

The name of the get velocity command.

Definition at line 66 of file [ElevatorSubsystem.hpp](#).

```
00066 {"GET POSITION"};
```

#### 5.71.4.4 CAP\_DISTANCE\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::elevator::ElevatorSubsystem::CAP_DISTANCE_STATE_←
NAME[ ] {"CAP DISTANCE"} [static], [constexpr], [private]
```

The name of the cap distance state.

Definition at line 72 of file [ElevatorSubsystem.hpp](#).

```
00072 {"CAP DISTANCE"};
```

#### 5.71.4.5 m\_elevator

```
std::unique_ptr<IElevator> wisco::robot::subsystems::elevator::ElevatorSubsystem::m_elevator
{} [private]
```

The elevator being adapted.

Definition at line 78 of file [ElevatorSubsystem.hpp](#).

```
00078 {};
```

#### 5.71.4.6 m\_distance\_sensor

```
std::unique_ptr<io::IDistanceSensor> wisco::robot::subsystems::elevator::ElevatorSubsystem::m_distance_sensor
{} [private]
```

The distance sensor being adapted.

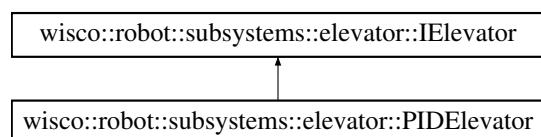
Definition at line 84 of file [ElevatorSubsystem.hpp](#).

```
00084 {};
```

## 5.72 wisco::robot::subsystems::elevator::IElevator Class Reference

Interface for elevators.

Inheritance diagram for wisco::robot::subsystems::elevator::IElevator:



## Public Member Functions

- virtual ~**IxElevator** ()=default  
*Destroy the IxElevator object.*
- virtual void **initialize** ()=0  
*Initializes the elevator.*
- virtual void **run** ()=0  
*Runs the elevator.*
- virtual double **getPosition** ()=0  
*Get the position of the elevator in inches.*
- virtual void **setPosition** (double position)=0  
*Set the position of the elevator in inches.*

### 5.72.1 Detailed Description

Interface for elevators.

#### Author

Nathan Sandvig

Definition at line 41 of file [IxElevator.hpp](#).

### 5.72.2 Member Function Documentation

#### 5.72.2.1 initialize()

```
virtual void wisco::robot::subsystems::elevator::IElevator::initialize () [pure virtual]
```

Initializes the elevator.

Implemented in [wisco::robot::subsystems::elevator::PIDElevator](#).

#### 5.72.2.2 run()

```
virtual void wisco::robot::subsystems::elevator::IElevator::run () [pure virtual]
```

Runs the elevator.

Implemented in [wisco::robot::subsystems::elevator::PIDElevator](#).

#### 5.72.2.3 getPosition()

```
virtual double wisco::robot::subsystems::elevator::IElevator::getPosition () [pure virtual]
```

Get the position of the elevator in inches.

#### Returns

double The elevator position in inches

Implemented in [wisco::robot::subsystems::elevator::PIDElevator](#).

#### 5.72.2.4 setPosition()

```
virtual void wisco::robot::subsystems::elevator::IElevator::setPosition (
    double position ) [pure virtual]
```

Set the position of the elevator in inches.

**Parameters**

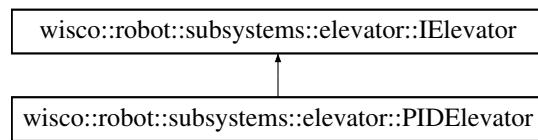
<i>position</i>	The elevator position in inches
-----------------	---------------------------------

Implemented in [wisco::robot::subsystems::elevator::PIDElevator](#).

## 5.73 wisco::robot::subsystems::elevator::PIDElevator Class Reference

An elevator controller with PID position control.

Inheritance diagram for wisco::robot::subsystems::elevator::PIDElevator:



### Public Member Functions

- void [\*\*initialize\*\*](#) () override  
*Initializes the intake.*
- void [\*\*run\*\*](#) () override  
*Runs the intake.*
- double [\*\*getPosition\*\*](#) () override  
*Get the position of the elevator in inches.*
- void [\*\*setPosition\*\*](#) (double position) override  
*Set the position of the elevator in inches.*
- void [\*\*setClock\*\*](#) (const std::unique\_ptr<[rtos::IClock](#)> &clock)  
*Set the rtos clock.*
- void [\*\*setDelayer\*\*](#) (const std::unique\_ptr<[rtos::IDelayer](#)> &delayer)  
*Set the rtos delayer.*
- void [\*\*setMutex\*\*](#) (std::unique\_ptr<[rtos::IMutex](#)> &mutex)  
*Set the thread mutex.*
- void [\*\*setTask\*\*](#) (std::unique\_ptr<[rtos::ITask](#)> &task)  
*Set the task handler.*
- void [\*\*setPID\*\*](#) ([control::PID](#) pid)  
*Set the PID controller.*
- void [\*\*setMotors\*\*](#) ([hal::MotorGroup](#) &motors)  
*Set the motors.*
- void [\*\*setRotationSensor\*\*](#) (std::unique\_ptr<[io::IRotationSensor](#)> &rotation\_sensor)  
*Set the rotation sensor.*
- void [\*\*setInchesPerRadian\*\*](#) (double inches\_per\_radian)  
*Set the inches per radian of the elevator.*

### Public Member Functions inherited from [wisco::robot::subsystems::elevator::IElevator](#)

- virtual ~[IElevator](#) ()=default  
*Destroy the [IElevator](#) object.*

### Private Member Functions

- void `taskUpdate ()`  
*Runs all the object-specific updates in the task loop.*
- void `updatePosition ()`  
*Updates the elevator position.*

### Static Private Member Functions

- static void `taskLoop (void *params)`  
*The task loop function for background updates.*

### Private Attributes

- `std::unique_ptr< rtos::IClock > m_clock {}`  
*The rtos clock.*
- `std::unique_ptr< rtos::IDelay > m_delayer {}`  
*The rtos delayer.*
- `std::unique_ptr< rtos::IMutex > m_mutex {}`  
*The mutex for thread safety.*
- `std::unique_ptr< rtos::ITask > m_task {}`  
*The background task handler.*
- `control::PID m_pid {}`  
*The position PID controller.*
- `hal::MotorGroup m_motors {}`  
*The motors on the elevator.*
- `std::unique_ptr< io::IRotationSensor > m_rotation_sensor {}`  
*The rotation sensor on the elevator.*
- double `m_inches_per_radian {}`  
*The number of movement inches per radian.*
- double `m_position {}`  
*The position setting of the elevator.*

### Static Private Attributes

- static constexpr uint8\_t `TASK_DELAY {10}`  
*The loop delay on the task.*

## 5.73.1 Detailed Description

An elevator controller with PID position control.

### Author

Nathan Sandvig

Definition at line 53 of file [PIDElevator.hpp](#).

## 5.73.2 Member Function Documentation

### 5.73.2.1 `taskLoop()`

```
void wisco::robot::subsystems::elevator::PIDElevator::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

**Parameters**

<i>params</i>	
---------------	--

Definition at line 11 of file PIDElevator.cpp.

```
00012 {
00013     void** parameters{static_cast<void**>(params)};
00014     PIDElevator* instance{static_cast<PIDElevator*>(parameters[0])};
00015
00016     while (true)
00017     {
00018         instance->taskUpdate();
00019     }
00020 }
```

**5.73.2.2 taskUpdate()**

```
void wisco::robot::subsystems::elevator::PIDElevator::taskUpdate () [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 22 of file PIDElevator.cpp.

```
00023 {
00024     updatePosition();
00025     m_delayer->delay(TASK_DELAY);
00026 }
```

**5.73.2.3 updatePosition()**

```
void wisco::robot::subsystems::elevator::PIDElevator::updatePosition () [private]
```

Updates the elevator position.

Definition at line 28 of file PIDElevator.cpp.

```
00029 {
00030     if (m_mutex)
00031         m_mutex->take();
00032
00033     double current_position{getPosition()};
00034
00035     double voltage{m_pid.getControlValue(current_position, m_position)};
00036     m_motors.setVoltage(voltage);
00037
00038     if (m_mutex)
00039         m_mutex->give();
00040 }
```

**5.73.2.4 initialize()**

```
void wisco::robot::subsystems::elevator::PIDElevator::initialize () [override], [virtual]
```

Initializes the intake.

Implements [wisco::robot::subsystems::elevator::IElevator](#).

Definition at line 42 of file PIDElevator.cpp.

```
00043 {
00044     m_pid.reset();
00045     m_motors.initialize();
00046     if (m_rotation_sensor)
00047         m_rotation_sensor->initialize();
00048 }
```

### 5.73.2.5 run()

```
void wisco::robot::subsystems::elevator::PIDElevator::run ( ) [override], [virtual]
```

Runs the intake.

Implements [wisco::robot::subsystems::elevator::IElevator](#).

Definition at line 50 of file [PIDElevator.cpp](#).

```
00051 {
00052     if (m_task)
00053     {
00054         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};
00055         params[0] = this;
00056         m_task->start(&PIDElevator::taskLoop, params);
00057     }
00058 }
```

### 5.73.2.6 getPosition()

```
double wisco::robot::subsystems::elevator::PIDElevator::getPosition ( ) [override], [virtual]
```

Get the position of the elevator in inches.

#### Returns

`double` The elevator position

Implements [wisco::robot::subsystems::elevator::IElevator](#).

Definition at line 60 of file [PIDElevator.cpp](#).

```
00061 {
00062     double position{};
00063     if (m_rotation_sensor)
00064     {
00065         position = m_rotation_sensor->getRotation() * m_inches_per_radian;
00066     }
00067     else
00068     {
00069         position = m_motors.getPosition() * m_inches_per_radian;
00070     }
00071     return position;
00072 }
```

### 5.73.2.7 setPosition()

```
void wisco::robot::subsystems::elevator::PIDElevator::setPosition (
    double position ) [override], [virtual]
```

Set the position of the elevator in inches.

#### Parameters

<code>position</code>	The position of the elevator
-----------------------	------------------------------

Implements [wisco::robot::subsystems::elevator::IElevator](#).

Definition at line 74 of file [PIDElevator.cpp](#).

```

00075 {
00076     if (m_mutex)
00077         m_mutex->take();
00078
00079     m_position = position;
00080
00081     if (m_mutex)
00082         m_mutex->give();
00083 }
```

### 5.73.2.8 setClock()

```
void wisco::robot::subsystems::elevator::PIDElevator::setClock (
    const std::unique_ptr< rtos::IClock > & clock )
```

Set the rtos clock.

#### Parameters

<i>clock</i>	The rtos clock
--------------	----------------

Definition at line 85 of file PIDElevator.cpp.

```

00086 {
00087     m_clock = clock->clone();
00088 }
```

### 5.73.2.9 setDelayer()

```
void wisco::robot::subsystems::elevator::PIDElevator::setDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 90 of file PIDElevator.cpp.

```

00091 {
00092     m_delayer = delayer->clone();
00093 }
```

### 5.73.2.10 setMutex()

```
void wisco::robot::subsystems::elevator::PIDElevator::setMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Set the thread mutex.

#### Parameters

<i>mutex</i>	The thread mutex
--------------	------------------

Definition at line 95 of file PIDElevator.cpp.

```
00096 {
00097     m_mutex = std::move(mutex);
00098 }
```

### 5.73.2.11 setTask()

```
void wisco::robot::subsystems::elevator::PIDElevator::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Set the task handler.

#### Parameters

<i>task</i>	The task handler
-------------	------------------

Definition at line 100 of file [PIDElevator.cpp](#).

```
00101 {
00102     m_task = std::move(task);
00103 }
```

### 5.73.2.12 setPID()

```
void wisco::robot::subsystems::elevator::PIDElevator::setPID (
    control::PID pid )
```

Set the PID controller.

#### Parameters

<i>pid</i>	The PID controller
------------	--------------------

Definition at line 105 of file [PIDElevator.cpp](#).

```
00106 {
00107     m_pid = pid;
00108 }
```

### 5.73.2.13 setMotors()

```
void wisco::robot::subsystems::elevator::PIDElevator::setMotors (
    hal::MotorGroup & motors )
```

Set the motors.

#### Parameters

<i>motors</i>	The motors
---------------	------------

Definition at line 110 of file [PIDElevator.cpp](#).

```
00111 {
00112     m_motors = motors;
00113 }
```

### 5.73.2.14 setRotationSensor()

```
void wisco::robot::subsystems::elevator::PIDElevator::setRotationSensor (
    std::unique_ptr< io::IRotationSensor > & rotation_sensor )
```

Set the rotation sensor.

#### Parameters

<i>rotation_sensor</i>	The rotation sensor
------------------------	---------------------

Definition at line 115 of file PIDElevator.cpp.

```
00116 {
00117     m_rotation_sensor = std::move(rotation_sensor);
00118 }
```

### 5.73.2.15 setInchesPerRadian()

```
void wisco::robot::subsystems::elevator::PIDElevator::setInchesPerRadian (
    double inches_per_radian )
```

Set the inches per radian of the elevator.

#### Parameters

<i>inches_per_radian</i>	The inches per radian of the elevator
--------------------------	---------------------------------------

Definition at line 120 of file PIDElevator.cpp.

```
00121 {
00122     m_inches_per_radian = inches_per_radian;
00123 }
```

## 5.73.3 Member Data Documentation

### 5.73.3.1 TASK\_DELAY

```
constexpr uint8_t wisco::robot::subsystems::elevator::PIDElevator::TASK_DELAY {10} [static],
[constexpr], [private]
```

The loop delay on the task.

Definition at line 60 of file PIDElevator.hpp.  
00060 {10};

### 5.73.3.2 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::elevator::PIDElevator::m_clock {}
[private]
```

The rtos clock.

Definition at line 73 of file PIDElevator.hpp.  
00073 {};

### 5.73.3.3 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::elevator::PIDElevator::m_delayer {}  
[private]
```

The rtos delayer.

Definition at line 79 of file [PIDElevator.hpp](#).  
00079 {};

### 5.73.3.4 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::elevator::PIDElevator::m_mutex {}  
[private]
```

The mutex for thread safety.

Definition at line 85 of file [PIDElevator.hpp](#).  
00085 {};

### 5.73.3.5 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::elevator::PIDElevator::m_task {} [private]
```

The background task handler.

Definition at line 91 of file [PIDElevator.hpp](#).  
00091 {};

### 5.73.3.6 m\_pid

```
control::PID wisco::robot::subsystems::elevator::PIDElevator::m_pid {} [private]
```

The position PID controller.

Definition at line 97 of file [PIDElevator.hpp](#).  
00097 {};

### 5.73.3.7 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::elevator::PIDElevator::m_motors {} [private]
```

The motors on the elevator.

Definition at line 103 of file [PIDElevator.hpp](#).  
00103 {};

### 5.73.3.8 m\_rotation\_sensor

```
std::unique_ptr<io::IRotationSensor> wisco::robot::subsystems::elevator::PIDElevator::m_<-
rotation_sensor {} [private]
```

The rotation sensor on the elevator.

Definition at line 109 of file [PIDElevator.hpp](#).  
00109 {};

### 5.73.3.9 m\_inches\_per\_radian

```
double wisco::robot::subsystems::elevator::PIDElevator::m_inches_per_radian {} [private]
```

The number of movement inches per radian.

Definition at line 115 of file [PIDElevator.hpp](#).  
00115 {};

### 5.73.3.10 m\_position

```
double wisco::robot::subsystems::elevator::PIDElevator::m_position {} [private]
```

The position setting of the elevator.

Definition at line 121 of file [PIDElevator.hpp](#).  
00121 {};

## 5.74 wisco::robot::subsystems::elevator::PIDElevatorBuilder Class Reference

Builder class for a pid-based elevator system.

### Public Member Functions

- [PIDElevatorBuilder \\* withClock](#) (const std::unique\_ptr<[rtos::IClock](#)> &clock)  
*Add an rtos clock to the build.*
- [PIDElevatorBuilder \\* withDelayer](#) (const std::unique\_ptr<[rtos::IDelayer](#)> &delayer)  
*Add an rtos delayer to the build.*
- [PIDElevatorBuilder \\* withMutex](#) (std::unique\_ptr<[rtos::IMutex](#)> &mutex)  
*Add a thread mutex to the build.*
- [PIDElevatorBuilder \\* withTask](#) (std::unique\_ptr<[rtos::ITask](#)> &task)  
*Add a task handler to the build.*
- [PIDElevatorBuilder \\* withPID](#) ([control::PID](#) pid)  
*Add a PID controller to the build.*
- [PIDElevatorBuilder \\* withMotor](#) (std::unique\_ptr<[io::IMotor](#)> &motor)  
*Add a motor to the build.*
- [PIDElevatorBuilder \\* withRotationSensor](#) (std::unique\_ptr<[io::IRotationSensor](#)> &rotation\_sensor)  
*Add a rotation sensor to the build.*
- [PIDElevatorBuilder \\* withInchesPerRadian](#) (double inches\_per\_radian)  
*Add an inches per radian constant to the build.*
- [std::unique\\_ptr<\[IElevator\]\(#\)> build \(\)](#)  
*Builds the elevator.*

## Private Attributes

- std::unique\_ptr<rtos::IClock> m\_clock {}  
*The rtos clock.*
- std::unique\_ptr<rtos::IDelay> m\_delay {}  
*The rtos delayer.*
- std::unique\_ptr<rtos::IMutex> m\_mutex {}  
*The mutex for thread safety.*
- std::unique\_ptr<rtos::ITask> m\_task {}  
*The background task handler.*
- control::PID m\_pid {}  
*The position PID controller.*
- hal::MotorGroup m\_motors {}  
*The motors on the elevator.*
- std::unique\_ptr<io::IRotationSensor> m\_rotation\_sensor {}  
*The rotation sensor on the elevator.*
- double m\_inches\_per\_radian {}  
*The number of movement inches per radian.*

## 5.74.1 Detailed Description

Builder class for a pid-based elevator system.

### Author

Nathan Sandvig

Definition at line 45 of file [PIDElevatorBuilder.hpp](#).

## 5.74.2 Member Function Documentation

### 5.74.2.1 withClock()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withClock (
    const std::unique_ptr<rtos::IClock> & clock )
```

Add an rtos clock to the build.

#### Parameters

<code>clock</code>	The rtos clock
--------------------	----------------

#### Returns

PIDElevatorBuilder\* This object for build chaining

Definition at line 11 of file [PIDElevatorBuilder.cpp](#).

```
00012 {
00013     m_clock = clock->clone();
00014     return this;
00015 }
```

### 5.74.2.2 withDelayer()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withDelayer ( 
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

#### Returns

PIDElevatorBuilder\* This object for build chaining

Definition at line 17 of file [PIDElevatorBuilder.cpp](#).

```
00018 { 
00019     m_delayer = delayer->clone(); 
00020     return this;
00021 }
```

### 5.74.2.3 withMutex()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withMutex ( 
    std::unique_ptr< rtos::IMutex > & mutex )
```

Add a thread mutex to the build.

#### Parameters

<i>mutex</i>	The thread mutex
--------------	------------------

#### Returns

PIDElevatorBuilder\* This object for build chaining

Definition at line 23 of file [PIDElevatorBuilder.cpp](#).

```
00024 { 
00025     m_mutex = std::move(mutex);
00026     return this;
00027 }
```

### 5.74.2.4 withTask()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withTask ( 
    std::unique_ptr< rtos::ITask > & task )
```

Add a task handler to the build.

#### Parameters

<i>task</i>	The task handler
-------------	------------------

**Returns**

PIDElevatorBuilder\* This object for build chaining

**Definition at line 29 of file PIDElevatorBuilder.cpp.**

```
00030 {
00031     m_task = std::move(task);
00032     return this;
00033 }
```

**5.74.2.5 withPID()**

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withPID (
    control::PID pid )
```

Add a PID controller to the build.

**Parameters**

<i>pid</i>	The PID controller
------------	--------------------

**Returns**

PIDElevatorBuilder\* This object for build chaining

**Definition at line 35 of file PIDElevatorBuilder.cpp.**

```
00036 {
00037     m_pid = pid;
00038     return this;
00039 }
```

**5.74.2.6 withMotor()**

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withMotor (
    std::unique_ptr< io::IMotor > & motor )
```

Add a motor to the build.

**Parameters**

<i>motor</i>	The motor
--------------	-----------

**Returns**

PIDElevatorBuilder\* This object for build chaining

**Definition at line 41 of file PIDElevatorBuilder.cpp.**

```
00042 {
00043     m_motors.addMotor(motor);
00044     return this;
00045 }
```

### 5.74.2.7 withRotationSensor()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withRotationSensor ( std::unique_ptr< io::IRotationSensor > & rotation_sensor )
```

Add a rotation sensor to the build.

#### Parameters

<i>rotation_sensor</i>	The rotation sensor
------------------------	---------------------

#### Returns

PIDElevatorBuilder\* This object for build chaining

Definition at line 47 of file PIDElevatorBuilder.cpp.

```
00048 {  
00049     m_rotation_sensor = std::move(rotation_sensor);  
00050     return this;  
00051 }
```

### 5.74.2.8 withInchesPerRadian()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withInchesPerRadian ( double inches_per_radian )
```

Add an inches per radian constant to the build.

#### Parameters

<i>inches_per_radian</i>	The inches per radian of the elevator
--------------------------	---------------------------------------

#### Returns

PIDElevatorBuilder\* This object for build chaining

Definition at line 53 of file PIDElevatorBuilder.cpp.

```
00054 {  
00055     m_inches_per_radian = inches_per_radian;  
00056     return this;  
00057 }
```

### 5.74.2.9 build()

```
std::unique_ptr< IElevator > wisco::robot::subsystems::elevator::PIDElevatorBuilder::build ( )
```

Builds the elevator.

**Returns**

```
std::unique_ptr<IElevator> The PIDElevator object built with the stored data
```

Definition at line 59 of file [PIDElevatorBuilder.cpp](#).

```
00060 {  
00061     std::unique_ptr<PIDElevator> pid_elevator{std::make_unique<PIDElevator>()};  
00062     pid_elevator->setClock(m_clock);  
00063     pid_elevator->setDelayer(m_delayer);  
00064     pid_elevator->setMutex(m_mutex);  
00065     pid_elevator->setTask(m_task);  
00066     pid_elevator->setPID(m_pid);  
00067     pid_elevator->setMotors(m_motors);  
00068     if (m_rotation_sensor)  
00069         pid_elevator->setRotationSensor(m_rotation_sensor);  
00070     pid_elevator->setInchesPerRadian(m_inches_per_radian);  
00071     return pid_elevator;  
00072 }
```

## 5.74.3 Member Data Documentation

### 5.74.3.1 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_clock  
{ } [private]
```

The rtos clock.

Definition at line 52 of file [PIDElevatorBuilder.hpp](#).

```
00052 {};
```

### 5.74.3.2 m\_delayer

```
std::unique_ptr<rtos::IDelay> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_←  
delay {} [private]
```

The rtos delayer.

Definition at line 58 of file [PIDElevatorBuilder.hpp](#).

```
00058 {};
```

### 5.74.3.3 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_mutex  
{ } [private]
```

The mutex for thread safety.

Definition at line 64 of file [PIDElevatorBuilder.hpp](#).

```
00064 {};
```

### 5.74.3.4 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_task {}  
[private]
```

The background task handler.

Definition at line 70 of file [PIDElevatorBuilder.hpp](#).

```
00070 {};
```

### 5.74.3.5 m\_pid

```
control::PID wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_pid {} [private]
```

The position PID controller.

Definition at line 76 of file [PIDElevatorBuilder.hpp](#).

```
00076 {};
```

### 5.74.3.6 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_motors {} [private]
```

The motors on the elevator.

Definition at line 82 of file [PIDElevatorBuilder.hpp](#).

```
00082 {};
```

### 5.74.3.7 m\_rotation\_sensor

```
std::unique_ptr<io::IRotationSensor> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_rotation_sensor {} [private]
```

The rotation sensor on the elevator.

Definition at line 88 of file [PIDElevatorBuilder.hpp](#).

```
00088 {};
```

### 5.74.3.8 m\_inches\_per\_radian

```
double wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_inches_per_radian {} [private]
```

The number of movement inches per radian.

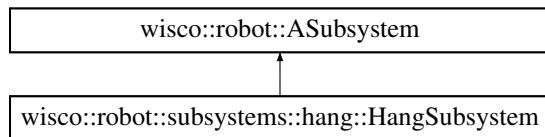
Definition at line 94 of file [PIDElevatorBuilder.hpp](#).

```
00094 {};
```

## 5.75 wisco::robot::subsystems::hang::HangSubsystem Class Reference

The subsystem adapter for hangs.

Inheritance diagram for wisco::robot::subsystems::hang::HangSubsystem:



## Public Member Functions

- **HangSubsystem** (std::unique\_ptr< [IClaw](#) > &claw, std::unique\_ptr< [IToggleArm](#) > &toggle\_arm, std::unique\_ptr< [IWinch](#) > &winch)

*Construct a new Hang Subsystem object.*

- void **initialize** () override

*Initializes the subsystem.*

- void **run** () override

*Runs the subsystem.*

- void **command** (std::string command\_name, va\_list &args) override

*Runs a command for the subsystem.*

- void \* **state** (std::string state\_name) override

*Gets a state of the subsystem.*

## Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem** ()=default

*Construct a new ASubsystem object.*

- **ASubsystem** (const [ASubsystem](#) &other)=default

*Construct a new ASubsystem object.*

- **ASubsystem** ([ASubsystem](#) &&other)=default

*Construct a new ASubsystem object.*

- **ASubsystem** (std::string name)

*Construct a new ASubsystem object.*

- virtual ~**ASubsystem** ()=default

*Destroy the ASubsystem object.*

- const std::string & **getName** () const

*Get the name of the subsystem.*

- **ASubsystem** & **operator=** (const [ASubsystem](#) &rhs)=default

*Copy assignment operator for ASubsystem.*

- **ASubsystem** & **operator=** ([ASubsystem](#) &&rhs)=default

*Move assignment operator for ASubsystem.*

## Private Attributes

- std::unique\_ptr< [IClaw](#) > **m\_claw** {}

*The claw being adapted.*

- std::unique\_ptr< [IToggleArm](#) > **m\_toggle\_arm** {}

*The toggle arm being adapted.*

- std::unique\_ptr< [IWinch](#) > **m\_winch** {}

*The winch being adapted.*

### Static Private Attributes

- static constexpr char `SUBSYSTEM_NAME` [] {"HANG"}  
*The name of the subsystem.*
- static constexpr char `CLOSE_CLAW_COMMAND_NAME` [] {"CLOSE CLAW"}  
*The name of the close claw command.*
- static constexpr char `OPEN_CLAW_COMMAND_NAME` [] {"OPEN CLAW"}  
*The name of the open claw command.*
- static constexpr char `LOWER_ARM_COMMAND_NAME` [] {"LOWER ARM"}  
*The name of the lower arm command.*
- static constexpr char `RAISE_ARM_COMMAND_NAME` [] {"RAISE ARM"}  
*The name of the raise arm command.*
- static constexpr char `ENGAGE_WINCH_COMMAND_NAME` [] {"ENGAGE WINCH"}  
*The name of the engage winch command.*
- static constexpr char `DISENGAGE_WINCH_COMMAND_NAME` [] {"DISENGAGE WINCH"}  
*The name of the disengage winch command.*
- static constexpr char `CLAW_CLOSED_STATE_NAME` [] {"CLAW CLOSED"}  
*The name of the claw closed state.*
- static constexpr char `CLAW_OPEN_STATE_NAME` [] {"CLAW OPEN"}  
*The name of the claw open state.*
- static constexpr char `ARM_DOWN_STATE_NAME` [] {"ARM DOWN"}  
*The name of the arm down state.*
- static constexpr char `ARM_UP_STATE_NAME` [] {"ARM UP"}  
*The name of the arm up state.*
- static constexpr char `WINCH_ENGAGED_STATE_NAME` [] {"WINCH ENGAGED"}  
*The name of the winch engaged state.*
- static constexpr char `WINCH_DISENGAGED_STATE_NAME` [] {"WINCH DISENGAGED"}  
*The name of the winch disengaged state.*

### 5.75.1 Detailed Description

The subsystem adapter for hangs.

#### Author

Nathan Sandvig

Definition at line 50 of file [HangSubsystem.hpp](#).

### 5.75.2 Constructor & Destructor Documentation

#### 5.75.2.1 HangSubsystem()

```
wisco::robot::subsystems::hang::HangSubsystem::HangSubsystem (
    std::unique_ptr< IClaw > & claw,
    std::unique_ptr< IToggleArm > & toggle_arm,
    std::unique_ptr< IWinch > & winch )
```

Construct a new Hang Subsystem object.

### Parameters

<i>claw</i>	The claw being adapted
<i>toggle_arm</i>	The toggle arm being adapted
<i>winch</i>	The winch being adapted

Definition at line 11 of file [HangSubsystem.cpp](#).

```
00014     : ASubsystem{SUBSYSTEM_NAME},
00015     m_claw{std::move(claw)},
00016     m_toggle_arm{std::move(toggle_arm)},
00017     m_winch{std::move(winch)}
00018 {
00019
00020 }
```

### 5.75.3 Member Function Documentation

#### 5.75.3.1 initialize()

```
void wisco::robot::subsystems::hang::HangSubsystem::initialize () [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 22 of file [HangSubsystem.cpp](#).

```
00023 {
00024     if (m_claw)
00025         m_claw->initialize();
00026     if (m_toggle_arm)
00027         m_toggle_arm->initialize();
00028     if (m_winch)
00029         m_winch->initialize();
00030 }
```

#### 5.75.3.2 run()

```
void wisco::robot::subsystems::hang::HangSubsystem::run () [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 32 of file [HangSubsystem.cpp](#).

```
00033 {
00034     if (m_claw)
00035         m_claw->run();
00036     if (m_toggle_arm)
00037         m_toggle_arm->run();
00038     if (m_winch)
00039         m_winch->run();
00040 }
```

#### 5.75.3.3 command()

```
void wisco::robot::subsystems::hang::HangSubsystem::command (
    std::string command_name,
    va_list & args) [override], [virtual]
```

Runs a command for the subsystem.

**Parameters**

<code>command_name</code>	The name of the command to run
<code>args</code>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 42 of file `HangSubsystem.cpp`.

```
00043 {
00044     if (command_name == CLOSE_CLAW_COMMAND_NAME)
00045     {
00046         if (m_claw)
00047             m_claw->close();
00048     }
00049     else if (command_name == OPEN_CLAW_COMMAND_NAME)
00050     {
00051         if (m_claw)
00052             m_claw->open();
00053     }
00054     else if (command_name == RAISE_ARM_COMMAND_NAME)
00055     {
00056         if (m_toggle_arm)
00057             m_toggle_arm->setUp();
00058     }
00059     else if (command_name == LOWER_ARM_COMMAND_NAME)
00060     {
00061         if (m_toggle_arm)
00062             m_toggle_arm->setDown();
00063     }
00064     else if (command_name == ENGAGE_WINCH_COMMAND_NAME)
00065     {
00066         if (m_winch)
00067             m_winch->engage();
00068     }
00069     else if (command_name == DISENGAGE_WINCH_COMMAND_NAME)
00070     {
00071         if (m_winch)
00072             m_winch->disengage();
00073     }
00074 }
```

**5.75.3.4 state()**

```
void * wisco::robot::subsystems::hang::HangSubsystem::state (
    std::string state_name) [override], [virtual]
```

Gets a state of the subsystem.

**Parameters**

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

**Returns**

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 76 of file `HangSubsystem.cpp`.

```
00077 {
00078     void* result=nullptr;
00079
00080     if (state_name == CLAW_CLOSED_STATE_NAME)
00081     {
00082         if (m_claw)
00083         {
```

```

00084         bool* closed{new bool{m_claw->isClosed()}};
00085         result = closed;
00086     }
00087 }
00088 else if (state_name == CLAW_OPEN_STATE_NAME)
00089 {
00090     if (m_claw)
00091     {
00092         bool* open{new bool{m_claw->isOpen()}};
00093         result = open;
00094     }
00095 }
00096 else if (state_name == ARM_DOWN_STATE_NAME)
00097 {
00098     if (m_toggle_arm)
00099     {
00100         bool* down{new bool{m_toggle_arm->isDown()}};
00101         result = down;
00102     }
00103 }
00104 else if (state_name == ARM_UP_STATE_NAME)
00105 {
00106     if (m_toggle_arm)
00107     {
00108         bool* up{new bool{m_toggle_arm->isUp()}};
00109         result = up;
00110     }
00111 }
00112 else if (state_name == WINCH_ENGAGED_STATE_NAME)
00113 {
00114     if (m_winch)
00115     {
00116         bool* engaged{new bool{m_winch->isEngaged()}};
00117         result = engaged;
00118     }
00119 }
00120 else if (state_name == WINCH_DISENGAGED_STATE_NAME)
00121 {
00122     if (m_winch)
00123     {
00124         bool* disengaged{new bool{m_winch->isDisengaged()}};
00125         result = disengaged;
00126     }
00127 }
00128
00129 return result;
00130 }

```

## 5.75.4 Member Data Documentation

### 5.75.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::SUBSYSTEM_NAME[ ] {"HANG"} [static],  
[constexpr], [private]
```

The name of the subsystem.

Definition at line 57 of file [HangSubsystem.hpp](#).  
00057 {"HANG"};

### 5.75.4.2 CLOSE\_CLAW\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::CLOSE_CLAW_COMMAND_NAME[ ] {"CLOSE  
CLAW"} [static], [constexpr], [private]
```

The name of the close claw command.

Definition at line 63 of file [HangSubsystem.hpp](#).  
00063 {"CLOSE CLAW"};

#### 5.75.4.3 OPEN\_CLAW\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::OPEN_CLAW_COMMAND_NAME[ ] { "OPEN  
CLAW" } [static], [constexpr], [private]
```

The name of the open claw command.

Definition at line 69 of file [HangSubsystem.hpp](#).  
00069 {"OPEN CLAW"};

#### 5.75.4.4 LOWER\_ARM\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::LOWER_ARM_COMMAND_NAME[ ] { "LOWER  
ARM" } [static], [constexpr], [private]
```

The name of the lower arm command.

Definition at line 75 of file [HangSubsystem.hpp](#).  
00075 {"LOWER ARM"};

#### 5.75.4.5 RAISE\_ARM\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::RAISE_ARM_COMMAND_NAME[ ] { "RAISE  
ARM" } [static], [constexpr], [private]
```

The name of the raise arm command.

Definition at line 81 of file [HangSubsystem.hpp](#).  
00081 {"RAISE ARM"};

#### 5.75.4.6 ENGAGE\_WINCH\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::ENGAGE_WINCH_COMMAND_NAME[ ] { "ENGAGE  
WINCH" } [static], [constexpr], [private]
```

The name of the engage winch command.

Definition at line 87 of file [HangSubsystem.hpp](#).  
00087 {"ENGAGE WINCH"};

#### 5.75.4.7 DISENGAGE\_WINCH\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::DISENGAGE_WINCH_COMMAND_NAME[ ] { "DISENGAGE  
WINCH" } [static], [constexpr], [private]
```

The name of the disengage winch command.

Definition at line 93 of file [HangSubsystem.hpp](#).  
00093 {"DISENGAGE WINCH"};

#### 5.75.4.8 CLAW\_CLOSED\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::CLAW_CLOSED_STATE_NAME[ ] { "CLAW  
CLOSED" } [static], [constexpr], [private]
```

The name of the claw closed state.

Definition at line 99 of file [HangSubsystem.hpp](#).

```
00099 {"CLAW CLOSED"};
```

#### 5.75.4.9 CLAW\_OPEN\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::CLAW_OPEN_STATE_NAME[ ] { "CLAW  
OPEN" } [static], [constexpr], [private]
```

The name of the claw open state.

Definition at line 105 of file [HangSubsystem.hpp](#).

```
00105 {"CLAW OPEN"};
```

#### 5.75.4.10 ARM\_DOWN\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::ARM_DOWN_STATE_NAME[ ] { "ARM  
DOWN" } [static], [constexpr], [private]
```

The name of the arm down state.

Definition at line 111 of file [HangSubsystem.hpp](#).

```
00111 {"ARM DOWN"};
```

#### 5.75.4.11 ARM\_UP\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::ARM_UP_STATE_NAME[ ] { "ARM UP" }  
[static], [constexpr], [private]
```

The name of the arm up state.

Definition at line 117 of file [HangSubsystem.hpp](#).

```
00117 {"ARM UP"};
```

#### 5.75.4.12 WINCH\_ENGAGED\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::WINCH_ENGAGED_STATE_NAME[ ] { "WINCH  
ENGAGED" } [static], [constexpr], [private]
```

The name of the winch engaged state.

Definition at line 123 of file [HangSubsystem.hpp](#).

```
00123 {"WINCH ENGAGED"};
```

#### 5.75.4.13 WINCH\_DISENGAGED\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::hang::HangSubsystem::WINCH_DISENGAGED_STATE_NAME[ ]
{ "WINCH DISENGAGED" } [static], [constexpr], [private]
```

The name of the winch disengaged state.

Definition at line 129 of file [HangSubsystem.hpp](#).  
00129 {"WINCH DISENGAGED"};

#### 5.75.4.14 m\_claw

```
std::unique_ptr<IClaw> wisco::robot::subsystems::hang::HangSubsystem::m_claw {} [private]
```

The claw being adapted.

Definition at line 135 of file [HangSubsystem.hpp](#).  
00135 {};

#### 5.75.4.15 m\_toggle\_arm

```
std::unique_ptr<IToggleArm> wisco::robot::subsystems::hang::HangSubsystem::m_toggle_arm {} [private]
```

The toggle arm being adapted.

Definition at line 141 of file [HangSubsystem.hpp](#).  
00141 {};

#### 5.75.4.16 m\_winch

```
std::unique_ptr<IWinch> wisco::robot::subsystems::hang::HangSubsystem::m_winch {} [private]
```

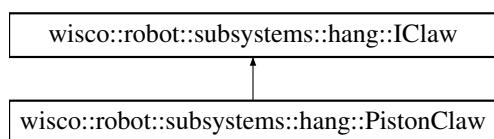
The winch being adapted.

Definition at line 147 of file [HangSubsystem.hpp](#).  
00147 {};

## 5.76 wisco::robot::subsystems::hang::IClaw Class Reference

Interface for claws with an open and closed position.

Inheritance diagram for wisco::robot::subsystems::hang::IClaw:



## Public Member Functions

- virtual ~**IClaw** ()=default  
*Destroy the **IClaw** object.*
- virtual void **initialize** ()=0  
*Initializes the claw.*
- virtual void **run** ()=0  
*Runs the claw.*
- virtual void **close** ()=0  
*Closes the claw.*
- virtual void **open** ()=0  
*Opens the claw.*
- virtual bool **isClosed** ()=0  
*Checks if the claw is closed.*
- virtual bool **isOpen** ()=0  
*Checks if the claw is open.*

### 5.76.1 Detailed Description

Interface for claws with an open and closed position.

#### Author

Nathan Sandvig

Definition at line 41 of file [IClaw.hpp](#).

### 5.76.2 Member Function Documentation

#### 5.76.2.1 initialize()

```
virtual void wisco::robot::subsystems::hang::IClaw::initialize () [pure virtual]
```

Initializes the claw.

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

#### 5.76.2.2 run()

```
virtual void wisco::robot::subsystems::hang::IClaw::run () [pure virtual]
```

Runs the claw.

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

### 5.76.2.3 close()

```
virtual void wisco::robot::subsystems::hang::IClaw::close ( ) [pure virtual]
```

Closes the claw.

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

### 5.76.2.4 open()

```
virtual void wisco::robot::subsystems::hang::IClaw::open ( ) [pure virtual]
```

Opens the claw.

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

### 5.76.2.5 isClosed()

```
virtual bool wisco::robot::subsystems::hang::IClaw::isClosed ( ) [pure virtual]
```

Checks if the claw is closed.

Returns

true The claw is closed

false The claw is not closed

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

### 5.76.2.6 isOpen()

```
virtual bool wisco::robot::subsystems::hang::IClaw::isOpen ( ) [pure virtual]
```

Checks if the claw is open.

Returns

true The claw is open

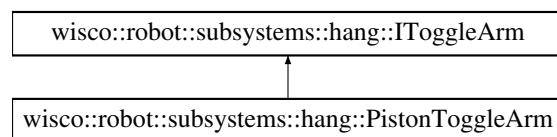
false The claw is not open

Implemented in [wisco::robot::subsystems::hang::PistonClaw](#).

## 5.77 wisco::robot::subsystems::hang::IToggleArm Class Reference

Interface for arms with an up and down position.

Inheritance diagram for wisco::robot::subsystems::hang::IToggleArm:



## Public Member Functions

- virtual ~**IToggleArm** ()=default  
*Destroy the [IToggleArm](#) object.*
- virtual void **initialize** ()=0  
*Initializes the arm.*
- virtual void **run** ()=0  
*Runs the arm.*
- virtual void **setDown** ()=0  
*Sets the arm to the down position.*
- virtual void **setUp** ()=0  
*Sets the arm to the up position.*
- virtual bool **isDown** ()=0  
*Checks if the arm is in the down position.*
- virtual bool **isUp** ()=0  
*Checks if the arm is in the up position.*

### 5.77.1 Detailed Description

Interface for arms with an up and down position.

#### Author

Nathan Sandvig

Definition at line 41 of file [IToggleArm.hpp](#).

### 5.77.2 Member Function Documentation

#### 5.77.2.1 initialize()

```
virtual void wisco::robot::subsystems::hang::IToggleArm::initialize ( ) [pure virtual]
```

Initializes the arm.

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

#### 5.77.2.2 run()

```
virtual void wisco::robot::subsystems::hang::IToggleArm::run ( ) [pure virtual]
```

Runs the arm.

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

**5.77.2.3 setDown()**

```
virtual void wisco::robot::subsystems::hang::IToggleArm::setDown ( ) [pure virtual]
```

Sets the arm to the down position.

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

**5.77.2.4 setUp()**

```
virtual void wisco::robot::subsystems::hang::IToggleArm::setUp ( ) [pure virtual]
```

Sets the arm to the up position.

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

**5.77.2.5 isDown()**

```
virtual bool wisco::robot::subsystems::hang::IToggleArm::isDown ( ) [pure virtual]
```

Checks if the arm is in the down position.

**Returns**

true The arm is in the down position

false The arm is not in the down position

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

**5.77.2.6 isUp()**

```
virtual bool wisco::robot::subsystems::hang::IToggleArm::isUp ( ) [pure virtual]
```

Checks if the arm is in the up position.

**Returns**

true The arm is in the up position

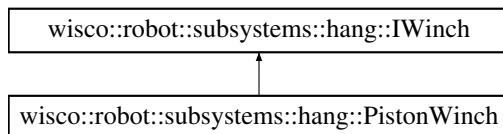
false The arm is not in the up position

Implemented in [wisco::robot::subsystems::hang::PistonToggleArm](#).

**5.78 wisco::robot::subsystems::hang::IWinch Class Reference**

Interface for winches with an engaged and disengaged position.

Inheritance diagram for `wisco::robot::subsystems::hang::IWinch`:



## Public Member Functions

- virtual ~**IWinch** ()=default  
*Destroy the **IWinch** object.*
- virtual void **initialize** ()=0  
*Initializes the winch.*
- virtual void **run** ()=0  
*Runs the winch.*
- virtual void **engage** ()=0  
*Engages the winch.*
- virtual void **disengage** ()=0  
*Disengages the winch.*
- virtual bool **isEngaged** ()=0  
*Checks if the winch is engaged.*
- virtual bool **isDisengaged** ()=0  
*Checks if the winch is disengaged.*

### 5.78.1 Detailed Description

Interface for winches with an engaged and disengaged position.

#### Author

Nathan Sandvig

Definition at line 41 of file [IWinch.hpp](#).

### 5.78.2 Member Function Documentation

#### 5.78.2.1 initialize()

```
virtual void wisco::robot::subsystems::hang::IWinch::initialize ( ) [pure virtual]
```

Initializes the winch.

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

#### 5.78.2.2 run()

```
virtual void wisco::robot::subsystems::hang::IWinch::run ( ) [pure virtual]
```

Runs the winch.

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

### 5.78.2.3 engage()

```
virtual void wisco::robot::subsystems::hang::IWinch::engage () [pure virtual]
```

Engages the winch.

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

### 5.78.2.4 disengage()

```
virtual void wisco::robot::subsystems::hang::IWinch::disengage () [pure virtual]
```

Disengages the winch.

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

### 5.78.2.5 isEngaged()

```
virtual bool wisco::robot::subsystems::hang::IWinch::isEngaged () [pure virtual]
```

Checks if the winch is engaged.

#### Returns

true The winch is engaged

false The winch is not engaged

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

### 5.78.2.6 isDisengaged()

```
virtual bool wisco::robot::subsystems::hang::IWinch::isDisengaged () [pure virtual]
```

Checks if the winch is disengaged.

#### Returns

true The winch is disengaged

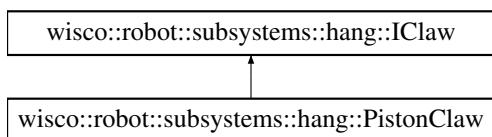
false The winch is not disengaged

Implemented in [wisco::robot::subsystems::hang::PistonWinch](#).

## 5.79 **wisco::robot::subsystems::hang::PistonClaw** Class Reference

A claw controlled using a piston.

Inheritance diagram for [wisco::robot::subsystems::hang::PistonClaw](#):



## Public Member Functions

- void `initialize ()` override  
*Initializes the claw.*
- void `run ()` override  
*Runs the claw.*
- void `close ()` override  
*Closes the claw.*
- void `open ()` override  
*Opens the claw.*
- bool `isClosed ()` override  
*Checks if the claw is closed.*
- bool `isOpen ()` override  
*Checks if the claw is open.*
- void `setPistons (hal::PistonGroup &pistons)`  
*Sets the pistons for the claw.*
- void `setClosedState (bool closed_state)`  
*Sets the state of the pistons when the claw is closed.*

## Public Member Functions inherited from `wisco::robot::subsystems::hang::IClaw`

- virtual ~`IClaw ()`=default  
*Destroy the `IClaw` object.*

## Private Attributes

- `hal::PistonGroup m_pistons {}`  
*The pistons for the claw.*
- bool `m_closed_state {}`  
*The state of the piston group when the claw is closed.*

### 5.79.1 Detailed Description

A claw controlled using a piston.

#### Author

Nathan Sandvig

Definition at line 45 of file `PistonClaw.hpp`.

### 5.79.2 Member Function Documentation

#### 5.79.2.1 `initialize()`

```
void wisco::robot::subsystems::hang::PistonClaw::initialize ( ) [override], [virtual]
```

Initializes the claw.

Implements `wisco::robot::subsystems::hang::IClaw`.

Definition at line 11 of file `PistonClaw.cpp`.

```
00012 {
00013     // No initialize code
00014 }
```

### 5.79.2.2 run()

```
void wisco::robot::subsystems::hang::PistonClaw::run ( ) [override], [virtual]
```

Runs the claw.

Implements [wisco::robot::subsystems::hang::IClaw](#).

Definition at line 16 of file [PistonClaw.cpp](#).

```
00017 {  
00018     // No run code  
00019 }
```

### 5.79.2.3 close()

```
void wisco::robot::subsystems::hang::PistonClaw::close ( ) [override], [virtual]
```

Closes the claw.

Implements [wisco::robot::subsystems::hang::IClaw](#).

Definition at line 21 of file [PistonClaw.cpp](#).

```
00022 {  
00023     m_pistons.setState(m_closed_state);  
00024 }
```

### 5.79.2.4 open()

```
void wisco::robot::subsystems::hang::PistonClaw::open ( ) [override], [virtual]
```

Opens the claw.

Implements [wisco::robot::subsystems::hang::IClaw](#).

Definition at line 26 of file [PistonClaw.cpp](#).

```
00027 {  
00028     m_pistons.setState(!m_closed_state);  
00029 }
```

### 5.79.2.5 isClosed()

```
bool wisco::robot::subsystems::hang::PistonClaw::isClosed ( ) [override], [virtual]
```

Checks if the claw is closed.

#### Returns

true The claw is closed

false The claw is not closed

Implements [wisco::robot::subsystems::hang::IClaw](#).

Definition at line 31 of file [PistonClaw.cpp](#).

```
00032 {  
00033     return m_pistons.getState() == m_closed_state;  
00034 }
```

### 5.79.2.6 isOpen()

```
bool wisco::robot::subsystems::hang::PistonClaw::isOpen ( ) [override], [virtual]
```

Checks if the claw is open.

#### Returns

true The claw is open  
false The claw is not open

Implements [wisco::robot::subsystems::hang::IClaw](#).

Definition at line 36 of file [PistonClaw.cpp](#).

```
00037 {  
00038     return m_pistons.getState() != m_closed_state;  
00039 }
```

### 5.79.2.7 setPistons()

```
void wisco::robot::subsystems::hang::PistonClaw::setPistons (  
    hal::PistonGroup & pistons )
```

Sets the pistons for the claw.

#### Parameters

<i>pistons</i>	The pistons for the claw
----------------	--------------------------

Definition at line 41 of file [PistonClaw.cpp](#).

```
00042 {  
00043     m_pistons = pistons;  
00044 }
```

### 5.79.2.8 setClosedState()

```
void wisco::robot::subsystems::hang::PistonClaw::setClosedState (   
    bool closed_state )
```

Sets the state of the pistons when the claw is closed.

#### Parameters

<i>closed_state</i>	The state of the pistons when the claw is closed
---------------------	--

Definition at line 46 of file [PistonClaw.cpp](#).

```
00047 {  
00048     m_closed_state = closed_state;  
00049 }
```

### 5.79.3 Member Data Documentation

#### 5.79.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonClaw::m_pistons {} [private]
```

The pistons for the claw.

Definition at line 52 of file [PistonClaw.hpp](#).  
00052 {};

#### 5.79.3.2 m\_closed\_state

```
bool wisco::robot::subsystems::hang::PistonClaw::m_closed_state {} [private]
```

The state of the piston group when the claw is closed.

Definition at line 58 of file [PistonClaw.hpp](#).  
00058 {};

## 5.80 wisco::robot::subsystems::hang::PistonClawBuilder Class Reference

Builder for a piston-based claw.

### Public Member Functions

- [PistonClawBuilder \\* withPiston \(std::unique\\_ptr< io::IPiston > &piston\)](#)  
*Adds a piston to the build.*
- [PistonClawBuilder \\* withClosedState \(bool closed\\_state\)](#)  
*Adds a closed state to the build.*
- [std::unique\\_ptr< IClaw > build \(\)](#)  
*Builds a piston claw.*

### Private Attributes

- [hal::PistonGroup m\\_pistons {}](#)  
*The pistons for the claw.*
- [bool m\\_closed\\_state {}](#)  
*The state of the piston group when the claw is closed.*

### 5.80.1 Detailed Description

Builder for a piston-based claw.

#### Author

Nathan Sandvig

Definition at line 44 of file [PistonClawBuilder.hpp](#).

## 5.80.2 Member Function Documentation

### 5.80.2.1 withPiston()

```
PistonClawBuilder * wisco::robot::subsystems::hang::PistonClawBuilder::withPiston ( std::unique_ptr< io::IPiston > & piston )
```

Adds a piston to the build.

**Parameters**

<i>piston</i>	The piston
---------------	------------

**Returns**

This object for build chaining

Definition at line 11 of file [PistonClawBuilder.cpp](#).

```
00012 {
00013     m_pistons.addPiston(piston);
00014     return this;
00015 }
```

**5.80.2.2 withClosedState()**

```
PistonClawBuilder * wisco::robot::subsystems::hang::PistonClawBuilder::withClosedState (
    bool closed_state )
```

Adds a closed state to the build.

**Parameters**

<i>closed_state</i>	The state of the pistons when the claw is closed
---------------------	--

**Returns**

This object for build chaining

Definition at line 17 of file [PistonClawBuilder.cpp](#).

```
00018 {
00019     m_closed_state = closed_state;
00020     return this;
00021 }
```

**5.80.2.3 build()**

```
std::unique_ptr< IClaw > wisco::robot::subsystems::hang::PistonClawBuilder::build ( )
```

Builds a piston claw.

**Returns**

std::unique\_ptr<IClaw> The piston claw as a claw interface object

Definition at line 23 of file [PistonClawBuilder.cpp](#).

```
00024 {
00025     std::unique_ptr<PistonClaw> piston_claw{std::make_unique<PistonClaw>()};
00026     piston_claw->setPistons(m_pistons);
00027     piston_claw->setClosedState(m_closed_state);
00028     return piston_claw;
00029 }
```

### 5.80.3 Member Data Documentation

#### 5.80.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonClawBuilder::m_pistons {} [private]
```

The pistons for the claw.

Definition at line 51 of file [PistonClawBuilder.hpp](#).

```
00051 {};
```

#### 5.80.3.2 m\_closed\_state

```
bool wisco::robot::subsystems::hang::PistonClawBuilder::m_closed_state {} [private]
```

The state of the piston group when the claw is closed.

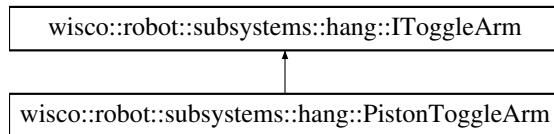
Definition at line 57 of file [PistonClawBuilder.hpp](#).

```
00057 {};
```

## 5.81 wisco::robot::subsystems::hang::PistonToggleArm Class Reference

Interface for arms with an up and down position.

Inheritance diagram for wisco::robot::subsystems::hang::PistonToggleArm:



### Public Member Functions

- void [initialize \(\)](#) override  
*Initializes the arm.*
- void [run \(\)](#) override  
*Runs the arm.*
- void [setDown \(\)](#) override  
*Sets the arm to the down position.*
- void [setUp \(\)](#) override  
*Sets the arm to the up position.*
- bool [isDown \(\)](#) override  
*Checks if the arm is in the down position.*
- bool [isUp \(\)](#) override  
*Checks if the arm is in the up position.*
- void [setPistons \(hal::PistonGroup &pistons\)](#)  
*Sets the pistons for the arm.*
- void [setUpState \(bool up\\_state\)](#)  
*Sets the state of the pistons when the arm is up.*

## Public Member Functions inherited from [wisco::robot::subsystems::hang::IToggleArm](#)

- virtual ~[IToggleArm](#) ()=default

*Destroy the [IToggleArm](#) object.*

### Private Attributes

- [hal::PistonGroup m\\_pistons {}](#)  
*The pistons on the arm.*
- [bool m\\_up\\_state {}](#)  
*The state of the pistons when the arm is up.*

## 5.81.1 Detailed Description

Interface for arms with an up and down position.

### Author

Nathan Sandvig

Definition at line 45 of file [PistonToggleArm.hpp](#).

## 5.81.2 Member Function Documentation

### 5.81.2.1 initialize()

```
void wisco::robot::subsystems::hang::PistonToggleArm::initialize ( ) [override], [virtual]
```

Initializes the arm.

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 11 of file [PistonToggleArm.cpp](#).

```
00012 {  
00013     // No initialization code  
00014 }
```

### 5.81.2.2 run()

```
void wisco::robot::subsystems::hang::PistonToggleArm::run ( ) [override], [virtual]
```

Runs the arm.

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 16 of file [PistonToggleArm.cpp](#).

```
00017 {  
00018     // No running code  
00019 }
```

### 5.81.2.3 setDown()

`void wisco::robot::subsystems::hang::PistonToggleArm::setDown ( ) [override], [virtual]`

Sets the arm to the down position.

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 21 of file [PistonToggleArm.cpp](#).

```
00022 {
00023     m_pistons.setState(!m_up_state);
00024 }
```

### 5.81.2.4 setUp()

`void wisco::robot::subsystems::hang::PistonToggleArm::setUp ( ) [override], [virtual]`

Sets the arm to the up position.

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 26 of file [PistonToggleArm.cpp](#).

```
00027 {
00028     m_pistons.setState(m_up_state);
00029 }
```

### 5.81.2.5 isDown()

`bool wisco::robot::subsystems::hang::PistonToggleArm::isDown ( ) [override], [virtual]`

Checks if the arm is in the down position.

Returns

true The arm is in the down position

false The arm is not in the down position

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 31 of file [PistonToggleArm.cpp](#).

```
00032 {
00033     return m_pistons.getState() != m_up_state;
00034 }
```

### 5.81.2.6 isUp()

`bool wisco::robot::subsystems::hang::PistonToggleArm::isUp ( ) [override], [virtual]`

Checks if the arm is in the up position.

Returns

true The arm is in the up position

false The arm is not in the up position

Implements [wisco::robot::subsystems::hang::IToggleArm](#).

Definition at line 36 of file [PistonToggleArm.cpp](#).

```
00037 {
00038     return m_pistons.getState() == m_up_state;
00039 }
```

### 5.81.2.7 setPistons()

`void wisco::robot::subsystems::hang::PistonToggleArm::setPistons (
 hal::PistonGroup & pistons )`

Sets the pistons for the arm.

**Parameters**

<i>pistons</i>	The pistons for the arm
----------------	-------------------------

Definition at line 41 of file [PistonToggleArm.cpp](#).

```
00042 {  
00043     m_pistons = pistons;  
00044 }
```

### 5.81.2.8 setUpState()

```
void wisco::robot::subsystems::hang::PistonToggleArm::setUpState (  
    bool up_state )
```

Sets the state of the pistons when the arm is up.

**Parameters**

<i>up_state</i>	The state of the pistons when the arm is up
-----------------	---

Definition at line 46 of file [PistonToggleArm.cpp](#).

```
00047 {  
00048     m_up_state = up_state;  
00049 }
```

## 5.81.3 Member Data Documentation

### 5.81.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonToggleArm::m_pistons {} [private]
```

The pistons on the arm.

Definition at line 52 of file [PistonToggleArm.hpp](#).

```
00052 {};
```

### 5.81.3.2 m\_up\_state

```
bool wisco::robot::subsystems::hang::PistonToggleArm::m_up_state {} [private]
```

The state of the pistons when the arm is up.

Definition at line 58 of file [PistonToggleArm.hpp](#).

```
00058 {};
```

## 5.82 wisco::robot::subsystems::hang::PistonToggleArmBuilder Class Reference

Interface for arms with an up and down position.

## Public Member Functions

- `PistonToggleArmBuilder * withPiston (std::unique_ptr< io::IPiston > &piston)`  
*Adds a piston to the build.*
- `PistonToggleArmBuilder * withUpState (bool up_state)`  
*Adds an up state to the build.*
- `std::unique_ptr< IToggleArm > build ()`  
*Builds a piston toggle arm.*

## Private Attributes

- `hal::PistonGroup m_pistons {}`  
*The pistons on the arm.*
- `bool m_up_state {}`  
*The state of the pistons when the arm is up.*

### 5.82.1 Detailed Description

Interface for arms with an up and down position.

#### Author

Nathan Sandvig

Definition at line 44 of file [PistonToggleArmBuilder.hpp](#).

### 5.82.2 Member Function Documentation

#### 5.82.2.1 withPiston()

```
PistonToggleArmBuilder * wisco::robot::subsystems::hang::PistonToggleArmBuilder::withPiston (
    std::unique_ptr< io::IPiston > & piston )
```

Adds a piston to the build.

#### Parameters

<code>piston</code>	The piston
---------------------	------------

#### Returns

This object for build chaining

Definition at line 11 of file [PistonToggleArmBuilder.cpp](#).

```
00012 {
00013     m_pistons.addPiston(piston);
00014     return this;
00015 }
```

### 5.82.2.2 withUpState()

```
PistonToggleArmBuilder * wisco::robot::subsystems::hang::PistonToggleArmBuilder::withUpState (
    bool up_state )
```

Adds an up state to the build.

#### Parameters

<i>up_state</i>	The state of the pistons when the arm is up
-----------------	---

#### Returns

This object for build chaining

Definition at line 17 of file [PistonToggleArmBuilder.cpp](#).

```
00018 {
00019     m_up_state = up_state;
00020     return this;
00021 }
```

### 5.82.2.3 build()

```
std::unique_ptr< IToggleArm > wisco::robot::subsystems::hang::PistonToggleArmBuilder::build (
)
```

Builds a piston toggle arm.

#### Returns

`std::unique_ptr<IToggleArm>` The piston toggle arm as a toggle arm interface object

Definition at line 23 of file [PistonToggleArmBuilder.cpp](#).

```
00024 {
00025     std::unique_ptr<PistonToggleArm> piston_toggle_arm{std::make_unique<PistonToggleArm>()};
00026     piston_toggle_arm->setPistons(m_pistons);
00027     piston_toggle_arm->setUpState(m_up_state);
00028     return piston_toggle_arm;
00029 }
```

## 5.82.3 Member Data Documentation

### 5.82.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonToggleArmBuilder::m_pistons {} [private]
```

The pistons on the arm.

Definition at line 51 of file [PistonToggleArmBuilder.hpp](#).

```
00051 {};
```

### 5.82.3.2 m\_up\_state

```
bool wisco::robot::subsystems::hang::PistonToggleArmBuilder::m_up_state {} [private]
```

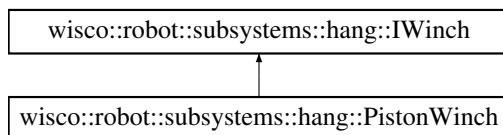
The state of the pistons when the arm is up.

Definition at line 57 of file [PistonToggleArmBuilder.hpp](#).  
00057 {};

## 5.83 wisco::robot::subsystems::hang::PistonWinch Class Reference

A winch controlled using a piston.

Inheritance diagram for wisco::robot::subsystems::hang::PistonWinch:



### Public Member Functions

- void [initialize](#) () override  
*Initializes the winch.*
- void [run](#) () override  
*Runs the winch.*
- void [engage](#) () override  
*Engages the winch.*
- void [disengage](#) () override  
*Disengages the winch.*
- bool [isEngaged](#) () override  
*Checks if the winch is engaged.*
- bool [isDisengaged](#) () override  
*Checks if the winch is disengaged.*
- void [setPistons](#) (hal::PistonGroup &pistons)  
*Sets the pistons for the winch.*
- void [setEngagedState](#) (bool engaged\_state)  
*Sets the state of the pistons when the winch is engaged.*

### Public Member Functions inherited from [wisco::robot::subsystems::hang::IWinch](#)

- virtual ~[IWinch](#) ()=default  
*Destroy the [IWinch](#) object.*

### Private Attributes

- `hal::PistonGroup m_pistons {}`  
*The pistons for the claw.*
- `bool m_engaged_state {}`  
*The state of the piston group when the winch is engaged.*

### 5.83.1 Detailed Description

A winch controlled using a piston.

#### Author

Nathan Sandvig

Definition at line 45 of file [PistonWinch.hpp](#).

### 5.83.2 Member Function Documentation

#### 5.83.2.1 initialize()

```
void wisco::robot::subsystems::hang::PistonWinch::initialize () [override], [virtual]
```

Initializes the winch.

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 11 of file [PistonWinch.cpp](#).

```
00012 {  
00013     // No initialize code  
00014 }
```

#### 5.83.2.2 run()

```
void wisco::robot::subsystems::hang::PistonWinch::run () [override], [virtual]
```

Runs the winch.

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 16 of file [PistonWinch.cpp](#).

```
00017 {  
00018     // No run code  
00019 }
```

#### 5.83.2.3 engage()

```
void wisco::robot::subsystems::hang::PistonWinch::engage () [override], [virtual]
```

Engages the winch.

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 21 of file [PistonWinch.cpp](#).

```
00022 {  
00023     m_pistons.setState(m_engaged_state);  
00024 }
```

### 5.83.2.4 disengage()

```
void wisco::robot::subsystems::hang::PistonWinch::disengage ( ) [override], [virtual]
```

Disengages the winch.

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 26 of file [PistonWinch.cpp](#).

```
00027 {  
00028     m_pistons.setState(!m_engaged_state);  
00029 }
```

### 5.83.2.5 isEngaged()

```
bool wisco::robot::subsystems::hang::PistonWinch::isEngaged ( ) [override], [virtual]
```

Checks if the winch is engaged.

#### Returns

true The winch is engaged

false The winch is not engaged

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 31 of file [PistonWinch.cpp](#).

```
00032 {  
00033     return m_pistons.getState() == m_engaged_state;  
00034 }
```

### 5.83.2.6 isDisengaged()

```
bool wisco::robot::subsystems::hang::PistonWinch::isDisengaged ( ) [override], [virtual]
```

Checks if the winch is disengaged.

#### Returns

true The winch is disengaged

false The winch is not disengaged

Implements [wisco::robot::subsystems::hang::IWinch](#).

Definition at line 36 of file [PistonWinch.cpp](#).

```
00037 {  
00038     return m_pistons.getState() != m_engaged_state;  
00039 }
```

### 5.83.2.7 setPistons()

```
void wisco::robot::subsystems::hang::PistonWinch::setPistons (   
    hal::PistonGroup & pistons )
```

Sets the pistons for the winch.

**Parameters**

<i>pistons</i>	The pistons for the winch
----------------	---------------------------

Definition at line 41 of file [PistonWinch.cpp](#).

```
00042 {  
00043     m_pistons = pistons;  
00044 }
```

### 5.83.2.8 setEngagedState()

```
void wisco::robot::subsystems::hang::PistonWinch::setEngagedState (  
    bool engaged_state )
```

Sets the state of the pistons when the winch is engaged.

**Parameters**

<i>engaged_state</i>	The state of the pistons when the winch is engaged
----------------------	--

Definition at line 46 of file [PistonWinch.cpp](#).

```
00047 {  
00048     m_engaged_state = engaged_state;  
00049 }
```

## 5.83.3 Member Data Documentation

### 5.83.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonWinch::m_pistons {} [private]
```

The pistons for the claw.

Definition at line 52 of file [PistonWinch.hpp](#).

```
00052 {};
```

### 5.83.3.2 m\_engaged\_state

```
bool wisco::robot::subsystems::hang::PistonWinch::m_engaged_state {} [private]
```

The state of the piston group when the winch is engaged.

Definition at line 58 of file [PistonWinch.hpp](#).

```
00058 {};
```

## 5.84 wisco::robot::subsystems::hang::PistonWinchBuilder Class Reference

Builder for a piston-based winch.

## Public Member Functions

- `PistonWinchBuilder * withPiston (std::unique_ptr< io::IPiston > &piston)`  
*Adds a piston to the build.*
- `PistonWinchBuilder * withEngagedState (bool engaged_state)`  
*Adds an engaged state to the build.*
- `std::unique_ptr< IWinch > build ()`  
*Builds a piston winch.*

## Private Attributes

- `hal::PistonGroup m_pistons {}`  
*The pistons for the winch.*
- `bool m_engaged_state {}`  
*The state of the piston group when the winch is engaged.*

### 5.84.1 Detailed Description

Builder for a piston-based winch.

#### Author

Nathan Sandvig

Definition at line 44 of file [PistonWinchBuilder.hpp](#).

### 5.84.2 Member Function Documentation

#### 5.84.2.1 withPiston()

```
PistonWinchBuilder * wisco::robot::subsystems::hang::PistonWinchBuilder::withPiston (
    std::unique_ptr< io::IPiston > & piston )
```

Adds a piston to the build.

#### Parameters

<code>piston</code>	The piston
---------------------	------------

#### Returns

This object for build chaining

Definition at line 11 of file [PistonWinchBuilder.cpp](#).

```
00012 {
00013     m_pistons.addPiston(piston);
00014     return this;
00015 }
```

### 5.84.2.2 withEngagedState()

```
PistonWinchBuilder * wisco::robot::subsystems::hang::PistonWinchBuilder::withEngagedState (
    bool engaged_state )
```

Adds an engaged state to the build.

#### Parameters

<i>engaged_state</i>	The state of the pistons when the winch is engaged
----------------------	--

#### Returns

This object for build chaining

Definition at line 17 of file [PistonWinchBuilder.cpp](#).

```
00018 {
00019     m_engaged_state = engaged_state;
00020     return this;
00021 }
```

### 5.84.2.3 build()

```
std::unique_ptr< IWinch > wisco::robot::subsystems::hang::PistonWinchBuilder::build ( )
```

Builds a piston winch.

#### Returns

`std::unique_ptr<IWinch>` The piston winch as a winch interface object

Definition at line 23 of file [PistonWinchBuilder.cpp](#).

```
00024 {
00025     std::unique_ptr<PistonWinch> piston_winch{std::make_unique<PistonWinch>()};
00026     piston_winch->setPistons(m_pistons);
00027     piston_winch->setEngagedState(m_engaged_state);
00028     return piston_winch;
00029 }
```

## 5.84.3 Member Data Documentation

### 5.84.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::hang::PistonWinchBuilder::m_pistons {} [private]
```

The pistons for the winch.

Definition at line 51 of file [PistonWinchBuilder.hpp](#).

```
00051 {};
```

### 5.84.3.2 m\_engaged\_state

```
bool wisco::robot::subsystems::hang::PistonWinchBuilder::m_engaged_state {} [private]
```

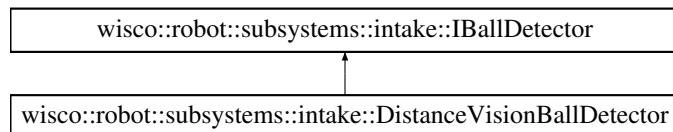
The state of the piston group when the winch is engaged.

Definition at line 57 of file [PistonWinchBuilder.hpp](#).  
00057 {};

## 5.85 wisco::robot::subsystems::intake::DistanceVisionBallDetector Class Reference

Ball detection system that uses a distance and vision sensor.

Inheritance diagram for wisco::robot::subsystems::intake::DistanceVisionBallDetector:



### Public Member Functions

- void [initialize](#) () override  
*Initializes the ball detector.*
- void [run](#) () override  
*Runs the ball detector.*
- double [getBallDistance](#) () override  
*Get the distance to the ball.*
- double [getBallAngle](#) () override  
*Get the angle to the ball.*
- void [setDistanceSensor](#) (std::unique\_ptr<io::IDistanceSensor> &distance\_sensor)  
*Sets the distance sensor.*

### Public Member Functions inherited from [wisco::robot::subsystems::intake::IBallDetector](#)

- virtual ~[IBallDetector](#) ()=default  
*Destroy the [IBallDetector](#) object.*

### Private Attributes

- std::unique\_ptr<io::IDistanceSensor> [m\\_distance\\_sensor](#) {}  
*The distance sensor.*

## 5.85.1 Detailed Description

Ball detection system that uses a distance and vision sensor.

### Author

Nathan Sandvig

Definition at line 48 of file [DistanceVisionBallDetector.hpp](#).

## 5.85.2 Member Function Documentation

### 5.85.2.1 initialize()

```
void wisco::robot::subsystems::intake::DistanceVisionBallDetector::initialize () [override],  
[virtual]
```

Initializes the ball detector.

Implements [wisco::robot::subsystems::intake::IBallDetector](#).

Definition at line 11 of file [DistanceVisionBallDetector.cpp](#).

```
00012 {  
00013     if (m_distance_sensor)  
00014         m_distance_sensor->initialize();  
00015 }
```

### 5.85.2.2 run()

```
void wisco::robot::subsystems::intake::DistanceVisionBallDetector::run () [override], [virtual]
```

Runs the ball detector.

Implements [wisco::robot::subsystems::intake::IBallDetector](#).

Definition at line 17 of file [DistanceVisionBallDetector.cpp](#).

```
00018 {  
00019     // No running code  
00020 }
```

### 5.85.2.3 getBallDistance()

```
double wisco::robot::subsystems::intake::DistanceVisionBallDetector::getBallDistance () [override],  
[virtual]
```

Get the distance to the ball.

### Returns

double The distance to the ball

Implements [wisco::robot::subsystems::intake::IBallDetector](#).

Definition at line 22 of file [DistanceVisionBallDetector.cpp](#).

```
00023 {  
00024     double distance{};  
00025     if (m_distance_sensor)  
00026         distance = m_distance_sensor->getDistance();  
00027     return distance;  
00028 }
```

### 5.85.2.4 getBallAngle()

```
double wisco::robot::subsystems::intake::DistanceVisionBallDetector::getBallAngle () [override],  
[virtual]
```

Get the angle to the ball.

#### Returns

double The angle to the ball

Implements [wisco::robot::subsystems::intake::IBallDetector](#).

Definition at line 30 of file [DistanceVisionBallDetector.cpp](#).

```
00031 {  
00032     double angle{};  
00033  
00034     // TODO vision sensor  
00035  
00036     return angle;  
00037 }
```

### 5.85.2.5 setDistanceSensor()

```
void wisco::robot::subsystems::intake::DistanceVisionBallDetector::setDistanceSensor (  
    std::unique_ptr< io::IDistanceSensor > & distance_sensor )
```

Sets the distance sensor.

#### Parameters

<i>distance_sensor</i>	The distance sensor
------------------------	---------------------

Definition at line 39 of file [DistanceVisionBallDetector.cpp](#).

```
00040 {  
00041     m_distance_sensor = std::move(distance_sensor);  
00042 }
```

## 5.85.3 Member Data Documentation

### 5.85.3.1 *m\_distance\_sensor*

```
std::unique_ptr<io::IDistanceSensor> wisco::robot::subsystems::intake::DistanceVisionBallDetector::m_distance_sensor {} [private]
```

The distance sensor.

Definition at line 55 of file [DistanceVisionBallDetector.hpp](#).

```
00055 {};
```

## 5.86 wisco::robot::subsystems::intake::DistanceVisionBallDetector Builder Class Reference

Ball detection system that uses a distance and vision sensor.

## Public Member Functions

- `DistanceVisionBallDetectorBuilder * withDistanceSensor (std::unique_ptr< io::IDistanceSensor > &distance_sensor)`  
*Adds a distance sensor to the build.*
- `std::unique_ptr< IBallDetector > build ()`  
*Builds the distance vision ball detector.*

## Private Attributes

- `std::unique_ptr< io::IDistanceSensor > m_distance_sensor {}`  
*The distance sensor.*

### 5.86.1 Detailed Description

Ball detection system that uses a distance and vision sensor.

#### Author

Nathan Sandvig

Definition at line 43 of file [DistanceVisionBallDetectorBuilder.hpp](#).

### 5.86.2 Member Function Documentation

#### 5.86.2.1 withDistanceSensor()

```
DistanceVisionBallDetectorBuilder * wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder::withDistanceSensor (
    std::unique_ptr< io::IDistanceSensor > & distance_sensor )
```

Adds a distance sensor to the build.

#### Parameters

<code>distance_sensor</code>	The distance sensor
------------------------------	---------------------

#### Returns

This object for build chaining

Definition at line 11 of file [DistanceVisionBallDetectorBuilder.cpp](#).

```
00012 {
00013     m_distance_sensor = std::move(distance_sensor);
00014     return this;
00015 }
```

### 5.86.2.2 build()

```
std::unique_ptr< IBallDetector > wisco::robot::subsystems::intake::DistanceVisionBallDetector<->
Builder::build ( )
```

Builds the distance vision ball detector.

#### Returns

`std::unique_ptr<IBallDetector>` The distance vision ball detector as an [IBallDetector](#) interface object

Definition at line 17 of file [DistanceVisionBallDetectorBuilder.cpp](#).

```
00018 {
00019     std::unique_ptr<DistanceVisionBallDetector>
00020         distance_vision_ball_detector{std::make_unique<DistanceVisionBallDetector>() };
00021     distance_vision_ball_detector->setDistanceSensor(m\_distance\_sensor);
00022     return distance_vision_ball_detector;
00022 }
```

## 5.86.3 Member Data Documentation

### 5.86.3.1 `m_distance_sensor`

```
std::unique_ptr<io::IDistanceSensor> wisco::robot::subsystems::intake::DistanceVisionBall<->
DetectorBuilder::m_distance_sensor {} [private]
```

The distance sensor.

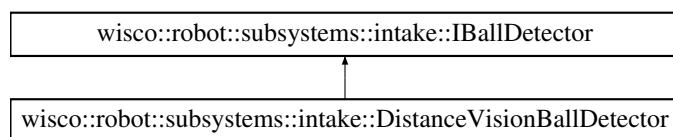
Definition at line 50 of file [DistanceVisionBallDetectorBuilder.hpp](#).

```
00050 {};
```

## 5.87 `wisco::robot::subsystems::intake::IBallDetector` Class Reference

Interface for ball detection system.

Inheritance diagram for `wisco::robot::subsystems::intake::IBallDetector`:



### Public Member Functions

- virtual ~[IBallDetector](#) ()=default  
*Destroy the [IBallDetector](#) object.*
- virtual void [initialize](#) ()=0  
*Initializes the ball detector.*
- virtual void [run](#) ()=0  
*Runs the ball detector.*
- virtual double [getBallDistance](#) ()=0  
*Get the distance to the ball.*
- virtual double [getBallAngle](#) ()=0  
*Get the angle to the ball.*

## 5.87.1 Detailed Description

Interface for ball detection system.

### Author

Nathan Sandvig

Definition at line 41 of file [IBallDetector.hpp](#).

## 5.87.2 Member Function Documentation

### 5.87.2.1 initialize()

```
virtual void wisco::robot::subsystems::intake::IBallDetector::initialize () [pure virtual]
```

Initializes the ball detector.

Implemented in [wisco::robot::subsystems::intake::DistanceVisionBallDetector](#).

### 5.87.2.2 run()

```
virtual void wisco::robot::subsystems::intake::IBallDetector::run () [pure virtual]
```

Runs the ball detector.

Implemented in [wisco::robot::subsystems::intake::DistanceVisionBallDetector](#).

### 5.87.2.3 getBallDistance()

```
virtual double wisco::robot::subsystems::intake::IBallDetector::getBallDistance () [pure virtual]
```

Get the distance to the ball.

#### Returns

double The distance to the ball

Implemented in [wisco::robot::subsystems::intake::DistanceVisionBallDetector](#).

### 5.87.2.4 getBallAngle()

```
virtual double wisco::robot::subsystems::intake::IBallDetector::getBallAngle () [pure virtual]
```

Get the angle to the ball.

#### Returns

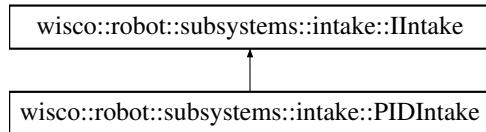
double The angle to the ball

Implemented in [wisco::robot::subsystems::intake::DistanceVisionBallDetector](#).

## 5.88 wisco::robot::subsystems::intake::IIntake Class Reference

Interface for intakes.

Inheritance diagram for wisco::robot::subsystems::intake::IIntake:



### Public Member Functions

- virtual ~**IIntake** ()=default  
*Destroy the **IIntake** object.*
- virtual void **initialize** ()=0  
*Initializes the intake.*
- virtual void **run** ()=0  
*Runs the intake.*
- virtual double **getVelocity** ()=0  
*Get the velocity of the intake in in/s.*
- virtual void **setVelocity** (double velocity)=0  
*Set the velocity of the intake.*
- virtual void **setVoltage** (double voltage)=0  
*Set the voltage of the intake directly.*

### 5.88.1 Detailed Description

Interface for intakes.

#### Author

Nathan Sandvig

Definition at line 41 of file [IIntake.hpp](#).

### 5.88.2 Member Function Documentation

#### 5.88.2.1 initialize()

```
virtual void wisco::robot::subsystems::intake::IIntake::initialize ( ) [pure virtual]
```

Initializes the intake.

Implemented in [wisco::robot::subsystems::intake::PIDIntake](#).

### 5.88.2.2 run()

```
virtual void wisco::robot::subsystems::intake::IIntake::run ( ) [pure virtual]
```

Runs the intake.

Implemented in [wisco::robot::subsystems::intake::PIDIntake](#).

### 5.88.2.3 getVelocity()

```
virtual double wisco::robot::subsystems::intake::IIntake::getVelocity ( ) [pure virtual]
```

Get the velocity of the intake in in/s.

#### Returns

double The intake velocity

Implemented in [wisco::robot::subsystems::intake::PIDIntake](#).

### 5.88.2.4 setVelocity()

```
virtual void wisco::robot::subsystems::intake::IIntake::setVelocity ( double velocity ) [pure virtual]
```

Set the velocity of the intake.

#### Parameters

<i>velocity</i>	The velocity of the intake in in/s
-----------------	------------------------------------

Implemented in [wisco::robot::subsystems::intake::PIDIntake](#).

### 5.88.2.5 setVoltage()

```
virtual void wisco::robot::subsystems::intake::IIntake::setVoltage ( double voltage ) [pure virtual]
```

Set the voltage of the intake directly.

#### Parameters

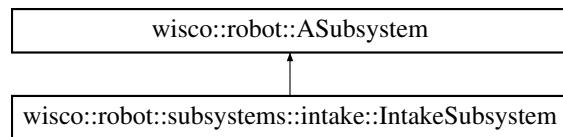
<i>voltage</i>	The voltage for the intake
----------------	----------------------------

Implemented in [wisco::robot::subsystems::intake::PIDIntake](#).

## 5.89 wisco::robot::subsystems::intake::IntakeSubsystem Class Reference

The subsystem adapter for intakes.

Inheritance diagram for wisco::robot::subsystems::intake::IntakeSubsystem:



### Public Member Functions

- **IntakeSubsystem** (std::unique\_ptr<[IIntake](#)> &intake, std::unique\_ptr<[IBallDetector](#)> &ball\_detector)  
*Construct a new Intake Subsystem object.*
- void **initialize** () override  
*Initializes the subsystem.*
- void **run** () override  
*Runs the subsystem.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the subsystem.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the subsystem.*

### Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem** ()=default  
*Construct a new ASubsystem object.*
- **ASubsystem** (const [ASubsystem](#) &other)=default  
*Construct a new ASubsystem object.*
- **ASubsystem** ([ASubsystem](#) &&other)=default  
*Construct a new ASubsystem object.*
- **ASubsystem** (std::string name)  
*Construct a new ASubsystem object.*
- virtual ~**ASubsystem** ()=default  
*Destroy the ASubsystem object.*
- const std::string & **getName** () const  
*Get the name of the subsystem.*
- **ASubsystem** & **operator=** (const [ASubsystem](#) &rhs)=default  
*Copy assignment operator for ASubsystem.*
- **ASubsystem** & **operator=** ([ASubsystem](#) &&rhs)=default  
*Move assignment operator for ASubsystem.*

### Private Attributes

- std::unique\_ptr<[IIntake](#)> **m\_intake** {}  
*The intake being adapted.*
- std::unique\_ptr<[IBallDetector](#)> **m\_ball\_detector** {}  
*The ball detector being adapted.*

## Static Private Attributes

- static constexpr char `SUBSYSTEM_NAME` [] {"INTAKE"}  
*The name of the subsystem.*
- static constexpr char `SET_VELOCITY_COMMAND_NAME` [] {"SET VELOCITY"}  
*The name of the set velocity command.*
- static constexpr char `SET_VOLTAGE_COMMAND_NAME` [] {"SET VOLTAGE"}  
*The name of the set voltage command.*
- static constexpr char `GET_VELOCITY_STATE_NAME` [] {"GET VELOCITY"}  
*The name of the get velocity command.*
- static constexpr char `GET BALL_DISTANCE_STATE_NAME` [] {"GET BALL DISTANCE"}  
*The name of the get ball distance command.*
- static constexpr char `GET BALL_ANGLE_STATE_NAME` [] {"GET BALL ANGLE"}  
*The name of the get ball angle command.*

## 5.89.1 Detailed Description

The subsystem adapter for intakes.

### Author

Nathan Sandvig

Definition at line 47 of file `IntakeSubsystem.hpp`.

## 5.89.2 Constructor & Destructor Documentation

### 5.89.2.1 IntakeSubsystem()

```
wisco::robot::subsystems::intake::IntakeSubsystem::IntakeSubsystem (
    std::unique_ptr< IIntake > & intake,
    std::unique_ptr< IBallDetector > & ball_detector )
```

Construct a new Intake Subsystem object.

#### Parameters

<code>intake</code>	The intake being adapted
<code>ball_detector</code>	The ball detector being adapted

Definition at line 11 of file `IntakeSubsystem.cpp`.

```
00012     : ASubsystem{SUBSYSTEM_NAME}, m_intake{std::move(intake)},
  m_ball_detector{std::move(ball_detector)}
00013 {
00014
00015 }
```

### 5.89.3 Member Function Documentation

#### 5.89.3.1 initialize()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::initialize ( ) [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [IntakeSubsystem.cpp](#).

```
00018 {
00019     if (m_intake)
00020         m_intake->initialize();
00021     if (m_ball_detector)
00022         m_ball_detector->initialize();
00023 }
```

#### 5.89.3.2 run()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::run ( ) [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 25 of file [IntakeSubsystem.cpp](#).

```
00026 {
00027     if (m_intake)
00028         m_intake->run();
00029     if (m_ball_detector)
00030         m_ball_detector->run();
00031 }
```

#### 5.89.3.3 command()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

##### Parameters

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 33 of file [IntakeSubsystem.cpp](#).

```
00034 {
00035     if (command_name == SET_VELOCITY_COMMAND_NAME)
00036     {
00037         double velocity{va_arg(args, double)};
00038         m_intake->setVelocity(velocity);
00039     }
00040     else if (command_name == SET_VOLTAGE_COMMAND_NAME)
00041     {
```

```

00042     double voltage(va_arg(args, double));
00043     m_intake->setVoltage(voltage);
00044 }
00045 }
```

### 5.89.3.4 state()

```
void * wisco::robot::subsystems::intake::IntakeSubsystem::state (
    std::string state_name) [override], [virtual]
```

Gets a state of the subsystem.

#### Parameters

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 47 of file [IntakeSubsystem.cpp](#).

```

00048 {
00049     void* result=nullptr;
00050
00051     if (state_name == GET_VELOCITY_STATE_NAME)
00052     {
00053         double* velocity=new double{m_intake->getVelocity()};
00054         result = velocity;
00055     }
00056     else if (state_name == GET BALL_DISTANCE_STATE_NAME)
00057     {
00058         double* distance=new double{m_ball_detector->getBallDistance()};
00059         result = distance;
00060     }
00061     else if (state_name == GET BALL_ANGLE_STATE_NAME)
00062     {
00063         double* angle=new double{m_ball_detector->getBallAngle()};
00064         result = angle;
00065     }
00066
00067     return result;
00068 }
```

## 5.89.4 Member Data Documentation

### 5.89.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::SUBSYSTEM_NAME[ ] {"INTAKE"}
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 54 of file [IntakeSubsystem.hpp](#).

```
00054 {"INTAKE"};
```

#### 5.89.4.2 SET\_VELOCITY\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::SET_VELOCITY_COMMAND_NAME[ ]
{"SET VELOCITY"} [static], [constexpr], [private]
```

The name of the set velocity command.

Definition at line 60 of file [IntakeSubsystem.hpp](#).

```
00060 {"SET VELOCITY"};
```

#### 5.89.4.3 SET\_VOLTAGE\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::SET_VOLTAGE_COMMAND_NAME[ ]
{"SET VOLTAGE"} [static], [constexpr], [private]
```

The name of the set voltage command.

Definition at line 66 of file [IntakeSubsystem.hpp](#).

```
00066 {"SET VOLTAGE"};
```

#### 5.89.4.4 GET\_VELOCITY\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::GET_VELOCITY_STATE_NAME[ ]
{"GET VELOCITY"} [static], [constexpr], [private]
```

The name of the get velocity command.

Definition at line 72 of file [IntakeSubsystem.hpp](#).

```
00072 {"GET VELOCITY"};
```

#### 5.89.4.5 GET BALL\_DISTANCE\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::GET_BALLDISTANCESTATE_NAME[ ]
{"GET BALL DISTANCE"} [static], [constexpr], [private]
```

The name of the get ball distance command.

Definition at line 78 of file [IntakeSubsystem.hpp](#).

```
00078 {"GET BALL DISTANCE"};
```

#### 5.89.4.6 GET\_BALL\_ANGLE\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::GET_BALLANDLESTATE_NAME[ ]
{"GET BALL ANGLE"} [static], [constexpr], [private]
```

The name of the get ball angle command.

Definition at line 84 of file [IntakeSubsystem.hpp](#).

```
00084 {"GET BALL ANGLE"};
```

#### 5.89.4.7 m\_intake

```
std::unique_ptr<IIntake> wisco::robot::subsystems::intake::IntakeSubsystem::m_intake {} [private]
```

The intake being adapted.

Definition at line 90 of file [IntakeSubsystem.hpp](#).  
00090 {};

#### 5.89.4.8 m\_ball\_detector

```
std::unique_ptr<IBallDetector> wisco::robot::subsystems::intake::IntakeSubsystem::m_ball_detector {} [private]
```

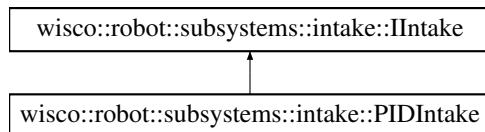
The ball detector being adapted.

Definition at line 96 of file [IntakeSubsystem.hpp](#).  
00096 {};

## 5.90 wisco::robot::subsystems::intake::PIDIntake Class Reference

An intake controller with PID velocity control.

Inheritance diagram for wisco::robot::subsystems::intake::PIDIntake:



### Public Member Functions

- void [initialize](#) () override  
*Initializes the intake.*
- void [run](#) () override  
*Runs the intake.*
- double [getVelocity](#) () override  
*Get the velocity of the intake in in/s.*
- void [setVelocity](#) (double velocity) override  
*Set the velocity of the intake.*
- void [setVoltage](#) (double voltage) override  
*Set the voltage of the intake directly.*
- void [setClock](#) (const std::unique\_ptr<[rtos::IClock](#)> &clock)  
*Set the rtos clock.*
- void [setDelayer](#) (const std::unique\_ptr<[rtos::IDelayLayer](#)> &delayer)  
*Set the rtos delayer.*
- void [setMutex](#) (std::unique\_ptr<[rtos::IMutex](#)> &mutex)  
*Set the thread mutex.*
- void [setTask](#) (std::unique\_ptr<[rtos::ITask](#)> &task)  
*Set the task handler.*
- void [setPID](#) ([control::PID](#) pid)  
*Set the PID controller.*
- void [setMotors](#) ([hal::MotorGroup](#) &motors)  
*Set the motors.*
- void [setRollerRadius](#) (double roller\_radius)  
*Set the radius of the roller.*

## Public Member Functions inherited from `wisco::robot::subsystems::intake::IIntake`

- virtual ~`IIntake` ()=default

*Destroy the `IIntake` object.*

## Private Member Functions

- void `taskUpdate` ()
 

*Runs all the object-specific updates in the task loop.*
- void `updateVelocity` ()
 

*Updates the intake velocity.*

## Static Private Member Functions

- static void `taskLoop` (void \*params)
 

*The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr<`rtos::IClock`> `m_clock` {}
 

*The rtos clock.*
- std::unique\_ptr<`rtos::IDelayer`> `m_delayer` {}
 

*The rtos delayer.*
- std::unique\_ptr<`rtos::IMutex`> `m_mutex` {}
 

*The mutex for thread safety.*
- std::unique\_ptr<`rtos::ITask`> `m_task` {}
 

*The background task handler.*
- `control::PID m_pid` {}
 

*The velocity PID controller.*
- `hal::MotorGroup m_motors` {}
 

*The motors on the intake.*
- double `m_roller_radius` {}
 

*The radius of the intake roller.*
- double `m_velocity` {}
 

*The velocity setting of the intake.*
- bool `velocity_control` {}
 

*Whether or not to control with velocity.*

## Static Private Attributes

- static constexpr uint8\_t `TASK_DELAY` {10}
 

*The loop delay on the task.*

## 5.90.1 Detailed Description

An intake controller with PID velocity control.

### Author

Nathan Sandvig

Definition at line 52 of file `PIDIntake.hpp`.

## 5.90.2 Member Function Documentation

### 5.90.2.1 taskLoop()

```
void wisco::robot::subsystems::intake::PIDIntake::taskLoop (
    void * params) [static], [private]
```

The task loop function for background updates.

#### Parameters

<i>params</i>	
---------------	--

Definition at line 11 of file [PIDIntake.cpp](#).

```
00012 {
00013     void** parameters{static_cast<void**>(params)};
00014     PIDIntake* instance{static_cast<PIDIntake*>(parameters[0])};
00015
00016     while (true)
00017     {
00018         instance->taskUpdate();
00019     }
00020 }
```

### 5.90.2.2 taskUpdate()

```
void wisco::robot::subsystems::intake::PIDIntake::taskUpdate () [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 22 of file [PIDIntake.cpp](#).

```
00023 {
00024     if (velocity_control)
00025         updateVelocity();
00026     m_delayer->delay(TASK_DELAY);
00027 }
```

### 5.90.2.3 updateVelocity()

```
void wisco::robot::subsystems::intake::PIDIntake::updateVelocity () [private]
```

Updates the intake velocity.

Definition at line 29 of file [PIDIntake.cpp](#).

```
00030 {
00031     if (m_mutex)
00032         m_mutex->take();
00033
00034     double velocity{getVelocity()};
00035     double voltage{m_pid.getControlValue(velocity, m_velocity)};
00036     m_motors.setVoltage(voltage);
00037
00038     if (m_mutex)
00039         m_mutex->give();
00040 }
```

#### 5.90.2.4 initialize()

```
void wisco::robot::subsystems::intake::PIDIntake::initialize ( ) [override], [virtual]
```

Initializes the intake.

Implements [wisco::robot::subsystems::intake::IIntake](#).

Definition at line 42 of file [PIDIntake.cpp](#).

```
00043 {
00044     m_pid.reset();
00045     m_motors.initialize();
00046 }
```

#### 5.90.2.5 run()

```
void wisco::robot::subsystems::intake::PIDIntake::run ( ) [override], [virtual]
```

Runs the intake.

Implements [wisco::robot::subsystems::intake::IIntake](#).

Definition at line 48 of file [PIDIntake.cpp](#).

```
00049 {
00050     if (m_task)
00051     {
00052         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};
00053         params[0] = this;
00054         m_task->start(&PIDIntake::taskLoop, params);
00055     }
00056 }
```

#### 5.90.2.6 getVelocity()

```
double wisco::robot::subsystems::intake::PIDIntake::getVelocity ( ) [override], [virtual]
```

Get the velocity of the intake in in/s.

##### Returns

double The intake velocity

Implements [wisco::robot::subsystems::intake::IIntake](#).

Definition at line 58 of file [PIDIntake.cpp](#).

```
00059 {
00060     return m_motors.getAngularVelocity() * m_roller_radius;
00061 }
```

#### 5.90.2.7 setVelocity()

```
void wisco::robot::subsystems::intake::PIDIntake::setVelocity (
    double velocity) [override], [virtual]
```

Set the velocity of the intake.

**Parameters**

<i>velocity</i>	The velocity of the intake in in/s
-----------------	------------------------------------

Implements [wisco::robot::subsystems::intake::IIntake](#).

Definition at line 63 of file [PIDIntake.cpp](#).

```
00064 {
00065     if (m_mutex)
00066         m_mutex->take();
00067
00068     m_velocity = velocity;
00069     velocity_control = true;
00070
00071     if (m_mutex)
00072         m_mutex->give();
00073 }
```

**5.90.2.8 setVoltage()**

```
void wisco::robot::subsystems::intake::PIDIntake::setVoltage (
    double voltage) [override], [virtual]
```

Set the voltage of the intake directly.

**Parameters**

<i>voltage</i>	The voltage for the intake
----------------	----------------------------

Implements [wisco::robot::subsystems::intake::IIntake](#).

Definition at line 75 of file [PIDIntake.cpp](#).

```
00076 {
00077     if (m_mutex)
00078         m_mutex->take();
00079
00080     m_pid.reset();
00081     m_motors.setVoltage(voltage);
00082     velocity_control = false;
00083
00084     if (m_mutex)
00085         m_mutex->give();
00086 }
```

**5.90.2.9 setClock()**

```
void wisco::robot::subsystems::intake::PIDIntake::setClock (
    const std::unique_ptr< rtos::IClock > & clock )
```

Set the rtos clock.

**Parameters**

<i>clock</i>	The rtos clock
--------------	----------------

Definition at line 88 of file [PIDIntake.cpp](#).

```
00089 {
00090     m_clock = clock->clone();
00091 }
```

### 5.90.2.10 setDelayer()

```
void wisco::robot::subsystems::intake::PIDIntake::setDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 93 of file [PIDIntake.cpp](#).

```
00094 {
00095     m_delayer = delayer->clone();
00096 }
```

### 5.90.2.11 setMutex()

```
void wisco::robot::subsystems::intake::PIDIntake::setMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Set the thread mutex.

#### Parameters

<i>mutex</i>	The thread mutex
--------------	------------------

Definition at line 98 of file [PIDIntake.cpp](#).

```
00099 {
00100     m_mutex = std::move(mutex);
00101 }
```

### 5.90.2.12 setTask()

```
void wisco::robot::subsystems::intake::PIDIntake::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Set the task handler.

#### Parameters

<i>task</i>	The task handler
-------------	------------------

Definition at line 103 of file [PIDIntake.cpp](#).

```
00104 {
00105     m_task = std::move(task);
00106 }
```

### 5.90.2.13 setPID()

```
void wisco::robot::subsystems::intake::PIDIntake::setPID (
    control::PID pid )
```

Set the PID controller.

**Parameters**

<i>pid</i>	The PID controller
------------	--------------------

Definition at line 108 of file [PIDIntake.cpp](#).

```
00109 {
00110     m_pid = pid;
00111 }
```

**5.90.2.14 setMotors()**

```
void wisco::robot::subsystems::intake::PIDIntake::setMotors (
    hal::MotorGroup & motors )
```

Set the motors.

**Parameters**

<i>motors</i>	The motors
---------------	------------

Definition at line 113 of file [PIDIntake.cpp](#).

```
00114 {
00115     m_motors = motors;
00116 }
```

**5.90.2.15 setRollerRadius()**

```
void wisco::robot::subsystems::intake::PIDIntake::setRollerRadius (
    double roller_radius )
```

Set the radius of the roller.

**Parameters**

<i>roller_radius</i>	The radius of the roller
----------------------	--------------------------

Definition at line 118 of file [PIDIntake.cpp](#).

```
00119 {
00120     m_roller_radius = roller_radius;
00121 }
```

**5.90.3 Member Data Documentation****5.90.3.1 TASK\_DELAY**

```
constexpr uint8_t wisco::robot::subsystems::intake::PIDIntake::TASK_DELAY {10} [static],
[constexpr], [private]
```

The loop delay on the task.

Definition at line 59 of file [PIDIntake.hpp](#).

```
00059 {10};
```

### 5.90.3.2 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::intake::PIDIntake::m_clock {} [private]
```

The rtos clock.

Definition at line 72 of file [PIDIntake.hpp](#).  
00072 {};

### 5.90.3.3 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::intake::PIDIntake::m_delayer {} [private]
```

The rtos delayer.

Definition at line 78 of file [PIDIntake.hpp](#).  
00078 {};

### 5.90.3.4 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::intake::PIDIntake::m_mutex {} [private]
```

The mutex for thread safety.

Definition at line 84 of file [PIDIntake.hpp](#).  
00084 {};

### 5.90.3.5 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::intake::PIDIntake::m_task {} [private]
```

The background task handler.

Definition at line 90 of file [PIDIntake.hpp](#).  
00090 {};

### 5.90.3.6 m\_pid

```
control::PID wisco::robot::subsystems::intake::PIDIntake::m_pid {} [private]
```

The velocity PID controller.

Definition at line 96 of file [PIDIntake.hpp](#).  
00096 {};

### 5.90.3.7 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::intake::PIDIntake::m_motors {} [private]
```

The motors on the intake.

Definition at line 102 of file [PIDIntake.hpp](#).  
00102 {};

### 5.90.3.8 m\_roller\_radius

```
double wisco::robot::subsystems::intake::PIDIntake::m_roller_radius {} [private]
```

The radius of the intake roller.

Definition at line 108 of file [PIDIntake.hpp](#).

```
00108 {};
```

### 5.90.3.9 m\_velocity

```
double wisco::robot::subsystems::intake::PIDIntake::m_velocity {} [private]
```

The velocity setting of the intake.

Definition at line 114 of file [PIDIntake.hpp](#).

```
00114 {};
```

### 5.90.3.10 velocity\_control

```
bool wisco::robot::subsystems::intake::PIDIntake::velocity_control {} [private]
```

Whether or not to control with velocity.

Definition at line 120 of file [PIDIntake.hpp](#).

```
00120 {};
```

## 5.91 wisco::robot::subsystems::intake::PIDIntakeBuilder Class Reference

A builder class for a PID-based intake subsystem.

### Public Member Functions

- `PIDIntakeBuilder * withClock (const std::unique_ptr< rtos::IClock > &clock)`  
*Add an rtos clock to the build.*
- `PIDIntakeBuilder * withDelayer (const std::unique_ptr< rtos::IDelay > &delayer)`  
*Add an rtos delayer to the build.*
- `PIDIntakeBuilder * withMutex (std::unique_ptr< rtos::IMutex > &mutex)`  
*Add a thread mutex to the build.*
- `PIDIntakeBuilder * withTask (std::unique_ptr< rtos::ITask > &task)`  
*Add a task handler to the build.*
- `PIDIntakeBuilder * withPID (control::PID pid)`  
*Add a PID controller to the build.*
- `PIDIntakeBuilder * withMotor (std::unique_ptr< io::IMotor > &motor)`  
*Add a motor to the build.*
- `PIDIntakeBuilder * withRollerRadius (double roller_radius)`  
*Add a roller radius to the build.*
- `std::unique_ptr< IIntake > build ()`  
*Builds the intake.*

## Private Attributes

- std::unique\_ptr< rtos::IClock > m\_clock {}  
*The rtos clock.*
- std::unique\_ptr< rtos::IDelay > m\_delay {}  
*The rtos delayer.*
- std::unique\_ptr< rtos::IMutex > m\_mutex {}  
*The mutex for thread safety.*
- std::unique\_ptr< rtos::ITask > m\_task {}  
*The background task handler.*
- control::PID m\_pid {}  
*The velocity PID controller.*
- hal::MotorGroup m\_motors {}  
*The motors on the intake.*
- double m\_roller\_radius {}  
*The radius of the intake roller.*

## 5.91.1 Detailed Description

A builder class for a PID-based intake subsystem.

### Author

Nathan Sandvig

Definition at line 43 of file [PIDIntakeBuilder.hpp](#).

## 5.91.2 Member Function Documentation

### 5.91.2.1 withClock()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withClock (
    const std::unique_ptr< rtos::IClock > & clock )
```

Add an rtos clock to the build.

#### Parameters

<code>clock</code>	The rtos clock
--------------------	----------------

#### Returns

PIDIntakeBuilder\* This object for build chaining

Definition at line 11 of file [PIDIntakeBuilder.cpp](#).

```
00012 {
00013     m_clock = clock->clone();
00014     return this;
00015 }
```

### 5.91.2.2 withDelayer()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

#### Returns

PIDIntakeBuilder\* This object for build chaining

Definition at line 17 of file [PIDIntakeBuilder.cpp](#).

```
00018 {
00019     m_delayer = delayer->clone();
00020     return this;
00021 }
```

### 5.91.2.3 withMutex()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Add a thread mutex to the build.

#### Parameters

<i>mutex</i>	The thread mutex
--------------	------------------

#### Returns

PIDIntakeBuilder\* This object for build chaining

Definition at line 23 of file [PIDIntakeBuilder.cpp](#).

```
00024 {
00025     m_mutex = std::move(mutex);
00026     return this;
00027 }
```

### 5.91.2.4 withTask()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withTask (
    std::unique_ptr< rtos::ITask > & task )
```

Add a task handler to the build.

#### Parameters

<i>task</i>	The task handler
-------------	------------------

**Returns**

PIDIntakeBuilder\* This object for build chaining

Definition at line 29 of file [PIDIntakeBuilder.cpp](#).

```
00030 {  
00031     m_task = std::move(task);  
00032     return this;  
00033 }
```

### 5.91.2.5 withPID()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withPID (  
    control::PID pid )
```

Add a PID controller to the build.

**Parameters**

<i>pid</i>	The PID controller
------------	--------------------

**Returns**

PIDIntakeBuilder\* This object for build chaining

Definition at line 35 of file [PIDIntakeBuilder.cpp](#).

```
00036 {  
00037     m_pid = pid;  
00038     return this;  
00039 }
```

### 5.91.2.6 withMotor()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withMotor (  
    std::unique_ptr< io::IMotor > & motor )
```

Add a motor to the build.

**Parameters**

<i>motor</i>	The motor
--------------	-----------

**Returns**

PIDIntakeBuilder\* This object for build chaining

Definition at line 41 of file [PIDIntakeBuilder.cpp](#).

```
00042 {  
00043     m_motors.addMotor(motor);  
00044     return this;  
00045 }
```

### 5.91.2.7 withRollerRadius()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withRollerRadius (
    double roller_radius )
```

Add a roller radius to the build.

#### Parameters

<i>roller_radius</i>	The radius of the roller
----------------------	--------------------------

#### Returns

PIDIntakeBuilder\* This object for build chaining

Definition at line 47 of file [PIDIntakeBuilder.cpp](#).

```
00048 {
00049     m_roller_radius = roller_radius;
00050     return this;
00051 }
```

### 5.91.2.8 build()

```
std::unique_ptr< IIntake > wisco::robot::subsystems::intake::PIDIntakeBuilder::build ( )
```

Builds the intake.

#### Returns

std::unique\_ptr<IIntake> The [PIDIntake](#) object built with the stored data

Definition at line 53 of file [PIDIntakeBuilder.cpp](#).

```
00054 {
00055     std::unique_ptr<PIDIntake> pid_intake{std::make_unique<PIDIntake>() };
00056     pid_intake->setClock(m_clock);
00057     pid_intake->setDelayer(m_delayer);
00058     pid_intake->setMutex(m_mutex);
00059     pid_intake->setTask(m_task);
00060     pid_intake->setPID(m_pid);
00061     pid_intake->setMotors(m_motors);
00062     pid_intake->setRollerRadius(m_roller_radius);
00063     return pid_intake;
00064 }
```

## 5.91.3 Member Data Documentation

### 5.91.3.1 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::intake::PIDIntakeBuilder::m_clock {}
[private]
```

The rtos clock.

Definition at line 50 of file [PIDIntakeBuilder.hpp](#).

```
00050 {};
```

### 5.91.3.2 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::intake::PIDIntakeBuilder::m_delayer
{} [private]
```

The rtos delayer.

Definition at line 56 of file [PIDIntakeBuilder.hpp](#).  
00056 {};

### 5.91.3.3 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::intake::PIDIntakeBuilder::m_mutex {} [private]
```

The mutex for thread safety.

Definition at line 62 of file [PIDIntakeBuilder.hpp](#).  
00062 {};

### 5.91.3.4 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::intake::PIDIntakeBuilder::m_task {} [private]
```

The background task handler.

Definition at line 68 of file [PIDIntakeBuilder.hpp](#).  
00068 {};

### 5.91.3.5 m\_pid

```
control::PID wisco::robot::subsystems::intake::PIDIntakeBuilder::m_pid {} [private]
```

The velocity PID controller.

Definition at line 74 of file [PIDIntakeBuilder.hpp](#).  
00074 {};

### 5.91.3.6 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::intake::PIDIntakeBuilder::m_motors {} [private]
```

The motors on the intake.

Definition at line 80 of file [PIDIntakeBuilder.hpp](#).  
00080 {};

### 5.91.3.7 m\_roller\_radius

```
double wisco::robot::subsystems::intake::PIDIntakeBuilder::m_roller_radius {} [private]
```

The radius of the intake roller.

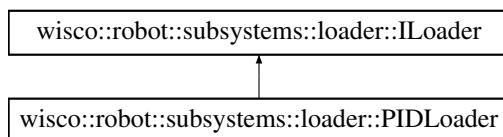
Definition at line 86 of file [PIDIntakeBuilder.hpp](#).

```
00086 {};
```

## 5.92 wisco::robot::subsystems::loader::ILoader Class Reference

Interface for loader subsystems.

Inheritance diagram for wisco::robot::subsystems::loader::ILoader:



### Public Member Functions

- virtual ~**ILoader** ()=default  
*Destroy the **ILoader** object.*
- virtual void **initialize** ()=0  
*Initializes the loader.*
- virtual void **run** ()=0  
*Runs the loader.*
- virtual void **doLoad** ()=0  
*Does one match load.*
- virtual void **doReady** ()=0  
*Gets the system ready for another match load.*
- virtual bool **isLoaded** ()=0  
*Checks if the match load is loaded.*
- virtual bool **isReady** ()=0  
*Checks if the loader is ready.*

### 5.92.1 Detailed Description

Interface for loader subsystems.

#### Author

Nathan Sandvig

Definition at line 41 of file [ILoader.hpp](#).

## 5.92.2 Member Function Documentation

### 5.92.2.1 initialize()

```
virtual void wisco::robot::subsystems::loader::ILoader::initialize () [pure virtual]
```

Initializes the loader.

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

### 5.92.2.2 run()

```
virtual void wisco::robot::subsystems::loader::ILoader::run () [pure virtual]
```

Runs the loader.

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

### 5.92.2.3 doLoad()

```
virtual void wisco::robot::subsystems::loader::ILoader::doLoad () [pure virtual]
```

Does one match load.

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

### 5.92.2.4 doReady()

```
virtual void wisco::robot::subsystems::loader::ILoader::doReady () [pure virtual]
```

Gets the system ready for another match load.

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

### 5.92.2.5 isLoaded()

```
virtual bool wisco::robot::subsystems::loader::ILoader::isLoaded () [pure virtual]
```

Checks if the match load is loaded.

#### Returns

true The match load is loaded

false The match load is not loaded

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

### 5.92.2.6 `isReady()`

```
virtual bool wisco::robot::subsystems::loader::ILoader::isReady() [pure virtual]
```

Checks if the loader is ready.

#### Returns

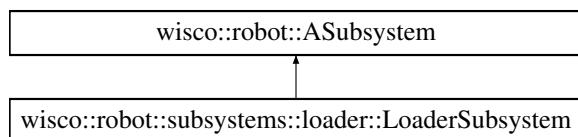
- true The loader is ready
- false The loader is not ready

Implemented in [wisco::robot::subsystems::loader::PIDLoader](#).

## 5.93 `wisco::robot::subsystems::loader::LoaderSubsystem` Class Reference

The subsystem adapter for loaders.

Inheritance diagram for `wisco::robot::subsystems::loader::LoaderSubsystem`:



### Public Member Functions

- `LoaderSubsystem (std::unique_ptr< ILoader > &loader)`  
*Construct a new LOADER Subsystem object.*
- `void initialize () override`  
*Initializes the subsystem.*
- `void run () override`  
*Runs the subsystem.*
- `void command (std::string command_name, va_list &args) override`  
*Runs a command for the subsystem.*
- `void * state (std::string state_name) override`  
*Gets a state of the subsystem.*

## Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem ()=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (const ASubsystem &other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (ASubsystem &&other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (std::string name)**  
*Construct a new ASubsystem object.*
- **virtual ~ASubsystem ()=default**  
*Destroy the ASubsystem object.*
- **const std::string & getName () const**  
*Get the name of the subsystem.*
- **ASubsystem & operator= (const ASubsystem &rhs)=default**  
*Copy assignment operator for ASubsystem.*
- **ASubsystem & operator= (ASubsystem &&rhs)=default**  
*Move assignment operator for ASubsystem.*

## Private Attributes

- **std::unique\_ptr< ILoader > m\_loader {}**  
*The loader being adapted.*

## Static Private Attributes

- **static constexpr char SUBSYSTEM\_NAME [] {"LOADER"}**  
*The name of the subsystem.*
- **static constexpr char DO\_LOAD\_COMMAND\_NAME [] {"DO LOAD"}**  
*The name of the do load command.*
- **static constexpr char DO\_READY\_COMMAND\_NAME [] {"DO READY"}**  
*The name of the do ready command.*
- **static constexpr char IS\_LOADED\_STATE\_NAME [] {"IS LOADED"}**  
*The name of the is loaded command.*
- **static constexpr char IS\_READY\_STATE\_NAME [] {"IS READY"}**  
*The name of the is ready command.*

## 5.93.1 Detailed Description

The subsystem adapter for loaders.

### Author

Nathan Sandvig

Definition at line 46 of file [LoaderSubsystem.hpp](#).

## 5.93.2 Constructor & Destructor Documentation

### 5.93.2.1 LoaderSubsystem()

```
wisco::robot::subsystems::loader::LoaderSubsystem::LoaderSubsystem (
    std::unique_ptr< ILoader > & loader )
```

Construct a new LOADER Subsystem object.

**Parameters**

<i>loader</i>	The loader being adapted
---------------	--------------------------

Definition at line 11 of file [LoaderSubsystem.cpp](#).

```
00012     : ASubsystem{SUBSYSTEM_NAME}, m_loader{std::move(loader)}
00013 {
00014
00015 }
```

### 5.93.3 Member Function Documentation

#### 5.93.3.1 initialize()

```
void wisco::robot::subsystems::loader::LoaderSubsystem::initialize ( ) [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [LoaderSubsystem.cpp](#).

```
00018 {
00019     if (m_loader)
00020         m_loader->initialize();
00021 }
```

#### 5.93.3.2 run()

```
void wisco::robot::subsystems::loader::LoaderSubsystem::run ( ) [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 23 of file [LoaderSubsystem.cpp](#).

```
00024 {
00025     if (m_loader)
00026         m_loader->run();
00027 }
```

#### 5.93.3.3 command()

```
void wisco::robot::subsystems::loader::LoaderSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 29 of file [LoaderSubsystem.cpp](#).

```
00030 {
00031     if (command_name == DO_LOAD_COMMAND_NAME)
00032     {
00033         m_loader->doLoad();
00034     }
00035     else if (command_name == DO_READY_COMMAND_NAME)
00036     {
00037         m_loader->doReady();
00038     }
00039 }
```

### 5.93.3.4 state()

```
void * wisco::robot::subsystems::loader::LoaderSubsystem::state (
    std::string state_name) [override], [virtual]
```

Gets a state of the subsystem.

#### Parameters

<i>state_name</i>	The name of the state to get
-------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 41 of file [LoaderSubsystem.cpp](#).

```
00042 {
00043     void* result=nullptr;
00044
00045     if (state_name == IS_LOADED_STATE_NAME)
00046     {
00047         bool* loaded=new bool{m_loader->isLoaded()};
00048         result = loaded;
00049     }
00050     else if (state_name == IS_READY_STATE_NAME)
00051     {
00052         bool* ready=new bool{m_loader->isReady()};
00053         result = ready;
00054     }
00055
00056     return result;
00057 }
```

## 5.93.4 Member Data Documentation

### 5.93.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::loader::LoaderSubsystem::SUBSYSTEM_NAME[] {"LOADER"}
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file [LoaderSubsystem.hpp](#).

```
00053 {"LOADER"};
```

#### 5.93.4.2 DO\_LOAD\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::loader::LoaderSubsystem::DO_LOAD_COMMAND_NAME[ ] {"DO
LOAD"} [static], [constexpr], [private]
```

The name of the do load command.

Definition at line 59 of file [LoaderSubsystem.hpp](#).

```
00059 {"DO LOAD"};
```

#### 5.93.4.3 DO\_READY\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::loader::LoaderSubsystem::DO_READY_COMMAND_NAME[ ] {"DO
READY"} [static], [constexpr], [private]
```

The name of the do ready command.

Definition at line 65 of file [LoaderSubsystem.hpp](#).

```
00065 {"DO READY"};
```

#### 5.93.4.4 IS\_LOADED\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::loader::LoaderSubsystem::IS_LOADED_STATE_NAME[ ] {"IS
LOADED"} [static], [constexpr], [private]
```

The name of the is loaded command.

Definition at line 71 of file [LoaderSubsystem.hpp](#).

```
00071 {"IS LOADED"};
```

#### 5.93.4.5 IS\_READY\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::loader::LoaderSubsystem::IS_READY_STATE_NAME[ ] {"IS
READY"} [static], [constexpr], [private]
```

The name of the is ready command.

Definition at line 77 of file [LoaderSubsystem.hpp](#).

```
00077 {"IS READY"};
```

#### 5.93.4.6 m\_loader

```
std::unique_ptr<ILoader> wisco::robot::subsystems::loader::LoaderSubsystem::m_loader {} [private]
```

The loader being adapted.

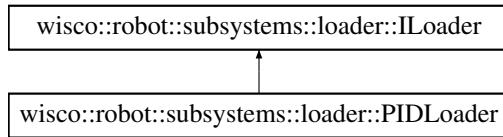
Definition at line 83 of file [LoaderSubsystem.hpp](#).

```
00083 {};
```

## 5.94 wisco::robot::subsystems::loader::PIDLoader Class Reference

An loader controller with PID position control.

Inheritance diagram for wisco::robot::subsystems::loader::PIDLoader:



### Public Member Functions

- void [initialize](#) () override  
*Initializes the loader.*
- void [run](#) () override  
*Runs the loader.*
- void [doLoad](#) () override  
*Does one match load.*
- void [doReady](#) () override  
*Gets the system ready for another match load.*
- bool [isLoaded](#) () override  
*Checks if the match load is loaded.*
- bool [isReady](#) () override  
*Checks if the loader is ready.*
- void [setClock](#) (const std::unique\_ptr< rtos::IClock > &clock)  
*Set the rtos clock.*
- void [setDelayer](#) (const std::unique\_ptr< rtos::IDelayer > &delayer)  
*Set the rtos delayer.*
- void [setMutex](#) (std::unique\_ptr< rtos::IMutex > &mutex)  
*Set the thread mutex.*
- void [setTask](#) (std::unique\_ptr< rtos::ITask > &task)  
*Set the task handler.*
- void [setPID](#) (control::PID pid)  
*Set the PID controller.*
- void [setMotors](#) (hal::MotorGroup &motors)  
*Set the motors.*
- void [setMatchLoadPosition](#) (double match\_load\_position)  
*Sets the position for match loads.*
- void [setReadyPosition](#) (double ready\_position)  
*Set the position for ready.*
- void [setPositionTolerance](#) (double position\_tolerance)  
*Set the allowed tolerance around the target positions.*

### Public Member Functions inherited from [wisco::robot::subsystems::loader::ILoader](#)

- virtual ~[ILoader](#) ()=default  
*Destroy the [ILoader](#) object.*

## Private Types

- enum class `EState` { **READY** , **LOADING** , **LOADED** , **READYING** }
- The states the loader can be in.*

## Private Member Functions

- void `taskUpdate` ()  
*Runs all the object-specific updates in the task loop.*
- void `updateState` ()  
*Updates the system state.*
- void `updatePosition` ()  
*Updates the loader position.*
- double `getPosition` ()  
*Gets the position of the loader.*

## Static Private Member Functions

- static void `taskLoop` (void \*params)  
*The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr<`rtos::IClock`> `m_clock` {}  
*The rtos clock.*
- std::unique\_ptr<`rtos::IDelayer`> `m_delayer` {}  
*The rtos delayer.*
- std::unique\_ptr<`rtos::IMutex`> `m_mutex` {}  
*The mutex for thread safety.*
- std::unique\_ptr<`rtos::ITask`> `m_task` {}  
*The background task handler.*
- `control::PID m_pid` {}  
*The position PID controller.*
- `hal::MotorGroup m_motors` {}  
*The motors on the loader.*
- double `m_match_load_position` {}  
*The position for match loads.*
- double `m_ready_position` {}  
*The position for ready.*
- double `m_position_tolerance` {}  
*The allowed tolerance around the target positions.*
- `EState state` {`EState::READY`}  
*The state of the loader.*
- double `target_position` {}  
*The position setting of the loader.*

## Static Private Attributes

- static constexpr uint8\_t `TASK_DELAY` {10}  
*The loop delay on the task.*

### 5.94.1 Detailed Description

An loader controller with PID position control.

#### Author

Nathan Sandvig

Definition at line 52 of file [PIDLoader.hpp](#).

### 5.94.2 Member Enumeration Documentation

#### 5.94.2.1 EState

```
enum class wisco::robot::subsystems::loader::PIDLoader::EState [strong], [private]
```

The states the loader can be in.

#### Author

Nathan Sandvig

Definition at line 60 of file [PIDLoader.hpp](#).

```
00061 {
00062     READY,
00063     LOADING,
00064     LOADED,
00065     READYING
00066 };
```

### 5.94.3 Member Function Documentation

#### 5.94.3.1 taskLoop()

```
void wisco::robot::subsystems::loader::PIDLoader::taskLoop (
    void * params) [static], [private]
```

The task loop function for background updates.

#### Parameters

<i>params</i>
---------------

Definition at line 11 of file [PIDLoader.cpp](#).

```
00012 {
00013     void** parameters{static_cast<void**>(params)};
00014     PIDLoader* instance{static_cast<PIDLoader*>(parameters[0])};
00015
00016     while (true)
00017     {
00018         instance->taskUpdate();
00019     }
00020 }
```

### 5.94.3.2 taskUpdate()

```
void wisco::robot::subsystems::loader::PIDLoader::taskUpdate ( ) [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 22 of file [PIDLoader.cpp](#).

```
00023 {
00024     if (m_mutex)
00025         m_mutex->take();
00026     updateState();
00027     updatePosition();
00028     if (m_mutex)
00029         m_mutex->give();
00030
00031     m_delayer->delay(TASK_DELAY);
00032 }
```

### 5.94.3.3 updateState()

```
void wisco::robot::subsystems::loader::PIDLoader::updateState ( ) [private]
```

Updates the system state.

Definition at line 34 of file [PIDLoader.cpp](#).

```
00035 {
00036     double position{getPosition()};
00037     switch (state)
00038     {
00039         case EState::READY:
00040             break;
00041         case EState::READYING:
00042             if (std::abs(position - m_ready_position) < m_position_tolerance)
00043                 state = EState::READY;
00044             break;
00045         case EState::LOADED:
00046             break;
00047         case EState::LOADING:
00048             if (std::abs(position - m_match_load_position) < m_position_tolerance)
00049                 state = EState::LOADED;
00050             break;
00051     }
00052 }
```

### 5.94.3.4 updatePosition()

```
void wisco::robot::subsystems::loader::PIDLoader::updatePosition ( ) [private]
```

Updates the loader position.

Definition at line 54 of file [PIDLoader.cpp](#).

```
00055 {
00056     double voltage{m_pid.getControlValue(getPosition(), target_position)};
00057     m_motors.setVoltage(voltage);
00058 }
```

### 5.94.3.5 getPosition()

```
double wisco::robot::subsystems::loader::PIDLoader::getPosition ( ) [private]
```

Gets the position of the loader.

Returns

double The position of the loader

Definition at line 60 of file [PIDLoader.cpp](#).

```
00061 {
00062     return m_motors.getPosition();
00063 }
```

### 5.94.3.6 initialize()

```
void wisco::robot::subsystems::loader::PIDLoader::initialize ( ) [override], [virtual]
```

Initializes the loader.

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 65 of file [PIDLoader.cpp](#).

```
00066 {  
00067     m_pid.reset();  
00068     m_motors.initialize();  
00069 }
```

### 5.94.3.7 run()

```
void wisco::robot::subsystems::loader::PIDLoader::run ( ) [override], [virtual]
```

Runs the loader.

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 71 of file [PIDLoader.cpp](#).

```
00072 {  
00073     if (m_task)  
00074     {  
00075         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};  
00076         params[0] = this;  
00077         m_task->start(&PIDLoader::taskLoop, params);  
00078     }  
00079 }
```

### 5.94.3.8 doLoad()

```
void wisco::robot::subsystems::loader::PIDLoader::doLoad ( ) [override], [virtual]
```

Does one match load.

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 81 of file [PIDLoader.cpp](#).

```
00082 {  
00083     if (m_mutex)  
00084         m_mutex->take();  
00085  
00086     if (state == EState::READY)  
00087     {  
00088         state = EState::LOADING;  
00089         target_position = m_match_load_position;  
00090     }  
00091  
00092     if (m_mutex)  
00093         m_mutex->give();  
00094 }
```

### 5.94.3.9 doReady()

```
void wisco::robot::subsystems::loader::PIDLoader::doReady () [override], [virtual]
```

Gets the system ready for another match load.

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 96 of file [PIDLoader.cpp](#).

```
00097 {
00098     if (m_mutex)
00099         m_mutex->take();
00100
00101     if (state == EState::LOADED)
00102     {
00103         state = EState::READYING;
00104         target_position = m_ready_position;
00105     }
00106
00107     if (m_mutex)
00108         m_mutex->give();
00109 }
```

### 5.94.3.10 isLoaded()

```
bool wisco::robot::subsystems::loader::PIDLoader::isLoaded () [override], [virtual]
```

Checks if the match load is loaded.

Returns

true The match load is loaded

false The match load is not loaded

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 111 of file [PIDLoader.cpp](#).

```
00112 {
00113     return state == EState::LOADED;
00114 }
```

### 5.94.3.11 isReady()

```
bool wisco::robot::subsystems::loader::PIDLoader::isReady () [override], [virtual]
```

Checks if the loader is ready.

Returns

true The loader is ready

false The loader is not ready

Implements [wisco::robot::subsystems::loader::ILoader](#).

Definition at line 116 of file [PIDLoader.cpp](#).

```
00117 {
00118     return state == EState::READY;
00119 }
```

### 5.94.3.12 setClock()

```
void wisco::robot::subsystems::loader::PIDLoader::setClock (
    const std::unique_ptr< rtos::IClock > & clock )
```

Set the rtos clock.

**Parameters**

<i>clock</i>	The rtos clock
--------------	----------------

Definition at line 121 of file [PIDLoader.cpp](#).

```
00122 {  
00123     m_clock = clock->clone();  
00124 }
```

**5.94.3.13 setDelayer()**

```
void wisco::robot::subsystems::loader::PIDLoader::setDelayer (  
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

**Parameters**

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 126 of file [PIDLoader.cpp](#).

```
00127 {  
00128     m_delayer = delayer->clone();  
00129 }
```

**5.94.3.14 setMutex()**

```
void wisco::robot::subsystems::loader::PIDLoader::setMutex (  
    std::unique_ptr< rtos::IMutex > & mutex )
```

Set the thread mutex.

**Parameters**

<i>mutex</i>	The thread mutex
--------------	------------------

Definition at line 131 of file [PIDLoader.cpp](#).

```
00132 {  
00133     m_mutex = std::move(mutex);  
00134 }
```

**5.94.3.15 setTask()**

```
void wisco::robot::subsystems::loader::PIDLoader::setTask (  
    std::unique_ptr< rtos::ITask > & task )
```

Set the task handler.

**Parameters**

<i>task</i>	The task handler
-------------	------------------

Definition at line 136 of file [PIDLoader.cpp](#).

```
00137 {
00138     m_task = std::move(task);
00139 }
```

### 5.94.3.16 setPID()

```
void wisco::robot::subsystems::loader::PIDLoader::setPID (
    control::PID pid)
```

Set the PID controller.

Parameters

<i>pid</i>	The PID controller
------------	--------------------

Definition at line 141 of file [PIDLoader.cpp](#).

```
00142 {
00143     m_pid = pid;
00144 }
```

### 5.94.3.17 setMotors()

```
void wisco::robot::subsystems::loader::PIDLoader::setMotors (
    hal::MotorGroup & motors)
```

Set the motors.

Parameters

<i>motors</i>	The motors
---------------	------------

Definition at line 146 of file [PIDLoader.cpp](#).

```
00147 {
00148     m_motors = motors;
00149 }
```

### 5.94.3.18 setMatchLoadPosition()

```
void wisco::robot::subsystems::loader::PIDLoader::setMatchLoadPosition (
    double match_load_position)
```

Sets the position for match loads.

Parameters

<i>match_load_position</i>	The position for match loads
----------------------------	------------------------------

Definition at line 151 of file [PIDLoader.cpp](#).

```
00152 {
00153     m_match_load_position = match_load_position;
00154 }
```

### 5.94.3.19 setReadyPosition()

```
void wisco::robot::subsystems::loader::PIDLoader::setReadyPosition (
    double ready_position )
```

Set the position for ready.

#### Parameters

<i>ready_position</i>	The position for ready
-----------------------	------------------------

Definition at line 156 of file [PIDLoader.cpp](#).

```
00157 {
00158     m_ready_position = ready_position;
00159 }
```

### 5.94.3.20 setPositionTolerance()

```
void wisco::robot::subsystems::loader::PIDLoader::setPositionTolerance (
    double position_tolerance )
```

Set the allowed tolerance around the target positions.

#### Parameters

<i>position_tolerance</i>	The allowed tolerance around the target positions
---------------------------	---

Definition at line 161 of file [PIDLoader.cpp](#).

```
00162 {
00163     m_position_tolerance = position_tolerance;
00164 }
```

## 5.94.4 Member Data Documentation

### 5.94.4.1 TASK\_DELAY

```
constexpr uint8_t wisco::robot::subsystems::loader::PIDLoader::TASK_DELAY {10} [static],
[constexpr], [private]
```

The loop delay on the task.

Definition at line 72 of file [PIDLoader.hpp](#).

```
00072 {10};
```

### 5.94.4.2 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::loader::PIDLoader::m_clock {} [private]
```

The rtos clock.

Definition at line 85 of file [PIDLoader.hpp](#).

```
00085 {};
```

#### 5.94.4.3 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::loader::PIDLoader::m_delayer {}  
[private]
```

The rtos delayer.

Definition at line 91 of file [PIDLoader.hpp](#).

```
00091 {};
```

#### 5.94.4.4 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::loader::PIDLoader::m_mutex {} [private]
```

The mutex for thread safety.

Definition at line 97 of file [PIDLoader.hpp](#).

```
00097 {};
```

#### 5.94.4.5 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::loader::PIDLoader::m_task {} [private]
```

The background task handler.

Definition at line 103 of file [PIDLoader.hpp](#).

```
00103 {};
```

#### 5.94.4.6 m\_pid

```
control::PID wisco::robot::subsystems::loader::PIDLoader::m_pid {} [private]
```

The position PID controller.

Definition at line 109 of file [PIDLoader.hpp](#).

```
00109 {};
```

#### 5.94.4.7 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::loader::PIDLoader::m_motors {} [private]
```

The motors on the loader.

Definition at line 115 of file [PIDLoader.hpp](#).

```
00115 {};
```

#### 5.94.4.8 m\_match\_load\_position

```
double wisco::robot::subsystems::loader::PIDLoader::m_match_load_position {} [private]
```

The position for match loads.

Definition at line 121 of file [PIDLoader.hpp](#).

```
00121 {};
```

#### 5.94.4.9 m\_ready\_position

```
double wisco::robot::subsystems::loader::PIDLoader::m_ready_position {} [private]
```

The position for ready.

Definition at line 127 of file [PIDLoader.hpp](#).  
00127 {};

#### 5.94.4.10 m\_position\_tolerance

```
double wisco::robot::subsystems::loader::PIDLoader::m_position_tolerance {} [private]
```

The allowed tolerance around the target positions.

Definition at line 133 of file [PIDLoader.hpp](#).  
00133 {};

#### 5.94.4.11 state

```
EState wisco::robot::subsystems::loader::PIDLoader::state {EState::READY} [private]
```

The state of the loader.

Definition at line 139 of file [PIDLoader.hpp](#).  
00139 {EState::READY};

#### 5.94.4.12 target\_position

```
double wisco::robot::subsystems::loader::PIDLoader::target_position {} [private]
```

The position setting of the loader.

Definition at line 145 of file [PIDLoader.hpp](#).  
00145 {};

## 5.95 wisco::robot::subsystems::loader::PIDLoaderBuilder Class Reference

Builder class for a pid-based loader system.

## Public Member Functions

- `PIDLoaderBuilder * withClock (const std::unique_ptr< rtos::IClock > &clock)`  
*Add a rtos clock to the build.*
- `PIDLoaderBuilder * withDelayer (const std::unique_ptr< rtos::IDelay > &delayer)`  
*Add a rtos delayer to the build.*
- `PIDLoaderBuilder * withMutex (std::unique_ptr< rtos::IMutex > &mutex)`  
*Add a thread mutex to the build.*
- `PIDLoaderBuilder * withTask (std::unique_ptr< rtos::ITask > &task)`  
*Add a task handler to the build.*
- `PIDLoaderBuilder * withPID (control::PID pid)`  
*Add a PID controller to the build.*
- `PIDLoaderBuilder * withMotor (std::unique_ptr< io::IMotor > &motor)`  
*Add a motor to the build.*
- `PIDLoaderBuilder * withMatchLoadPosition (double match_load_position)`  
*Add a position for match loads to the build.*
- `PIDLoaderBuilder * withReadyPosition (double ready_position)`  
*Add a position for ready to the build.*
- `PIDLoaderBuilder * withPositionTolerance (double position_tolerance)`  
*Add an allowed tolerance around the target positions to the build.*
- `std::unique_ptr< ILoader > build ()`  
*Builds the pid loader.*

## Private Attributes

- `std::unique_ptr< rtos::IClock > m_clock {}`  
*The rtos clock.*
- `std::unique_ptr< rtos::IDelay > m_delay {}`  
*The rtos delayer.*
- `std::unique_ptr< rtos::IMutex > m_mutex {}`  
*The mutex for thread safety.*
- `std::unique_ptr< rtos::ITask > m_task {}`  
*The background task handler.*
- `control::PID m_pid {}`  
*The position PID controller.*
- `hal::MotorGroup m_motors {}`  
*The motors on the loader.*
- `double m_match_load_position {}`  
*The position for match loads.*
- `double m_ready_position {}`  
*The position for ready.*
- `double m_position_tolerance {}`  
*The allowed tolerance around the target positions.*

### 5.95.1 Detailed Description

Builder class for a pid-based loader system.

#### Author

Nathan Sandvig

Definition at line 43 of file `PIDLoaderBuilder.hpp`.

## 5.95.2 Member Function Documentation

### 5.95.2.1 withClock()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withClock (
    const std::unique_ptr< rtos::IClock > & clock )
```

Add a rtos clock to the build.

#### Parameters

<i>clock</i>	The rtos clock
--------------	----------------

#### Returns

This object for build chaining

Definition at line 11 of file PIDLoaderBuilder.cpp.

```
00012 {
00013     m_clock = clock->clone();
00014     return this;
00015 }
```

### 5.95.2.2 withDelayer()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withDelayer (
    const std::unique_ptr< rtos::IDelayer > & delayer )
```

Add a rtos delayer to the build.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

#### Returns

This object for build chaining

Definition at line 17 of file PIDLoaderBuilder.cpp.

```
00018 {
00019     m_delayer = delayer->clone();
00020     return this;
00021 }
```

### 5.95.2.3 withMutex()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Add a thread mutex to the build.

**Parameters**

<i>mutex</i>	The thread mutex
--------------	------------------

**Returns**

This object for build chaining

Definition at line 23 of file [PIDLoaderBuilder.cpp](#).

```
00024 {
00025     m_mutex = std::move(mutex);
00026     return this;
00027 }
```

**5.95.2.4 withTask()**

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withTask (
    std::unique_ptr< rtos::ITask > & task )
```

Add a task handler to the build.

**Parameters**

<i>task</i>	The task handler
-------------	------------------

**Returns**

This object for build chaining

Definition at line 29 of file [PIDLoaderBuilder.cpp](#).

```
00030 {
00031     m_task = std::move(task);
00032     return this;
00033 }
```

**5.95.2.5 withPID()**

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withPID (
    control::PID pid )
```

Add a PID controller to the build.

**Parameters**

<i>pid</i>	The PID controller
------------	--------------------

**Returns**

This object for build chaining

Definition at line 35 of file [PIDLoaderBuilder.cpp](#).

```
00036 {
00037     m_pid = pid;
00038     return this;
00039 }
```

### 5.95.2.6 withMotor()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withMotor (
    std::unique_ptr< io::IMotor > & motor )
```

Add a motor to the build.

#### Parameters

<i>motor</i>	The motor
--------------	-----------

#### Returns

This object for build chaining

Definition at line 41 of file [PIDLoaderBuilder.cpp](#).

```
00042 {
00043     m_motors.addMotor(motor);
00044     return this;
00045 }
```

### 5.95.2.7 withMatchLoadPosition()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withMatchLoadPosition (
    double match_load_position )
```

Add a position for match loads to the build.

#### Parameters

<i>match_load_position</i>	The position for match loads
----------------------------	------------------------------

#### Returns

This object for build chaining

Definition at line 47 of file [PIDLoaderBuilder.cpp](#).

```
00048 {
00049     m_match_load_position = match_load_position;
00050     return this;
00051 }
```

### 5.95.2.8 withReadyPosition()

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withReadyPosition (
    double ready_position )
```

Add a position for ready to the build.

**Parameters**

<i>ready_position</i>	The position for ready
-----------------------	------------------------

**Returns**

This object for build chaining

Definition at line 53 of file PIDLoaderBuilder.cpp.

```
00054 {
00055     m_ready_position = ready_position;
00056     return this;
00057 }
```

**5.95.2.9 withPositionTolerance()**

```
PIDLoaderBuilder * wisco::robot::subsystems::loader::PIDLoaderBuilder::withPositionTolerance (
    double position_tolerance )
```

Add an allowed tolerance around the target positions to the build.

**Parameters**

<i>position_tolerance</i>	The allowed tolerance around the target positions
---------------------------	---

**Returns**

This object for build chaining

Definition at line 59 of file PIDLoaderBuilder.cpp.

```
00060 {
00061     m_position_tolerance = position_tolerance;
00062     return this;
00063 }
```

**5.95.2.10 build()**

```
std::unique_ptr< ILoader > wisco::robot::subsystems::loader::PIDLoaderBuilder::build ( )
```

Builds the pid loader.

**Returns**

`std::unique_ptr<ILoader>` The pid loader as a loader interface

Definition at line 65 of file PIDLoaderBuilder.cpp.

```
00066 {
00067     std::unique_ptr<PIDLoader> pid_loader{std::make_unique<PIDLoader>()};
00068     pid_loader->setClock(m_clock);
00069     pid_loader->setDelayer(m_delayer);
00070     pid_loader->setMutex(m_mutex);
00071     pid_loader->setTask(m_task);
00072     pid_loader->setMotors(m_motors);
00073     pid_loader->setPID(m_pid);
00074     pid_loader->setMatchLoadPosition(m_match_load_position);
00075     pid_loader->setReadyPosition(m_ready_position);
00076     pid_loader->setPositionTolerance(m_position_tolerance);
00077     return pid_loader;
00078 }
```

### 5.95.3 Member Data Documentation

#### 5.95.3.1 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::loader::PIDLoaderBuilder::m_clock {}  
[private]
```

The rtos clock.

Definition at line 50 of file [PIDLoaderBuilder.hpp](#).  
00050 {};

#### 5.95.3.2 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::loader::PIDLoaderBuilder::m_delayer {}  
[private]
```

The rtos delayer.

Definition at line 56 of file [PIDLoaderBuilder.hpp](#).  
00056 {};

#### 5.95.3.3 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::loader::PIDLoaderBuilder::m_mutex {}  
[private]
```

The mutex for thread safety.

Definition at line 62 of file [PIDLoaderBuilder.hpp](#).  
00062 {};

#### 5.95.3.4 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::loader::PIDLoaderBuilder::m_task {}  
[private]
```

The background task handler.

Definition at line 68 of file [PIDLoaderBuilder.hpp](#).  
00068 {};

#### 5.95.3.5 m\_pid

```
control::PID wisco::robot::subsystems::loader::PIDLoaderBuilder::m_pid {} [private]
```

The position PID controller.

Definition at line 74 of file [PIDLoaderBuilder.hpp](#).  
00074 {};

### 5.95.3.6 m\_motors

```
hal::MotorGroup wisco::robot::subsystems::loader::PIDLoaderBuilder::m_motors {} [private]
```

The motors on the loader.

Definition at line 80 of file [PIDLoaderBuilder.hpp](#).

```
00080 {};
```

### 5.95.3.7 m\_match\_load\_position

```
double wisco::robot::subsystems::loader::PIDLoaderBuilder::m_match_load_position {} [private]
```

The position for match loads.

Definition at line 86 of file [PIDLoaderBuilder.hpp](#).

```
00086 {};
```

### 5.95.3.8 m\_ready\_position

```
double wisco::robot::subsystems::loader::PIDLoaderBuilder::m_ready_position {} [private]
```

The position for ready.

Definition at line 92 of file [PIDLoaderBuilder.hpp](#).

```
00092 {};
```

### 5.95.3.9 m\_position\_tolerance

```
double wisco::robot::subsystems::loader::PIDLoaderBuilder::m_position_tolerance {} [private]
```

The allowed tolerance around the target positions.

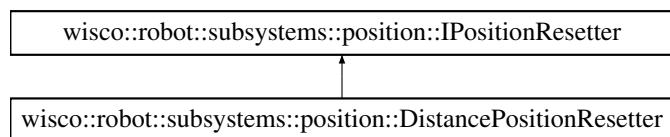
Definition at line 98 of file [PIDLoaderBuilder.hpp](#).

```
00098 {};
```

## 5.96 wisco::robot::subsystems::position::DistancePositionResetter Class Reference

Interface for position resetting subsystems.

Inheritance diagram for wisco::robot::subsystems::position::DistancePositionResetter:



## Public Member Functions

- void [initialize \(\)](#) override  
*Initializes the position resetting system.*
- void [run \(\)](#) override  
*Runs the position tracking system.*
- double [getResetX \(double theta\)](#) override  
*Get the reset position for X.*
- double [getResetY \(double theta\)](#) override  
*Get the reset position for Y.*
- void [setDistanceSensor \(std::unique\\_ptr< io::IDistanceSensor > &distance\\_sensor\)](#)  
*Set the distance sensor.*
- void [setLocalX \(double local\\_x\)](#)  
*Set the local x-offset.*
- void [setLocalY \(double local\\_y\)](#)  
*Set the local y-offset.*
- void [setLocalTheta \(double local\\_theta\)](#)  
*Set the local angle-offset.*

## Public Member Functions inherited from wisco::robot::subsystems::position::IPositionResetter

- virtual ~[IPositionResetter \(\)](#)=default  
*Destroy the [IPositionResetter](#) object.*

## Private Member Functions

- double [bindRadians \(double radians\)](#)  
*Binds radian values between pi and -pi.*

## Private Attributes

- std::unique\_ptr< [io::IDistanceSensor](#) > [m\\_distance\\_sensor {}](#)  
*The distance sensor used to reset the position.*
- double [m\\_local\\_x {}](#)  
*The local x-offset of the distance sensor.*
- double [m\\_local\\_y {}](#)  
*The local y-offset of the distance sensor.*
- double [m\\_local\\_theta {}](#)  
*The local angle-offset of the distance sensor.*

## Static Private Attributes

- static constexpr double [NEAR\\_WALL {0}](#)  
*The coordinate of the near wall.*
- static constexpr double [FAR\\_WALL {144}](#)  
*The coordinate of the far wall.*

## 5.96.1 Detailed Description

Interface for position resetting subsystems.

### Author

Nathan Sandvig

Definition at line 48 of file [DistancePositionResetter.hpp](#).

## 5.96.2 Member Function Documentation

### 5.96.2.1 bindRadians()

```
double wisco::robot::subsystems::position::DistancePositionResetter::bindRadians (
    double radians ) [private]
```

Binds radian values between pi and -pi.

#### Parameters

<i>radians</i>	The raw radian value
----------------	----------------------

#### Returns

double The bound radian value

Definition at line 11 of file [DistancePositionResetter.cpp](#).

```
00012 {
00013     while (radians > M_PI)
00014         radians -= (2 * M_PI);
00015     while (radians < -M_PI)
00016         radians += (2 * M_PI);
00017     return radians;
00018 }
```

### 5.96.2.2 initialize()

```
void wisco::robot::subsystems::position::DistancePositionResetter::initialize () [override],
[virtual]
```

Initializes the position resetting system.

Implements [wisco::robot::subsystems::position::IPositionResetter](#).

Definition at line 20 of file [DistancePositionResetter.cpp](#).

```
00021 {
00022     if (m_distance_sensor)
00023         m_distance_sensor->initialize();
00024 }
```

### 5.96.2.3 run()

```
void wisco::robot::subsystems::position::DistancePositionResetter::run () [override], [virtual]
```

Runs the position tracking system.

Implements [wisco::robot::subsystems::position::IPositionResetter](#).

Definition at line 26 of file [DistancePositionResetter.cpp](#).

```
00027 {  
00028     // No running code  
00029 }
```

### 5.96.2.4 getResetX()

```
double wisco::robot::subsystems::position::DistancePositionResetter::getResetX (  
    double theta) [override], [virtual]
```

Get the reset position for X.

#### Parameters

<i>theta</i>	The current robot angle
--------------	-------------------------

#### Returns

double The X reset position

Implements [wisco::robot::subsystems::position::IPositionResetter](#).

Definition at line 31 of file [DistancePositionResetter.cpp](#).

```
00032 {  
00033     double reset_x{};  
00034     if (m_distance_sensor)  
00035     {  
00036         theta = bindRadians(theta);  
00037         double distance{m_distance_sensor->getDistance()};  
00038         double wall_theta{bindRadians(theta + m_local_theta)};  
00039         bool near_wall{std::abs(wall_theta) > M_PI / 2};  
00040         if (near_wall)  
00041             wall_theta = bindRadians(wall_theta + M_PI);  
00042         double sensor_distance{distance * std::cos(wall_theta)};  
00043         double local_x_distance{m_local_x * std::cos(theta)};  
00044         double local_y_distance{m_local_y * std::cos(bindRadians(theta + (M_PI / 2)))};  
00045         double wall_distance{sensor_distance + local_x_distance + local_y_distance};  
00046         if (near_wall)  
00047             reset_x = NEAR_WALL + wall_distance;  
00048         else  
00049             reset_x = FAR_WALL - wall_distance;  
00050     }  
00051     return reset_x;  
00052 }
```

### 5.96.2.5 getResetY()

```
double wisco::robot::subsystems::position::DistancePositionResetter::getResetY (  
    double theta) [override], [virtual]
```

Get the reset position for Y.

**Parameters**

<i>theta</i>	The current robot angle
--------------	-------------------------

**Returns**

double The Y reset position

Implements [wisco::robot::subsystems::position::IPositionResetter](#).

Definition at line 54 of file [DistancePositionResetter.cpp](#).

```
00055 {
00056     theta -= (M_PI / 2);
00057     return getResetX(theta);
00058 }
```

**5.96.2.6 setDistanceSensor()**

```
void wisco::robot::subsystems::position::DistancePositionResetter::setDistanceSensor (
    std::unique_ptr< io::IDistanceSensor > & distance_sensor )
```

Set the distance sensor.

**Parameters**

<i>distance_sensor</i>	The distance sensor
------------------------	---------------------

Definition at line 60 of file [DistancePositionResetter.cpp](#).

```
00061 {
00062     if (distance_sensor)
00063         m_distance_sensor = std::move(distance_sensor);
00064 }
```

**5.96.2.7 setLocalX()**

```
void wisco::robot::subsystems::position::DistancePositionResetter::setLocalX (
    double local_x )
```

Set the local x-offset.

**Parameters**

<i>local_x</i>	The local x-offset
----------------	--------------------

Definition at line 66 of file [DistancePositionResetter.cpp](#).

```
00067 {
00068     m_local_x = local_x;
00069 }
```

**5.96.2.8 setLocalY()**

```
void wisco::robot::subsystems::position::DistancePositionResetter::setLocalY (
```

```
double local_y )
```

Set the local y-offset.

#### Parameters

<i>local_y</i>	The local y-offset
----------------	--------------------

Definition at line 71 of file [DistancePositionResetter.cpp](#).

```
00072 {
00073     m_local_y = local_y;
00074 }
```

### 5.96.2.9 setLocalTheta()

```
void wisco::robot::subsystems::position::DistancePositionResetter::setLocalTheta (
    double local_theta )
```

Set the local angle-offset.

#### Parameters

<i>local_theta</i>	The local angle-offset
--------------------	------------------------

Definition at line 76 of file [DistancePositionResetter.cpp](#).

```
00077 {
00078     m_local_theta = local_theta;
00079 }
```

## 5.96.3 Member Data Documentation

### 5.96.3.1 NEAR\_WALL

```
constexpr double wisco::robot::subsystems::position::DistancePositionResetter::NEAR_WALL {0}
[static], [constexpr], [private]
```

The coordinate of the near wall.

Definition at line 55 of file [DistancePositionResetter.hpp](#).

```
00055 {0};
```

### 5.96.3.2 FAR\_WALL

```
constexpr double wisco::robot::subsystems::position::DistancePositionResetter::FAR_WALL {144}
[static], [constexpr], [private]
```

The coordinate of the far wall.

Definition at line 61 of file [DistancePositionResetter.hpp](#).

```
00061 {144};
```

### 5.96.3.3 m\_distance\_sensor

```
std::unique_ptr<io::IDistanceSensor> wisco::robot::subsystems::position::DistancePositionResetter::m_distance_sensor {} [private]
```

The distance sensor used to reset the position.

Definition at line 67 of file [DistancePositionResetter.hpp](#).

```
00067 {};
```

### 5.96.3.4 m\_local\_x

```
double wisco::robot::subsystems::position::DistancePositionResetter::m_local_x {} [private]
```

The local x-offset of the distance sensor.

Definition at line 73 of file [DistancePositionResetter.hpp](#).

```
00073 {};
```

### 5.96.3.5 m\_local\_y

```
double wisco::robot::subsystems::position::DistancePositionResetter::m_local_y {} [private]
```

The local y-offset of the distance sensor.

Definition at line 79 of file [DistancePositionResetter.hpp](#).

```
00079 {};
```

### 5.96.3.6 m\_local\_theta

```
double wisco::robot::subsystems::position::DistancePositionResetter::m_local_theta {} [private]
```

The local angle-offset of the distance sensor.

Definition at line 85 of file [DistancePositionResetter.hpp](#).

```
00085 {};
```

## 5.97 wisco::robot::subsystems::position::DistancePositionResetter Builder Class Reference

Builder for distance position resetters.

### Public Member Functions

- `DistancePositionResetterBuilder * withDistanceSensor (std::unique_ptr<io::IDistanceSensor> &distance_sensor)`  
*Add a distance sensor to the build.*
- `DistancePositionResetterBuilder * withLocalX (double local_x)`  
*Add a local x-offset to the build.*
- `DistancePositionResetterBuilder * withLocalY (double local_y)`  
*Add a local y-offset to the build.*
- `DistancePositionResetterBuilder * withLocalTheta (double local_theta)`  
*Add a local angle-offset to the build.*
- `std::unique_ptr<IPositionResetter> build ()`  
*Builds the distance position resetter system.*

**Private Attributes**

- std::unique\_ptr< io::IDistanceSensor > m\_distance\_sensor {}  
*The distance sensor used to reset the position.*
- double m\_local\_x {}  
*The local x-offset of the distance sensor.*
- double m\_local\_y {}  
*The local y-offset of the distance sensor.*
- double m\_local\_theta {}  
*The local angle-offset of the distance sensor.*

**5.97.1 Detailed Description**

Builder for distance position resetters.

**Author**

Nathan Sandvig

Definition at line 43 of file [DistancePositionResetterBuilder.hpp](#).

**5.97.2 Member Function Documentation****5.97.2.1 withDistanceSensor()**

```
DistancePositionResetterBuilder * wisco::robot::subsystems::position::DistancePositionResetterBuilder::withDistanceSensor (
    std::unique_ptr< io::IDistanceSensor > & distance_sensor )
```

Add a distance sensor to the build.

**Parameters**

<i>distance_sensor</i>	The distance sensor
------------------------	---------------------

**Returns**

This object for build chaining

Definition at line 11 of file [DistancePositionResetterBuilder.cpp](#).

```
00012 {
00013     m_distance_sensor = std::move(distance_sensor);
00014     return this;
00015 }
```

**5.97.2.2 withLocalX()**

```
DistancePositionResetterBuilder * wisco::robot::subsystems::position::DistancePositionResetterBuilder::withLocalX (
    double local_x )
```

Add a local x-offset to the build.

**Parameters**

<i>local_x</i>	The local x-offset
----------------	--------------------

**Returns**

This object for build chaining

Definition at line 17 of file [DistancePositionResetterBuilder.cpp](#).

```
00018 {
00019     m_local_x = local_x;
00020     return this;
00021 }
```

**5.97.2.3 withLocalY()**

```
DistancePositionResetterBuilder * wisco::robot::subsystems::position::DistancePositionResetterBuilder::withLocalY (
    double local_y )
```

Add a local y-offset to the build.

**Parameters**

<i>local_y</i>	The local y-offset
----------------	--------------------

**Returns**

This object for build chaining

Definition at line 23 of file [DistancePositionResetterBuilder.cpp](#).

```
00024 {
00025     m_local_y = local_y;
00026     return this;
00027 }
```

**5.97.2.4 withLocalTheta()**

```
DistancePositionResetterBuilder * wisco::robot::subsystems::position::DistancePositionResetterBuilder::withLocalTheta (
    double local_theta )
```

Add a local angle-offset to the build.

**Parameters**

<i>local_theta</i>	The local angle-offset
--------------------	------------------------

**Returns**

This object for build chaining

Definition at line 29 of file [DistancePositionResetterBuilder.cpp](#).

```
00030 {
00031     m_local_theta = local_theta;
00032     return this;
00033 }
```

**5.97.2.5 build()**

```
std::unique_ptr< IPositionResetter > wisco::robot::subsystems::position::DistancePositionResetterBuilder::build ( )
```

Builds the distance position resetter system.

**Returns**

`std::unique_ptr<IPositionResetter>` The distance position resetter as a position resetting interface

Definition at line 35 of file [DistancePositionResetterBuilder.cpp](#).

```
00036 {
00037     std::unique_ptr<DistancePositionResetter>
00038         distance_position_resetter{std::make_unique<DistancePositionResetter>()};
00039     distance_position_resetter->setDistanceSensor(m_distance_sensor);
00040     distance_position_resetter->setLocalX(m_local_x);
00041     distance_position_resetter->setLocalY(m_local_y);
00042     distance_position_resetter->setLocalTheta(m_local_theta);
00043     return distance_position_resetter;
00044 }
```

**5.97.3 Member Data Documentation****5.97.3.1 m\_distance\_sensor**

```
std::unique_ptr<io::IDistanceSensor> wisco::robot::subsystems::position::DistancePositionResetterBuilder::m_distance_sensor {} [private]
```

The distance sensor used to reset the position.

Definition at line 50 of file [DistancePositionResetterBuilder.hpp](#).

```
00050 {};
```

**5.97.3.2 m\_local\_x**

```
double wisco::robot::subsystems::position::DistancePositionResetterBuilder::m_local_x {} [private]
```

The local x-offset of the distance sensor.

Definition at line 56 of file [DistancePositionResetterBuilder.hpp](#).

```
00056 {};
```

### 5.97.3.3 m\_local\_y

```
double wisco::robot::subsystems::position::DistancePositionResetterBuilder::m_local_y {} [private]
```

The local y-offset of the distance sensor.

Definition at line 62 of file [DistancePositionResetterBuilder.hpp](#).  
00062 {};

### 5.97.3.4 m\_local\_theta

```
double wisco::robot::subsystems::position::DistancePositionResetterBuilder::m_local_theta {} [private]
```

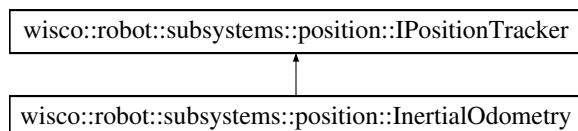
The local angle-offset of the distance sensor.

Definition at line 68 of file [DistancePositionResetterBuilder.hpp](#).  
00068 {};

## 5.98 wisco::robot::subsystems::position::InertialOdometry Class Reference

An odometry system based on a heading sensor with two distance tracking sensors.

Inheritance diagram for wisco::robot::subsystems::position::InertialOdometry:



### Public Member Functions

- void [initialize \(\)](#) override  
*Initializes the position tracking system.*
- void [run \(\)](#) override  
*Runs the position tracking system.*
- void [setPosition \(Position position\)](#) override  
*Set the position of the position tracking system.*
- Position [getPosition \(\)](#) override  
*Get the position of the system.*
- void [setClock \(std::unique\\_ptr< rtos::IClock > &clock\)](#)  
*Set the system clock.*
- void [setDelayer \(std::unique\\_ptr< rtos::IDelayer > &delayer\)](#)  
*Set the rtos delayer.*
- void [setMutex \(std::unique\\_ptr< rtos::IMutex > &mutex\)](#)  
*Set the os mutex.*
- void [setTask \(std::unique\\_ptr< rtos::ITask > &task\)](#)

- Set the rtos task handler.
- void `setHeadingSensor` (std::unique\_ptr< `io::IHeadingSensor` > &heading\_sensor)
  - Set the heading sensor.
- void `setLinearDistanceTrackingSensor` (std::unique\_ptr< `io::IDistanceTrackingSensor` > &linear\_distance\_tracking\_sensor)
  - Set the linear distance tracking sensor.
- void `setLinearDistanceTrackingOffset` (double linear\_distance\_tracking\_offset)
  - Set the linear distance tracking offset.
- void `setStrafeDistanceTrackingSensor` (std::unique\_ptr< `io::IDistanceTrackingSensor` > &strafe\_distance\_tracking\_sensor)
  - Set the strafe distance tracking sensor.
- void `setStrafeDistanceTrackingOffset` (double strafe\_distance\_tracking\_offset)
  - Set the strafe distance tracking offset.

## Public Member Functions inherited from `wisco::robot::subsystems::position::IPositionTracker`

- virtual ~`IPositionTracker` ()=default

*Destroy the `IPositionTracker` object.*

## Private Member Functions

- void `taskUpdate` ()
  - Runs all the object-specific updates in the task loop.*
- void `updatePosition` ()
  - Updates the position of the system.*

## Static Private Member Functions

- static void `taskLoop` (void \*params)
  - The task loop function for background updates.*

## Private Attributes

- std::unique\_ptr< `rtos::IClock` > `m_clock` {}
  - The system clock.*
- std::unique\_ptr< `rtos::IDelay` > `m_delayer` {}
  - The system delayer.*
- std::unique\_ptr< `rtos::IMutex` > `m_mutex` {}
  - The os mutex.*
- std::unique\_ptr< `rtos::ITask` > `m_task` {}
  - The task handler.*
- std::unique\_ptr< `io::IHeadingSensor` > `m_heading_sensor` {}
  - The sensor to track the robot's heading.*
- std::unique\_ptr< `io::IDistanceTrackingSensor` > `m_linear_distance_tracking_sensor` {}
  - The sensor to track the robot's linear movement.*
- double `m_linear_distance_tracking_offset` {}
  - The offset from the linear distance tracking sensor to the tracking center This value is left-justified, I.E. the offset to the left of center.*

- `std::unique_ptr< io::IDistanceTrackingSensor > m_strafe_distance_tracking_sensor {}`  
*The sensor to track the robot's strafe movement.*
- `double m_strafe_distance_tracking_offset {}`  
*The offset from the strafe distance tracking sensor to the tracking center This value is front-justified, I.E. the offset to the front of center.*
- `Position m_position {}`  
*The current position of the system.*
- `double last_heading {}`  
*The last value of the heading sensor.*
- `double last_linear_distance {}`  
*The last value of the linear distance tracking sensor.*
- `double last_strafe_distance {}`  
*The last value of the strafe distance tracking sensor.*
- `uint32_t last_time {}`  
*The last value of the system clock time.*

### Static Private Attributes

- `static constexpr uint8_t TASK_DELAY {10}`  
*The loop delay on the task.*
- `static constexpr double TIME_UNIT_CONVERTER {1000}`  
*Converts the time units for velocity.*

### 5.98.1 Detailed Description

An odometry system based on a heading sensor with two distance tracking sensors.

#### Author

Nathan Sandvig

Definition at line 53 of file [InertialOdometry.hpp](#).

### 5.98.2 Member Function Documentation

#### 5.98.2.1 taskLoop()

```
void wisco::robot::subsystems::position::InertialOdometry::taskLoop (
    void * params ) [static], [private]
```

The task loop function for background updates.

#### Parameters

<code>params</code>	<input type="text"/>
---------------------	----------------------

Definition at line 12 of file [InertialOdometry.cpp](#).

```

00013 {
00014     void** parameters{static_cast<void**>(params)};
00015     InertialOdometry* instance{static_cast<InertialOdometry*>(parameters[0])};
00016
00017     while (true)
00018     {
00019         instance->taskUpdate();
00020     }
00021 }
```

### 5.98.2.2 taskUpdate()

```
void wisco::robot::subsystems::position::InertialOdometry::taskUpdate () [private]
```

Runs all the object-specific updates in the task loop.

Definition at line 23 of file [InertialOdometry.cpp](#).

```

00024 {
00025     updatePosition();
00026     m_delayer->delay(TASK_DELAY);
00027 }
```

### 5.98.2.3 updatePosition()

```
void wisco::robot::subsystems::position::InertialOdometry::updatePosition () [private]
```

Updates the position of the system.

Definition at line 29 of file [InertialOdometry.cpp](#).

```

00030 {
00031     if (m_mutex)
00032         m_mutex->take();
00033
00034     double current_heading{};
00035     double current_linear_distance{};
00036     double current_strafe_distance{};
00037     uint32_t current_time{};
00038
00039     if (m_heading_sensor)
00040         current_heading = m_heading_sensor->getRotation();
00041     if (m_linear_distance_tracking_sensor)
00042         current_linear_distance = m_linear_distance_tracking_sensor->getDistance();
00043     if (m_strafe_distance_tracking_sensor)
00044         current_strafe_distance = m_strafe_distance_tracking_sensor->getDistance();
00045     if (m_clock)
00046         current_time = m_clock->getTime();
00047
00048     double heading_change{current_heading - last_heading};
00049     double linear_change{current_linear_distance - last_linear_distance};
00050     double strafe_change{current_strafe_distance - last_strafe_distance};
00051     uint32_t time_change{current_time - last_time};
00052
00053     double local_x{};
00054     double local_y{};
00055     double local_theta{};
00056     if (heading_change != 0.0)
00057     {
00058         double linear_radius{(linear_change / heading_change) - m_linear_distance_tracking_offset};
00059         double strafe_radius{(strafe_change / heading_change) - m_strafe_distance_tracking_offset};
00060         local_x = 2 * std::sin(heading_change / 2) * strafe_radius;
00061         local_y = 2 * std::sin(heading_change / 2) * linear_radius;
00062         local_theta = (last_heading + current_heading) / 2;
00063     }
00064     else
00065     {
00066         local_x = strafe_change;
00067         local_y = linear_change;
00068         local_theta = current_heading;
00069     }
00070
00071     double global_x{(local_x * std::sin(local_theta)) + (local_y * std::cos(local_theta))};
00072     double global_y{(local_y * std::sin(local_theta)) + (-local_x * std::cos(local_theta))};
00073
00074     m_position.x += global_x;
```

```

00075     m_position.y += global_y;
00076     m_position.theta = current_heading;
00077
00078     if (m_clock)
00079     {
00080         m_position.xV = global_x / (time_change / TIME_UNIT_CONVERTER);
00081         m_position.yV = global_y / (time_change / TIME_UNIT_CONVERTER);
00082         m_position.thetaV = heading_change / (time_change / TIME_UNIT_CONVERTER);
00083     }
00084
00085     last_heading = current_heading;
00086     last_linear_distance = current_linear_distance;
00087     last_strafe_distance = current_strafe_distance;
00088     last_time = current_time;
00089
00090     if (m_mutex)
00091         m_mutex->give();
00092 }
```

#### 5.98.2.4 initialize()

void wisco::robot::subsystems::position::InertialOdometry::initialize () [override], [virtual]

Initializes the position tracking system.

Implements [wisco::robot::subsystems::position::IPositionTracker](#).

Definition at line 94 of file [InertialOdometry.cpp](#).

```

00095 {
00096     if (m_heading_sensor)
00097     {
00098         m_heading_sensor->initialize();
00099         last_heading = m_heading_sensor->getRotation();
00100     }
00101     if (m_linear_distance_tracking_sensor)
00102     {
00103         m_linear_distance_tracking_sensor->initialize();
00104         last_linear_distance = m_linear_distance_tracking_sensor->getDistance();
00105     }
00106     if (m_strafe_distance_tracking_sensor)
00107     {
00108         m_strafe_distance_tracking_sensor->initialize();
00109         last_strafe_distance = m_strafe_distance_tracking_sensor->getDistance();
00110     }
00111     if (m_clock)
00112         last_time = m_clock->getTime();
00113 }
```

#### 5.98.2.5 run()

void wisco::robot::subsystems::position::InertialOdometry::run () [override], [virtual]

Runs the position tracking system.

Implements [wisco::robot::subsystems::position::IPositionTracker](#).

Definition at line 115 of file [InertialOdometry.cpp](#).

```

00116 {
00117     if (m_task)
00118     {
00119         void** params{static_cast<void**>(malloc(1 * sizeof(void*)))};
00120         params[0] = this;
00121         m_task->start(&InertialOdometry::taskLoop, params);
00122     }
00123 }
```

#### 5.98.2.6 setPosition()

void wisco::robot::subsystems::position::InertialOdometry::setPosition (
 Position position ) [override], [virtual]

Set the position of the position tracking system.

**Parameters**

<i>position</i>	The new position
-----------------	------------------

Implements [wisco::robot::subsystems::position::IPositionTracker](#).

Definition at line 125 of file [InertialOdometry.cpp](#).

```
00126 {
00127     if (m_mutex)
00128         m_mutex->take();
00129     m_position = position;
00130     if (m_heading_sensor)
00131         m_heading_sensor->setRotation(position.theta);
00132     last_heading = position.theta;
00133     if (m_mutex)
00134         m_mutex->give();
00135 }
```

**5.98.2.7 getPosition()**

`Position` [wisco::robot::subsystems::position::InertialOdometry::getPosition](#) ( ) [override], [virtual]

Get the position of the system.

**Returns**

`Position` The position of the system

Implements [wisco::robot::subsystems::position::IPositionTracker](#).

Definition at line 137 of file [InertialOdometry.cpp](#).

```
00138 {
00139     Position position{};
00140     if (m_mutex)
00141         m_mutex->take();
00142     position = m_position;
00143     if (m_mutex)
00144         m_mutex->give();
00145     return position;
00146 }
```

**5.98.2.8 setClock()**

```
void wisco::robot::subsystems::position::InertialOdometry::setClock (
    std::unique_ptr< rtos::IClock > & clock )
```

Set the system clock.

**Parameters**

<i>clock</i>	The system clock
--------------	------------------

Definition at line 148 of file [InertialOdometry.cpp](#).

```
00149 {
00150     m_clock = std::move(clock);
00151 }
```

### 5.98.2.9 setDelayer()

```
void wisco::robot::subsystems::position::InertialOdometry::setDelayer (
    std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

#### Parameters

<i>delayer</i>	The rtos delayer
----------------	------------------

Definition at line 153 of file [InertialOdometry.cpp](#).

```
00154 {
00155     m_delayer = std::move(delayer);
00156 }
```

### 5.98.2.10 setMutex()

```
void wisco::robot::subsystems::position::InertialOdometry::setMutex (
    std::unique_ptr< rtos::IMutex > & mutex )
```

Set the os mutex.

#### Parameters

<i>mutex</i>	The os mutex
--------------	--------------

Definition at line 158 of file [InertialOdometry.cpp](#).

```
00159 {
00160     m_mutex = std::move(mutex);
00161 }
```

### 5.98.2.11 setTask()

```
void wisco::robot::subsystems::position::InertialOdometry::setTask (
    std::unique_ptr< rtos::ITask > & task )
```

Set the rtos task handler.

#### Parameters

<i>task</i>	The rtos task handler
-------------	-----------------------

Definition at line 163 of file [InertialOdometry.cpp](#).

```
00164 {
00165     m_task = std::move(task);
00166 }
```

### 5.98.2.12 setHeadingSensor()

```
void wisco::robot::subsystems::position::InertialOdometry::setHeadingSensor (
    std::unique_ptr< io::IHeadingSensor > & heading_sensor )
```

Set the heading sensor.

**Parameters**

<i>heading_sensor</i>	The heading sensor
-----------------------	--------------------

Definition at line 168 of file [InertialOdometry.cpp](#).

```
00169 {
00170     m_heading_sensor = std::move(heading_sensor);
00171 }
```

**5.98.2.13 setLinearDistanceTrackingSensor()**

```
void wisco::robot::subsystems::position::InertialOdometry::setLinearDistanceTrackingSensor (
    std::unique_ptr< io::IDistanceTrackingSensor > & linear_distance_tracking_sensor
)
```

Set the linear distance tracking sensor.

**Parameters**

<i>linear_distance_tracking_sensor</i>	The linear distance tracking sensor
--	-------------------------------------

Definition at line 173 of file [InertialOdometry.cpp](#).

```
00174 {
00175     m_linear_distance_tracking_sensor = std::move(linear_distance_tracking_sensor);
00176 }
```

**5.98.2.14 setLinearDistanceTrackingOffset()**

```
void wisco::robot::subsystems::position::InertialOdometry::setLinearDistanceTrackingOffset (
    double linear_distance_tracking_offset )
```

Set the linear distance tracking offset.

**Parameters**

<i>linear_distance_tracking_offset</i>	The linear distance tracking offset
--	-------------------------------------

Definition at line 178 of file [InertialOdometry.cpp](#).

```
00179 {
00180     m_linear_distance_tracking_offset = linear_distance_tracking_offset;
00181 }
```

**5.98.2.15 setStrafeDistanceTrackingSensor()**

```
void wisco::robot::subsystems::position::InertialOdometry::setStrafeDistanceTrackingSensor (
    std::unique_ptr< io::IDistanceTrackingSensor > & strafe_distance_tracking_sensor
)
```

Set the strafe distance tracking sensor.

**Parameters**

<i>strafe_distance_tracking_sensor</i>	The strafe distance tracking sensor
--	-------------------------------------

Definition at line 183 of file [InertialOdometry.cpp](#).

```
00184 {
00185     m_strafe_distance_tracking_sensor = std::move(strafe_distance_tracking_sensor);
00186 }
```

**5.98.2.16 setStrafeDistanceTrackingOffset()**

```
void wisco::robot::subsystems::position::InertialOdometry::setStrafeDistanceTrackingOffset (
    double strafe_distance_tracking_offset )
```

Set the strafe distance tracking offset.

**Parameters**

<i>strafe_distance_tracking_offset</i>	The strafe distance tracking offset
--	-------------------------------------

Definition at line 188 of file [InertialOdometry.cpp](#).

```
00189 {
00190     m_strafe_distance_tracking_offset = strafe_distance_tracking_offset;
00191 }
```

**5.98.3 Member Data Documentation****5.98.3.1 TASK\_DELAY**

```
constexpr uint8_t wisco::robot::subsystems::position::InertialOdometry::TASK_DELAY {10} [static],
[constexpr], [private]
```

The loop delay on the task.

Definition at line 60 of file [InertialOdometry.hpp](#).

```
00060 {10};
```

**5.98.3.2 TIME\_UNIT\_CONVERTER**

```
constexpr double wisco::robot::subsystems::position::InertialOdometry::TIME_UNIT_CONVERTER
{1000} [static], [constexpr], [private]
```

Converts the time units for velocity.

Definition at line 66 of file [InertialOdometry.hpp](#).

```
00066 {1000};
```

### 5.98.3.3 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::position::InertialOdometry::m_clock {}  
[private]
```

The system clock.

Definition at line 79 of file [InertialOdometry.hpp](#).  
00079 {};

### 5.98.3.4 m\_delayer

```
std::unique_ptr<rtos::IDelay> wisco::robot::subsystems::position::InertialOdometry::m_↔  
delayer {} [private]
```

The system delayer.

Definition at line 85 of file [InertialOdometry.hpp](#).  
00085 {};

### 5.98.3.5 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::position::InertialOdometry::m_mutex {}  
[private]
```

The os mutex.

Definition at line 91 of file [InertialOdometry.hpp](#).  
00091 {};

### 5.98.3.6 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::position::InertialOdometry::m_task {}  
[private]
```

The task handler.

Definition at line 97 of file [InertialOdometry.hpp](#).  
00097 {};

### 5.98.3.7 m\_heading\_sensor

```
std::unique_ptr<io::IHeadingSensor> wisco::robot::subsystems::position::InertialOdometry::m_↔  
heading_sensor {} [private]
```

The sensor to track the robot's heading.

Definition at line 103 of file [InertialOdometry.hpp](#).  
00103 {};

### 5.98.3.8 m\_linear\_distance\_tracking\_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::Inertial->
Odometry::m_linear_distance_tracking_sensor {} [private]
```

The sensor to track the robot's linear movement.

Definition at line 109 of file [InertialOdometry.hpp](#).  
00109 {};

### 5.98.3.9 m\_linear\_distance\_tracking\_offset

```
double wisco::robot::subsystems::position::InertialOdometry::m_linear_distance_tracking_offset
{} [private]
```

The offset from the linear distance tracking sensor to the tracking center This value is left-justified, I.E. the offset to the left of center.

Definition at line 116 of file [InertialOdometry.hpp](#).  
00116 {};

### 5.98.3.10 m\_strafe\_distance\_tracking\_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::Inertial->
Odometry::m_strafe_distance_tracking_sensor {} [private]
```

The sensor to track the robot's strafe movement.

Definition at line 122 of file [InertialOdometry.hpp](#).  
00122 {};

### 5.98.3.11 m\_strafe\_distance\_tracking\_offset

```
double wisco::robot::subsystems::position::InertialOdometry::m_strafe_distance_tracking_offset
{} [private]
```

The offset from the strafe distance tracking sensor to the tracking center This value is front-justified, I.E. the offset to the front of center.

Definition at line 129 of file [InertialOdometry.hpp](#).  
00129 {};

### 5.98.3.12 m\_position

```
Position wisco::robot::subsystems::position::InertialOdometry::m_position {} [private]
```

The current position of the system.

Definition at line 135 of file [InertialOdometry.hpp](#).  
00135 {};

### 5.98.3.13 last\_heading

```
double wisco::robot::subsystems::position::InertialOdometry::last_heading {} [private]
```

The last value of the heading sensor.

Definition at line 141 of file [InertialOdometry.hpp](#).

```
00141 {};
```

### 5.98.3.14 last\_linear\_distance

```
double wisco::robot::subsystems::position::InertialOdometry::last_linear_distance {} [private]
```

The last value of the linear distance tracking sensor.

Definition at line 147 of file [InertialOdometry.hpp](#).

```
00147 {};
```

### 5.98.3.15 last\_strafe\_distance

```
double wisco::robot::subsystems::position::InertialOdometry::last_strafe_distance {} [private]
```

The last value of the strafe distance tracking sensor.

Definition at line 153 of file [InertialOdometry.hpp](#).

```
00153 {};
```

### 5.98.3.16 last\_time

```
uint32_t wisco::robot::subsystems::position::InertialOdometry::last_time {} [private]
```

The last value of the system clock time.

Definition at line 159 of file [InertialOdometry.hpp](#).

```
00159 {};
```

## 5.99 wisco::robot::subsystems::position::InertialOdometryBuilder Class Reference

Builder class for the inertial odometry class.

## Public Member Functions

- `InertialOdometryBuilder * withClock (std::unique_ptr< wisco::rtos::IClock > &clock)`  
*Adds a system clock to the builder.*
- `InertialOdometryBuilder * withDelayer (std::unique_ptr< wisco::rtos::IDelay > &delayer)`  
*Adds a system delayer to the builder.*
- `InertialOdometryBuilder * withMutex (std::unique_ptr< wisco::rtos::IMutex > &mutex)`  
*Adds a system mutex to the builder.*
- `InertialOdometryBuilder * withTask (std::unique_ptr< wisco::rtos::ITask > &task)`  
*Adds a system task to the builder.*
- `InertialOdometryBuilder * withHeadingSensor (std::unique_ptr< wisco::io::IHeadingSensor > &heading_sensor)`  
*Adds a heading sensor to the builder.*
- `InertialOdometryBuilder * withLinearDistanceTrackingSensor (std::unique_ptr< wisco::io::IDistanceTrackingSensor > &linear_distance_tracking_sensor)`  
*Adds a linear distance tracking sensor to the builder.*
- `InertialOdometryBuilder * withLinearDistanceTrackingOffset (double linear_distance_tracking_offset)`  
*Adds a linear distance tracking offset to the builder.*
- `InertialOdometryBuilder * withStrafeDistanceTrackingSensor (std::unique_ptr< wisco::io::IDistanceTrackingSensor > &strafe_distance_tracking_sensor)`  
*Adds a strafe distance tracking sensor to the builder.*
- `InertialOdometryBuilder * withStrafeDistanceTrackingOffset (double strafe_distance_tracking_offset)`  
*Adds a strafe distance tracking offset to the builder.*
- `std::unique_ptr< IPositionTracker > build ()`  
*Builds the inertial odometry system.*

## Private Attributes

- `std::unique_ptr< rtos::IClock > m_clock {}`  
*The system clock.*
- `std::unique_ptr< rtos::IDelay > m_delayer {}`  
*The system delayer.*
- `std::unique_ptr< rtos::IMutex > m_mutex {}`  
*The os mutex.*
- `std::unique_ptr< rtos::ITask > m_task {}`  
*The task handler.*
- `std::unique_ptr< io::IHeadingSensor > m_heading_sensor {}`  
*The sensor to track the robot's heading.*
- `std::unique_ptr< io::IDistanceTrackingSensor > m_linear_distance_tracking_sensor {}`  
*The sensor to track the robot's linear movement.*
- `double m_linear_distance_tracking_offset {}`  
*The offset from the linear distance tracking sensor to the tracking center This value is left-justified, I.E. the offset to the left of center.*
- `std::unique_ptr< io::IDistanceTrackingSensor > m_strafe_distance_tracking_sensor {}`  
*The sensor to track the robot's strafe movement.*
- `double m_strafe_distance_tracking_offset {}`  
*The offset from the strafe distance tracking sensor to the tracking center This value is front-justified, I.E. the offset to the front of center.*

## 5.99.1 Detailed Description

Builder class for the inertial odometry class.

### Author

Nathan Sandvig

Definition at line 43 of file [InertialOdometryBuilder.hpp](#).

## 5.99.2 Member Function Documentation

### 5.99.2.1 withClock()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
Clock (
    std::unique_ptr< wisco::rtos::IClock > & clock )
```

Adds a system clock to the builder.

#### Parameters

<i>clock</i>	The system clock
--------------	------------------

#### Returns

`InertialOdometryBuilder*` This builder for build chaining

Definition at line 11 of file [InertialOdometryBuilder.cpp](#).

```
00012 {
00013     m_clock = std::move(clock);
00014     return this;
00015 }
```

### 5.99.2.2 withDelayer()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
Delayer (
    std::unique_ptr< wisco::rtos::IDelayer > & delayer )
```

Adds a system delayer to the builder.

#### Parameters

<i>delayer</i>	The system delayer
----------------	--------------------

#### Returns

`InertialOdometryBuilder*` This builder for build chaining

Definition at line 17 of file [InertialOdometryBuilder.cpp](#).

```
00018 {
00019     m_delayer = std::move(delayer);
00020     return this;
00021 }
```

### 5.99.2.3 withMutex()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
Mutex (
    std::unique_ptr< wisco::rtos::IMutex > & mutex )
```

Adds a system mutex to the builder.

#### Parameters

<i>mutex</i>	The system mutex
--------------	------------------

#### Returns

[InertialOdometryBuilder\\*](#) This builder for build chaining

Definition at line 23 of file [InertialOdometryBuilder.cpp](#).

```
00024 {
00025     m_mutex = std::move(mutex);
00026     return this;
00027 }
```

### 5.99.2.4 withTask()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
Task (
    std::unique_ptr< wisco::rtos::ITask > & task )
```

Adds a system task to the builder.

#### Parameters

<i>task</i>	The system task
-------------	-----------------

#### Returns

[InertialOdometryBuilder\\*](#) This builder for build chaining

Definition at line 29 of file [InertialOdometryBuilder.cpp](#).

```
00030 {
00031     m_task = std::move(task);
00032     return this;
00033 }
```

### 5.99.2.5 withHeadingSensor()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
HeadingSensor (
    std::unique_ptr< wisco::io::IHeadingSensor > & heading_sensor )
```

Adds a heading sensor to the builder.

#### Parameters

<i>heading_sensor</i>	The heading sensor
-----------------------	--------------------

#### Returns

InertialOdometryBuilder\* This builder for build chaining

Definition at line 35 of file [InertialOdometryBuilder.cpp](#).

```
00036 {
00037     m_heading_sensor = std::move(heading_sensor);
00038     return this;
00039 }
```

### 5.99.2.6 withLinearDistanceTrackingSensor()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
LinearDistanceTrackingSensor (
    std::unique_ptr< wisco::io::IDistanceTrackingSensor > & linear_distance_tracking←
_sensor )
```

Adds a linear distance tracking sensor to the builder.

#### Parameters

<i>linear_distance_tracking_sensor</i>	The linear distance tracking sensor
--	-------------------------------------

#### Returns

InertialOdometryBuilder\* This builder for build chaining

Definition at line 41 of file [InertialOdometryBuilder.cpp](#).

```
00042 {
00043     m_linear_distance_tracking_sensor = std::move(linear_distance_tracking_sensor);
00044     return this;
00045 }
```

### 5.99.2.7 withLinearDistanceTrackingOffset()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
LinearDistanceTrackingOffset (
    double linear_distance_tracking_offset )
```

Adds a linear distance tracking offset to the builder.

#### Parameters

<i>linear_distance_tracking_offset</i>	The linear distance tracking offset
--	-------------------------------------

**Returns**

InertialOdometryBuilder\* This builder for build chaining

Definition at line 47 of file [InertialOdometryBuilder.cpp](#).

```
00048 {
00049     m_linear_distance_tracking_offset = linear_distance_tracking_offset;
00050     return this;
00051 }
```

**5.99.2.8 withStrafeDistanceTrackingSensor()**

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
StrafeDistanceTrackingSensor (
    std::unique_ptr< wisco::io::IDistanceTrackingSensor > & strafe_distance_tracking←
_sensor )
```

Adds a strafe distance tracking sensor to the builder.

**Parameters**

<i>strafe_distance_tracking_sensor</i>	The strafe distance tracking sensor
--	-------------------------------------

**Returns**

InertialOdometryBuilder\* This builder for build chaining

Definition at line 53 of file [InertialOdometryBuilder.cpp](#).

```
00054 {
00055     m_strafe_distance_tracking_sensor = std::move(strafe_distance_tracking_sensor);
00056     return this;
00057 }
```

**5.99.2.9 withStrafeDistanceTrackingOffset()**

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with←
StrafeDistanceTrackingOffset (
    double strafe_distance_tracking_offset )
```

Adds a strafe distance tracking offset to the builder.

**Parameters**

<i>strafe_distance_tracking_offset</i>	The strafe distance tracking offset
--	-------------------------------------

**Returns**

InertialOdometryBuilder\* This builder for build chaining

Definition at line 59 of file [InertialOdometryBuilder.cpp](#).

```
00060 {
00061     m_strafe_distance_tracking_offset = strafe_distance_tracking_offset;
00062     return this;
00063 }
```

### 5.99.2.10 build()

```
std::unique_ptr< IPositionTracker > wisco::robot::subsystems::position::InertialOdometry<-
Builder::build ( )
```

Builds the inertial odometry system.

#### Returns

`std::unique_ptr<IPositionTracker>` The inertial odometry system as a position tracking interface

Definition at line 65 of file [InertialOdometryBuilder.cpp](#).

```
00066 {
00067     std::unique_ptr<InertialOdometry> inertial_odometry{std::make_unique<InertialOdometry>()};
00068     inertial_odometry->setClock(m_clock);
00069     inertial_odometry->setDelayer(m_delayer);
00070     inertial_odometry->setMutex(m_mutex);
00071     inertial_odometry->setTask(m_task);
00072     inertial_odometry->setHeadingSensor(m_heading_sensor);
00073     inertial_odometry->setLinearDistanceTrackingSensor(m_linear_distance_tracking_sensor);
00074     inertial_odometry->setLinearDistanceTrackingOffset(m_linear_distance_tracking_offset);
00075     inertial_odometry->setStrafeDistanceTrackingSensor(m_strafe_distance_tracking_sensor);
00076     inertial_odometry->setStrafeDistanceTrackingOffset(m_strafe_distance_tracking_offset);
00077     return inertial_odometry;
00078 }
```

## 5.99.3 Member Data Documentation

### 5.99.3.1 m\_clock

```
std::unique_ptr<rtos::IClock> wisco::robot::subsystems::position::InertialOdometryBuilder::m<-
_clock {} [private]
```

The system clock.

Definition at line 50 of file [InertialOdometryBuilder.hpp](#).

```
00050 {};
```

### 5.99.3.2 m\_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::position::InertialOdometryBuilder<-
::m_delayer {} [private]
```

The system delayer.

Definition at line 56 of file [InertialOdometryBuilder.hpp](#).

```
00056 {};
```

### 5.99.3.3 m\_mutex

```
std::unique_ptr<rtos::IMutex> wisco::robot::subsystems::position::InertialOdometryBuilder::m<-
_mutex {} [private]
```

The os mutex.

Definition at line 62 of file [InertialOdometryBuilder.hpp](#).

```
00062 {};
```

### 5.99.3.4 m\_task

```
std::unique_ptr<rtos::ITask> wisco::robot::subsystems::position::InertialOdometryBuilder::m_task {} [private]
```

The task handler.

Definition at line 68 of file [InertialOdometryBuilder.hpp](#).

```
00068 {};
```

### 5.99.3.5 m\_heading\_sensor

```
std::unique_ptr<io::IHeadingSensor> wisco::robot::subsystems::position::InertialOdometryBuilder::m_heading_sensor {} [private]
```

The sensor to track the robot's heading.

Definition at line 74 of file [InertialOdometryBuilder.hpp](#).

```
00074 {};
```

### 5.99.3.6 m\_linear\_distance\_tracking\_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::InertialOdometryBuilder::m_linear_distance_tracking_sensor {} [private]
```

The sensor to track the robot's linear movement.

Definition at line 80 of file [InertialOdometryBuilder.hpp](#).

```
00080 {};
```

### 5.99.3.7 m\_linear\_distance\_tracking\_offset

```
double wisco::robot::subsystems::position::InertialOdometryBuilder::m_linear_distance_tracking_offset {} [private]
```

The offset from the linear distance tracking sensor to the tracking center This value is left-justified, I.E. the offset to the left of center.

Definition at line 87 of file [InertialOdometryBuilder.hpp](#).

```
00087 {};
```

### 5.99.3.8 m\_strafe\_distance\_tracking\_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::InertialOdometryBuilder::m_strafe_distance_tracking_sensor {} [private]
```

The sensor to track the robot's strafe movement.

Definition at line 93 of file [InertialOdometryBuilder.hpp](#).

```
00093 {};
```

### 5.99.3.9 m\_strafe\_distance\_tracking\_offset

```
double wisco::robot::subsystems::position::InertialOdometryBuilder::m_strafe_distance_tracking_offset {} [private]
```

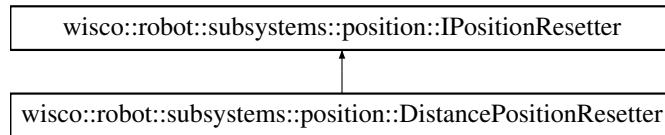
The offset from the strafe distance tracking sensor to the tracking center This value is front-justified, I.E. the offset to the front of center.

Definition at line 100 of file [InertialOdometryBuilder.hpp](#).  
00100 {};

## 5.100 wisco::robot::subsystems::position::IPositionResetter Class Reference

Interface for position resetting subsystems.

Inheritance diagram for wisco::robot::subsystems::position::IPositionResetter:



### Public Member Functions

- virtual ~**IPositionResetter** ()=default  
*Destroy the [IPositionResetter](#) object.*
- virtual void **initialize** ()=0  
*Initializes the position resetting system.*
- virtual void **run** ()=0  
*Runs the position tracking system.*
- virtual double **getResetX** (double theta)=0  
*Get the reset position for X.*
- virtual double **getResetY** (double theta)=0  
*Get the reset position for Y.*

### 5.100.1 Detailed Description

Interface for position resetting subsystems.

#### Author

Nathan Sandvig

Definition at line 41 of file [IPositionResetter.hpp](#).

## 5.100.2 Member Function Documentation

### 5.100.2.1 initialize()

```
virtual void wisco::robot::subsystems::position::IPositionResetter::initialize () [pure virtual]
```

Initializes the position resetting system.

Implemented in [wisco::robot::subsystems::position::DistancePositionResetter](#).

### 5.100.2.2 run()

```
virtual void wisco::robot::subsystems::position::IPositionResetter::run () [pure virtual]
```

Runs the position tracking system.

Implemented in [wisco::robot::subsystems::position::DistancePositionResetter](#).

### 5.100.2.3 getResetX()

```
virtual double wisco::robot::subsystems::position::IPositionResetter::getResetX (
    double theta ) [pure virtual]
```

Get the reset position for X.

#### Parameters

<i>theta</i>	The current robot angle
--------------	-------------------------

#### Returns

double The X reset position

Implemented in [wisco::robot::subsystems::position::DistancePositionResetter](#).

### 5.100.2.4 getResetY()

```
virtual double wisco::robot::subsystems::position::IPositionResetter::getResetY (
    double theta ) [pure virtual]
```

Get the reset position for Y.

#### Parameters

<i>theta</i>	The current robot angle
--------------	-------------------------

**Returns**

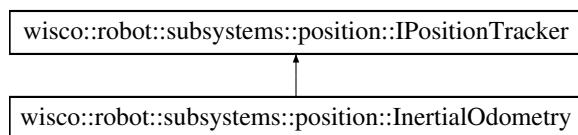
```
double The Y reset position
```

Implemented in [wisco::robot::subsystems::position::DistancePositionResetter](#).

## 5.101 wisco::robot::subsystems::position::IPositionTracker Class Reference

Interface for position tracking subsystems.

Inheritance diagram for wisco::robot::subsystems::position::IPositionTracker:



### Public Member Functions

- virtual ~**IPositionTracker** ()=default  
*Destroy the [IPositionTracker](#) object.*
- virtual void **initialize** ()=0  
*Initializes the position tracking system.*
- virtual void **run** ()=0  
*Runs the position tracking system.*
- virtual void **setPosition** ([Position](#) position)=0  
*Set the position of the position tracking system.*
- virtual [Position](#) **getPosition** ()=0  
*Get the position of the system.*

### 5.101.1 Detailed Description

Interface for position tracking subsystems.

#### Author

Nathan Sandvig

Definition at line 43 of file [IPositionTracker.hpp](#).

### 5.101.2 Member Function Documentation

#### 5.101.2.1 initialize()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::initialize ( ) [pure virtual]
```

Initializes the position tracking system.

Implemented in [wisco::robot::subsystems::position::InertialOdometry](#).

### 5.101.2.2 run()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::run () [pure virtual]
```

Runs the position tracking system.

Implemented in [wisco::robot::subsystems::position::InertialOdometry](#).

### 5.101.2.3 setPosition()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::setPosition (
    Position position ) [pure virtual]
```

Set the position of the position tracking system.

#### Parameters

<code>position</code>	The new position
-----------------------	------------------

Implemented in [wisco::robot::subsystems::position::InertialOdometry](#).

### 5.101.2.4 getPosition()

```
virtual Position wisco::robot::subsystems::position::IPositionTracker::getPosition () [pure
virtual]
```

Get the position of the system.

#### Returns

`Position` The position of the system

Implemented in [wisco::robot::subsystems::position::InertialOdometry](#).

## 5.102 **wisco::robot::subsystems::position::Position** Struct Reference

Holds a robot position.

### Public Attributes

- double `x` {}  
*The X-coordinate.*
- double `y` {}  
*The Y-coordinate.*
- double `theta` {}  
*The angle.*
- double `xV` {}  
*The X-velocity.*
- double `yV` {}  
*The Y-velocity.*
- double `thetaV` {}  
*The angular velocity.*

### 5.102.1 Detailed Description

Holds a robot position.

#### Author

Nathan Sandvig

Definition at line 41 of file [Position.hpp](#).

### 5.102.2 Member Data Documentation

#### 5.102.2.1 x

```
double wisco::robot::subsystems::position::Position::x {}
```

The X-coordinate.

Definition at line 47 of file [Position.hpp](#).  
00047 {};

#### 5.102.2.2 y

```
double wisco::robot::subsystems::position::Position::y {}
```

The Y-coordinate.

Definition at line 53 of file [Position.hpp](#).  
00053 {};

#### 5.102.2.3 theta

```
double wisco::robot::subsystems::position::Position::theta {}
```

The angle.

Definition at line 59 of file [Position.hpp](#).  
00059 {};

#### 5.102.2.4 xv

```
double wisco::robot::subsystems::position::Position::xv {}
```

The X-velocity.

Definition at line 65 of file [Position.hpp](#).  
00065 {};

### 5.102.2.5 yV

```
double wisco::robot::subsystems::position::Position::yV {}
```

The Y-velocity.

Definition at line 71 of file [Position.hpp](#).

```
00071 {};
```

### 5.102.2.6 thetaV

```
double wisco::robot::subsystems::position::Position::thetaV {}
```

The angular velocity.

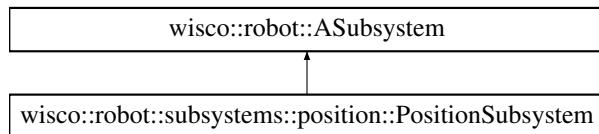
Definition at line 77 of file [Position.hpp](#).

```
00077 {};
```

## 5.103 wisco::robot::subsystems::position::PositionSubsystem Class Reference

Adapter from a position tracker to a robot subsystem.

Inheritance diagram for wisco::robot::subsystems::position::PositionSubsystem:



### Public Member Functions

- **PositionSubsystem** (std::unique\_ptr<[IPositionTracker](#)> &position\_tracker, std::unique\_ptr<[IPositionResetter](#)> &position\_resetter)  
*Construct a new Position Subsystem object.*
- void **initialize** () override  
*Initializes the subsystem.*
- void **run** () override  
*Runs the subsystems.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the subsystem.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the subsystem.*

## Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem ()=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (const ASubsystem &other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (ASubsystem &&other)=default**  
*Construct a new ASubsystem object.*
- **ASubsystem (std::string name)**  
*Construct a new ASubsystem object.*
- **virtual ~ASubsystem ()=default**  
*Destroy the ASubsystem object.*
- **const std::string & getName () const**  
*Get the name of the subsystem.*
- **ASubsystem & operator= (const ASubsystem &rhs)=default**  
*Copy assignment operator for ASubsystem.*
- **ASubsystem & operator= (ASubsystem &&rhs)=default**  
*Move assignment operator for ASubsystem.*

## Private Attributes

- **std::unique\_ptr<IPositionTracker> m\_position\_tracker {}**  
*The position tracker being adapted.*
- **std::unique\_ptr<IPositionResetter> m\_position\_resetter {}**  
*The position resetter being adapted.*

## Static Private Attributes

- **static constexpr char SUBSYSTEM\_NAME [] {"POSITION TRACKER"}**  
*The name of the subsystem.*
- **static constexpr char SET\_POSITION\_COMMAND\_NAME [] {"SET POSITION"}**  
*The name of the set position command.*
- **static constexpr char RESET\_X\_COMMAND\_NAME [] {"RESET X"}**  
*The name of the reset x command.*
- **static constexpr char RESET\_Y\_COMMAND\_NAME [] {"RESET Y"}**  
*The name of the reset y command.*
- **static constexpr char GET\_POSITION\_STATE\_NAME [] {"GET POSITION"}**  
*The name of the get position command.*

### 5.103.1 Detailed Description

Adapter from a position tracker to a robot subsystem.

#### Author

Nathan Sandvig

Definition at line 47 of file [PositionSubsystem.hpp](#).

## 5.103.2 Constructor & Destructor Documentation

### 5.103.2.1 PositionSubsystem()

```
wisco::robot::subsystems::position::PositionSubsystem::PositionSubsystem (
    std::unique_ptr< IPositionTracker > & position_tracker,
    std::unique_ptr< IPositionResetter > & position_resetter )
```

Construct a new [Position Subsystem](#) object.

#### Parameters

<i>position_tracker</i>	The position tracker being adapted
<i>position_resetter</i>	The position resetter being adapted

Definition at line 12 of file [PositionSubsystem.cpp](#).

```
00014     : ASubsystem{SUBSYSTEM_NAME},
00015     m_position_tracker{std::move(position_tracker)},
00016     m_position_resetter{std::move(position_resetter)}
00017 {
00018
00019 }
```

## 5.103.3 Member Function Documentation

### 5.103.3.1 initialize()

```
void wisco::robot::subsystems::position::PositionSubsystem::initialize () [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 21 of file [PositionSubsystem.cpp](#).

```
00022 {
00023     if (m_position_tracker)
00024         m_position_tracker->initialize();
00025     if (m_position_resetter)
00026         m_position_resetter->initialize();
00027 }
```

### 5.103.3.2 run()

```
void wisco::robot::subsystems::position::PositionSubsystem::run () [override], [virtual]
```

Runs the subsystems.

Implements [wisco::robot::ASubsystem](#).

Definition at line 29 of file [PositionSubsystem.cpp](#).

```
00030 {
00031     if (m_position_tracker)
00032         m_position_tracker->run();
00033     if (m_position_resetter)
00034         m_position_resetter->run();
00035 }
```

### 5.103.3.3 command()

```
void wisco::robot::subsystems::position::PositionSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 37 of file [PositionSubsystem.cpp](#).

```
00038 {
00039     if (command_name == SET_POSITION_COMMAND_NAME)
00040     {
00041         Position position{va_arg(args, double), va_arg(args, double), va_arg(args, double)};
00042         m_position_tracker->setPosition(position);
00043     }
00044     else if (command_name == RESET_X_COMMAND_NAME)
00045     {
00046         if (m_position_tracker && m_position_resetter)
00047         {
00048             Position position{m_position_tracker->getPosition()};
00049             double reset_x{m_position_resetter->getResetX(position.theta)};
00050             position.x = reset_x;
00051             m_position_tracker->setPosition(position);
00052         }
00053     }
00054     else if (command_name == RESET_Y_COMMAND_NAME)
00055     {
00056         if (m_position_tracker && m_position_resetter)
00057         {
00058             Position position{m_position_tracker->getPosition()};
00059             double reset_y{m_position_resetter->getResetY(position.theta)};
00060             position.y = reset_y;
00061             m_position_tracker->setPosition(position);
00062         }
00063     }
00064 }
```

**5.103.3.4 state()**

```
void * wisco::robot::subsystems::position::PositionSubsystem::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the subsystem.

**Parameters**

<i>state_name</i>	The name of the state to get
-------------------	------------------------------

**Returns**

*void\** The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 66 of file [PositionSubsystem.cpp](#).

```
00067 {
00068     void* result{};
00069
00070     if (state_name == GET_POSITION_STATE_NAME)
00071     {
00072         Position* position{new Position{m_position_tracker->getPosition()}};
00073         result = position;
00074     }
00075
00076     return result;
00077 }
```

## 5.103.4 Member Data Documentation

### 5.103.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::SUBSYSTEM_NAME[ ] {"POSITION  
TRACKER"} [static], [constexpr], [private]
```

The name of the subsystem.

Definition at line [54](#) of file [PositionSubsystem.hpp](#).  
00054 {"POSITION TRACKER"};

### 5.103.4.2 SET\_POSITION\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::SET_POSITION_COMMAND_NAME[ ] {"SET POSITION"} [static], [constexpr], [private]
```

The name of the set position command.

Definition at line [60](#) of file [PositionSubsystem.hpp](#).  
00060 {"SET POSITION"};

### 5.103.4.3 RESET\_X\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::RESET_X_COMMAND_NAME[ ] {"RESET X"} [static], [constexpr], [private]
```

The name of the reset x command.

Definition at line [66](#) of file [PositionSubsystem.hpp](#).  
00066 {"RESET X"};

### 5.103.4.4 RESET\_Y\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::RESET_Y_COMMAND_NAME[ ] {"RESET Y"} [static], [constexpr], [private]
```

The name of the reset y command.

Definition at line [72](#) of file [PositionSubsystem.hpp](#).  
00072 {"RESET Y"};

### 5.103.4.5 GET\_POSITION\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::GET_POSITION_STATE_NAME[ ] {"GET POSITION"} [static], [constexpr], [private]
```

The name of the get position command.

Definition at line [78](#) of file [PositionSubsystem.hpp](#).  
00078 {"GET POSITION"};

### 5.103.4.6 m\_position\_tracker

```
std::unique_ptr<IPositionTracker> wisco::robot::subsystems::position::PositionSubsystem::m_<->
position_tracker {} [private]
```

The position tracker being adapted.

Definition at line 84 of file [PositionSubsystem.hpp](#).

```
00084 {};
```

### 5.103.4.7 m\_position\_resetter

```
std::unique_ptr<IPositionResetter> wisco::robot::subsystems::position::PositionSubsystem::m_<->
position_resetter {} [private]
```

The position resetter being adapted.

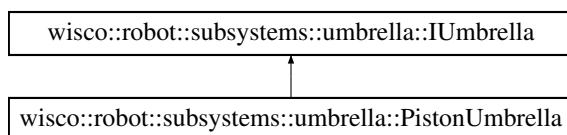
Definition at line 90 of file [PositionSubsystem.hpp](#).

```
00090 {};
```

## 5.104 wisco::robot::subsystems::umbrella::IUmbrella Class Reference

Interface for umbrellas with an out and in position.

Inheritance diagram for wisco::robot::subsystems::umbrella::IUmbrella:



### Public Member Functions

- virtual ~**IUmbrella** ()=default  
*Destroy the **IUmbrella** object.*
- virtual void **initialize** ()=0  
*Initializes the umbrella.*
- virtual void **run** ()=0  
*Runs the umbrella.*
- virtual void **setIn** ()=0  
*Sets the umbrella to the in position.*
- virtual void **setOut** ()=0  
*Sets the umbrella to the out position.*
- virtual bool **isIn** ()=0  
*Checks if the umbrella is in.*
- virtual bool **isOut** ()=0  
*Checks if the umbrella is out.*

## 5.104.1 Detailed Description

Interface for umbrellas with an out and in position.

### Author

Nathan Sandvig

Definition at line 41 of file [IUmbrella.hpp](#).

## 5.104.2 Member Function Documentation

### 5.104.2.1 initialize()

```
virtual void wisco::robot::subsystems::umbrella::IUmbrella::initialize () [pure virtual]
```

Initializes the umbrella.

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

### 5.104.2.2 run()

```
virtual void wisco::robot::subsystems::umbrella::IUmbrella::run () [pure virtual]
```

Runs the umbrella.

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

### 5.104.2.3 setIn()

```
virtual void wisco::robot::subsystems::umbrella::IUmbrella::setIn () [pure virtual]
```

Sets the umbrella to the in position.

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

### 5.104.2.4 setOut()

```
virtual void wisco::robot::subsystems::umbrella::IUmbrella::setOut () [pure virtual]
```

Sets the umbrella to the out position.

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

### 5.104.2.5 `isIn()`

```
virtual bool wisco::robot::subsystems::umbrella::IUmbrella::isIn () [pure virtual]
```

Checks if the umbrella is in.

#### Returns

- true The umbrella is in
- false The umbrella is not in

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

### 5.104.2.6 `isOut()`

```
virtual bool wisco::robot::subsystems::umbrella::IUmbrella::isOut () [pure virtual]
```

Checks if the umbrella is out.

#### Returns

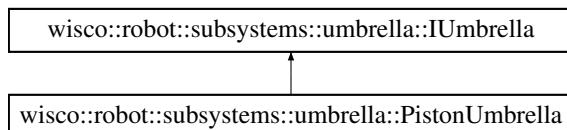
- true The umbrella is out
- false The umbrella is not out

Implemented in [wisco::robot::subsystems::umbrella::PistonUmbrella](#).

## 5.105 `wisco::robot::subsystems::umbrella::PistonUmbrella` Class Reference

A umbrella controlled using a piston.

Inheritance diagram for `wisco::robot::subsystems::umbrella::PistonUmbrella`:



### Public Member Functions

- void `initialize` () override  
*Initializes the umbrella.*
- void `run` () override  
*Runs the umbrella.*
- void `setIn` () override  
*Closes the umbrella.*
- void `setOut` () override  
*Opens the umbrella.*
- bool `isIn` () override  
*Checks if the umbrella is in.*
- bool `isOut` () override  
*Checks if the umbrella is out.*
- void `setPistons` (`hal::PistonGroup` &pistons)  
*Sets the pistons for the umbrella.*
- void `setOutState` (bool out\_state)  
*Sets the state of the pistons when the umbrella is out.*

## Public Member Functions inherited from [wisco::robot::subsystems::umbrella::IUmbrella](#)

- virtual ~[IUmbrella](#) ()=default

*Destroy the [IUmbrella](#) object.*

## Private Attributes

- [hal::PistonGroup m\\_pistons {}](#)  
*The pistons for the umbrella.*
- [bool m\\_out\\_state {}](#)  
*The state of the piston group when the umbrella is out.*

## 5.105.1 Detailed Description

A umbrella controlled using a piston.

### Author

Nathan Sandvig

Definition at line 45 of file [PistonUmbrella.hpp](#).

## 5.105.2 Member Function Documentation

### 5.105.2.1 initialize()

```
void wisco::robot::subsystems::umbrella::PistonUmbrella::initialize ( ) [override], [virtual]
```

Initializes the umbrella.

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 11 of file [PistonUmbrella.cpp](#).

```
00012 {  
00013     // No initialization code  
00014 }
```

### 5.105.2.2 run()

```
void wisco::robot::subsystems::umbrella::PistonUmbrella::run ( ) [override], [virtual]
```

Runs the umbrella.

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 16 of file [PistonUmbrella.cpp](#).

```
00017 {  
00018     // No running code  
00019 }
```

### 5.105.2.3 setIn()

void wisco::robot::subsystems::umbrella::PistonUmbrella::setIn ( ) [override], [virtual]

Closes the umbrella.

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 21 of file [PistonUmbrella.cpp](#).

```
00022 {
00023     m_pistons.setState(!m_out_state);
00024 }
```

### 5.105.2.4 setOut()

void wisco::robot::subsystems::umbrella::PistonUmbrella::setOut ( ) [override], [virtual]

Opens the umbrella.

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 26 of file [PistonUmbrella.cpp](#).

```
00027 {
00028     m_pistons.setState(m_out_state);
00029 }
```

### 5.105.2.5 isIn()

bool wisco::robot::subsystems::umbrella::PistonUmbrella::isIn ( ) [override], [virtual]

Checks if the umbrella is in.

Returns

true The umbrella is in

false The umbrella is not in

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 31 of file [PistonUmbrella.cpp](#).

```
00032 {
00033     return m_pistons.getState() != m_out_state;
00034 }
```

### 5.105.2.6 isOut()

bool wisco::robot::subsystems::umbrella::PistonUmbrella::isOut ( ) [override], [virtual]

Checks if the umbrella is out.

Returns

true The umbrella is out

false The umbrella is not out

Implements [wisco::robot::subsystems::umbrella::IUmbrella](#).

Definition at line 36 of file [PistonUmbrella.cpp](#).

```
00037 {
00038     return m_pistons.getState() == m_out_state;
00039 }
```

### 5.105.2.7 setPistons()

```
void wisco::robot::subsystems::umbrella::PistonUmbrella::setPistons (
    hal::PistonGroup & pistons )
```

Sets the pistons for the umbrella.

**Parameters**

<i>pistons</i>	The pistons for the umbrella
----------------	------------------------------

Definition at line 41 of file [PistonUmbrella.cpp](#).

```
00042 {  
00043     m_pistons = pistons;  
00044 }
```

### 5.105.2.8 setOutState()

```
void wisco::robot::subsystems::umbrella::PistonUmbrella::setOutState (  
    bool out_state )
```

Sets the state of the pistons when the umbrella is out.

**Parameters**

<i>out_state</i>	The state of the pistons when the umbrella is out
------------------	---

Definition at line 46 of file [PistonUmbrella.cpp](#).

```
00047 {  
00048     m_out_state = out_state;  
00049 }
```

## 5.105.3 Member Data Documentation

### 5.105.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::umbrella::PistonUmbrella::m_pistons {} [private]
```

The pistons for the umbrella.

Definition at line 52 of file [PistonUmbrella.hpp](#).

```
00052 {};
```

### 5.105.3.2 m\_out\_state

```
bool wisco::robot::subsystems::umbrella::PistonUmbrella::m_out_state {} [private]
```

The state of the piston group when the umbrella is out.

Definition at line 58 of file [PistonUmbrella.hpp](#).

```
00058 {};
```

## 5.106 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder Class Reference

Builder class for piston umbrellas.

## Public Member Functions

- `PistonUmbrellaBuilder * withPiston (std::unique_ptr< io::IPiston > &piston)`  
*Adds a piston to the build.*
- `PistonUmbrellaBuilder * withOutState (bool out_state)`  
*Adds an out state to the build.*
- `std::unique_ptr< IUmbrella > build ()`  
*Builds the piston umbrella.*

## Private Attributes

- `hal::PistonGroup m_pistons {}`  
*The pistons for the umbrella.*
- `bool m_out_state {}`  
*The state of the piston group when the umbrella is out.*

### 5.106.1 Detailed Description

Builder class for piston umbrellas.

#### Author

Nathan Sandvig

Definition at line 43 of file [PistonUmbrellaBuilder.hpp](#).

### 5.106.2 Member Function Documentation

#### 5.106.2.1 withPiston()

```
PistonUmbrellaBuilder * wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder::withPiston
(
    std::unique_ptr< io::IPiston > & piston )
```

Adds a piston to the build.

#### Parameters

<code>piston</code>	The piston
---------------------	------------

#### Returns

This object for build chaining

Definition at line 11 of file [PistonUmbrellaBuilder.cpp](#).

```
00012 {
00013     m_pistons.addPiston(piston);
00014     return this;
00015 }
```

### 5.106.2.2 withOutState()

```
PistonUmbrellaBuilder * wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder::withOutState (
    bool out_state )
```

Adds an out state to the build.

#### Parameters

<i>out_state</i>	The state of the pistons when the umbrella is out
------------------	---

#### Returns

This object for build chaining

Definition at line 17 of file [PistonUmbrellaBuilder.cpp](#).

```
00018 {
00019     m_out_state = out_state;
00020     return this;
00021 }
```

### 5.106.2.3 build()

```
std::unique_ptr< IUmbrella > wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder::build()
```

Builds the piston umbrella.

#### Returns

`std::unique_ptr<IUmbrella>` The piston umbrella as an umbrella interface

Definition at line 23 of file [PistonUmbrellaBuilder.cpp](#).

```
00024 {
00025     std::unique_ptr<PistonUmbrella> piston_umbrella{std::make_unique<PistonUmbrella>()};
00026     piston_umbrella->setPistons(m_pistons);
00027     piston_umbrella->setOutState(m_out_state);
00028     return piston_umbrella;
00029 }
```

## 5.106.3 Member Data Documentation

### 5.106.3.1 m\_pistons

```
hal::PistonGroup wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder::m_pistons {} [private]
```

The pistons for the umbrella.

Definition at line 50 of file [PistonUmbrellaBuilder.hpp](#).

```
00050 {};
```

### 5.106.3.2 `m_out_state`

```
bool wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder::m_out_state {} [private]
```

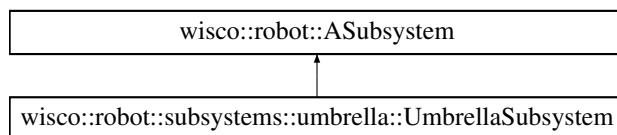
The state of the piston group when the umbrella is out.

Definition at line 56 of file [PistonUmbrellaBuilder.hpp](#).  
00056 {};

## 5.107 `wisco::robot::subsystems::umbrella::UmbrellaSubsystem` Class Reference

The subsystem adapter for umbrellas.

Inheritance diagram for `wisco::robot::subsystems::umbrella::UmbrellaSubsystem`:



### Public Member Functions

- [`UmbrellaSubsystem` \(std::unique\\_ptr< `IUmbrella` > &umbrella\)](#)  
*Construct a new UMBRELLA Subsystem object.*
- `void initialize ()` override  
*Initializes the subsystem.*
- `void run ()` override  
*Runs the subsystem.*
- `void command (std::string command_name, va_list &args)` override  
*Runs a command for the subsystem.*
- `void * state (std::string state_name)` override  
*Gets a state of the subsystem.*

### Public Member Functions inherited from `wisco::robot::ASubsystem`

- `ASubsystem ()`=default  
*Construct a new ASubsystem object.*
- `ASubsystem (const ASubsystem &other)`=default  
*Construct a new ASubsystem object.*
- `ASubsystem (ASubsystem &&other)`=default  
*Construct a new ASubsystem object.*
- `ASubsystem (std::string name)`  
*Construct a new ASubsystem object.*
- `virtual ~ASubsystem ()`=default  
*Destroy the ASubsystem object.*
- `const std::string & getName ()` const  
*Get the name of the subsystem.*
- `ASubsystem & operator= (const ASubsystem &rhs)`=default  
*Copy assignment operator for ASubsystem.*
- `ASubsystem & operator= (ASubsystem &&rhs)`=default  
*Move assignment operator for ASubsystem.*

**Private Attributes**

- std::unique\_ptr< [IUmbrella](#) > [m\\_umbrella](#) {}  
*The umbrella being adapted.*

**Static Private Attributes**

- static constexpr char [SUBSYSTEM\\_NAME](#) [] {"UMBRELLA"}  
*The name of the subsystem.*
- static constexpr char [SET\\_IN\\_COMMAND\\_NAME](#) [] {"SET IN"}  
*The name of the set in command.*
- static constexpr char [SET\\_OUT\\_COMMAND\\_NAME](#) [] {"SET OUT"}  
*The name of the set out command.*
- static constexpr char [IS\\_IN\\_STATE\\_NAME](#) [] {"IS IN"}  
*The name of the is in command.*
- static constexpr char [IS\\_OUT\\_STATE\\_NAME](#) [] {"IS OUT"}  
*The name of the is out command.*

**5.107.1 Detailed Description**

The subsystem adapter for umbrellas.

**Author**

Nathan Sandvig

Definition at line 46 of file [UmbrellaSubsystem.hpp](#).

**5.107.2 Constructor & Destructor Documentation****5.107.2.1 UmbrellaSubsystem()**

```
wisco::robot::subsystems::umbrella::UmbrellaSubsystem::UmbrellaSubsystem (
    std::unique_ptr< IUmbrella > & umbrella )
```

Construct a new UMBRELLA Subsystem object.

**Parameters**

<i>umbrella</i>	The umbrella being adapted
-----------------	----------------------------

Definition at line 11 of file [UmbrellaSubsystem.cpp](#).

```
00012     : ASubsystem{SUBSYSTEM\_NAME}, m\_umbrella{std::move(umbrella)}
00013 {
00014
00015 }
```

### 5.107.3 Member Function Documentation

#### 5.107.3.1 initialize()

```
void wisco::robot::subsystems::umbrella::UmbrellaSubsystem::initialize () [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [UmbrellaSubsystem.cpp](#).

```
00018 {
00019     if (m_umbrella)
00020         m_umbrella->initialize();
00021 }
```

#### 5.107.3.2 run()

```
void wisco::robot::subsystems::umbrella::UmbrellaSubsystem::run () [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 23 of file [UmbrellaSubsystem.cpp](#).

```
00024 {
00025     if (m_umbrella)
00026         m_umbrella->run();
00027 }
```

#### 5.107.3.3 command()

```
void wisco::robot::subsystems::umbrella::UmbrellaSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

##### Parameters

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 29 of file [UmbrellaSubsystem.cpp](#).

```
00030 {
00031     if (command_name == SET_IN_COMMAND_NAME)
00032     {
00033         m_umbrella->setIn();
00034     }
00035     else if (command_name == SET_OUT_COMMAND_NAME)
00036     {
00037         m_umbrella->setOut();
00038     }
00039 }
```

### 5.107.3.4 state()

```
void * wisco::robot::subsystems::umbrella::UmbrellaSubsystem::state (
    std::string state_name ) [override], [virtual]
```

Gets a state of the subsystem.

#### Parameters

<code>state_name</code>	The name of the state to get
-------------------------	------------------------------

#### Returns

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 41 of file [UmbrellaSubsystem.cpp](#).

```
00042 {
00043     void* result=nullptr;
00044
00045     if (state_name == IS_IN_STATE_NAME)
00046     {
00047         bool* loaded{new bool{m_umbrella->isIn()}};
00048         result = loaded;
00049     }
00050     else if (state_name == IS_OUT_STATE_NAME)
00051     {
00052         bool* ready{new bool{m_umbrella->isOut()}};
00053         result = ready;
00054     }
00055
00056     return result;
00057 }
```

## 5.107.4 Member Data Documentation

### 5.107.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::umbrella::UmbrellaSubsystem::SUBSYSTEM_NAME[ ] {"UMBRELLA"}  
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file [UmbrellaSubsystem.hpp](#).

```
00053 {"UMBRELLA"};
```

### 5.107.4.2 SET\_IN\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::umbrella::UmbrellaSubsystem::SET_IN_COMMAND_NAME[ ] {"SET IN"}  
[static], [constexpr], [private]
```

The name of the set in command.

Definition at line 59 of file [UmbrellaSubsystem.hpp](#).

```
00059 {"SET IN"};
```

### 5.107.4.3 SET\_OUT\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::umbrella::UmbrellaSubsystem::SET_OUT_COMMAND_NAME[ ] {"SET OUT"} [static], [constexpr], [private]
```

The name of the set out command.

Definition at line 65 of file [UmbrellaSubsystem.hpp](#).

```
00065 {"SET OUT"};
```

### 5.107.4.4 IS\_IN\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::umbrella::UmbrellaSubsystem::IS_IN_STATE_NAME[ ] {"IS IN"} [static], [constexpr], [private]
```

The name of the is in command.

Definition at line 71 of file [UmbrellaSubsystem.hpp](#).

```
00071 {"IS IN"};
```

### 5.107.4.5 IS\_OUT\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::umbrella::UmbrellaSubsystem::IS_OUT_STATE_NAME[ ] {"IS OUT"} [static], [constexpr], [private]
```

The name of the is out command.

Definition at line 77 of file [UmbrellaSubsystem.hpp](#).

```
00077 {"IS OUT"};
```

### 5.107.4.6 m\_umbrella

```
std::unique_ptr<IUmbrella> wisco::robot::subsystems::umbrella::UmbrellaSubsystem::m_umbrella {} [private]
```

The umbrella being adapted.

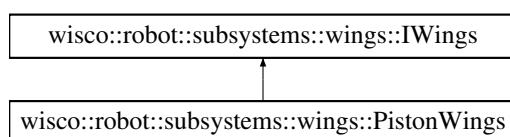
Definition at line 83 of file [UmbrellaSubsystem.hpp](#).

```
00083 {};
```

## 5.108 wisco::robot::subsystems::wings::IWings Class Reference

Interface for wings subsystems.

Inheritance diagram for wisco::robot::subsystems::wings::IWings:



## Public Member Functions

- virtual ~[IWings](#) ()=default  
*Destroy the [IWings](#) object.*
- virtual void [initialize](#) ()=0  
*Initializes the wings.*
- virtual void [run](#) ()=0  
*Runs the wings.*
- virtual void [setLeftWing](#) (bool out)=0  
*Sets the position of the left wing.*
- virtual void [setRightWing](#) (bool out)=0  
*Sets the position of the right wing.*
- virtual bool [getLeftWing](#) ()=0  
*Gets the position of the left wing.*
- virtual bool [getRightWing](#) ()=0  
*Gets the position of the right wing.*

### 5.108.1 Detailed Description

Interface for wings subsystems.

#### Author

Nathan Sandvig

Definition at line 41 of file [IWings.hpp](#).

### 5.108.2 Member Function Documentation

#### 5.108.2.1 [initialize\(\)](#)

```
virtual void wisco::robot::subsystems::wings::IWings::initialize ( ) [pure virtual]
```

Initializes the wings.

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

#### 5.108.2.2 [run\(\)](#)

```
virtual void wisco::robot::subsystems::wings::IWings::run ( ) [pure virtual]
```

Runs the wings.

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

#### 5.108.2.3 [setLeftWing\(\)](#)

```
virtual void wisco::robot::subsystems::wings::IWings::setLeftWing ( bool out ) [pure virtual]
```

Sets the position of the left wing.

**Parameters**

<i>out</i>	True for out, false for in
------------	----------------------------

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

**5.108.2.4 setRightWing()**

```
virtual void wisco::robot::subsystems::wings::IWings::setRightWing (   
    bool out ) [pure virtual]
```

Sets the position of the right wing.

**Parameters**

<i>out</i>	True for out, false for in
------------	----------------------------

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

**5.108.2.5 getLeftWing()**

```
virtual bool wisco::robot::subsystems::wings::IWings::getLeftWing ( ) [pure virtual]
```

Gets the position of the left wing.

**Returns**

true The wing is out

false The wing is in

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

**5.108.2.6 getRightWing()**

```
virtual bool wisco::robot::subsystems::wings::IWings::getRightWing ( ) [pure virtual]
```

Gets the position of the right wing.

**Returns**

true The wing is out

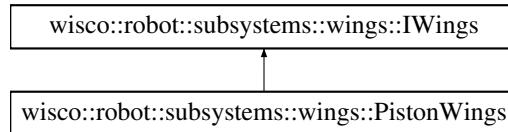
false The wing is in

Implemented in [wisco::robot::subsystems::wings::PistonWings](#).

## 5.109 **wisco::robot::subsystems::wings::PistonWings** Class Reference

Wings controlled using pistons.

Inheritance diagram for wisco::robot::subsystems::wings::PistonWings:



### Public Member Functions

- void **initialize** () override  
*Initializes the wings.*
- void **run** () override  
*Runs the wings.*
- void **setLeftWing** (bool out) override  
*Sets the position of the left wing.*
- void **setRightWing** (bool out) override  
*Sets the position of the right wing.*
- bool **getLeftWing** () override  
*Gets the position of the left wing.*
- bool **getRightWing** () override  
*Gets the position of the right wing.*
- void **setLeftPistons** (hal::PistonGroup &left\_pistons)  
*Sets the left wing pistons.*
- void **setRightPistons** (hal::PistonGroup &right\_pistons)  
*Sets the right wing pistons.*
- void **setOutState** (bool out\_state)  
*Sets the state of the pistons when the wings are out.*

### Public Member Functions inherited from **wisco::robot::subsystems::wings::IWings**

- virtual ~**IWings** ()=default  
*Destroy the **IWings** object.*

### Private Attributes

- hal::PistonGroup **m\_left\_pistons** {}  
*The pistons for the left wing.*
- hal::PistonGroup **m\_right\_pistons** {}  
*The pistons for the right wing.*
- bool **m\_out\_state** {}  
*The state of the pistons when the wings are out.*

### 5.109.1 Detailed Description

Wings controlled using pistons.

#### Author

Nathan Sandvig

Definition at line 45 of file [PistonWings.hpp](#).

### 5.109.2 Member Function Documentation

#### 5.109.2.1 initialize()

```
void wisco::robot::subsystems::wings::PistonWings::initialize () [override], [virtual]
```

Initializes the wings.

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 11 of file [PistonWings.cpp](#).

```
00012 {  
00013     // No initialization code  
00014 }
```

#### 5.109.2.2 run()

```
void wisco::robot::subsystems::wings::PistonWings::run () [override], [virtual]
```

Runs the wings.

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 16 of file [PistonWings.cpp](#).

```
00017 {  
00018     // No running code  
00019 }
```

#### 5.109.2.3 setLeftWing()

```
void wisco::robot::subsystems::wings::PistonWings::setLeftWing (  
    bool out ) [override], [virtual]
```

Sets the position of the left wing.

#### Parameters

<i>out</i>	True for out, false for in
------------	----------------------------

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 21 of file [PistonWings.cpp](#).

```
00022 {
00023     if (out)
00024         m_left_pistons.setState(m_out_state);
00025     else
00026         m_left_pistons.setState(!m_out_state);
00027 }
```

#### 5.109.2.4 setRightWing()

```
void wisco::robot::subsystems::wings::PistonWings::setRightWing (
    bool out) [override], [virtual]
```

Sets the position of the right wing.

##### Parameters

<i>out</i>	True for out, false for in
------------	----------------------------

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 29 of file [PistonWings.cpp](#).

```
00030 {
00031     if (out)
00032         m_right_pistons.setState(m_out_state);
00033     else
00034         m_right_pistons.setState(!m_out_state);
00035 }
```

#### 5.109.2.5 getLeftWing()

```
bool wisco::robot::subsystems::wings::PistonWings::getLeftWing () [override], [virtual]
```

Gets the position of the left wing.

##### Returns

true The wing is out  
false The wing is in

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 37 of file [PistonWings.cpp](#).

```
00038 {
00039     return m_left_pistons.getState() == m_out_state;
00040 }
```

#### 5.109.2.6 getRightWing()

```
bool wisco::robot::subsystems::wings::PistonWings::getRightWing () [override], [virtual]
```

Gets the position of the right wing.

##### Returns

true The wing is out  
false The wing is in

Implements [wisco::robot::subsystems::wings::IWings](#).

Definition at line 42 of file [PistonWings.cpp](#).

```
00043 {
00044     return m_right_pistons.getState() == m_out_state;
00045 }
```

### 5.109.2.7 setLeftPistons()

```
void wisco::robot::subsystems::wings::PistonWings::setLeftPistons (
    hal::PistonGroup & left_pistons )
```

Sets the left wing pistons.

#### Parameters

<i>left_pistons</i>	The left wing pistons
---------------------	-----------------------

Definition at line 47 of file [PistonWings.cpp](#).

```
00048 {
00049     m_left_pistons = left_pistons;
00050 }
```

### 5.109.2.8 setRightPistons()

```
void wisco::robot::subsystems::wings::PistonWings::setRightPistons (
    hal::PistonGroup & right_pistons )
```

Sets the right wing pistons.

#### Parameters

<i>right_pistons</i>	The right wing pistons
----------------------	------------------------

Definition at line 52 of file [PistonWings.cpp](#).

```
00053 {
00054     m_right_pistons = right_pistons;
00055 }
```

### 5.109.2.9 setOutState()

```
void wisco::robot::subsystems::wings::PistonWings::setOutState (
    bool out_state )
```

Sets the state of the pistons when the wings are out.

#### Parameters

<i>out_state</i>	The state of the pistons when the wings are out
------------------	---

Definition at line 57 of file [PistonWings.cpp](#).

```
00058 {
00059     m_out_state = out_state;
00060 }
```

### 5.109.3 Member Data Documentation

#### 5.109.3.1 m\_left\_pistons

```
hal::PistonGroup wisco::robot::subsystems::wings::PistonWings::m_left_pistons {} [private]
```

The pistons for the left wing.

Definition at line 52 of file [PistonWings.hpp](#).  
00052 {};

#### 5.109.3.2 m\_right\_pistons

```
hal::PistonGroup wisco::robot::subsystems::wings::PistonWings::m_right_pistons {} [private]
```

The pistons for the right wing.

Definition at line 58 of file [PistonWings.hpp](#).  
00058 {};

#### 5.109.3.3 m\_out\_state

```
bool wisco::robot::subsystems::wings::PistonWings::m_out_state {} [private]
```

The state of the pistons when the wings are out.

Definition at line 64 of file [PistonWings.hpp](#).  
00064 {};

## 5.110 wisco::robot::subsystems::wings::PistonWingsBuilder Class Reference

Builder class for the piston wings class.

### Public Member Functions

- [PistonWingsBuilder \\* withLeftPiston \(std::unique\\_ptr< io::IPiston > &left\\_piston\)](#)  
*Adds a left wing piston to the build.*
- [PistonWingsBuilder \\* withRightPiston \(std::unique\\_ptr< io::IPiston > &right\\_piston\)](#)  
*Adds a right wing piston to the build.*
- [PistonWingsBuilder \\* withOutState \(bool out\\_state\)](#)  
*Adds an out state to the build.*
- [std::unique\\_ptr< IWings > build \(\)](#)  
*Builds the piston wings system.*

## Private Attributes

- `hal::PistonGroup m_left_pistons {}`  
*The pistons for the left wing.*
- `hal::PistonGroup m_right_pistons {}`  
*The pistons for the right wing.*
- `bool m_out_state {}`  
*The state of the pistons when the wings are out.*

### 5.110.1 Detailed Description

Builder class for the piston wings class.

#### Author

Nathan Sandvig

Definition at line 43 of file [PistonWingsBuilder.hpp](#).

### 5.110.2 Member Function Documentation

#### 5.110.2.1 withLeftPiston()

```
PistonWingsBuilder * wisco::robot::subsystems::wings::PistonWingsBuilder::withLeftPiston (
    std::unique_ptr< io::IPiston > & left_piston )
```

Adds a left wing piston to the build.

#### Parameters

<code>left_piston</code>	The left wing piston
--------------------------	----------------------

#### Returns

This object for build chaining

Definition at line 11 of file [PistonWingsBuilder.cpp](#).

```
00012 {
00013     m_left_pistons.addPiston(left_piston);
00014     return this;
00015 }
```

#### 5.110.2.2 withRightPiston()

```
PistonWingsBuilder * wisco::robot::subsystems::wings::PistonWingsBuilder::withRightPiston (
    std::unique_ptr< io::IPiston > & right_piston )
```

Adds a right wing piston to the build.

**Parameters**

<i>right_piston</i>	The right wing piston
---------------------	-----------------------

**Returns**

This object for build chaining

Definition at line 17 of file [PistonWingsBuilder.cpp](#).

```
00018 {
00019     m_right_pistons.addPiston(right_piston);
00020     return this;
00021 }
```

**5.110.2.3 withOutState()**

```
PistonWingsBuilder * wisco::robot::subsystems::wings::PistonWingsBuilder::withOutState (
    bool out_state )
```

Adds an out state to the build.

**Parameters**

<i>out_state</i>	The state of the pistons when the wings are out
------------------	---

**Returns**

This object for build chaining

Definition at line 23 of file [PistonWingsBuilder.cpp](#).

```
00024 {
00025     m_out_state = out_state;
00026     return this;
00027 }
```

**5.110.2.4 build()**

```
std::unique_ptr< IWings > wisco::robot::subsystems::wings::PistonWingsBuilder::build ( )
```

Builds the piston wings system.

**Returns**

`std::unique_ptr<IWings>` The piston wings system as a wings interface

Definition at line 29 of file [PistonWingsBuilder.cpp](#).

```
00030 {
00031     std::unique_ptr<PistonWings> piston_wings{std::make_unique<PistonWings>()};
00032     piston_wings->setLeftPistons(m_left_pistons);
00033     piston_wings->setRightPistons(m_right_pistons);
00034     piston_wings->setOutState(m_out_state);
00035     return piston_wings;
00036 }
```

### 5.110.3 Member Data Documentation

#### 5.110.3.1 m\_left\_pistons

```
hal::PistonGroup wisco::robot::subsystems::wings::PistonWingsBuilder::m_left_pistons {} [private]
```

The pistons for the left wing.

Definition at line 50 of file [PistonWingsBuilder.hpp](#).  
00050 {};

#### 5.110.3.2 m\_right\_pistons

```
hal::PistonGroup wisco::robot::subsystems::wings::PistonWingsBuilder::m_right_pistons {} [private]
```

The pistons for the right wing.

Definition at line 56 of file [PistonWingsBuilder.hpp](#).  
00056 {};

#### 5.110.3.3 m\_out\_state

```
bool wisco::robot::subsystems::wings::PistonWingsBuilder::m_out_state {} [private]
```

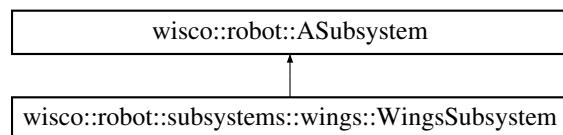
The state of the pistons when the wings are out.

Definition at line 62 of file [PistonWingsBuilder.hpp](#).  
00062 {};

## 5.111 wisco::robot::subsystems::wings::WingsSubsystem Class Reference

The subsystem adapter for wings.

Inheritance diagram for wisco::robot::subsystems::wings::WingsSubsystem:



### Public Member Functions

- **WingsSubsystem** (std::unique\_ptr< [IWings](#) > &wings)  
*Construct a new Wings Subsystem object.*
- void **initialize** () override  
*Initializes the subsystem.*
- void **run** () override  
*Runs the subsystem.*
- void **command** (std::string command\_name, va\_list &args) override  
*Runs a command for the subsystem.*
- void \* **state** (std::string state\_name) override  
*Gets a state of the subsystem.*

## Public Member Functions inherited from [wisco::robot::ASubsystem](#)

- **ASubsystem ()=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem (const [ASubsystem](#) &other)=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem ([ASubsystem](#) &&other)=default**  
*Construct a new [ASubsystem](#) object.*
- **ASubsystem (std::string name)**  
*Construct a new [ASubsystem](#) object.*
- **virtual ~ASubsystem ()=default**  
*Destroy the [ASubsystem](#) object.*
- **const std::string & getName () const**  
*Get the name of the subsystem.*
- **[ASubsystem](#) & operator= (const [ASubsystem](#) &rhs)=default**  
*Copy assignment operator for [ASubsystem](#).*
- **[ASubsystem](#) & operator= ([ASubsystem](#) &&rhs)=default**  
*Move assignment operator for [ASubsystem](#).*

## Private Attributes

- **std::unique\_ptr< [IWings](#) > m\_wings {}**  
*The wings being adapted.*

## Static Private Attributes

- **static constexpr char SUBSYSTEM\_NAME [] {"WINGS"}**  
*The name of the subsystem.*
- **static constexpr char SET\_LEFT\_WING\_COMMAND\_NAME [] {"SET LEFT WING"}**  
*The name of the set left wing command.*
- **static constexpr char SET\_RIGHT\_WING\_COMMAND\_NAME [] {"SET RIGHT WING"}**  
*The name of the set right wing command.*
- **static constexpr char GET\_LEFT\_WING\_STATE\_NAME [] {"GET LEFT WING"}**  
*The name of the get left wing state.*
- **static constexpr char GET\_RIGHT\_WING\_STATE\_NAME [] {"GET RIGHT WING"}**  
*The name of the get right wing state.*

### 5.111.1 Detailed Description

The subsystem adapter for wings.

#### Author

Nathan Sandvig

Definition at line 46 of file [WingsSubsystem.hpp](#).

### 5.111.2 Constructor & Destructor Documentation

#### 5.111.2.1 WingsSubsystem()

```
wisco::robot::subsystems::wings::WingsSubsystem::WingsSubsystem (
    std::unique_ptr< IWings > & wings )
```

Construct a new Wings Subsystem object.

**Parameters**

<i>wings</i>	The wings being adapted
--------------	-------------------------

Definition at line 11 of file [WingsSubsystem.cpp](#).

```
00012     : m_wings{std::move(wings)}
00013 {
00014
00015 }
```

### 5.111.3 Member Function Documentation

#### 5.111.3.1 initialize()

```
void wisco::robot::subsystems::wings::WingsSubsystem::initialize ( ) [override], [virtual]
```

Initializes the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 17 of file [WingsSubsystem.cpp](#).

```
00018 {
00019     if (m_wings)
00020         m_wings->initialize();
00021 }
```

#### 5.111.3.2 run()

```
void wisco::robot::subsystems::wings::WingsSubsystem::run ( ) [override], [virtual]
```

Runs the subsystem.

Implements [wisco::robot::ASubsystem](#).

Definition at line 23 of file [WingsSubsystem.cpp](#).

```
00024 {
00025     if (m_wings)
00026         m_wings->run();
00027 }
```

#### 5.111.3.3 command()

```
void wisco::robot::subsystems::wings::WingsSubsystem::command (
    std::string command_name,
    va_list & args ) [override], [virtual]
```

Runs a command for the subsystem.

**Parameters**

<i>command_name</i>	The name of the command to run
<i>args</i>	The parameters for the command

Implements [wisco::robot::ASubsystem](#).

Definition at line 29 of file [WingsSubsystem.cpp](#).

```
00030 {
00031     if (command_name == SET_LEFT_WING_COMMAND_NAME)
00032     {
00033         bool out{va_arg(args, bool)};
00034         m_wings->setLeftWing(out);
00035     }
00036     else if (command_name == SET_RIGHT_WING_COMMAND_NAME)
00037     {
00038         bool out{va_arg(args, bool)};
00039         m_wings->setRightWing(out);
00040     }
00041 }
```

#### 5.111.3.4 state()

```
void * wisco::robot::subsystems::wings::WingsSubsystem::state (
    std::string state_name) [override], [virtual]
```

Gets a state of the subsystem.

**Parameters**

<i>state_name</i>	The name of the state to get
-------------------	------------------------------

**Returns**

`void*` The current value of that state

Implements [wisco::robot::ASubsystem](#).

Definition at line 43 of file [WingsSubsystem.cpp](#).

```
00044 {
00045     void* result=nullptr;
00046
00047     if (state_name == GET_LEFT_WING_STATE_NAME)
00048     {
00049         bool* out{new bool{m_wings->getLeftWing()}};
00050         result = out;
00051     }
00052     else if (state_name == GET_RIGHT_WING_STATE_NAME)
00053     {
00054         bool* out{new bool{m_wings->getRightWing()}};
00055         result = out;
00056     }
00057
00058     return result;
00059 }
```

### 5.111.4 Member Data Documentation

#### 5.111.4.1 SUBSYSTEM\_NAME

```
constexpr char wisco::robot::subsystems::wings::WingsSubsystem::SUBSYSTEM_NAME[] {"WINGS"}
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file [WingsSubsystem.hpp](#).

```
00053 {"WINGS"};
```

#### 5.111.4.2 SET\_LEFT\_WING\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::wings::WingsSubsystem::SET_LEFT_WING_COMMAND_NAME[ ]  
{"SET LEFT WING"} [static], [constexpr], [private]
```

The name of the set left wing command.

Definition at line 59 of file [WingsSubsystem.hpp](#).

```
00059 {"SET LEFT WING"};
```

#### 5.111.4.3 SET\_RIGHT\_WING\_COMMAND\_NAME

```
constexpr char wisco::robot::subsystems::wings::WingsSubsystem::SET_RIGHT_WING_COMMAND_NAME[ ]  
{"SET RIGHT WING"} [static], [constexpr], [private]
```

The name of the set right wing command.

Definition at line 65 of file [WingsSubsystem.hpp](#).

```
00065 {"SET RIGHT WING"};
```

#### 5.111.4.4 GET\_LEFT\_WING\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::wings::WingsSubsystem::GET_LEFT_WING_STATE_NAME[ ]  
{"GET LEFT WING"} [static], [constexpr], [private]
```

The name of the get left wing state.

Definition at line 71 of file [WingsSubsystem.hpp](#).

```
00071 {"GET LEFT WING"};
```

#### 5.111.4.5 GET\_RIGHT\_WING\_STATE\_NAME

```
constexpr char wisco::robot::subsystems::wings::WingsSubsystem::GET_RIGHT_WING_STATE_NAME[ ]  
{"GET RIGHT WING"} [static], [constexpr], [private]
```

The name of the get right wing state.

Definition at line 77 of file [WingsSubsystem.hpp](#).

```
00077 {"GET RIGHT WING"};
```

#### 5.111.4.6 m\_wings

```
std::unique_ptr<IWings> wisco::robot::subsystems::wings::WingsSubsystem::m_wings {} [private]
```

The wings being adapted.

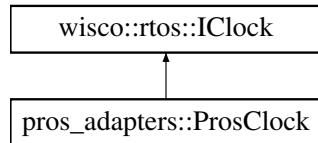
Definition at line 83 of file [WingsSubsystem.hpp](#).

```
00083 {};
```

## 5.112 wisco::rtos::IClock Class Reference

Interface for an rtos system clock.

Inheritance diagram for wisco::rtos::IClock:



### Public Member Functions

- virtual ~**IClock** ()=default  
*Destroy the **IClock** object.*
- virtual std::unique\_ptr< **IClock** > **clone** () const =0  
*Clones the **IClock** object.*
- virtual uint32\_t **getTime** ()=0  
*Get the clock time in milliseconds.*

### 5.112.1 Detailed Description

Interface for an rtos system clock.

#### Author

Nathan Sandvig

Definition at line 28 of file [IClock.hpp](#).

### 5.112.2 Member Function Documentation

#### 5.112.2.1 **clone()**

```
virtual std::unique_ptr< IClock > wisco::rtos::IClock::clone ( ) const [pure virtual]
```

Clones the **IClock** object.

#### Returns

`std::unique_ptr<IClock>` The cloned **IClock** object

Implemented in [pros\\_adapters::ProsClock](#).

### 5.112.2.2 `getTime()`

```
virtual uint32_t wisco::rtos::IClock::getTime ( ) [pure virtual]
```

Get the clock time in milliseconds.

#### Returns

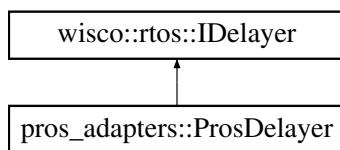
`uint32_t` The clock time in milliseconds

Implemented in [pros\\_adapters::ProsClock](#).

## 5.113 `wisco::rtos::IDelayer` Class Reference

Interface for rtos delay systems.

Inheritance diagram for `wisco::rtos::IDelayer`:



#### Public Member Functions

- `virtual ~IDelayer ()=default`  
*Destroy the `IDelayer` object.*
- `virtual std::unique_ptr< wisco::rtos::IDelayer > clone () const =0`  
*Clones the `IDelayer` object.*
- `virtual void delay (uint32_t millis)=0`  
*Delays the rtos system for a number of milliseconds.*
- `virtual void delayUntil (uint32_t time)=0`  
*Delays the rtos system until a certain system time in milliseconds.*

### 5.113.1 Detailed Description

Interface for rtos delay systems.

#### Author

Nathan Sandvig

Definition at line 28 of file [IDelayer.hpp](#).

## 5.113.2 Member Function Documentation

### 5.113.2.1 clone()

```
virtual std::unique_ptr< wisco::rtos::IDelayer > wisco::rtos::IDelayer::clone ( ) const [pure virtual]
```

Clones the [IDelayer](#) object.

#### Returns

`std::unique_ptr<IDelayer>` The cloned [IDelayer](#) object

Implemented in [pros\\_adapters::ProsDelayer](#).

### 5.113.2.2 delay()

```
virtual void wisco::rtos::IDelayer::delay ( uint32_t millis ) [pure virtual]
```

Delays the rtos system for a number of milliseconds.

#### Parameters

<code>millis</code>	The number of milliseconds to delay
---------------------	-------------------------------------

Implemented in [pros\\_adapters::ProsDelayer](#).

### 5.113.2.3 delayUntil()

```
virtual void wisco::rtos::IDelayer::delayUntil ( uint32_t time ) [pure virtual]
```

Delays the rtos system until a certain system time in milliseconds.

#### Parameters

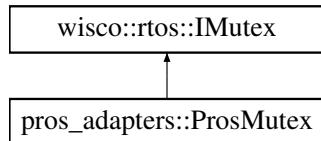
<code>time</code>	The time in milliseconds to delay until
-------------------	---

Implemented in [pros\\_adapters::ProsDelayer](#).

## 5.114 wisco::rtos::IMutex Class Reference

Interface for rtos mutexes.

Inheritance diagram for wisco::rtos::IMutex:



## Public Member Functions

- virtual ~**IMutex** ()=default  
*Destroy the **IMutex** object.*
- virtual void **take** ()=0  
*Takes the mutex and locks it.*
- virtual void **give** ()=0  
*Gives the mutex and unlocks it.*

### 5.114.1 Detailed Description

Interface for rtos mutexes.

#### Author

Nathan Sandvig

Definition at line 25 of file [IMutex.hpp](#).

### 5.114.2 Member Function Documentation

#### 5.114.2.1 **take()**

```
virtual void wisco::rtos::IMutex::take ( ) [pure virtual]
```

Takes the mutex and locks it.

Implemented in [pros\\_adapters::ProsMutex](#).

#### 5.114.2.2 **give()**

```
virtual void wisco::rtos::IMutex::give ( ) [pure virtual]
```

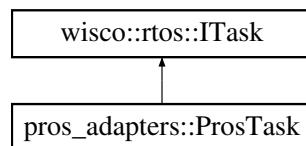
Gives the mutex and unlocks it.

Implemented in [pros\\_adapters::ProsMutex](#).

## 5.115 wisco::rtos::ITask Class Reference

Interface for an rtos task system.

Inheritance diagram for wisco::rtos::ITask:



### Public Member Functions

- virtual ~**ITask** ()=default  
*Destroy the **ITask** object.*
- virtual void **start** (void(\*function)(void \*), void \*parameters)=0  
*Starts the task.*
- virtual void **remove** ()=0  
*Removes the task from the system.*
- virtual void **suspend** ()=0  
*Suspends the task in the scheduler.*
- virtual void **resume** ()=0  
*Resumes the task in the scheduler.*
- virtual void **join** ()=0  
*Waits for the task to finish.*

### 5.115.1 Detailed Description

Interface for an rtos task system.

#### Author

Nathan Sandvig

Definition at line 25 of file [ITask.hpp](#).

### 5.115.2 Member Function Documentation

#### 5.115.2.1 start()

```
virtual void wisco::rtos::ITask::start (
    void(*)(void *) function,
    void * parameters ) [pure virtual]
```

Starts the task.

**Parameters**

<i>function</i>	The function to run in the task
<i>parameters</i>	The parameters for the function

Implemented in [pros\\_adapters::ProsTask](#).

**5.115.2.2 remove()**

```
virtual void wisco::rtos::ITask::remove () [pure virtual]
```

Removes the task from the system.

Implemented in [pros\\_adapters::ProsTask](#).

**5.115.2.3 suspend()**

```
virtual void wisco::rtos::ITask::suspend () [pure virtual]
```

Suspends the task in the scheduler.

Implemented in [pros\\_adapters::ProsTask](#).

**5.115.2.4 resume()**

```
virtual void wisco::rtos::ITask::resume () [pure virtual]
```

Resumes the task in the scheduler.

Implemented in [pros\\_adapters::ProsTask](#).

**5.115.2.5 join()**

```
virtual void wisco::rtos::ITask::join () [pure virtual]
```

Waits for the task to finish.

Implemented in [pros\\_adapters::ProsTask](#).

## 5.116 wisco::SystemConfiguration Struct Reference

Holds the system configuration information.

## Public Attributes

- std::unique\_ptr<IAlliance> alliance {}  
*The system alliance.*
- std::unique\_ptr<IAutonomous> autonomous {}  
*The system autonomous.*
- std::unique\_ptr<IConfiguration> configuration {}  
*The system configuration.*
- std::unique\_ptr<IProfile> profile {}  
*The system profile.*

### 5.116.1 Detailed Description

Holds the system configuration information.

#### Author

Nathan Sandvig

Definition at line 24 of file [SystemConfiguration.hpp](#).

### 5.116.2 Member Data Documentation

#### 5.116.2.1 alliance

```
std::unique_ptr<IAlliance> wisco::SystemConfiguration::alliance {}
```

The system alliance.

Definition at line 30 of file [SystemConfiguration.hpp](#).  
00030 {};

#### 5.116.2.2 autonomous

```
std::unique_ptr<IAutonomous> wisco::SystemConfiguration::autonomous {}
```

The system autonomous.

Definition at line 36 of file [SystemConfiguration.hpp](#).  
00036 {};

#### 5.116.2.3 configuration

```
std::unique_ptr<IConfiguration> wisco::SystemConfiguration::configuration {}
```

The system configuration.

Definition at line 42 of file [SystemConfiguration.hpp](#).  
00042 {};

### 5.116.2.4 profile

```
std::unique_ptr<IProfile> wisco::SystemConfiguration::profile {}
```

The system profile.

Definition at line 48 of file [SystemConfiguration.hpp](#).  
00048 {};

## 5.117 wisco::testing::pros\_testing::DriveTest Class Reference

Tests a pros-based drive.

### Public Member Functions

- [DriveTest](#) (std::unique\_ptr< pros::MotorGroup > &left\_drive\_motors, std::unique\_ptr< pros::MotorGroup > &right\_drive\_motors, std::unique\_ptr< pros::Imu > &heading\_sensor, std::unique\_ptr< pros::Rotation > &linear\_sensor, double linear\_counts\_per\_inch)  
*Construct a new Drive Test object.*
- void [initialize](#) ()  
*Initializes the drive testing system.*
- void [runLinearTest](#) ()  
*Runs the linear motion test.*
- void [runTurningTest](#) ()  
*Runs the turning motion test.*

### Private Attributes

- std::unique\_ptr< pros::MotorGroup > [m\\_left\\_drive\\_motors](#) {}  
*The left drive motors.*
- std::unique\_ptr< pros::MotorGroup > [m\\_right\\_drive\\_motors](#) {}  
*The right drive motors.*
- std::unique\_ptr< pros::Imu > [m\\_heading\\_sensor](#) {}  
*The heading sensor.*
- std::unique\_ptr< pros::Rotation > [m\\_linear\\_sensor](#) {}  
*The linear distance tracking sensor.*
- double [m\\_linear\\_counts\\_per\\_inch](#)  
*The CPI of the linear distance tracking sensor.*

## Static Private Attributes

- static constexpr char **LINEAR\_FILE\_NAME** [] {"drive\_linear\_test.csv"}  
*The name of the output file for linear drive testing.*
- static constexpr char **TURNING\_FILE\_NAME** [] {"drive\_turning\_test.csv"}  
*The name of the output file for turning drive testing.*
- static constexpr double **MILLIS\_TO\_S** {1.0 / 1000}  
*Converts milliseconds to seconds.*
- static constexpr double **HEADING\_TO\_RADIANS** {-M\_PI / 18000}  
*Converts heading to radians.*
- static constexpr double **INCHES\_TO\_METERS** {2.54 / 100}  
*Converts inches to meters.*
- static constexpr uint32\_t **V\_TO\_MV** {1000}  
*Converts Volts to millivolts.*
- static constexpr uint8\_t **TEST\_V** {8}  
*The number of millivolts to use for testing.*
- static constexpr uint32\_t **TEST\_DURATION** {500}  
*The duration of the tests in ms.*

## 5.117.1 Detailed Description

Tests a pros-based drive.

### Author

Nathan Sandvig

Definition at line 48 of file [DriveTest.hpp](#).

## 5.117.2 Constructor & Destructor Documentation

### 5.117.2.1 DriveTest()

```
wisco::testing::pros_testing::DriveTest::DriveTest (
    std::unique_ptr< pros::MotorGroup > & left_drive_motors,
    std::unique_ptr< pros::MotorGroup > & right_drive_motors,
    std::unique_ptr< pros::Imu > & heading_sensor,
    std::unique_ptr< pros::Rotation > & linear_sensor,
    double linear_counts_per_inch )
```

Construct a new Drive Test object.

#### Parameters

<i>left_drive_motors</i>	The left drive motors
<i>right_drive_motors</i>	The right drive motors
<i>heading_sensor</i>	The heading sensor
<i>linear_sensor</i>	The linear distance tracking sensor
<i>linear_counts_per_inch</i>	The linear distance tracking sensor CPI

Definition at line 9 of file DriveTest.cpp.

```
00013     :
00014     m_left_drive_motors{std::move(left_drive_motors)},
00015     m_right_drive_motors{std::move(right_drive_motors)},
00016     m_heading_sensor{std::move(heading_sensor)},
00017     m_linear_sensor{std::move(linear_sensor)},
00018     m_linear_counts_per_inch{linear_counts_per_inch}
00019 {
00020
00021 }
```

## 5.117.3 Member Function Documentation

### 5.117.3.1 initialize()

```
void wisco::testing::pros_testing::DriveTest::initialize ( )
```

Initializes the drive testing system.

Definition at line 23 of file DriveTest.cpp.

```
00024 {
00025     m_heading_sensor->reset(true);
00026     m_linear_sensor->reset();
00027     m_heading_sensor->set_data_rate(5);
00028     m_linear_sensor->set_data_rate(5);
00029 }
```

### 5.117.3.2 runLinearTest()

```
void wisco::testing::pros_testing::DriveTest::runLinearTest ( )
```

Runs the linear motion test.

Definition at line 31 of file DriveTest.cpp.

```
00032 {
00033     std::string test_output_file_path{FILE_PATH};
00034     test_output_file_path = test_output_file_path.append(LINEAR_FILE_NAME);
00035     std::ofstream test_output_file{test_output_file_path};
00036     if (test_output_file.fail())
00037         return;
00038
00039     test_output_file << "linear_velocity,time\n";
00040     test_output_file.flush();
00041
00042     if (m_left_drive_motors && m_right_drive_motors && m_heading_sensor && m_linear_sensor)
00043     {
00044         m_left_drive_motors->move_voltage(TEST_V * V_TO_MV);
00045         m_right_drive_motors->move_voltage(TEST_V * V_TO_MV);
00046
00047         uint32_t start_time{pros::millis()};
00048         double last_position{m_linear_sensor->get_position() / m_linear_counts_per_inch *
00049             INCHES_TO_METERS};
00050         double last_time{pros::millis() * MILLIS_TO_S};
00051         while (pros::millis() - start_time < TEST_DURATION)
00052         {
00053             double current_position{m_linear_sensor->get_position() / m_linear_counts_per_inch *
00054                 INCHES_TO_METERS};
00055             if (current_position != last_position)
00056             {
00057                 double position_change{current_position - last_position};
00058                 last_position = current_position;
00059
00060                 double current_time{pros::millis() * MILLIS_TO_S};
00061                 double time_change{current_time - last_time};
00062                 last_time = current_time;
00063
00064                 double velocity{position_change / time_change};
00065                 test_output_file << velocity << ',' << current_time << '\n';
00066                 test_output_file.flush();
00067             }
00068             pros::delay(1);
00069         }
00070         m_left_drive_motors->move_voltage(0);
00071         m_right_drive_motors->move_voltage(0);
00072     }
00073     test_output_file.close();
00074 }
```

### 5.117.3.3 runTurningTest()

```
void wisco::testing::pros_testing::DriveTest::runTurningTest( )
```

Runs the turning motion test.

Definition at line 76 of file [DriveTest.cpp](#).

```
00077 {
00078     std::string test_output_file_path{FILE_PATH};
00079     test_output_file_path = test_output_file_path.append(TURNING_FILE_NAME);
00080     std::ofstream test_output_file{test_output_file_path};
00081     if (test_output_file.fail())
00082         return;
00083
00084     test_output_file << "angular_velocity,time\n";
00085     test_output_file.flush();
00086
00087     if (m_left_drive_motors && m_right_drive_motors && m_heading_sensor && m_linear_sensor)
00088     {
00089         m_left_drive_motors->move_voltage(-TEST_V * V_TO_MV);
00090         m_right_drive_motors->move_voltage(TEST_V * V_TO_MV);
00091
00092         uint32_t start_time{pros::millis()};
00093         double last_rotation{m_heading_sensor->get_rotation() * HEADING_TO_RADIANS};
00094         double last_time{pros::millis() * MILLIS_TO_S};
00095         while (pros::millis() - start_time < TEST_DURATION)
00096         {
00097             double current_rotation{m_heading_sensor->get_rotation() * HEADING_TO_RADIANS};
00098             if (current_rotation != last_rotation)
00099             {
00100                 double rotation_change{current_rotation - last_rotation};
00101                 last_rotation = current_rotation;
00102
00103                 double current_time{pros::millis() * MILLIS_TO_S};
00104                 double time_change{current_time - last_time};
00105                 last_time = current_time;
00106
00107                 double velocity{rotation_change / time_change};
00108                 test_output_file << velocity << ',' << current_time << '\n';
00109                 test_output_file.flush();
00110             }
00111             pros::delay(1);
00112         }
00113
00114         m_left_drive_motors->move_voltage(0);
00115         m_right_drive_motors->move_voltage(0);
00116     }
00117
00118     test_output_file.close();
00119 }
```

## 5.117.4 Member Data Documentation

### 5.117.4.1 LINEAR\_FILE\_NAME

```
constexpr char wisco::testing::pros_testing::DriveTest::LINEAR_FILE_NAME[] {"drive_linear_↔
test.csv"} [static], [constexpr], [private]
```

The name of the output file for linear drive testing.

Definition at line 55 of file [DriveTest.hpp](#).

```
00055 {"drive_linear_test.csv"};
```

### 5.117.4.2 TURNING\_FILE\_NAME

```
constexpr char wisco::testing::pros_testing::DriveTest::TURNING_FILE_NAME[] {"drive_turning_↔
test.csv"} [static], [constexpr], [private]
```

The name of the output file for turning drive testing.

Definition at line 61 of file [DriveTest.hpp](#).

```
00061 {"drive_turning_test.csv"};
```

#### 5.117.4.3 **MILLIS\_TO\_S**

```
constexpr double wisco::testing::pros_testing::DriveTest::MILLIS_TO_S {1.0 / 1000} [static],  
[constexpr], [private]
```

Converts milliseconds to seconds.

Definition at line 67 of file [DriveTest.hpp](#).

```
00067 {1.0 / 1000};
```

#### 5.117.4.4 **HEADING\_TO\_RADIANS**

```
constexpr double wisco::testing::pros_testing::DriveTest::HEADING_TO_RADIANS {-M_PI / 18000}  
[static], [constexpr], [private]
```

Converts heading to radians.

Definition at line 73 of file [DriveTest.hpp](#).

```
00073 {-M_PI / 18000};
```

#### 5.117.4.5 **INCHES\_TO\_METERS**

```
constexpr double wisco::testing::pros_testing::DriveTest::INCHES_TO_METERS {2.54 / 100} [static],  
[constexpr], [private]
```

Converts inches to meters.

Definition at line 79 of file [DriveTest.hpp](#).

```
00079 {2.54 / 100};
```

#### 5.117.4.6 **V\_TO\_MV**

```
constexpr uint32_t wisco::testing::pros_testing::DriveTest::V_TO_MV {1000} [static], [constexpr],  
[private]
```

Converts Volts to milliVolts.

Definition at line 85 of file [DriveTest.hpp](#).

```
00085 {1000};
```

#### 5.117.4.7 **TEST\_V**

```
constexpr uint8_t wisco::testing::pros_testing::DriveTest::TEST_V {8} [static], [constexpr],  
[private]
```

The number of millivolts to use for testing.

Definition at line 91 of file [DriveTest.hpp](#).

```
00091 {8};
```

#### 5.117.4.8 TEST\_DURATION

```
constexpr uint32_t wisco::testing::pros_testing::DriveTest::TEST_DURATION {500} [static],  
[constexpr], [private]
```

The duration of the tests in ms.

Definition at line 97 of file [DriveTest.hpp](#).  
00097 {500};

#### 5.117.4.9 m\_left\_drive\_motors

```
std::unique_ptr<pros::MotorGroup> wisco::testing::pros_testing::DriveTest::m_left_drive_motors  
{ } [private]
```

The left drive motors.

Definition at line 103 of file [DriveTest.hpp](#).  
00103 {};

#### 5.117.4.10 m\_right\_drive\_motors

```
std::unique_ptr<pros::MotorGroup> wisco::testing::pros_testing::DriveTest::m_right_drive_motors  
{ } [private]
```

The right drive motors.

Definition at line 109 of file [DriveTest.hpp](#).  
00109 {};

#### 5.117.4.11 m\_heading\_sensor

```
std::unique_ptr<pros::Imu> wisco::testing::pros_testing::DriveTest::m_heading_sensor {} [private]
```

The heading sensor.

Definition at line 115 of file [DriveTest.hpp](#).  
00115 {};

#### 5.117.4.12 m\_linear\_sensor

```
std::unique_ptr<pros::Rotation> wisco::testing::pros_testing::DriveTest::m_linear_sensor {}  
[private]
```

The linear distance tracking sensor.

Definition at line 121 of file [DriveTest.hpp](#).  
00121 {};

#### 5.117.4.13 `m_linear_counts_per_inch`

```
double wisco::testing::pros_testing::DriveTest::m_linear_counts_per_inch [private]
```

The CPI of the linear distance tracking sensor.

Definition at line 127 of file [DriveTest.hpp](#).

## 5.118 `wisco::testing::TestFactory` Class Reference

Factory to build test classes.

### Static Public Member Functions

- static std::unique\_ptr< [pros\\_testing::DriveTest](#) > `createDriveTest ()`  
*Create a Drive Test object.*

### Static Private Attributes

- static const std::vector< int8\_t > `LEFT_DRIVE_PORTS` {11, 12, -13, -14}  
*The left drive motor ports.*
- static const std::vector< int8\_t > `RIGHT_DRIVE_PORTS` {17, 18, -19, -20}  
*The right drive motor ports.*
- static constexpr int8\_t `INERTIAL_PORT` {9}  
*The port for the inertial sensor.*
- static constexpr int8\_t `LINEAR_TRACKING_PORT` {8}  
*The port for the linear distance tracking sensor.*
- static constexpr double `LINEAR_COUNTS_PER_INCH` {4696.375}  
*The CPI of the linear distance tracking sensor.*

### 5.118.1 Detailed Description

Factory to build test classes.

#### Author

Nathan Sandvig

Definition at line 30 of file [TestFactory.hpp](#).

## 5.118.2 Member Function Documentation

### 5.118.2.1 createDriveTest()

```
std::unique_ptr< pros_testing::DriveTest > wisco::testing::TestFactory::createDriveTest ( )  
[static]
```

Create a Drive Test object.

#### Returns

`std::unique_ptr<pros_testing::DriveTest>` The drive test object

Definition at line 10 of file [TestFactory.cpp](#).

```
00011 {  
00012     std::unique_ptr<pros::MotorGroup>  
00013         left_drive_motors{std::make_unique<pros::MotorGroup>(LEFT_DRIVE_PORTS)};  
00014     std::unique_ptr<pros::MotorGroup>  
00015         right_drive_motors{std::make_unique<pros::MotorGroup>(RIGHT_DRIVE_PORTS)};  
00016     std::unique_ptr<pros::Imu> heading_sensor{std::make_unique<pros::Imu>(INERTIAL_PORT)};  
00017     std::unique_ptr<pros::Rotation>  
00018         linear_tracking_sensor{std::make_unique<pros::Rotation>(LINEAR_TRACKING_PORT)};  
00019     return std::make_unique<pros_testing::DriveTest>(left_drive_motors, right_drive_motors,  
00020             heading_sensor, linear_tracking_sensor, LINEAR_COUNTS_PER_INCH);  
00021 }
```

## 5.118.3 Member Data Documentation

### 5.118.3.1 LEFT\_DRIVE\_PORTS

```
const std::vector< int8_t > wisco::testing::TestFactory::LEFT_DRIVE_PORTS {11, 12, -13, -14}  
[static], [private]
```

The left drive motor ports.

Definition at line 37 of file [TestFactory.hpp](#).

### 5.118.3.2 RIGHT\_DRIVE\_PORTS

```
const std::vector< int8_t > wisco::testing::TestFactory::RIGHT_DRIVE_PORTS {17, 18, -19, -20}  
[static], [private]
```

The right drive motor ports.

Definition at line 43 of file [TestFactory.hpp](#).

### 5.118.3.3 INERTIAL\_PORT

```
constexpr int8_t wisco::testing::TestFactory::INERTIAL_PORT {9} [static], [constexpr], [private]
```

The port for the inertial sensor.

Definition at line 49 of file [TestFactory.hpp](#).

### 5.118.3.4 LINEAR\_TRACKING\_PORT

```
constexpr int8_t wisco::testing::TestFactory::LINEAR_TRACKING_PORT {8} [static], [constexpr], [private]
```

The port for the linear distance tracking sensor.

Definition at line 55 of file [TestFactory.hpp](#).

```
00055 {8};
```

### 5.118.3.5 LINEAR\_COUNTS\_PER\_INCH

```
constexpr double wisco::testing::TestFactory::LINEAR_COUNTS_PER_INCH {4696.375} [static], [constexpr], [private]
```

The CPI of the linear distance tracking sensor.

Definition at line 61 of file [TestFactory.hpp](#).

```
00061 {4696.375};
```

## 5.119 wisco::user::drive::DifferentialDriveOperator Class Reference

Runs the operator-controlled differential drive voltage settings.

### Public Member Functions

- [DifferentialDriveOperator](#) (const std::shared\_ptr< [user::IController](#) > &controller, const std::shared\_ptr< [robot::Robot](#) > &robot)
   
*Construct a new Drive Operator object.*
- void [setDriveVoltage](#) ([EChassisControlMode](#) control\_mode)
   
*Set the drive voltage.*

### Private Member Functions

- void [updateDriveVoltage](#) (double left\_voltage, double right\_voltage)
   
*Updates the voltage of the drive subsystem.*
- void [updateArcade](#) (double forward, double turn)
   
*Updates the drive using arcade inputs.*
- void [updateSingleArcadeLeft](#) ()
   
*Update the drive voltage for single left stick arcade drive.*
- void [updateSingleArcadeRight](#) ()
   
*Update the drive voltage for single right stick arcade drive.*
- void [updateSplitArcadeLeft](#) ()
   
*Update the drive voltage for split stick arcade with left stick forward control.*
- void [updateSplitArcadeRight](#) ()
   
*Update the drive voltage for split stick arcade with right stick forward control.*
- void [updateTank](#) ()
   
*Update the drive voltage for tank control.*

### Private Attributes

- std::shared\_ptr< user::IController > m\_controller {}  
*The user input controller.*
- std::shared\_ptr< robot::Robot > m\_robot {}  
*The robot being controlled.*

### Static Private Attributes

- static constexpr char DIFFERENTIAL\_DRIVE\_SUBSYSTEM\_NAME [] {"DIFFERENTIAL DRIVE"}  
*The name of the differential drive subsystem.*
- static constexpr char SET\_VOLTAGE\_COMMAND [] {"SET VOLTAGE"}  
*The command to set drive voltage.*
- static constexpr double VOLTAGE\_CONVERSION {12.0}  
*Converts controller input to voltage.*

## 5.119.1 Detailed Description

Runs the operator-controlled differential drive voltage settings.

### Author

Nathan Sandvig

Definition at line 39 of file [DifferentialDriveOperator.hpp](#).

## 5.119.2 Constructor & Destructor Documentation

### 5.119.2.1 DifferentialDriveOperator()

```
wisco::user::drive::DifferentialDriveOperator::DifferentialDriveOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Drive Operator object.

#### Parameters

controller	The user input controller
robot	The robot to control

Definition at line 59 of file [DifferentialDriveOperator.cpp](#).

```
00061     : m_controller(controller), m_robot(robot)
00062 {
00063
00064 }
```

### 5.119.3 Member Function Documentation

#### 5.119.3.1 updateDriveVoltage()

```
void wisco::user::drive::DifferentialDriveOperator::updateDriveVoltage (
    double left_voltage,
    double right_voltage ) [private]
```

Updates the voltage of the drive subsystem.

##### Parameters

<i>left_voltage</i>	The left drive voltage
<i>right_voltage</i>	The right drive voltage

Definition at line 9 of file DifferentialDriveOperator.cpp.

```
00010 {
00011     if (m_robot)
00012     {
00013         m_robot->sendCommand(DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME, SET_VOLTAGE_COMMAND, left_voltage,
00014             right_voltage);
00015     }
```

#### 5.119.3.2 updateArcade()

```
void wisco::user::drive::DifferentialDriveOperator::updateArcade (
    double forward,
    double turn ) [private]
```

Updates the drive using arcade inputs.

##### Parameters

<i>forward</i>	The forward voltage
<i>turn</i>	The turn voltage (positive to the right)

Definition at line 17 of file DifferentialDriveOperator.cpp.

```
00018 {
00019     double left_voltage{(forward + turn) * VOLTAGE_CONVERSION};
00020     double right_voltage{(forward - turn) * VOLTAGE_CONVERSION};
00021     updateDriveVoltage(left_voltage, right_voltage);
00022 }
```

#### 5.119.3.3 updateSingleArcadeLeft()

```
void wisco::user::drive::DifferentialDriveOperator::updateSingleArcadeLeft ( ) [private]
```

Update the drive voltage for single left stick arcade drive.

Definition at line 24 of file DifferentialDriveOperator.cpp.

```
00025 {
00026     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y)};
00027     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_X)};
00028     updateArcade(forward, turn);
00029 }
```

### 5.119.3.4 updateSingleArcadeRight()

```
void wisco::user::drive::DifferentialDriveOperator::updateSingleArcadeRight () [private]
```

Update the drive voltage for single right stick arcade drive.

Definition at line 31 of file [DifferentialDriveOperator.cpp](#).

```
00032 {
00033     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y)};
00034     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_X)};
00035     updateArcade(forward, turn);
00036 }
```

### 5.119.3.5 updateSplitArcadeLeft()

```
void wisco::user::drive::DifferentialDriveOperator::updateSplitArcadeLeft () [private]
```

Update the drive voltage for split stick arcade with left stick forward control.

Definition at line 38 of file [DifferentialDriveOperator.cpp](#).

```
00039 {
00040     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y)};
00041     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_X)};
00042     updateArcade(forward, turn);
00043 }
```

### 5.119.3.6 updateSplitArcadeRight()

```
void wisco::user::drive::DifferentialDriveOperator::updateSplitArcadeRight () [private]
```

Update the drive voltage for split stick arcade with right stick forward control.

Definition at line 45 of file [DifferentialDriveOperator.cpp](#).

```
00046 {
00047     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y)};
00048     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_X)};
00049     updateArcade(forward, turn);
00050 }
```

### 5.119.3.7 updateTank()

```
void wisco::user::drive::DifferentialDriveOperator::updateTank () [private]
```

Update the drive voltage for tank control.

Definition at line 52 of file [DifferentialDriveOperator.cpp](#).

```
00053 {
00054     double left_voltage{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y) *
VOLTAGE_CONVERSION};
00055     double right_voltage{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y) *
VOLTAGE_CONVERSION};
00056     updateDriveVoltage(left_voltage, right_voltage);
00057 }
```

### 5.119.3.8 setDriveVoltage()

```
void wisco::user::drive::DifferentialDriveOperator::setDriveVoltage (
    EChassisControlMode control_mode )
```

Set the drive voltage.

**Parameters**

<i>control_mode</i>	The control mode of the drive
---------------------	-------------------------------

Definition at line 66 of file DifferentialDriveOperator.cpp.

```
00067 {
00068     if (!m_controller)
00069     {
00070         updateDriveVoltage(0, 0);
00071         return;
00072     }
00073
00074     switch (control_mode)
00075     {
00076         case EChassisControlMode::SINGLE_ARCADE_LEFT:
00077             updateSingleArcadeLeft();
00078             break;
00079         case EChassisControlMode::SINGLE_ARCADE_RIGHT:
00080             updateSingleArcadeRight();
00081             break;
00082         case EChassisControlMode::SPLIT_ARCADE_LEFT:
00083             updateSplitArcadeLeft();
00084             break;
00085         case EChassisControlMode::SPLIT_ARCADE_RIGHT:
00086             updateSplitArcadeRight();
00087             break;
00088         case EChassisControlMode::TANK:
00089             updateTank();
00090             break;
00091     }
00092 }
```

## 5.119.4 Member Data Documentation

### 5.119.4.1 DIFFERENTIAL\_DRIVE\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::drive::DifferentialDriveOperator::DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME[] {"DIFFERENTIAL DRIVE"} [static], [constexpr], [private]
```

The name of the differential drive subsystem.

Definition at line 46 of file DifferentialDriveOperator.hpp.  
00046 {"DIFFERENTIAL DRIVE"};

### 5.119.4.2 SET\_VOLTAGE\_COMMAND

```
constexpr char wisco::user::drive::DifferentialDriveOperator::SET_VOLTAGE_COMMAND[] {"SET VOLTAGE"} [static], [constexpr], [private]
```

The command to set drive voltage.

Definition at line 52 of file DifferentialDriveOperator.hpp.  
00052 {"SET VOLTAGE"};

### 5.119.4.3 VOLTAGE\_CONVERSION

```
constexpr double wisco::user::drive::DifferentialDriveOperator::VOLTAGE_CONVERSION {12.0} [static], [constexpr], [private]
```

Converts controller input to voltage.

Definition at line 58 of file DifferentialDriveOperator.hpp.  
00058 {12.0};

#### 5.119.4.4 m\_controller

```
std::shared_ptr<user::IController> wisco::user::drive::DifferentialDriveOperator::m_controller  
{ } [private]
```

The user input controller.

Definition at line 64 of file [DifferentialDriveOperator.hpp](#).

```
00064 {};
```

#### 5.119.4.5 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::drive::DifferentialDriveOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 70 of file [DifferentialDriveOperator.hpp](#).

```
00070 {};
```

## 5.120 wisco::user::elevator::ElevatorOperator Class Reference

Runs the operator-controlled elevator position settings.

### Public Member Functions

- `ElevatorOperator (const std::shared_ptr< user::IController > &controller, const std::shared_ptr< robot::Robot > &robot)`  
*Construct a new Elevator Operator object.*
- `void setElevatorPosition (const std::unique_ptr< IProfile > &profile)`  
*Set the elevator position.*

### Private Types

- enum class `EToggleState { IN , FIELD , MATCH_LOAD , POLE_HANG , PARTNER_HANG }`  
*The available states for elevator toggles.*

## Private Member Functions

- `double getElevatorPosition ()`  
*Gets the current elevator position.*
- `double getCapDistance ()`  
*Gets the current cap distance.*
- `bool getHangArmUp ()`  
*Get the hang arm up state.*
- `void updateElevatorPosition (double position)`  
*Updates the position of the elevator subsystem.*
- `void updatePoleHangPosition ()`  
*Updates the pole hang position using the distance sensor.*
- `void updateManual (EControllerDigital in, EControllerDigital out)`  
*Updates the elevator position based on manual control.*
- `void updatePresetSplit (EControllerDigital in, EControllerDigital field, EControllerDigital match_load, EControllerDigital pole_hang, EControllerDigital partner_hang)`  
*Updates the elevator position based on preset split control.*
- `void updatePresetToggle (EControllerDigital toggle)`  
*Updates the elevator position based on a toggle.*
- `void updatePresetLadder (EControllerDigital in, EControllerDigital out)`  
*Updates the elevator position based on a ladder toggle system.*
- `void updatePresetLadderIntake (EControllerDigital in, EControllerDigital out, EControllerDigital intake, EControllerDigital outtake)`  
*Updates the elevator position based on a ladder toggle system with an intake override.*

## Private Attributes

- `std::shared_ptr< user::IController > m_controller {}`  
*The user input controller.*
- `std::shared_ptr< robot::Robot > m_robot {}`  
*The robot being controlled.*
- `EToggleState toggle_state {EToggleState::IN}`  
*The state stored for toggle mode.*
- `bool manual_input {}`  
*Whether or not there is currently manual input.*

## Static Private Attributes

- `static constexpr char ELEVATOR_SUBSYSTEM_NAME [] {"ELEVATOR"}`  
*The name of the elevator subsystem.*
- `static constexpr char HANG_SUBSYSTEM_NAME [] {"HANG"}`  
*The name of the hang subsystem.*
- `static constexpr char SET_POSITION_COMMAND [] {"SET POSITION"}`  
*The command to set elevator position.*
- `static constexpr char GET_POSITION_STATE [] {"GET POSITION"}`  
*The state to get elevator position.*
- `static constexpr char CAP_DISTANCE_STATE_NAME [] {"CAP DISTANCE"}`  
*The name of the cap distance state.*
- `static constexpr char HANG_ARM_UP_STATE_NAME [] {"ARM UP"}`  
*The name of the hang arm up state.*

- static constexpr double [IN\\_POSITION](#) {0.0}  
*The in position for the elevator.*
- static constexpr double [FIELD\\_POSITION](#) {3.0}  
*The field position for the elevator.*
- static constexpr double [MATCH\\_LOAD\\_POSITION](#) {8.0}  
*The match loading position for the elevator.*
- static constexpr double [POLE\\_HANG\\_POSITION](#) {16.0}  
*The pole hang position for the elevator.*
- static constexpr double [PARTNER\\_HANG\\_POSITION](#) {18.0}  
*The partner hang position for the elevator.*
- static constexpr double [POLE\\_HANG\\_DISTANCE](#) {2.0}  
*The distance to the cap for a pole hang.*
- static constexpr double [CAP\\_DETECTED\\_DISTANCE](#) {5.0}  
*The distance within which a cap is considered detected.*

### 5.120.1 Detailed Description

Runs the operator-controlled elevator position settings.

#### Author

Nathan Sandvig

Definition at line [40](#) of file [ElevatorOperator.hpp](#).

### 5.120.2 Member Enumeration Documentation

#### 5.120.2.1 EToggleState

```
enum class wisco::user::elevator::ElevatorOperator::EToggleState [strong], [private]
```

The available states for elevator toggles.

Definition at line [47](#) of file [ElevatorOperator.hpp](#).

```
00048  {
00049      IN,
00050      FIELD,
00051      MATCH_LOAD,
00052      POLE_HANG,
00053      PARTNER_HANG
00054  };
```

### 5.120.3 Constructor & Destructor Documentation

#### 5.120.3.1 ElevatorOperator()

```
wisco::user::elevator::ElevatorOperator::ElevatorOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Elevator Operator object.

**Parameters**

<i>controller</i>	The user input controller
<i>robot</i>	The robot to control

Definition at line 9 of file [ElevatorOperator.cpp](#).

```
00011     : m_controller(controller), m_robot(robot)
00012 {
00013
00014 }
```

## 5.120.4 Member Function Documentation

### 5.120.4.1 getElevatorPosition()

```
double wisco::user::elevator::ElevatorOperator::getElevatorPosition () [private]
```

Gets the current elevator position.

**Returns**

```
double The current elevator position
```

Definition at line 16 of file [ElevatorOperator.cpp](#).

```
00017 {
00018     double* result{static_cast<double*>(m_robot->getState(ELEVATOR_SUBSYSTEM_NAME,
00019                                         GET_POSITION_STATE))};
00020     double position{*result};
00021     delete result;
00022     return position;
00023 }
```

### 5.120.4.2 getCapDistance()

```
double wisco::user::elevator::ElevatorOperator::getCapDistance () [private]
```

Gets the current cap distance.

**Returns**

```
double The current cap distance
```

Definition at line 24 of file [ElevatorOperator.cpp](#).

```
00025 {
00026     double* result{static_cast<double*>(m_robot->getState(ELEVATOR_SUBSYSTEM_NAME,
00027                                         CAP_DISTANCE_STATE_NAME))};
00028     double distance{*result};
00029     delete result;
00030 }
```

### 5.120.4.3 getHangArmUp()

```
bool wisco::user::elevator::ElevatorOperator::getHangArmUp ( ) [private]
```

Get the hang arm up state.

#### Returns

- true The hang arm is up
- false The hang arm is down

Definition at line 32 of file [ElevatorOperator.cpp](#).

```
00033 {
00034     bool* result{static_cast<bool*>(m_robot->getState(HANG_SUBSYSTEM_NAME, HANG_ARM_UP_STATE_NAME))};
00035     bool hang_arm_up{*result};
00036     delete result;
00037     return hang_arm_up;
00038 }
```

### 5.120.4.4 updateElevatorPosition()

```
void wisco::user::elevator::ElevatorOperator::updateElevatorPosition (
    double position) [private]
```

Updates the position of the elevator subsystem.

#### Parameters

<i>position</i>	The elevator position
-----------------	-----------------------

Definition at line 40 of file [ElevatorOperator.cpp](#).

```
00041 {
00042     m_robot->sendCommand(ELEVATOR_SUBSYSTEM_NAME, SET_POSITION_COMMAND, position);
00043 }
```

### 5.120.4.5 updatePoleHangPosition()

```
void wisco::user::elevator::ElevatorOperator::updatePoleHangPosition ( ) [private]
```

Updates the pole hang position using the distance sensor.

Definition at line 45 of file [ElevatorOperator.cpp](#).

```
00046 {
00047     if (getHangArmUp())
00048     {
00049         double distance{getCapDistance()};
00050         if (distance != 0 && distance < CAP_DETECTED_DISTANCE)
00051         {
00052             double position{getElevatorPosition()};
00053             updateElevatorPosition(position - (distance - POLE_HANG_DISTANCE));
00054         }
00055     }
00056 }
```

#### 5.120.4.6 updateManual()

```
void wisco::user::elevator::ElevatorOperator::updateManual (
    EControllerDigital in,
    EControllerDigital out ) [private]
```

Updates the elevator position based on manual control.

**Parameters**

<i>in</i>	The digital control for moving the elevator in
<i>out</i>	The digital control for moving the elevator out

Definition at line 58 of file [ElevatorOperator.cpp](#).

```
00059 {
00060     bool move_in{m_controller->getDigital(in)};
00061     bool move_out{m_controller->getDigital(out)};
00062     if (move_in && move_out)
00063     {
00064         updateElevatorPosition(getElevatorPosition());
00065         manual_input = true;
00066     }
00067     else if (move_in)
00068     {
00069         updateElevatorPosition(IN_POSITION);
00070         manual_input = true;
00071     }
00072     else if (move_out)
00073     {
00074         updateElevatorPosition(PARTNER_HANG_POSITION);
00075         manual_input = true;
00076     }
00077     else if (manual_input)
00078     {
00079         updateElevatorPosition(getElevatorPosition());
00080         manual_input = false;
00081     }
00082 }
```

**5.120.4.7 updatePresetSplit()**

```
void wisco::user::elevator::ElevatorOperator::updatePresetSplit (
    EControllerDigital in,
    EControllerDigital field,
    EControllerDigital match_load,
    EControllerDigital pole_hang,
    EControllerDigital partner_hang ) [private]
```

Updates the elevator position based on preset split control.

**Parameters**

<i>in</i>	The digital control for the in position
<i>field</i>	The digital control for the field position
<i>match_load</i>	The digital control for the match load position
<i>pole_hang</i>	The digital control for the pole hang position
<i>partner_hang</i>	The digital control for the partner hang position

Definition at line 84 of file [ElevatorOperator.cpp](#).

```
00085 {
00086     bool move_in{m_controller->getNewDigital(in)};
00087     bool move_field{m_controller->getNewDigital(field)};
00088     bool move_match_load{m_controller->getNewDigital(match_load)};
00089     bool move_pole_hang{m_controller->getNewDigital(pole_hang)};
00090     bool move_partner_hang{m_controller->getNewDigital(partner_hang)};
00091
00092     if (move_in && !move_field && !move_match_load && !move_pole_hang && !move_partner_hang)
00093         updateElevatorPosition(IN_POSITION);
00094     else if (!move_in && move_field && !move_match_load && !move_pole_hang && !move_partner_hang)
00095         updateElevatorPosition(FIELD_POSITION);
00096     else if (!move_in && !move_field && move_match_load && !move_pole_hang && !move_partner_hang)
00097         updateElevatorPosition(MATCH_LOAD_POSITION);
00098     else if (!move_in && !move_field && !move_match_load && move_pole_hang && !move_partner_hang)
```

```

00099     updateElevatorPosition(POLE_HANG_POSITION);
00100 else if (!move_in && !move_field && !move_match_load && !move_pole_hang && move_partner_hang)
00101     updateElevatorPosition(PARTNER_HANG_POSITION);
00102 }
```

### 5.120.4.8 updatePresetToggle()

```
void wisco::user::elevator::ElevatorOperator::updatePresetToggle (
    EControllerDigital toggle) [private]
```

Updates the elevator position based on a toggle.

#### Parameters

<i>toggle</i>	The digital control for the toggle
---------------	------------------------------------

Definition at line 104 of file [ElevatorOperator.cpp](#).

```

00105 {
00106     if (m_controller->getNewDigital(toggle))
00107     {
00108         switch (toggle_state)
00109         {
00110             case EToggleState::IN:
00111                 updateElevatorPosition(FIELD_POSITION);
00112                 toggle_state = EToggleState::FIELD;
00113                 break;
00114             case EToggleState::FIELD:
00115                 updateElevatorPosition(MATCH_LOAD_POSITION);
00116                 toggle_state = EToggleState::MATCH_LOAD;
00117                 break;
00118             case EToggleState::MATCH_LOAD:
00119                 updateElevatorPosition(POLE_HANG_POSITION);
00120                 toggle_state = EToggleState::POLE_HANG;
00121                 break;
00122             case EToggleState::POLE_HANG:
00123                 updateElevatorPosition(PARTNER_HANG_POSITION);
00124                 toggle_state = EToggleState::PARTNER_HANG;
00125                 break;
00126             case EToggleState::PARTNER_HANG:
00127                 updateElevatorPosition(IN_POSITION);
00128                 toggle_state = EToggleState::IN;
00129                 break;
00130         }
00131     }
00132 }
```

### 5.120.4.9 updatePresetLadder()

```
void wisco::user::elevator::ElevatorOperator::updatePresetLadder (
    EControllerDigital in,
    EControllerDigital out) [private]
```

Updates the elevator position based on a ladder toggle system.

#### Parameters

<i>in</i>	The digital control for the next position inward
<i>out</i>	The digital control for the next position outward

Definition at line 134 of file [ElevatorOperator.cpp](#).

```

00135 {
00136     bool move_in{m_controller->getNewDigital(in)};
00137     bool move_out{m_controller->getNewDigital(out)};
```

```

00138
00139     if (move_in && !move_out)
00140     {
00141         switch (toggle_state)
00142         {
00143             case EToggleState::IN:
00144                 break;
00145             case EToggleState::FIELD:
00146                 updateElevatorPosition(IN_POSITION);
00147                 toggle_state = EToggleState::IN;
00148                 break;
00149             case EToggleState::MATCH_LOAD:
00150                 updateElevatorPosition(FIELD_POSITION);
00151                 toggle_state = EToggleState::FIELD;
00152                 break;
00153             case EToggleState::POLE_HANG:
00154                 updateElevatorPosition(MATCH_LOAD_POSITION);
00155                 toggle_state = EToggleState::MATCH_LOAD;
00156             case EToggleState::PARTNER_HANG:
00157                 updateElevatorPosition(POLE_HANG_POSITION);
00158                 toggle_state = EToggleState::POLE_HANG;
00159                 break;
00160         }
00161     }
00162     else if (!move_in && move_out)
00163     {
00164         switch (toggle_state)
00165         {
00166             case EToggleState::IN:
00167                 updateElevatorPosition(FIELD_POSITION);
00168                 toggle_state = EToggleState::FIELD;
00169                 break;
00170             case EToggleState::FIELD:
00171                 updateElevatorPosition(MATCH_LOAD_POSITION);
00172                 toggle_state = EToggleState::MATCH_LOAD;
00173                 break;
00174             case EToggleState::MATCH_LOAD:
00175                 updateElevatorPosition(POLE_HANG_POSITION);
00176                 toggle_state = EToggleState::POLE_HANG;
00177                 break;
00178             case EToggleState::POLE_HANG:
00179                 updateElevatorPosition(PARTNER_HANG_POSITION);
00180                 toggle_state = EToggleState::PARTNER_HANG;
00181                 break;
00182             case EToggleState::PARTNER_HANG:
00183                 break;
00184         }
00185     }
00186 }
```

#### 5.120.4.10 updatePresetLadderIntake()

```

void wisco::user::elevator::ElevatorOperator::updatePresetLadderIntake (
    EControllerDigital in,
    EControllerDigital out,
    EControllerDigital intake,
    EControllerDigital outtake ) [private]
```

Updates the elevator position based on a ladder toggle system with an intake override.

##### Parameters

<i>in</i>	The digital control for the next position inward
<i>out</i>	The digital control for the next position outward
<i>intake</i>	The digital control for the intake
<i>outtake</i>	The digital control for the outtake

Definition at line 188 of file [ElevatorOperator.cpp](#).

```

00189 {
00190     bool move_in{m_controller->getNewDigital(in)};
00191     bool move_out{m_controller->getNewDigital(out)};
00192     bool move_intake{m_controller->getNewDigital(intake)};
```

```

00193     bool move_outtake{m_controller->getNewDigital(outtake)};
00194
00195     if (move_intake && toggle_state == EToggleState::IN)
00196     {
00197         updateElevatorPosition(FIELD_POSITION);
00198         toggle_state = EToggleState::FIELD;
00199     }
00200     else if (move_outtake && toggle_state == EToggleState::FIELD)
00201     {
00202         updateElevatorPosition(IN_POSITION);
00203         toggle_state = EToggleState::IN;
00204     }
00205     else if (move_in && !move_out)
00206     {
00207         switch (toggle_state)
00208         {
00209             case EToggleState::IN:
00210                 break;
00211             case EToggleState::FIELD:
00212                 updateElevatorPosition(IN_POSITION);
00213                 toggle_state = EToggleState::IN;
00214                 break;
00215             case EToggleState::MATCH_LOAD:
00216                 updateElevatorPosition(FIELD_POSITION);
00217                 toggle_state = EToggleState::FIELD;
00218                 break;
00219             case EToggleState::POLE_HANG:
00220                 updateElevatorPosition(MATCH_LOAD_POSITION);
00221                 toggle_state = EToggleState::MATCH_LOAD;
00222                 break;
00223             case EToggleState::PARTNER_HANG:
00224                 updateElevatorPosition(POLE_HANG_POSITION);
00225                 toggle_state = EToggleState::POLE_HANG;
00226                 break;
00227         }
00228     }
00229     else if (!move_in && move_out)
00230     {
00231         switch (toggle_state)
00232         {
00233             case EToggleState::IN:
00234                 updateElevatorPosition(FIELD_POSITION);
00235                 toggle_state = EToggleState::FIELD;
00236                 break;
00237             case EToggleState::FIELD:
00238                 updateElevatorPosition(MATCH_LOAD_POSITION);
00239                 toggle_state = EToggleState::MATCH_LOAD;
00240                 break;
00241             case EToggleState::MATCH_LOAD:
00242                 updateElevatorPosition(POLE_HANG_POSITION);
00243                 toggle_state = EToggleState::POLE_HANG;
00244                 break;
00245             case EToggleState::POLE_HANG:
00246                 updateElevatorPosition(PARTNER_HANG_POSITION);
00247                 toggle_state = EToggleState::PARTNER_HANG;
00248                 break;
00249             case EToggleState::PARTNER_HANG:
00250                 break;
00251         }
00252     }
00253 }
```

#### 5.120.4.11 setElevatorPosition()

```
void wisco::user::elevator::ElevatorOperator::setElevatorPosition (
    const std::unique_ptr< IProfile > & profile )
```

Set the elevator position.

##### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 255 of file [ElevatorOperator.cpp](#).

```
00256 {
00257     EControllerDigital in{profile->getDigitalControlMapping(EControl::ELEVATOR_IN)};
```

```

00258     EControllerDigital field{profile->getDigitalControlMapping(EControl::ELEVATOR_FIELD)};
00259     EControllerDigital match_load{profile->getDigitalControlMapping(EControl::ELEVATOR_MATCH_LOAD)};
00260     EControllerDigital pole_hang{profile->getDigitalControlMapping(EControl::ELEVATOR_POLE_HANG)};
00261     EControllerDigital
00262         partner_hang{profile->getDigitalControlMapping(EControl::ELEVATOR_PARTNER_HANG)};
00263     EControllerDigital out{profile->getDigitalControlMapping(EControl::ELEVATOR_OUT)};
00264     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::ELEVATOR_TOGGLE)};
00265     EControllerDigital intake{profile->getDigitalControlMapping(EControl::INTAKE_IN)};
00266     EControllerDigital outtake{profile->getDigitalControlMapping(EControl::INTAKE_OUT)};
00267
00268     switch (static_cast<EElevatorControlMode>(profile->getControlMode(EControlType::ELEVATOR)))
00269     {
00270     case EElevatorControlMode::MANUAL:
00271         updateManual(in, out);
00272         break;
00273     case EElevatorControlMode::PRESET_SPLIT:
00274         updatePresetSplit(in, field, match_load, pole_hang, partner_hang);
00275         break;
00276     case EElevatorControlMode::PRESET_TOGGLE_LADDER:
00277         updatePresetLadder(in, out);
00278         break;
00279     case EElevatorControlMode::PRESET_TOGGLE_SINGLE:
00280         updatePresetToggle(toggle);
00281         break;
00282     case EElevatorControlMode::PRESET_TOGGLE_LADDER_INTAKE:
00283         updatePresetLadderIntake(in, out, intake, outtake);
00284         break;
00285     }
00286     if (toggle_state == EToggleState::POLE_HANG)
00287         updatePoleHangPosition();
00288 }

```

## 5.120.5 Member Data Documentation

### 5.120.5.1 ELEVATOR\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::elevator::ElevatorOperator::ELEVATOR_SUBSYSTEM_NAME[] {"ELEVATOR"}  
[static], [constexpr], [private]
```

The name of the elevator subsystem.

Definition at line 60 of file [ElevatorOperator.hpp](#).  
00060 {"ELEVATOR"};

### 5.120.5.2 HANG\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::elevator::ElevatorOperator::HANG_SUBSYSTEM_NAME[] {"HANG"} [static],  
[constexpr], [private]
```

The name of the hang subsystem.

Definition at line 66 of file [ElevatorOperator.hpp](#).  
00066 {"HANG"};

### 5.120.5.3 SET\_POSITION\_COMMAND

```
constexpr char wisco::user::elevator::ElevatorOperator::SET_POSITION_COMMAND[] {"SET POSITION"}  
[static], [constexpr], [private]
```

The command to set elevator position.

Definition at line 72 of file [ElevatorOperator.hpp](#).  
00072 {"SET POSITION"};

#### 5.120.5.4 GET\_POSITION\_STATE

```
constexpr char wisco::user::elevator::ElevatorOperator::GET_POSITION_STATE[ ] {"GET POSITION"}  
[static], [constexpr], [private]
```

The state to get elevator position.

Definition at line 78 of file [ElevatorOperator.hpp](#).

```
00078 {"GET POSITION"};
```

#### 5.120.5.5 CAP\_DISTANCE\_STATE\_NAME

```
constexpr char wisco::user::elevator::ElevatorOperator::CAP_DISTANCE_STATE_NAME[ ] {"CAP DISTANCE"}  
[static], [constexpr], [private]
```

The name of the cap distance state.

Definition at line 84 of file [ElevatorOperator.hpp](#).

```
00084 {"CAP DISTANCE"};
```

#### 5.120.5.6 HANG\_ARM\_UP\_STATE\_NAME

```
constexpr char wisco::user::elevator::ElevatorOperator::HANG_ARM_UP_STATE_NAME[ ] {"ARM UP"}  
[static], [constexpr], [private]
```

The name of the hang arm up state.

Definition at line 90 of file [ElevatorOperator.hpp](#).

```
00090 {"ARM UP"};
```

#### 5.120.5.7 IN\_POSITION

```
constexpr double wisco::user::elevator::ElevatorOperator::IN_POSITION {0.0} [static], [constexpr],  
[private]
```

The in position for the elevator.

Definition at line 96 of file [ElevatorOperator.hpp](#).

```
00096 {0.0};
```

#### 5.120.5.8 FIELD\_POSITION

```
constexpr double wisco::user::elevator::ElevatorOperator::FIELD_POSITION {3.0} [static],  
[constexpr], [private]
```

The field position for the elevator.

Definition at line 102 of file [ElevatorOperator.hpp](#).

```
00102 {3.0};
```

### 5.120.5.9 MATCH\_LOAD\_POSITION

```
constexpr double wisco::user::elevator::ElevatorOperator::MATCH_LOAD_POSITION {8.0} [static],  
[constexpr], [private]
```

The match loading position for the elevator.

Definition at line 108 of file [ElevatorOperator.hpp](#).  
00108 {8.0};

### 5.120.5.10 POLE\_HANG\_POSITION

```
constexpr double wisco::user::elevator::ElevatorOperator::POLE_HANG_POSITION {16.0} [static],  
[constexpr], [private]
```

The pole hang position for the elevator.

Definition at line 114 of file [ElevatorOperator.hpp](#).  
00114 {16.0};

### 5.120.5.11 PARTNER\_HANG\_POSITION

```
constexpr double wisco::user::elevator::ElevatorOperator::PARTNER_HANG_POSITION {18.0} [static],  
[constexpr], [private]
```

The partner hang position for the elevator.

Definition at line 120 of file [ElevatorOperator.hpp](#).  
00120 {18.0};

### 5.120.5.12 POLE\_HANG\_DISTANCE

```
constexpr double wisco::user::elevator::ElevatorOperator::POLE_HANG_DISTANCE {2.0} [static],  
[constexpr], [private]
```

The distance to the cap for a pole hang.

Definition at line 126 of file [ElevatorOperator.hpp](#).  
00126 {2.0};

### 5.120.5.13 CAP\_DETECTED\_DISTANCE

```
constexpr double wisco::user::elevator::ElevatorOperator::CAP_DETECTED_DISTANCE {5.0} [static],  
[constexpr], [private]
```

The distance within which a cap is considered detected.

Definition at line 132 of file [ElevatorOperator.hpp](#).  
00132 {5.0};

### 5.120.5.14 m\_controller

```
std::shared_ptr<user::IController> wisco::user::elevator::ElevatorOperator::m_controller {} [private]
```

The user input controller.

Definition at line 138 of file [ElevatorOperator.hpp](#).  
00138 {};

### 5.120.5.15 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::elevator::ElevatorOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 144 of file [ElevatorOperator.hpp](#).  
00144 {};

### 5.120.5.16 toggle\_state

```
EToggleState wisco::user::elevator::ElevatorOperator::toggle_state {EToggleState::IN} [private]
```

The state stored for toggle mode.

Definition at line 150 of file [ElevatorOperator.hpp](#).  
00150 {EToggleState::IN};

### 5.120.5.17 manual\_input

```
bool wisco::user::elevator::ElevatorOperator::manual_input {} [private]
```

Whether or not there is currently manual input.

Definition at line 156 of file [ElevatorOperator.hpp](#).  
00156 {};

## 5.121 wisco::user::hang::HangOperator Class Reference

Runs the operator-controlled hang settings.

### Public Member Functions

- **HangOperator** (const std::shared\_ptr< user::IController > &controller, const std::shared\_ptr< robot::Robot > &robot)  
*Construct a new Hang Operator object.*
- void **setHangState** (const std::unique\_ptr< IProfile > &profile)  
*Set the hang state.*

### Private Types

- enum class `EToggleState` { `INACTIVE` , `RAISED` , `GRABBED` , `HUNG` }
- The available states for hang toggles.*

### Private Member Functions

- void `closeClaw` ()  
*Closes the claw.*
- void `openClaw` ()  
*Opens the claw.*
- void `lowerArm` ()  
*Lowers the arm.*
- void `raiseArm` ()  
*Raises the arm.*
- void `engageWinch` ()  
*Engages the winch.*
- void `disengageWinch` ()  
*Disengages the winch.*
- void `setInactiveState` ()  
*Sets the hang to the inactive state.*
- void `setRaisedState` ()  
*Sets the hang to the raised state.*
- void `setGrabbedState` ()  
*Sets the hang to the grabbed state.*
- void `setHungState` ()  
*Sets the hang to the hung state.*
- void `updatePresetSplit` (`EControllerDigital` inactive, `EControllerDigital` raise, `EControllerDigital` grab, `EControllerDigital` hang)  
*Updates the hang state based on preset split control.*
- void `updatePresetToggle` (`EControllerDigital` toggle)  
*Updates the hang state based on a toggle.*
- void `updatePresetLadder` (`EControllerDigital` next, `EControllerDigital` previous)  
*Updates the hang state based on a ladder toggle system.*
- void `updatePresetReset` (`EControllerDigital` toggle, `EControllerDigital` reset)  
*Updates the hang state based on a toggle and a reset.*

### Private Attributes

- `std::shared_ptr< user::IController > m_controller {}`  
*The user input controller.*
- `std::shared_ptr< robot::Robot > m_robot {}`  
*The robot being controlled.*
- `EToggleState toggle_state {EToggleState::INACTIVE}`  
*The state stored for toggle mode.*

## Static Private Attributes

- static constexpr char `HANG_SUBSYSTEM_NAME` [] {"HANG"}  
*The name of the hang subsystem.*
- static constexpr char `CLOSE_CLAW_COMMAND_NAME` [] {"CLOSE CLAW"}  
*The name of the close claw command.*
- static constexpr char `OPEN_CLAW_COMMAND_NAME` [] {"OPEN CLAW"}  
*The name of the open claw command.*
- static constexpr char `LOWER_ARM_COMMAND_NAME` [] {"LOWER ARM"}  
*The name of the lower arm command.*
- static constexpr char `RAISE_ARM_COMMAND_NAME` [] {"RAISE ARM"}  
*The name of the raise arm command.*
- static constexpr char `ENGAGE_WINCH_COMMAND_NAME` [] {"ENGAGE WINCH"}  
*The name of the engage winch command.*
- static constexpr char `DISENGAGE_WINCH_COMMAND_NAME` [] {"DISENGAGE WINCH"}  
*The name of the disengage winch command.*

### 5.121.1 Detailed Description

Runs the operator-controlled hang settings.

#### Author

Nathan Sandvig

Definition at line 40 of file `HangOperator.hpp`.

### 5.121.2 Member Enumeration Documentation

#### 5.121.2.1 EToggleState

```
enum class wisco::user::hang::HangOperator::EToggleState [strong], [private]
```

The available states for hang toggles.

Definition at line 47 of file `HangOperator.hpp`.

```
00048     {
00049         INACTIVE,
00050         RAISED,
00051         GRABBED,
00052         HUNG
00053     };
```

### 5.121.3 Constructor & Destructor Documentation

#### 5.121.3.1 HangOperator()

```
wisco::user::hang::HangOperator::HangOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Hang Operator object.

**Parameters**

<i>controller</i>	The user input controller
<i>robot</i>	The robot to control

Definition at line 9 of file [HangOperator.cpp](#).

```
00011     : m_controller(controller), m_robot(robot)
00012 {
00013
00014 }
```

## 5.121.4 Member Function Documentation

### 5.121.4.1 closeClaw()

```
void wisco::user::hang::HangOperator::closeClaw () [private]
```

Closes the claw.

Definition at line 16 of file [HangOperator.cpp](#).

```
00017 {
00018     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, CLOSE_CLAW_COMMAND_NAME);
00019 }
```

### 5.121.4.2 openClaw()

```
void wisco::user::hang::HangOperator::openClaw () [private]
```

Opens the claw.

Definition at line 21 of file [HangOperator.cpp](#).

```
00022 {
00023     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, OPEN_CLAW_COMMAND_NAME);
00024 }
```

### 5.121.4.3 lowerArm()

```
void wisco::user::hang::HangOperator::lowerArm () [private]
```

Lowers the arm.

Definition at line 26 of file [HangOperator.cpp](#).

```
00027 {
00028     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, LOWER_ARM_COMMAND_NAME);
00029 }
```

### 5.121.4.4 raiseArm()

```
void wisco::user::hang::HangOperator::raiseArm () [private]
```

Raises the arm.

Definition at line 31 of file [HangOperator.cpp](#).

```
00032 {
00033     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, RAISE_ARM_COMMAND_NAME);
00034 }
```

#### 5.121.4.5 engageWinch()

```
void wisco::user::hang::HangOperator::engageWinch () [private]
```

Engages the winch.

Definition at line 36 of file [HangOperator.cpp](#).

```
00037 {  
00038     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, ENGAGE_WINCH_COMMAND_NAME);  
00039 }
```

#### 5.121.4.6 disengageWinch()

```
void wisco::user::hang::HangOperator::disengageWinch () [private]
```

Disengages the winch.

Definition at line 41 of file [HangOperator.cpp](#).

```
00042 {  
00043     m_robot->sendCommand(HANG_SUBSYSTEM_NAME, DISENGAGE_WINCH_COMMAND_NAME);  
00044 }
```

#### 5.121.4.7 setInactiveState()

```
void wisco::user::hang::HangOperator::setInactiveState () [private]
```

Sets the hang to the inactive state.

Definition at line 46 of file [HangOperator.cpp](#).

```
00047 {  
00048     toggle_state = EToggleState::INACTIVE;  
00049     openClaw();  
00050     lowerArm();  
00051     disengageWinch();  
00052 }
```

#### 5.121.4.8 setRaisedState()

```
void wisco::user::hang::HangOperator::setRaisedState () [private]
```

Sets the hang to the raised state.

Definition at line 54 of file [HangOperator.cpp](#).

```
00055 {  
00056     toggle_state = EToggleState::RAISED;  
00057     openClaw();  
00058     raiseArm();  
00059     disengageWinch();  
00060 }
```

#### 5.121.4.9 setGrabbedState()

```
void wisco::user::hang::HangOperator::setGrabbedState () [private]
```

Sets the hang to the grabbed state.

Definition at line 62 of file [HangOperator.cpp](#).

```
00063 {  
00064     toggle_state = EToggleState::GRABBED;  
00065     closeClaw();  
00066     raiseArm();  
00067     disengageWinch();  
00068 }
```

### 5.121.4.10 setHungState()

```
void wisco::user::hang::HangOperator::setHungState ( ) [private]
```

Sets the hang to the hung state.

Definition at line 70 of file [HangOperator.cpp](#).

```
00071 {
00072     toggle_state = EToggleState::HUNG;
00073     closeClaw();
00074     lowerArm();
00075     engageWinch();
00076 }
```

### 5.121.4.11 updatePresetSplit()

```
void wisco::user::hang::HangOperator::updatePresetSplit (
    EControllerDigital inactive,
    EControllerDigital raise,
    EControllerDigital grab,
    EControllerDigital hang ) [private]
```

Updates the hang state based on preset split control.

#### Parameters

<i>inactive</i>	The digital control for the inactive state
<i>raise</i>	The digital control for the raise state
<i>grab</i>	The digital control for the grab state
<i>hang</i>	The digital control for the hang state

Definition at line 78 of file [HangOperator.cpp](#).

```
00079 {
00080     bool set_inactive{m_controller->getNewDigital(inactive)};
00081     bool set_raise{m_controller->getNewDigital(raise)};
00082     bool set_grab{m_controller->getNewDigital(grab)};
00083     bool set_hang{m_controller->getNewDigital(hang)};
00084
00085     if (set_inactive && !set_raise && !set_grab && !set_hang)
00086         setInactiveState();
00087     else if (!set_inactive && set_raise && !set_grab && !set_hang)
00088         setRaisedState();
00089     else if (!set_inactive && !set_raise && set_grab && !set_hang)
00090         setGrabbedState();
00091     else if (!set_inactive && !set_raise && !set_grab && set_hang)
00092         setHungState();
00093 }
```

### 5.121.4.12 updatePresetToggle()

```
void wisco::user::hang::HangOperator::updatePresetToggle (
    EControllerDigital toggle ) [private]
```

Updates the hang state based on a toggle.

#### Parameters

<i>toggle</i>	The digital control for the toggle
---------------	------------------------------------

Definition at line 95 of file [HangOperator.cpp](#).

```
00096 {
00097     bool set_toggle{m_controller->getNewDigital(toggle)};
00098
00099     if (set_toggle)
00100     {
00101         switch (toggle_state)
00102         {
00103             case EToggleState::INACTIVE:
00104                 setRaisedState();
00105                 break;
00106             case EToggleState::RAISED:
00107                 setGrabbedState();
00108                 break;
00109             case EToggleState::GRABBED:
00110                 setHungState();
00111                 break;
00112             case EToggleState::HUNG:
00113                 setInactiveState();
00114                 break;
00115         }
00116     }
00117 }
```

#### 5.121.4.13 updatePresetLadder()

```
void wisco::user::hang::HangOperator::updatePresetLadder (
    EControllerDigital next,
    EControllerDigital previous ) [private]
```

Updates the hang state based on a ladder toggle system.

##### Parameters

<i>next</i>	The digital control for the next state
<i>previous</i>	The digital control for the next state

Definition at line 119 of file [HangOperator.cpp](#).

```
00120 {
00121     bool set_next{m_controller->getNewDigital(next)};
00122     bool set_previous{m_controller->getNewDigital(previous)};
00123
00124     if (set_next && !set_previous)
00125     {
00126         switch (toggle_state)
00127         {
00128             case EToggleState::INACTIVE:
00129                 setRaisedState();
00130                 break;
00131             case EToggleState::RAISED:
00132                 setGrabbedState();
00133                 break;
00134             case EToggleState::GRABBED:
00135                 setHungState();
00136                 break;
00137             case EToggleState::HUNG:
00138                 break;
00139         }
00140     }
00141     else if (!set_next && set_previous)
00142     {
00143         switch (toggle_state)
00144         {
00145             case EToggleState::INACTIVE:
00146                 break;
00147             case EToggleState::RAISED:
00148                 setInactiveState();
00149                 break;
00150             case EToggleState::GRABBED:
00151                 setRaisedState();
00152                 break;
00153             case EToggleState::HUNG:
00154                 setGrabbedState();
00155                 break;
```

```
00156         }
00157     }
00158 }
```

#### 5.121.4.14 updatePresetReset()

```
void wisco::user::hang::HangOperator::updatePresetReset (
    EControllerDigital toggle,
    EControllerDigital reset ) [private]
```

Updates the hang state based on a toggle and a reset.

##### Parameters

<i>toggle</i>	The digital control for the toggle
<i>reset</i>	The digital control for the reset

Definition at line 160 of file [HangOperator.cpp](#).

```
00161 {
00162     bool set_toggle{m_controller->getNewDigital(toggle)};
00163     bool set_reset{m_controller->getNewDigital(reset)};
00164
00165     if (set_toggle && !set_reset)
00166     {
00167         switch (toggle_state)
00168         {
00169             case EToggleState::INACTIVE:
00170                 setRaisedState();
00171                 break;
00172             case EToggleState::RAISED:
00173                 setGrabbedState();
00174                 break;
00175             case EToggleState::GRABBED:
00176                 setHungState();
00177                 break;
00178             case EToggleState::HUNG:
00179                 setInactiveState();
00180                 break;
00181         }
00182     }
00183     else if (!set_toggle && set_reset)
00184     {
00185         setInactiveState();
00186     }
00187 }
```

#### 5.121.4.15 setHangState()

```
void wisco::user::hang::HangOperator::setHangState (
    const std::unique_ptr< IProfile > & profile )
```

Set the hang state.

##### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 189 of file [HangOperator.cpp](#).

```
00190 {
00191     EControllerDigital grab{profile->getDigitalControlMapping(EControl::HANG_GRAB)};
00192     EControllerDigital hang{profile->getDigitalControlMapping(EControl::HANG_HANG)};
00193     EControllerDigital inactive{profile->getDigitalControlMapping(EControl::HANG_INACTIVE)};
00194     EControllerDigital next{profile->getDigitalControlMapping(EControl::HANG_NEXT)};
```

```

00195     EControllerDigital previous{profile->getDigitalControlMapping(EControl::HANG_PREVIOUS)};
00196     EControllerDigital raise{profile->getDigitalControlMapping(EControl::HANG_RAISE)};
00197     EControllerDigital reset{profile->getDigitalControlMapping(EControl::HANG_RESET)};
00198     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::HANG_TOGGLE)};
00199
00200     switch (static_cast<EHangControlMode>(profile->getControlMode(EControlType::HANG)))
00201     {
00202         case EHangControlMode::PRESET_SPLIT:
00203             updatePresetSplit(inactive, raise, grab, hang);
00204             break;
00205         case EHangControlMode::PRESET_TOGGLE_LADDER:
00206             updatePresetLadder(next, previous);
00207             break;
00208         case EHangControlMode::PRESET_TOGGLE_RESET:
00209             updatePresetReset(toggle, reset);
00210             break;
00211         case EHangControlMode::PRESET_TOGGLE_SINGLE:
00212             updatePresetToggle(toggle);
00213             break;
00214     }
00215 }
```

## 5.121.5 Member Data Documentation

### 5.121.5.1 HANG\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::hang::HangOperator::HANG_SUBSYSTEM_NAME[] {"HANG"} [static],
[constexpr], [private]
```

The name of the hang subsystem.

Definition at line 59 of file [HangOperator.hpp](#).  
00059 {"HANG"};

### 5.121.5.2 CLOSE\_CLAW\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::CLOSE_CLAW_COMMAND_NAME[] {"CLOSE CLAW"} [static],
[constexpr], [private]
```

The name of the close claw command.

Definition at line 65 of file [HangOperator.hpp](#).  
00065 {"CLOSE CLAW"};

### 5.121.5.3 OPEN\_CLAW\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::OPEN_CLAW_COMMAND_NAME[] {"OPEN CLAW"} [static],
[constexpr], [private]
```

The name of the open claw command.

Definition at line 71 of file [HangOperator.hpp](#).  
00071 {"OPEN CLAW"};

### 5.121.5.4 LOWER\_ARM\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::LOWER_ARM_COMMAND_NAME[] {"LOWER ARM"} [static],
[constexpr], [private]
```

The name of the lower arm command.

Definition at line 77 of file [HangOperator.hpp](#).  
00077 {"LOWER ARM"};

### 5.121.5.5 RAISE\_ARM\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::RAISE_ARM_COMMAND_NAME[ ] {"RAISE ARM"} [static],  
[constexpr], [private]
```

The name of the raise arm command.

Definition at line 83 of file [HangOperator.hpp](#).

```
00083 {"RAISE ARM"};
```

### 5.121.5.6 ENGAGE\_WINCH\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::ENGAGE_WINCH_COMMAND_NAME[ ] {"ENGAGE WINCH"}  
[static], [constexpr], [private]
```

The name of the engage winch command.

Definition at line 89 of file [HangOperator.hpp](#).

```
00089 {"ENGAGE WINCH"};
```

### 5.121.5.7 DISENGAGE\_WINCH\_COMMAND\_NAME

```
constexpr char wisco::user::hang::HangOperator::DISENGAGE_WINCH_COMMAND_NAME[ ] {"DISENGAGE  
WINCH"} [static], [constexpr], [private]
```

The name of the disengage winch command.

Definition at line 95 of file [HangOperator.hpp](#).

```
00095 {"DISENGAGE WINCH"};
```

### 5.121.5.8 m\_controller

```
std::shared_ptr<user::IController> wisco::user::hang::HangOperator::m_controller {} [private]
```

The user input controller.

Definition at line 101 of file [HangOperator.hpp](#).

```
00101 {};
```

### 5.121.5.9 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::hang::HangOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 107 of file [HangOperator.hpp](#).

```
00107 {};
```

### 5.121.5.10 toggle\_state

```
EToggleState wisco::user::hang::HangOperator::toggle_state {EToggleState::INACTIVE} [private]
```

The state stored for toggle mode.

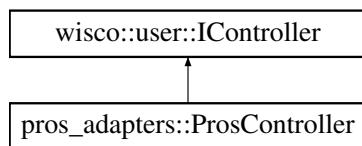
Definition at line 113 of file [HangOperator.hpp](#).

```
00113 {EToggleState::INACTIVE};
```

## 5.122 wisco::user::IController Class Reference

Interface for a controller.

Inheritance diagram for wisco::user::IController:



### Public Member Functions

- virtual ~**IController** ()=default  
*Destroy the **IController** object.*
- virtual void **initialize** ()=0  
*Initializes the controller.*
- virtual void **run** ()=0  
*Runs the controller.*
- virtual double **getAnalog** (EControllerAnalog analog\_channel)=0  
*Get the analog input of a channel from the controller.*
- virtual bool **getDigital** (EControllerDigital digital\_channel)=0  
*Get the digital input of a channel from the controller.*
- virtual bool **getNewDigital** (EControllerDigital digital\_channel)=0  
*Check for a new digital input of a channel from the controller.*
- virtual void **rumble** (std::string pattern)=0  
*Rumbles the controller.*

### 5.122.1 Detailed Description

Interface for a controller.

#### Author

Nathan Sandvig

Definition at line 30 of file [IController.hpp](#).

## 5.122.2 Member Function Documentation

### 5.122.2.1 initialize()

```
virtual void wisco::user::IController::initialize ( ) [pure virtual]
```

Initializes the controller.

Implemented in [pros\\_adapters::ProsController](#).

### 5.122.2.2 run()

```
virtual void wisco::user::IController::run ( ) [pure virtual]
```

Runs the controller.

Implemented in [pros\\_adapters::ProsController](#).

### 5.122.2.3 getAnalog()

```
virtual double wisco::user::IController::getAnalog ( EControllerAnalog analog_channel ) [pure virtual]
```

Get the analog input of a channel from the controller.

#### Parameters

<i>analog_channel</i>	The channel to read analog input from
-----------------------	---------------------------------------

#### Returns

`double` The value of the analog channel

Implemented in [pros\\_adapters::ProsController](#).

### 5.122.2.4 getDigital()

```
virtual bool wisco::user::IController::getDigital ( EControllerDigital digital_channel ) [pure virtual]
```

Get the digital input of a channel from the controller.

#### Parameters

<i>digital_channel</i>	The channel to read digital input from
------------------------	--

**Returns**

- true The digital channel is active
- false The digital channel is not active

Implemented in [pros\\_adapters::ProsController](#).

**5.122.2.5 getNewDigital()**

```
virtual bool wisco::user::IController::getNewDigital (
    EControllerDigital digital_channel ) [pure virtual]
```

Check for a new digital input of a channel from the controller.

**Parameters**

<i>digital_channel</i>	The channel to read digital input from
------------------------	--

**Returns**

- true The digital channel has a new input
- false The digital channel does not have a new input

Implemented in [pros\\_adapters::ProsController](#).

**5.122.2.6 rumble()**

```
virtual void wisco::user::IController::rumble (
    std::string pattern ) [pure virtual]
```

Rumbles the controller.

**Parameters**

<i>pattern</i>	The rumble pattern to follow Up to 8 characters, '.' short, '-' long, '' pause
----------------	--

Implemented in [pros\\_adapters::ProsController](#).

**5.123 wisco::user::intake::IntakeOperator Class Reference**

Runs the operator-controlled intake voltage settings.

**Public Member Functions**

- **IntakeOperator** (const std::shared\_ptr<[user::IController](#)> &controller, const std::shared\_ptr<[robot::Robot](#)> &robot)  
*Construct a new Intake Operator object.*
- void **setIntakeVoltage** (const std::unique\_ptr<[IProfile](#)> &profile)  
*Set the intake voltage.*

### Private Types

- enum class `EToggleState` { `OFF` , `IN` , `OUT` }

*The available states for intake toggles.*

### Private Member Functions

- void `updateIntakeVoltage` (double voltage)  
*Updates the voltage of the intake subsystem.*
- void `updateToggleVoltage` ()  
*Updates the intake voltage based on toggle state.*
- void `updateSingleToggle` (`EControllerDigital` toggle)  
*Update the voltage for single button toggle.*
- void `updateSplitHold` (`EControllerDigital` in, `EControllerDigital` out)  
*Update the voltage for split button hold.*
- void `updateSplitToggle` (`EControllerDigital` in, `EControllerDigital` out)  
*Update the voltage for split button toggle.*

### Private Attributes

- `std::shared_ptr< user::IController > m_controller {}`  
*The user input controller.*
- `std::shared_ptr< robot::Robot > m_robot {}`  
*The robot being controlled.*
- `EToggleState toggle_state {EToggleState::OFF}`  
*The state stored for toggle mode.*

### Static Private Attributes

- static constexpr char `INTAKE_SUBSYSTEM_NAME` [] {"INTAKE"}  
*The name of the intake subsystem.*
- static constexpr char `SET_VOLTAGE_COMMAND` [] {"SET VOLTAGE"}  
*The command to set intake voltage.*
- static constexpr double `VOLTAGE_SETTING` {12.0}  
*The voltage to run the intake at.*

## 5.123.1 Detailed Description

Runs the operator-controlled intake voltage settings.

### Author

Nathan Sandvig

Definition at line 40 of file `IntakeOperator.hpp`.

## 5.123.2 Member Enumeration Documentation

### 5.123.2.1 EToggleState

```
enum class wisco::user::intake::IntakeOperator::EToggleState [strong], [private]
```

The available states for intake toggles.

Definition at line 47 of file [IntakeOperator.hpp](#).

```
00048     {
00049         OFF,
00050         IN,
00051         OUT
00052     };
```

## 5.123.3 Constructor & Destructor Documentation

### 5.123.3.1 IntakeOperator()

```
wisco::user::intake::IntakeOperator::IntakeOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Intake Operator object.

Parameters

<i>controller</i>	The user input controller
<i>robot</i>	The robot to control

Definition at line 76 of file [IntakeOperator.cpp](#).

```
00078     : m_controller(controller), m_robot(robot)
00079 {
00080
00081 }
```

## 5.123.4 Member Function Documentation

### 5.123.4.1 updateIntakeVoltage()

```
void wisco::user::intake::IntakeOperator::updateIntakeVoltage (
    double voltage) [private]
```

Updates the voltage of the intake subsystem.

Parameters

<i>voltage</i>	The intake voltage
----------------	--------------------

Definition at line 9 of file [IntakeOperator.cpp](#).

```
00010 {
00011     m_robot->sendCommand(INTAKE_SUBSYSTEM_NAME, SET_VOLTAGE_COMMAND, voltage);
00012 }
```

### 5.123.4.2 updateToggleVoltage()

```
void wisco::user::intake::IntakeOperator::updateToggleVoltage ( ) [private]
```

Updates the intake voltage based on toggle state.

Definition at line 14 of file [IntakeOperator.cpp](#).

```
00015 {
00016     switch(toggle_state)
00017     {
00018         case EToggleState::OFF:
00019             updateIntakeVoltage(VOLTAGE_SETTING);
00020             break;
00021         case EToggleState::IN:
00022             updateIntakeVoltage(-VOLTAGE_SETTING);
00023             break;
00024         case EToggleState::OUT:
00025             updateIntakeVoltage(0);
00026             break;
00027     }
00028 }
```

### 5.123.4.3 updateSingleToggle()

```
void wisco::user::intake::IntakeOperator::updateSingleToggle (
    EControllerDigital toggle ) [private]
```

Update the voltage for single button toggle.

Definition at line 30 of file [IntakeOperator.cpp](#).

```
00031 {
00032     if (m_controller->getNewDigital(toggle))
00033     {
00034         switch(toggle_state)
00035         {
00036             case EToggleState::OFF:
00037                 toggle_state = EToggleState::IN;
00038                 break;
00039             case EToggleState::IN:
00040                 toggle_state = EToggleState::OUT;
00041                 break;
00042             case EToggleState::OUT:
00043                 toggle_state = EToggleState::OFF;
00044                 break;
00045         }
00046         updateToggleVoltage();
00047     }
00048 }
```

### 5.123.4.4 updateSplitHold()

```
void wisco::user::intake::IntakeOperator::updateSplitHold (
    EControllerDigital in,
    EControllerDigital out ) [private]
```

Update the voltage for split button hold.

Definition at line 50 of file [IntakeOperator.cpp](#).

```
00051 {
00052     double voltage{ (m_controller->getDigital(in) - m_controller->getDigital(out)) * VOLTAGE_SETTING};
00053     updateIntakeVoltage(voltage);
00054 }
```

### 5.123.4.5 updateSplitToggle()

```
void wisco::user::intake::IntakeOperator::updateSplitToggle (
    EControllerDigital in,
    EControllerDigital out ) [private]
```

Update the voltage for split button toggle.

Definition at line 56 of file [IntakeOperator.cpp](#).

```
00057 {
00058     if (m_controller->getNewDigital(in))
00059     {
00060         if (toggle_state == EToggleState::IN)
00061             toggle_state = EToggleState::OFF;
00062         else
00063             toggle_state = EToggleState::IN;
00064         updateToggleVoltage();
00065     }
00066     else if (m_controller->getNewDigital(out))
00067     {
00068         if (toggle_state == EToggleState::OUT)
00069             toggle_state = EToggleState::OFF;
00070         else
00071             toggle_state = EToggleState::OUT;
00072         updateToggleVoltage();
00073     }
00074 }
```

### 5.123.4.6 setIntakeVoltage()

```
void wisco::user::intake::IntakeOperator::setIntakeVoltage (
    const std::unique_ptr< IProfile > & profile )
```

Set the intake voltage.

#### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 83 of file [IntakeOperator.cpp](#).

```
00084 {
00085     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::INTAKE_TOGGLE)};
00086     EControllerDigital in{profile->getDigitalControlMapping(EControl::INTAKE_IN)};
00087     EControllerDigital out{profile->getDigitalControlMapping(EControl::INTAKE_OUT)};
00088     switch(static_cast<EIntakeControlMode>(profile->getControlMode(EControlType::INTAKE)))
00089     {
00090         case EIntakeControlMode::SINGLE_TOGGLE:
00091             updateSingleToggle(toggle);
00092             break;
00093         case EIntakeControlMode::SPLIT_HOLD:
00094             updateSplitHold(in, out);
00095             break;
00096         case EIntakeControlMode::SPLIT_TOGGLE:
00097             updateSplitToggle(in, out);
00098             break;
00099     }
00100 }
```

## 5.123.5 Member Data Documentation

### 5.123.5.1 INTAKE\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::intake::IntakeOperator::INTAKE_SUBSYSTEM_NAME[] {"INTAKE"} [static],
[constexpr], [private]
```

The name of the intake subsystem.

Definition at line 58 of file [IntakeOperator.hpp](#).  
00058 {"INTAKE"};

### 5.123.5.2 SET\_VOLTAGE\_COMMAND

```
constexpr char wisco::user::intake::IntakeOperator::SET_VOLTAGE_COMMAND[] {"SET VOLTAGE"}  
[static], [constexpr], [private]
```

The command to set intake voltage.

Definition at line 64 of file [IntakeOperator.hpp](#).  
00064 {"SET VOLTAGE"};

### 5.123.5.3 VOLTAGE\_SETTING

```
constexpr double wisco::user::intake::IntakeOperator::VOLTAGE_SETTING (12.0) [static], [constexpr],  
[private]
```

The voltage to run the intake at.

Definition at line 70 of file [IntakeOperator.hpp](#).  
00070 {12.0};

### 5.123.5.4 m\_controller

```
std::shared_ptr<user::IController> wisco::user::intake::IntakeOperator::m_controller {} [private]
```

The user input controller.

Definition at line 76 of file [IntakeOperator.hpp](#).  
00076 {};

### 5.123.5.5 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::intake::IntakeOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 82 of file [IntakeOperator.hpp](#).  
00082 {};

### 5.123.5.6 toggle\_state

```
EToggleState wisco::user::intake::IntakeOperator::toggle_state {EToggleState::OFF} [private]
```

The state stored for toggle mode.

Definition at line 88 of file [IntakeOperator.hpp](#).  
00088 {EToggleState::OFF};

## 5.124 wisco::user::loader::LoaderOperator Class Reference

Runs the operator-controlled loader settings.

### Public Member Functions

- `LoaderOperator` (const std::shared\_ptr<`user::IController`> &controller, const std::shared\_ptr<`robot::Robot`> &robot)  
*Construct a new Loader Operator object.*
- void `setLoaderPosition` (const std::unique\_ptr<`IProfile`> &profile)  
*Set the loader position.*

### Private Types

- enum class `EToggleState` { `LOAD` , `READY` }

*The available states for loader toggles.*

### Private Member Functions

- void `doLoad` ()  
*Sends a do load command to the loader.*
- void `doReady` ()  
*Sends a do ready command to the loader.*
- bool `isLoaded` ()  
*Gets the is loaded state from the loader.*
- bool `isReady` ()  
*Gets the is ready state from the loader.*
- void `updateHold` (`EControllerDigital` hold)  
*Updates the loader position based on a hold.*
- void `updateMacro` (`EControllerDigital` toggle)  
*Updates the loader position based on a macro.*
- void `updateSingleToggle` (`EControllerDigital` toggle)  
*Updates the loader position based on a single toggle.*
- void `updateSplitToggle` (`EControllerDigital` load, `EControllerDigital` ready)  
*Updates the loader position based on a split toggle.*

### Private Attributes

- std::shared\_ptr<`user::IController`> `m_controller` {}  
*The user input controller.*
- std::shared\_ptr<`robot::Robot`> `m_robot` {}  
*The robot being controlled.*
- `EToggleState` `toggle_state` {`EToggleState::READY`}  
*The state stored for toggle mode.*

## Static Private Attributes

- static constexpr char **SUBSYSTEM\_NAME** [] {"LOADER"}  
*The name of the subsystem.*
- static constexpr char **DO\_LOAD\_COMMAND\_NAME** [] {"DO LOAD"}  
*The name of the do load command.*
- static constexpr char **DO\_READY\_COMMAND\_NAME** [] {"DO READY"}  
*The name of the do ready command.*
- static constexpr char **IS\_LOADED\_STATE\_NAME** [] {"IS LOADED"}  
*The name of the is loaded command.*
- static constexpr char **IS\_READY\_STATE\_NAME** [] {"IS READY"}  
*The name of the is ready command.*

## 5.124.1 Detailed Description

Runs the operator-controlled loader settings.

### Author

Nathan Sandvig

Definition at line 40 of file [LoaderOperator.hpp](#).

## 5.124.2 Member Enumeration Documentation

### 5.124.2.1 EToggleState

```
enum class wisco::user::loader::LoaderOperator::EToggleState [strong], [private]
```

The available states for loader toggles.

Definition at line 47 of file [LoaderOperator.hpp](#).

```
00048     {
00049         LOAD,
00050         READY
00051     };
```

## 5.124.3 Constructor & Destructor Documentation

### 5.124.3.1 LoaderOperator()

```
wisco::user::loader::LoaderOperator::LoaderOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Loader Operator object.

#### Parameters

<i>controller</i>	The user input controller
<i>robot</i>	The robot to control

Definition at line 9 of file [LoaderOperator.cpp](#).

```
00011     : m_controller(controller),
00012     m_robot(robot)
00013 {
00014
00015 }
```

## 5.124.4 Member Function Documentation

### 5.124.4.1 doLoad()

```
void wisco::user::loader::LoaderOperator::doLoad ( ) [private]
```

Sends a do load command to the loader.

Definition at line 17 of file [LoaderOperator.cpp](#).

```
00018 {
00019     m_robot->sendCommand(SUBSYSTEM_NAME, DO_LOAD_COMMAND_NAME);
00020 }
```

### 5.124.4.2 doReady()

```
void wisco::user::loader::LoaderOperator::doReady ( ) [private]
```

Sends a do ready command to the loader.

Definition at line 22 of file [LoaderOperator.cpp](#).

```
00023 {
00024     m_robot->sendCommand(SUBSYSTEM_NAME, DO_READY_COMMAND_NAME);
00025 }
```

### 5.124.4.3 isLoaded()

```
bool wisco::user::loader::LoaderOperator::isLoaded ( ) [private]
```

Gets the is loaded state from the loader.

#### Returns

true The loader is loaded

false The loader is not loaded

Definition at line 27 of file [LoaderOperator.cpp](#).

```
00028 {
00029     bool* result{static_cast<bool*>(m_robot->getState(SUBSYSTEM_NAME, IS_LOADED_STATE_NAME))};
00030     bool loaded{*result};
00031     delete result;
00032     return loaded;
00033 }
```

#### 5.124.4.4 isReady()

```
bool wisco::user::loader::LoaderOperator::isReady ( ) [private]
```

Gets the is ready state from the loader.

##### Returns

- true The loader is ready
- false The loader is not ready

Definition at line 35 of file [LoaderOperator.cpp](#).

```
00036 {
00037     bool* result{static_cast<bool*>(m_robot->getState(SUBSYSTEM_NAME, IS_READY_STATE_NAME))};
00038     bool ready{*result};
00039     delete result;
00040     return ready;
00041 }
```

#### 5.124.4.5 updateHold()

```
void wisco::user::loader::LoaderOperator::updateHold (
    EControllerDigital hold ) [private]
```

Updates the loader position based on a hold.

##### Parameters

<i>hold</i>	The hold digital input
-------------	------------------------

Definition at line 43 of file [LoaderOperator.cpp](#).

```
00044 {
00045     bool set_hold{m_controller->getDigital(hold)};
00046
00047     if (set_hold)
00048         doLoad();
00049     else
00050         doReady();
00051 }
```

#### 5.124.4.6 updateMacro()

```
void wisco::user::loader::LoaderOperator::updateMacro (
    EControllerDigital toggle ) [private]
```

Updates the loader position based on a macro.

##### Parameters

<i>toggle</i>	The start toggle digital input
---------------	--------------------------------

Definition at line 53 of file [LoaderOperator.cpp](#).

```
00054 {
00055     bool set_toggle{m_controller->getNewDigital(toggle)};
00056 }
```

```

00057     if (set_toggle && isReady())
00058         doLoad();
00059     else if (isLoaded())
00060         doReady();
00061 }
```

#### 5.124.4.7 updateSingleToggle()

```
void wisco::user::loader::LoaderOperator::updateSingleToggle (
    EControllerDigital toggle ) [private]
```

Updates the loader position based on a single toggle.

##### Parameters

<i>toggle</i>	The toggle digital input
---------------	--------------------------

Definition at line 63 of file [LoaderOperator.cpp](#).

```

00064 {
00065     bool set_toggle{m_controller->getNewDigital(toggle)};
00066
00067     if (set_toggle)
00068     {
00069         switch (toggle_state)
00070         {
00071             case EToggleState::LOAD:
00072                 toggle_state = EToggleState::READY;
00073                 doReady();
00074                 break;
00075             case EToggleState::READY:
00076                 toggle_state = EToggleState::LOAD;
00077                 doLoad();
00078                 break;
00079         }
00080     }
00081 }
```

#### 5.124.4.8 updateSplitToggle()

```
void wisco::user::loader::LoaderOperator::updateSplitToggle (
    EControllerDigital load,
    EControllerDigital ready ) [private]
```

Updates the loader position based on a split toggle.

##### Parameters

<i>load</i>	The load digital input
<i>ready</i>	The ready digital input

Definition at line 83 of file [LoaderOperator.cpp](#).

```

00084 {
00085     bool set_load{m_controller->getNewDigital(load)};
00086     bool set_ready{m_controller->getNewDigital(ready)};
00087
00088     if (set_load && !set_ready)
00089     {
00090         toggle_state = EToggleState::LOAD;
00091         doLoad();
00092     }
00093     else if (!set_load && set_ready)
00094     {
00095         toggle_state = EToggleState::READY;
```

```
00096     doReady();
00097 }
00098 }
```

#### 5.124.4.9 setLoaderPosition()

```
void wisco::user::loader::LoaderOperator::setLoaderPosition (
    const std::unique_ptr< IProfile > & profile )
```

Set the loader position.

##### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 100 of file [LoaderOperator.cpp](#).

```
00101 {
00102     EControllerDigital hold{profile->getDigitalControlMapping(EControl::LOADER_HOLD)};
00103     EControllerDigital load{profile->getDigitalControlMapping(EControl::LOADER_LOAD)};
00104     EControllerDigital ready{profile->getDigitalControlMapping(EControl::LOADER_READY)};
00105     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::LOADER_TOGGLE)};
00106
00107     switch (static_cast<ELoaderControlMode>(profile->getControlMode(EControlType::LOADER)))
00108     {
00109         case ELoaderControlMode::HOLD:
00110             updateHold(hold);
00111             break;
00112         case ELoaderControlMode::MACRO:
00113             updateMacro(toggle);
00114             break;
00115         case ELoaderControlMode::SINGLE_TOGGLE:
00116             updateSingleToggle(toggle);
00117             break;
00118         case ELoaderControlMode::SPLIT_TOGGLE:
00119             updateSplitToggle(load, ready);
00120             break;
00121     }
00122 }
```

## 5.124.5 Member Data Documentation

### 5.124.5.1 SUBSYSTEM\_NAME

```
constexpr char wisco::user::loader::LoaderOperator::SUBSYSTEM_NAME[] {"LOADER"} [static],
[constexpr], [private]
```

The name of the subsystem.

Definition at line 57 of file [LoaderOperator.hpp](#).

```
00057 {"LOADER"};
```

### 5.124.5.2 DO\_LOAD\_COMMAND\_NAME

```
constexpr char wisco::user::loader::LoaderOperator::DO_LOAD_COMMAND_NAME[] {"DO LOAD"} [static],
[constexpr], [private]
```

The name of the do load command.

Definition at line 63 of file [LoaderOperator.hpp](#).

```
00063 {"DO LOAD"};
```

### 5.124.5.3 DO\_READY\_COMMAND\_NAME

```
constexpr char wisco::user::loader::LoaderOperator::DO_READY_COMMAND_NAME[] {"DO READY"} [static],  
[constexpr], [private]
```

The name of the do ready command.

Definition at line 69 of file [LoaderOperator.hpp](#).

```
00069 {"DO READY"};
```

### 5.124.5.4 IS\_LOADED\_STATE\_NAME

```
constexpr char wisco::user::loader::LoaderOperator::IS_LOADED_STATE_NAME[] {"IS LOADED"} [static],  
[constexpr], [private]
```

The name of the is loaded command.

Definition at line 75 of file [LoaderOperator.hpp](#).

```
00075 {"IS LOADED"};
```

### 5.124.5.5 IS\_READY\_STATE\_NAME

```
constexpr char wisco::user::loader::LoaderOperator::IS_READY_STATE_NAME[] {"IS READY"} [static],  
[constexpr], [private]
```

The name of the is ready command.

Definition at line 81 of file [LoaderOperator.hpp](#).

```
00081 {"IS READY"};
```

### 5.124.5.6 m\_controller

```
std::shared_ptr<user::IController> wisco::user::loader::LoaderOperator::m_controller {} [private]
```

The user input controller.

Definition at line 87 of file [LoaderOperator.hpp](#).

```
00087 {};
```

### 5.124.5.7 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::loader::LoaderOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 93 of file [LoaderOperator.hpp](#).

```
00093 {};
```

### 5.124.5.8 toggle\_state

```
EToggleState wisco::user::loader::LoaderOperator::toggle_state {EToggleState::READY} [private]
```

The state stored for toggle mode.

Definition at line 99 of file [LoaderOperator.hpp](#).  
00099 {EToggleState::READY};

## 5.125 wisco::user::umbrella::UmbrellaOperator Class Reference

Runs the operator-controlled umbrella settings.

### Public Member Functions

- **UmbrellaOperator** (const std::shared\_ptr< user::IController > &controller, const std::shared\_ptr< robot::Robot > &robot)  
*Construct a new Umbrella Operator object.*
- void **setUmbrellaPosition** (const std::unique\_ptr< IProfile > &profile)  
*Set the umbrella position.*

### Private Types

- enum class **EToggleState** { **IN** , **OUT** }  
*The available states for umbrella toggles.*

### Private Member Functions

- void **setIn** ()  
*Sets the umbrella in.*
- void **setOut** ()  
*Sets the umbrella out.*
- void **updateHold** (EControllerDigital hold)  
*Updates the umbrella position based on a hold.*
- void **updateSplit** (EControllerDigital in, EControllerDigital out)  
*Updates the umbrella position based on a split.*
- void **updateToggle** (EControllerDigital toggle)  
*Updates the umbrella position based on a toggle.*

### Private Attributes

- std::shared\_ptr< user::IController > **m\_controller** {}  
*The user input controller.*
- std::shared\_ptr< robot::Robot > **m\_robot** {}  
*The robot being controlled.*
- **EToggleState toggle\_state** {EToggleState::IN}  
*The state stored for toggle mode.*

## Static Private Attributes

- static constexpr char `UMBRELLA_SUBSYSTEM_NAME` [] {"UMBRELLA"}  
*The name of the umbrella subsystem.*
- static constexpr char `SET_IN_COMMAND` [] {"SET IN"}  
*The command to set in.*
- static constexpr char `SET_OUT_COMMAND` [] {"SET OUT"}  
*The command to set out.*

### 5.125.1 Detailed Description

Runs the operator-controlled umbrella settings.

#### Author

Nathan Sandvig

Definition at line 40 of file [UmbrellaOperator.hpp](#).

### 5.125.2 Member Enumeration Documentation

#### 5.125.2.1 EToggleState

```
enum class wisco::user::umbrella::UmbrellaOperator::EToggleState [strong], [private]
```

The available states for umbrella toggles.

Definition at line 47 of file [UmbrellaOperator.hpp](#).

```
00048     {
00049         IN,
00050         OUT
00051     };
```

### 5.125.3 Constructor & Destructor Documentation

#### 5.125.3.1 UmbrellaOperator()

```
wisco::user::umbrella::UmbrellaOperator::UmbrellaOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Umbrella Operator object.

#### Parameters

<code>controller</code>	The user input controller
<code>robot</code>	The robot to control

Definition at line 9 of file [UmbrellaOperator.cpp](#).

```
00011     : m_controller(controller),
00012     m_robot(robot)
00013 {
00014
00015 }
```

## 5.125.4 Member Function Documentation

### 5.125.4.1 setIn()

```
void wisco::user::umbrella::UmbrellaOperator::setIn ( ) [private]
```

Sets the umbrella in.

Definition at line 17 of file [UmbrellaOperator.cpp](#).

```
00018 {
00019     toggle_state = EToggleState::IN;
00020     m_robot->sendCommand(UMBRELLA_SUBSYSTEM_NAME, SET_IN_COMMAND);
00021 }
```

### 5.125.4.2 setOut()

```
void wisco::user::umbrella::UmbrellaOperator::setOut ( ) [private]
```

Sets the umbrella out.

Definition at line 23 of file [UmbrellaOperator.cpp](#).

```
00024 {
00025     toggle_state = EToggleState::OUT;
00026     m_robot->sendCommand(UMBRELLA_SUBSYSTEM_NAME, SET_OUT_COMMAND);
00027 }
```

### 5.125.4.3 updateHold()

```
void wisco::user::umbrella::UmbrellaOperator::updateHold (
    EControllerDigital hold) [private]
```

Updates the umbrella position based on a hold.

#### Parameters

<i>hold</i>	The hold digital input
-------------	------------------------

Definition at line 29 of file [UmbrellaOperator.cpp](#).

```
00030 {
00031     bool set_hold{m_controller->getDigital(hold)};
00032
00033     if (set_hold)
00034         setOut();
00035     else
00036         setIn();
00037 }
```

### 5.125.4.4 updateSplit()

```
void wisco::user::umbrella::UmbrellaOperator::updateSplit (
```

```
EControllerDigital in,
EControllerDigital out ) [private]
```

Updates the umbrella position based on a split.

#### Parameters

<i>in</i>	The in digital input
<i>out</i>	The out digital input

Definition at line 39 of file [UmbrellaOperator.cpp](#).

```
00040 {
00041     bool set_in{m_controller->getNewDigital(in)};
00042     bool set_out{m_controller->getNewDigital(out)};
00043
00044     if (set_in && !set_out)
00045         setIn();
00046     else if (!set_in && set_out)
00047         setOut();
00048 }
```

#### 5.125.4.5 updateToggle()

```
void wisco::user::umbrella::UmbrellaOperator::updateToggle (
    EControllerDigital toggle ) [private]
```

Updates the umbrella position based on a toggle.

#### Parameters

<i>toggle</i>	The toggle digital input
---------------	--------------------------

Definition at line 50 of file [UmbrellaOperator.cpp](#).

```
00051 {
00052     bool set_toggle{m_controller->getNewDigital(toggle)};
00053
00054     if (set_toggle)
00055     {
00056         switch (toggle_state)
00057         {
00058             case EToggleState::IN:
00059                 setOut();
00060                 break;
00061             case EToggleState::OUT:
00062                 setIn();
00063                 break;
00064         }
00065     }
00066 }
```

#### 5.125.4.6 setUmbrellaPosition()

```
void wisco::user::umbrella::UmbrellaOperator::setUmbrellaPosition (
    const std::unique_ptr< IProfile > & profile )
```

Set the umbrella position.

#### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 68 of file [UmbrellaOperator.cpp](#).

```
00069 {
00070     EControllerDigital hold{profile->getDigitalControlMapping(EControl::UMBRELLA_HOLD)};
00071     EControllerDigital in{profile->getDigitalControlMapping(EControl::UMBRELLA_IN)};
00072     EControllerDigital out{profile->getDigitalControlMapping(EControl::UMBRELLA_OUT)};
00073     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::UMBRELLA_TOGGLE)};
00074
00075     switch (static_cast<EUmbrellaControlMode>(profile->getControlMode(EControlType::UMBRELLA)))
00076     {
00077         case EUmbrellaControlMode::HOLD:
00078             updateHold(hold);
00079             break;
00080         case EUmbrellaControlMode::SINGLE_TOGGLE:
00081             updateToggle(toggle);
00082             break;
00083         case EUmbrellaControlMode::SPLIT_TOGGLE:
00084             updateSplit(in, out);
00085             break;
00086     }
00087 }
```

## 5.125.5 Member Data Documentation

### 5.125.5.1 UMBRELLA\_SUBSYSTEM\_NAME

constexpr char wisco::user::umbrella::UmbrellaOperator::UMBRELLA\_SUBSYSTEM\_NAME[] {"UMBRELLA"}  
 [static], [constexpr], [private]

The name of the umbrella subsystem.

Definition at line 57 of file [UmbrellaOperator.hpp](#).

```
00057 {"UMBRELLA"};
```

### 5.125.5.2 SET\_IN\_COMMAND

constexpr char wisco::user::umbrella::UmbrellaOperator::SET\_IN\_COMMAND[] {"SET IN"} [static],  
 [constexpr], [private]

The command to set in.

Definition at line 63 of file [UmbrellaOperator.hpp](#).

```
00063 {"SET IN"};
```

### 5.125.5.3 SET\_OUT\_COMMAND

constexpr char wisco::user::umbrella::UmbrellaOperator::SET\_OUT\_COMMAND[] {"SET OUT"} [static],  
 [constexpr], [private]

The command to set out.

Definition at line 69 of file [UmbrellaOperator.hpp](#).

```
00069 {"SET OUT"};
```

### 5.125.5.4 m\_controller

std::shared\_ptr<[user::IController](#)> wisco::user::umbrella::UmbrellaOperator::m\_controller {}  
 [private]

The user input controller.

Definition at line 75 of file [UmbrellaOperator.hpp](#).

```
00075 {};
```

### 5.125.5.5 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::umbrella::UmbrellaOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 81 of file [UmbrellaOperator.hpp](#).

```
00081 {};
```

### 5.125.5.6 toggle\_state

```
EToggleState wisco::user::umbrella::UmbrellaOperator::toggle_state {EToggleState::IN} [private]
```

The state stored for toggle mode.

Definition at line 87 of file [UmbrellaOperator.hpp](#).

```
00087 {EToggleState::IN};
```

## 5.126 wisco::user::wings::WingsOperator Class Reference

Runs the operator-controlled wings settings.

### Public Member Functions

- `WingsOperator` (const std::shared\_ptr< user::IController > &controller, const std::shared\_ptr< robot::Robot > &robot)
   
*Construct a new Wings Operator object.*
- void `setWingsPosition` (const std::unique\_ptr< IProfile > &profile)
   
*Set the wings position.*

### Private Types

- enum class `EToggleState` { **IN** , **OUT** }
   
*The available states for wings toggles.*

### Private Member Functions

- void `setLeftWing` (bool out)
   
*Sets the position of the left wing.*
- void `setRightWing` (bool out)
   
*Sets the position of the right wing.*
- void `setWings` (bool out)
   
*Set the wings to a specific position.*
- void `updateDualHold` (`EControllerDigital` hold)
   
*Updates the wing position based on a combined hold.*
- void `updateDualSplit` (`EControllerDigital` in, `EControllerDigital` out)
   
*Updates the wing position based on a combined split.*
- void `updateDualToggle` (`EControllerDigital` toggle)
   
*Updates the wing position based on a combined toggle.*
- void `updateSingleHold` (`EControllerDigital` left\_hold, `EControllerDigital` right\_hold)
   
*Updates the wing position based on a separate hold.*
- void `updateSingleSplit` (`EControllerDigital` left\_in, `EControllerDigital` left\_out, `EControllerDigital` right\_in, `EControllerDigital` right\_out)
   
*Updates the wing position based on a separate split.*
- void `updateSingleToggle` (`EControllerDigital` left\_toggle, `EControllerDigital` right\_toggle)
   
*Updates the wing position based on a separate toggle.*

**Private Attributes**

- std::shared\_ptr< user::IController > m\_controller {}  
*The user input controller.*
- std::shared\_ptr< robot::Robot > m\_robot {}  
*The robot being controlled.*
- EToggleState toggle\_state {EToggleState::IN}  
*The state stored for toggle mode.*
- EToggleState left\_toggle\_state {EToggleState::IN}  
*The state stored for the left piston for toggle mode.*
- EToggleState right\_toggle\_state {EToggleState::IN}  
*The state stored for the right piston for toggle mode.*

**Static Private Attributes**

- static constexpr char WINGS\_SUBSYSTEM\_NAME [] {"WINGS"}  
*The name of the wings subsystem.*
- static constexpr char SET\_LEFT\_WING\_COMMAND [] {"SET LEFT WING"}  
*The command to set the left wing position.*
- static constexpr char SET\_RIGHT\_WING\_COMMAND [] {"SET RIGHT WING"}  
*The command to set the right wing position.*

**5.126.1 Detailed Description**

Runs the operator-controlled wings settings.

**Author**

Nathan Sandvig

Definition at line 40 of file [WingsOperator.hpp](#).

**5.126.2 Member Enumeration Documentation****5.126.2.1 EToggleState**

```
enum class wisco::user::wings::WingsOperator::EToggleState [strong], [private]
```

The available states for wings toggles.

Definition at line 47 of file [WingsOperator.hpp](#).

```
00048 {
00049     IN,
00050     OUT
00051 };
```

**5.126.3 Constructor & Destructor Documentation****5.126.3.1 WingsOperator()**

```
wisco::user::wings::WingsOperator::WingsOperator (
    const std::shared_ptr< user::IController > & controller,
    const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Wings Operator object.

**Parameters**

<i>controller</i>	The user input controller
<i>robot</i>	The robot to control

Definition at line 9 of file [WingsOperator.cpp](#).

```
00011     : m_controller(controller),
00012     m_robot(robot)
00013 {
00014
00015 }
```

## 5.126.4 Member Function Documentation

### 5.126.4.1 setLeftWing()

```
void wisco::user::wings::WingsOperator::setLeftWing (
    bool out) [private]
```

Sets the position of the left wing.

**Parameters**

<i>out</i>	True for out, false for in
------------	----------------------------

Definition at line 17 of file [WingsOperator.cpp](#).

```
00018 {
00019     m_robot->sendCommand(WINGS_SUBSYSTEM_NAME, SET_LEFT_WING_COMMAND, out);
00020 }
```

### 5.126.4.2 setRightWing()

```
void wisco::user::wings::WingsOperator::setRightWing (
    bool out) [private]
```

Sets the position of the right wing.

**Parameters**

<i>out</i>	True for out, false for in
------------	----------------------------

Definition at line 22 of file [WingsOperator.cpp](#).

```
00023 {
00024     m_robot->sendCommand(WINGS_SUBSYSTEM_NAME, SET_RIGHT_WING_COMMAND, out);
00025 }
```

### 5.126.4.3 setWings()

```
void wisco::user::wings::WingsOperator::setWings (
    bool out) [private]
```

Set the wings to a specific position.

**Parameters**

<i>out</i>	True for out, false for in
------------	----------------------------

Definition at line 27 of file [WingsOperator.cpp](#).

```
00028 {  
00029     setLeftWing(out);  
00030     setRightWing(out);  
00031 }
```

**5.126.4.4 updateDualHold()**

```
void wisco::user::wings::WingsOperator::updateDualHold (  
    EControllerDigital hold )  [private]
```

Updates the wing position based on a combined hold.

**Parameters**

<i>hold</i>	The combined hold digital input
-------------	---------------------------------

Definition at line 33 of file [WingsOperator.cpp](#).

```
00034 {  
00035     bool set_hold{m_controller->getDigital(hold)};  
00036  
00037     if (set_hold)  
00038         setWings(true);  
00039     else  
00040         setWings(false);  
00041 }
```

**5.126.4.5 updateDualSplit()**

```
void wisco::user::wings::WingsOperator::updateDualSplit (  
    EControllerDigital in,  
    EControllerDigital out )  [private]
```

Updates the wing position based on a combined split.

**Parameters**

<i>in</i>	The combined in digital input
<i>out</i>	The combined out digital input

Definition at line 43 of file [WingsOperator.cpp](#).

```
00044 {  
00045     bool set_in{m_controller->getNewDigital(in)};  
00046     bool set_out{m_controller->getNewDigital(out)};  
00047  
00048     if (set_in && !set_out)  
00049         setWings(false);  
00050     else if (!set_in && set_out)  
00051         setWings(true);  
00052 }
```

#### 5.126.4.6 updateDualToggle()

```
void wisco::user::wings::WingsOperator::updateDualToggle (
    EControllerDigital toggle ) [private]
```

Updates the wing position based on a combined toggle.

##### Parameters

<i>toggle</i>	The combined toggle digital input
---------------	-----------------------------------

Definition at line 54 of file [WingsOperator.cpp](#).

```
00055 {
00056     bool set_toggle{m_controller->getNewDigital(toggle)};
00057
00058     if (set_toggle)
00059     {
00060         switch (toggle_state)
00061         {
00062             case EToggleState::IN:
00063                 toggle_state = EToggleState::OUT;
00064                 setWings(true);
00065                 break;
00066             case EToggleState::OUT:
00067                 toggle_state = EToggleState::IN;
00068                 setWings(false);
00069                 break;
00070         }
00071     }
00072 }
```

#### 5.126.4.7 updateSingleHold()

```
void wisco::user::wings::WingsOperator::updateSingleHold (
    EControllerDigital left_hold,
    EControllerDigital right_hold ) [private]
```

Updates the wing position based on a separate hold.

##### Parameters

<i>left_hold</i>	The left hold digital input
<i>right_hold</i>	The right hold digital input

Definition at line 74 of file [WingsOperator.cpp](#).

```
00075 {
00076     bool set_left_hold{m_controller->getDigital(left_hold)};
00077     bool set_right_hold{m_controller->getDigital(right_hold)};
00078
00079     if (set_left_hold)
00080         setLeftWing(true);
00081     else
00082         setLeftWing(false);
00083
00084     if (set_right_hold)
00085         setRightWing(true);
00086     else
00087         setRightWing(false);
00088 }
```

#### 5.126.4.8 updateSingleSplit()

```
void wisco::user::wings::WingsOperator::updateSingleSplit (
```

```

EControllerDigital left_in,
EControllerDigital left_out,
EControllerDigital right_in,
EControllerDigital right_out ) [private]

```

Updates the wing position based on a separate split.

#### Parameters

<i>left_in</i>	The left in digital input
<i>left_out</i>	The left out digital input
<i>right_in</i>	The right in digital input
<i>right_out</i>	The right out digital input

Definition at line 90 of file [WingsOperator.cpp](#).

```

00091 {
00092     bool set_left_in{m_controller->getDigital(left_in)};
00093     bool set_left_out{m_controller->getDigital(left_out)};
00094     bool set_right_in{m_controller->getDigital(right_in)};
00095     bool set_right_out{m_controller->getDigital(right_out)};
00096
00097     if (set_left_in && !set_left_out)
00098         setLeftWing(false);
00099     else if (!set_left_in && set_left_out)
00100         setLeftWing(true);
00101
00102     if (set_right_in && !set_right_out)
00103         setRightWing(false);
00104     else if (!set_right_in && set_right_out)
00105         setRightWing(true);
00106 }

```

#### 5.126.4.9 updateSingleToggle()

```

void wisco::user::wings::WingsOperator::updateSingleToggle (
    EControllerDigital left_toggle,
    EControllerDigital right_toggle ) [private]

```

Updates the wing position based on a separate toggle.

#### Parameters

<i>left_toggle</i>	The left toggle digital input
<i>right_toggle</i>	The right toggle digital input

Definition at line 108 of file [WingsOperator.cpp](#).

```

00109 {
00110     bool set_left_toggle{m_controller->getNewDigital(left_toggle)};
00111     bool set_right_toggle{m_controller->getNewDigital(right_toggle)};
00112
00113     if (set_left_toggle)
00114     {
00115         switch (left_toggle_state)
00116         {
00117             case EToggleState::IN:
00118                 left_toggle_state = EToggleState::OUT;
00119                 setLeftWing(true);
00120                 break;
00121             case EToggleState::OUT:
00122                 left_toggle_state = EToggleState::IN;
00123                 setLeftWing(false);
00124                 break;
00125         }
00126     }

```

```

00127
00128     if (set_right_toggle)
00129     {
00130         switch (right_toggle_state)
00131         {
00132             case EToggleState::IN:
00133                 right_toggle_state = EToggleState::OUT;
00134                 setRightWing(true);
00135                 break;
00136             case EToggleState::OUT:
00137                 right_toggle_state = EToggleState::IN;
00138                 setRightWing(false);
00139                 break;
00140         }
00141     }
00142 }
```

### 5.126.4.10 setWingsPosition()

```
void wisco::user::wings::WingsOperator::setWingsPosition (
    const std::unique_ptr< IProfile > & profile )
```

Set the wings position.

#### Parameters

<i>profile</i>	The driver profile
----------------	--------------------

Definition at line 144 of file [WingsOperator.cpp](#).

```

00145 {
00146     EControllerDigital hold{profile->getDigitalControlMapping(EControl::WINGS_HOLD)};
00147     EControllerDigital in{profile->getDigitalControlMapping(EControl::WINGS_IN)};
00148     EControllerDigital out{profile->getDigitalControlMapping(EControl::WINGS_OUT)};
00149     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::WINGS_TOGGLE)};
00150     EControllerDigital left_hold{profile->getDigitalControlMapping(EControl::WINGS_LEFT_HOLD)};
00151     EControllerDigital left_in{profile->getDigitalControlMapping(EControl::WINGS_LEFT_IN)};
00152     EControllerDigital left_out{profile->getDigitalControlMapping(EControl::WINGS_LEFT_OUT)};
00153     EControllerDigital left_toggle{profile->getDigitalControlMapping(EControl::WINGS_LEFT_TOGGLE)};
00154     EControllerDigital right_hold{profile->getDigitalControlMapping(EControl::WINGS_RIGHT_HOLD)};
00155     EControllerDigital right_in{profile->getDigitalControlMapping(EControl::WINGS_RIGHT_IN)};
00156     EControllerDigital right_out{profile->getDigitalControlMapping(EControl::WINGS_RIGHT_OUT)};
00157     EControllerDigital right_toggle{profile->getDigitalControlMapping(EControl::WINGS_RIGHT_TOGGLE)};
00158
00159     switch(static_cast<EWingsControlMode>(profile->getControlMode(EControlType::WINGS)))
00160     {
00161         case EWingsControlMode::DUAL_HOLD:
00162             updateDualHold(hold);
00163             break;
00164         case EWingsControlMode::DUAL_SPLIT:
00165             updateDualSplit(in, out);
00166             break;
00167         case EWingsControlMode::DUAL_TOGGLE:
00168             updateDualToggle(toggle);
00169             break;
00170         case EWingsControlMode::SINGLE_HOLD:
00171             updateSingleHold(left_hold, right_hold);
00172             break;
00173         case EWingsControlMode::SINGLE_SPLIT:
00174             updateSingleSplit(left_in, left_out, right_in, right_out);
00175             break;
00176         case EWingsControlMode::SINGLE_TOGGLE:
00177             updateSingleToggle(left_toggle, right_toggle);
00178             break;
00179     }
00180 }
```

## 5.126.5 Member Data Documentation

### 5.126.5.1 WINGS\_SUBSYSTEM\_NAME

```
constexpr char wisco::user::wings::WingsOperator::WINGS_SUBSYSTEM_NAME[] {"WINGS"} [static],
[constexpr], [private]
```

The name of the wings subsystem.

Definition at line 57 of file [WingsOperator.hpp](#).

```
00057 {"WINGS"};
```

### 5.126.5.2 SET\_LEFT\_WING\_COMMAND

```
constexpr char wisco::user::wings::WingsOperator::SET_LEFT_WING_COMMAND[] {"SET LEFT WING"}  
[static], [constexpr], [private]
```

The command to set the left wing position.

Definition at line 63 of file [WingsOperator.hpp](#).

```
00063 {"SET LEFT WING"};
```

### 5.126.5.3 SET\_RIGHT\_WING\_COMMAND

```
constexpr char wisco::user::wings::WingsOperator::SET_RIGHT_WING_COMMAND[] {"SET RIGHT WING"}  
[static], [constexpr], [private]
```

The command to set the right wing position.

Definition at line 69 of file [WingsOperator.hpp](#).

```
00069 {"SET RIGHT WING"};
```

### 5.126.5.4 m\_controller

```
std::shared_ptr<user::IController> wisco::user::wings::WingsOperator::m_controller {} [private]
```

The user input controller.

Definition at line 75 of file [WingsOperator.hpp](#).

```
00075 {};
```

### 5.126.5.5 m\_robot

```
std::shared_ptr<robot::Robot> wisco::user::wings::WingsOperator::m_robot {} [private]
```

The robot being controlled.

Definition at line 81 of file [WingsOperator.hpp](#).

```
00081 {};
```

### 5.126.5.6 toggle\_state

```
EToggleState wisco::user::wings::WingsOperator::toggle_state {EToggleState::IN} [private]
```

The state stored for toggle mode.

Definition at line 87 of file [WingsOperator.hpp](#).

```
00087 {EToggleState::IN};
```

### 5.126.5.7 left\_toggle\_state

```
EToggleState wisco::user::wings::WingsOperator::left_toggle_state {EToggleState::IN} [private]
```

The state stored for the left piston for toggle mode.

Definition at line 93 of file [WingsOperator.hpp](#).

```
00093 {EToggleState::IN};
```

### 5.126.5.8 right\_toggle\_state

```
EToggleState wisco::user::wings::WingsOperator::right_toggle_state {EToggleState::IN} [private]
```

The state stored for the right piston for toggle mode.

Definition at line 99 of file [WingsOperator.hpp](#).

```
00099 {EToggleState::IN};
```

# Index

accumulated\_error  
    wisco::control::PID, 220

AControl  
    wisco::control::AControl, 145

addAlliance  
    wisco::IMenu, 240  
    wisco::menu::MenuAdapter, 272

addAutonomous  
    wisco::IMenu, 240  
    wisco::menu::MenuAdapter, 273

addConfiguration  
    wisco::IMenu, 241  
    wisco::menu::MenuAdapter, 273

addControl  
    wisco::control::ControlSystem, 174

addMotor  
    wisco::hal::MotorGroup, 226

addOption  
    wisco::menu::LvglMenu, 264

addPiston  
    wisco::hal::PistonGroup, 230

addProfile  
    wisco::IMenu, 241  
    wisco::menu::MenuAdapter, 274

addSubsystem  
    wisco::robot::Robot, 296

alliance  
    wisco::SystemConfiguration, 525

ALLIANCE\_NAME  
    wisco::alliances::BlueAlliance, 86  
    wisco::alliances::RedAlliance, 87  
    wisco::alliances::SkillsAlliance, 88

ALLIANCE\_OPTION\_NAME  
    wisco::menu::MenuAdapter, 276

alliances  
    wisco::menu::MenuAdapter, 276

ANALOG\_CONTROL\_MAP  
    wisco::profiles::HenryProfile, 286  
    wisco::profiles::JohnProfile, 290

ANALOG\_CONVERSION  
    pros\_adapters::ProsController, 49

ANALOG\_MAP  
    pros\_adapters::ProsController, 50

ANGULAR\_VELOCITY\_CONSTANT  
    pros\_adapters::ProsEXPMotor, 61  
    pros\_adapters::ProsV5Motor, 83

ARM\_DOWN\_STATE\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem,  
        374

ARM\_UP\_STATE\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem,  
        374

ASubsystem  
    wisco::robot::ASubsystem, 292, 293

autonomous  
    wisco::MatchController, 259  
    wisco::SystemConfiguration, 525

autonomous\_manager  
    wisco::MatchController, 260

AUTONOMOUS\_NAME  
    wisco::autons::BlueMatchAuton, 99  
    wisco::autons::BlueSkillsAuton, 104  
    wisco::autons::OrangeMatchAuton, 107  
    wisco::autons::OrangeSkillsAuton, 109

AUTONOMOUS\_OPTION\_NAME  
    wisco::menu::MenuAdapter, 276

autonomous\_routines  
    wisco::menu::MenuAdapter, 276

AutonomousManager  
    wisco::AutonomousManager, 89

BALL\_DETECTOR\_DISTANCE\_CONSTANT  
    wisco::configs::BlueConfiguration, 131

BALL\_DETECTOR\_DISTANCE\_OFFSET  
    wisco::configs::BlueConfiguration, 131

BALL\_DETECTOR\_DISTANCE\_PORT  
    wisco::configs::BlueConfiguration, 131

bindRadians  
    wisco::robot::subsystems::position::DistancePositionResetter,  
        454

BOOMERANG\_CONTROL\_NAME  
    wisco::autons::BlueMatchAuton, 99

BOOMERANG\_GO\_TO\_POSITION\_COMMAND\_NAME  
    wisco::autons::BlueMatchAuton, 100

BOOMERANG\_LEAD  
    wisco::configs::BlueConfiguration, 132

BOOMERANG\_LINEAR\_KD  
    wisco::configs::BlueConfiguration, 132

BOOMERANG\_LINEAR\_KI  
    wisco::configs::BlueConfiguration, 131

BOOMERANG\_LINEAR\_KP  
    wisco::configs::BlueConfiguration, 131

BOOMERANG\_PAUSE\_COMMAND\_NAME  
    wisco::autons::BlueMatchAuton, 100

BOOMERANG\_RESUME\_COMMAND\_NAME  
    wisco::autons::BlueMatchAuton, 100

BOOMERANG\_ROTATIONAL\_KD  
    wisco::configs::BlueConfiguration, 132

BOOMERANG\_ROTATIONAL\_KI

wisco::configs::BlueConfiguration, 132  
**BOOMERANG\_ROTATIONAL\_KP**  
 wisco::configs::BlueConfiguration, 132  
**BOOMERANG\_TARGET\_REACHED\_STATE\_NAME**  
 wisco::autons::BlueMatchAuton, 101  
**BOOMERANG\_TARGET\_TOLERANCE**  
 wisco::configs::BlueConfiguration, 133  
**BoomerangControl**  
 wisco::control::boomerang::BoomerangControl, 149  
**boomerangGoToPoint**  
 wisco::autons::BlueMatchAuton, 93  
**boomerangPause**  
 wisco::autons::BlueMatchAuton, 94  
**boomerangResume**  
 wisco::autons::BlueMatchAuton, 94  
**boomerangTargetReached**  
 wisco::autons::BlueMatchAuton, 94  
**build**  
 wisco::control::boomerang::PIDBoomerangBuilder, 172  
 wisco::control::motion::PIDTurnBuilder, 201  
 wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder, 317  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 343  
 wisco::robot::subsystems::elevator::PIDElevatorBuilder, 365  
 wisco::robot::subsystems::hang::PistonClawBuilder, 387  
 wisco::robot::subsystems::hang::PistonToggleArmBuilder, 393  
 wisco::robot::subsystems::hang::PistonWinchBuilder, 399  
 wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder, 403  
 wisco::robot::subsystems::intake::PIDIntakeBuilder, 426  
 wisco::robot::subsystems::loader::PIDLoaderBuilder, 450  
 wisco::robot::subsystems::position::DistancePositionResetBuilder, 461  
 wisco::robot::subsystems::position::InertialOdometryBuilder, 479  
 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder, 499  
 wisco::robot::subsystems::wings::PistonWingsBuilder, 513  
**buildController**  
 wisco::configs::BlueConfiguration, 115  
 wisco::configs::OrangeConfiguration, 143  
 wisco::IConfiguration, 239  
**buildControlSystem**  
 wisco::configs::BlueConfiguration, 114  
 wisco::configs::OrangeConfiguration, 142  
 wisco::IConfiguration, 238  
**buildRobot**  
 wisco::configs::BlueConfiguration, 116  
 wisco::configs::OrangeConfiguration, 143  
 wisco::IConfiguration, 239  
**button\_default\_style**  
 wisco::menu::LvglMenu, 269  
**button\_matrix\_items\_style**  
 wisco::menu::LvglMenu, 270  
**button\_matrix\_main\_style**  
 wisco::menu::LvglMenu, 270  
**button\_pressed\_style**  
 wisco::menu::LvglMenu, 270  
**BUTTONS\_PER\_LINE**  
 wisco::menu::LvglMenu, 269  
**c0**  
 wisco::control::path::QuinticBezier, 213  
**c1**  
 wisco::control::path::QuinticBezier, 213  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c2**  
 wisco::control::path::QuinticBezier, 214  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c3**  
 wisco::control::path::QuinticBezier, 214  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c4**  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c5**  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c6**  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
**c7**  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 336  
**calculate**  
 wisco::control::path::QuinticBezierSpline, 215  
 calculateAngleToTarget  
 wisco::control::motion::PIDTurn, 188  
 calculateCarrotPoint  
 wisco::control::boomerang::PIDBoomerang, 159  
 calculateDistanceToTarget  
 wisco::control::boomerang::PIDBoomerang, 159  
**calculateDriveVelocity**  
 wisco::control::motion::PIDTurn, 189  
**calculatePoint**  
 wisco::control::path::QuinticBezier, 212  
**CAP\_DETECTED\_DISTANCE**  
 wisco::user::elevator::ElevatorOperator, 551  
**CAP\_DISTANCE\_STATE\_NAME**  
 wisco::robot::subsystems::elevator::ElevatorSubsystem, 351  
 wisco::user::elevator::ElevatorOperator, 550  
**cartridge\_map**  
 pros\_adapters::ProsV5Motor, 82

choices  
  wisco::menu::Option, 278

CLAW\_CLOSED\_STATE\_NAME  
  wisco::robot::subsystems::hang::HangSubsystem,  
    373

CLAW\_OPEN\_STATE\_NAME  
  wisco::robot::subsystems::hang::HangSubsystem,  
    374

clone  
  pros\_adapters::ProsClock, 43  
  pros\_adapters::ProsDelay, 53  
  wisco::rtos::IClock, 519  
  wisco::rtos::IDelay, 521

close  
  wisco::robot::subsystems::hang::IClaw, 376  
  wisco::robot::subsystems::hang::PistonClaw, 383

CLOSE\_CLAW\_COMMAND\_NAME  
  wisco::robot::subsystems::hang::HangSubsystem,  
    372

  wisco::user::hang::HangOperator, 560

closeClaw  
  wisco::user::hang::HangOperator, 555

COLUMN\_WIDTH  
  wisco::menu::LvglMenu, 269

command  
  wisco::control::AControl, 146  
  wisco::control::boomerang::BoomerangControl,  
    150

  wisco::control::motion::MotionControl, 181

  wisco::robot::ASubsystem, 294

  wisco::robot::subsystems::drive::DifferentialDriveSubsystem,  
    304

  wisco::robot::subsystems::elevator::ElevatorSubsystem,  
    349

  wisco::robot::subsystems::hang::HangSubsystem,  
    370

  wisco::robot::subsystems::intake::IntakeSubsystem,  
    410

  wisco::robot::subsystems::loader::LoaderSubsystem,  
    432

  wisco::robot::subsystems::position::PositionSubsystem,  
    489

  wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
    502

  wisco::robot::subsystems::wings::WingsSubsystem,  
    516

competitionInitialize  
  wisco::MatchController, 259

complete  
  wisco::menu::LvglMenu, 271

configuration  
  wisco::SystemConfiguration, 525

CONFIGURATION\_FILE  
  wisco::menu::LvglMenu, 269

CONFIGURATION\_NAME  
  wisco::configs::BlueConfiguration, 121  
  wisco::configs::OrangeConfiguration, 144

CONFIGURATION\_OPTION\_NAME

wisco::menu::MenuAdapter, 276

container\_default\_style  
  wisco::menu::LvglMenu, 270

container\_pressed\_style  
  wisco::menu::LvglMenu, 270

CONTROL\_DELAY  
  wisco::OPControlManager, 281

CONTROL\_MODE\_MAP  
  wisco::profiles::HenryProfile, 285  
  wisco::profiles::JohnProfile, 290

CONTROL\_NAME  
  wisco::control::boomerang::BoomerangControl,  
    151

  wisco::control::motion::MotionControl, 182

control\_robot  
  wisco::control::boomerang::PIDBoomerang, 166  
  wisco::control::motion::PITurn, 196

control\_system  
  wisco::MatchController, 261

controller  
  wisco::MatchController, 261

controls  
  wisco::control::ControlSystem, 176

createDriveTest  
  wisco::testing::TestFactory, 533

createMatchController  
  MatchControllerFactory, 41

CurveVelocityProfile  
  wisco::robot::subsystems::drive::CurveVelocityProfile,  
    300

  delay  
    pros\_adapters::ProsDelay, 53  
    wisco::rtos::IDelay, 521

  delayUntil  
    pros\_adapters::ProsDelay, 53  
    wisco::rtos::IDelay, 521

DIFFERENTIAL\_DRIVE\_SUBSYSTEM\_NAME  
  wisco::user::drive::DifferentialDriveOperator, 538

DifferentialDriveOperator  
  wisco::user::drive::DifferentialDriveOperator, 535

DifferentialDriveSubsystem  
  wisco::robot::subsystems::drive::DifferentialDriveSubsystem,  
    303

DIGITAL\_CONTROL\_MAP  
  wisco::profiles::HenryProfile, 286  
  wisco::profiles::JohnProfile, 291

DIGITAL\_MAP  
  pros\_adapters::ProsController, 50

disabled  
  wisco::MatchController, 259

disengage  
  wisco::robot::subsystems::hang::IWinch, 381  
  wisco::robot::subsystems::hang::PistonWinch, 395

DISENGAGE\_WINCH\_COMMAND\_NAME  
  wisco::robot::subsystems::hang::HangSubsystem,  
    373

  wisco::user::hang::HangOperator, 561

disengageWinch

wisco::user::hang::HangOperator, 556  
**display**  
 wisco::IMenu, 241  
 wisco::menu::MenuAdapter, 274  
**displayMenu**  
 wisco::menu::LvglMenu, 268  
**DistanceBooleanMode**  
 wisco::hal, 20  
**DistanceBooleanSensor**  
 wisco::hal::DistanceBooleanSensor, 222  
**DO\_LOAD\_COMMAND\_NAME**  
 wisco::robot::subsystems::loader::LoaderSubsystem, 433  
 wisco::user::loader::LoaderOperator, 575  
**DO\_READY\_COMMAND\_NAME**  
 wisco::robot::subsystems::loader::LoaderSubsystem, 434  
 wisco::user::loader::LoaderOperator, 575  
**doLoad**  
 wisco::robot::subsystems::loader::ILoader, 429  
 wisco::robot::subsystems::loader::PIDLoader, 439  
 wisco::user::loader::LoaderOperator, 572  
**doReady**  
 wisco::robot::subsystems::loader::ILoader, 429  
 wisco::robot::subsystems::loader::PIDLoader, 439  
 wisco::user::loader::LoaderOperator, 572  
**drawMainMenu**  
 wisco::menu::LvglMenu, 265  
**drawSettingsMenu**  
 wisco::menu::LvglMenu, 266  
**DRIVE\_GEAR\_RATIO**  
 wisco::configs::BlueConfiguration, 129  
**DRIVE\_GET\_RADIUS\_STATE\_NAME**  
 wisco::control::motion::PIDTurn, 195  
**DRIVE\_KINEMATIC**  
 wisco::configs::BlueConfiguration, 124  
**DRIVE\_LEFT\_MOTOR\_1\_GEARSET**  
 wisco::configs::BlueConfiguration, 125  
**DRIVE\_LEFT\_MOTOR\_1\_PORT**  
 wisco::configs::BlueConfiguration, 125  
**DRIVE\_LEFT\_MOTOR\_2\_GEARSET**  
 wisco::configs::BlueConfiguration, 125  
**DRIVE\_LEFT\_MOTOR\_2\_PORT**  
 wisco::configs::BlueConfiguration, 125  
**DRIVE\_LEFT\_MOTOR\_3\_GEARSET**  
 wisco::configs::BlueConfiguration, 126  
**DRIVE\_LEFT\_MOTOR\_3\_PORT**  
 wisco::configs::BlueConfiguration, 125  
**DRIVE\_LEFT\_MOTOR\_4\_GEARSET**  
 wisco::configs::BlueConfiguration, 126  
**DRIVE\_LEFT\_MOTOR\_4\_PORT**  
 wisco::configs::BlueConfiguration, 126  
**DRIVE\_MASS**  
 wisco::configs::BlueConfiguration, 128  
**DRIVE\_MOMENT\_OF\_INERTIA**  
 wisco::configs::BlueConfiguration, 128  
**DRIVE\_RADIUS**  
 wisco::configs::BlueConfiguration, 128  
**DRIVE\_RIGHT\_MOTOR\_1\_GEARSET**  
 wisco::configs::BlueConfiguration, 126  
**DRIVE\_RIGHT\_MOTOR\_1\_PORT**  
 wisco::configs::BlueConfiguration, 126  
**DRIVE\_RIGHT\_MOTOR\_2\_GEARSET**  
 wisco::configs::BlueConfiguration, 127  
**DRIVE\_RIGHT\_MOTOR\_2\_PORT**  
 wisco::configs::BlueConfiguration, 127  
**DRIVE\_RIGHT\_MOTOR\_3\_GEARSET**  
 wisco::configs::BlueConfiguration, 127  
**DRIVE\_RIGHT\_MOTOR\_3\_PORT**  
 wisco::configs::BlueConfiguration, 127  
**DRIVE\_RIGHT\_MOTOR\_4\_GEARSET**  
 wisco::configs::BlueConfiguration, 128  
**DRIVE\_RIGHT\_MOTOR\_4\_PORT**  
 wisco::configs::BlueConfiguration, 127  
**DRIVE\_SET\_VELOCITY\_COMMAND\_NAME**  
 wisco::control::boomerang::PIDBoomerang, 165  
 wisco::control::motion::PITurn, 195  
**DRIVE\_SUBSYSTEM\_NAME**  
 wisco::control::boomerang::PIDBoomerang, 164  
 wisco::control::motion::PITurn, 194  
**DRIVE\_VELOCITY\_PROFILE\_JERK\_RATE**  
 wisco::configs::BlueConfiguration, 124  
**DRIVE\_VELOCITY\_PROFILE\_MAX\_ACCELERATION**  
 wisco::configs::BlueConfiguration, 124  
**DRIVE\_VELOCITY\_TO\_VOLTAGE**  
 wisco::configs::BlueConfiguration, 128  
**DRIVE\_WHEEL\_RADIUS**  
 wisco::configs::BlueConfiguration, 129  
**driver\_profiles**  
 wisco::menu::MenuAdapter, 277  
**DriveTest**  
 wisco::testing::pros\_testing::DriveTest, 527  
**EChassisControlMode**  
 wisco::user::drive, 35  
**EControl**  
 wisco::user, 33  
**EControllerAnalog**  
 wisco::user, 33  
**EControllerDigital**  
 wisco::user, 34  
**EControlType**  
 wisco::user, 34  
**EElevatorControlMode**  
 wisco::user::elevator, 36  
**EHangControlMode**  
 wisco::user::hang, 37  
**EIntakeControlMode**  
 wisco::user::intake, 38  
**ELEVATOR\_DISTANCE\_CONSTANT**  
 wisco::configs::BlueConfiguration, 135  
**ELEVATOR\_DISTANCE\_OFFSET**  
 wisco::configs::BlueConfiguration, 135  
**ELEVATOR\_DISTANCE\_PORT**  
 wisco::configs::BlueConfiguration, 135  
**ELEVATOR\_INCHES\_PER\_RADIAN**  
 wisco::configs::BlueConfiguration, 134

ELEVATOR\_KD  
    wisco::configs::BlueConfiguration, 133

ELEVATOR\_KI  
    wisco::configs::BlueConfiguration, 133

ELEVATOR\_KP  
    wisco::configs::BlueConfiguration, 133

ELEVATOR\_MOTOR\_1\_GEARSET  
    wisco::configs::BlueConfiguration, 134

ELEVATOR\_MOTOR\_1\_PORT  
    wisco::configs::BlueConfiguration, 133

ELEVATOR\_MOTOR\_2\_GEARSET  
    wisco::configs::BlueConfiguration, 134

ELEVATOR\_MOTOR\_2\_PORT  
    wisco::configs::BlueConfiguration, 134

ELEVATOR\_ROTATION\_SENSOR\_PORT  
    wisco::configs::BlueConfiguration, 134

ELEVATOR\_SUBSYSTEM\_NAME  
    wisco::user::elevator::ElevatorOperator, 549

ElevatorOperator  
    wisco::user::elevator::ElevatorOperator, 541

ElevatorSubsystem  
    wisco::robot::subsystems::elevator::ElevatorSubsystem, 348

ELoaderControlMode  
    wisco::user::loader, 39

engage  
    wisco::robot::subsystems::hang::IWinch, 380  
    wisco::robot::subsystems::hang::PistonWinch, 395

ENGAGE\_WINCH\_COMMAND\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem, 373  
    wisco::user::hang::HangOperator, 561

engageWinch  
    wisco::user::hang::HangOperator, 555

EState  
    wisco::robot::subsystems::loader::PIDLoader, 437

EToggleState  
    wisco::user::elevator::ElevatorOperator, 541  
    wisco::user::hang::HangOperator, 554  
    wisco::user::intake::IntakeOperator, 566  
    wisco::user::loader::LoaderOperator, 571  
    wisco::user::umbrella::UmbrellaOperator, 578  
    wisco::user::wings::WingsOperator, 583

ETurnDirection  
    wisco::control::motion, 18

EUmbrellaControlMode  
    wisco::user::umbrella, 40

EWingsControlMode  
    wisco::user::wings, 40

extend  
    pros\_adapters::ProsPiston, 69  
    wisco::io::IPiston, 252

extended  
    pros\_adapters::ProsPiston, 71

FAR\_WALL  
    wisco::robot::subsystems::position::DistancePositionResetter, 457

FIELD\_POSITION

wisco::user::elevator::ElevatorOperator, 550

FILE\_PATH  
    wisco::testing::pros\_testing, 31

forced\_direction\_reached  
    wisco::control::motion::PIDTurn, 198

GEAR\_RATIO  
    pros\_adapters::ProsEXPMotor, 61

GET\_BALANCE\_ANGLE\_STATE\_NAME  
    wisco::robot::subsystems::intake::IntakeSubsystem, 412

GET\_BALANCE\_DISTANCE\_STATE\_NAME  
    wisco::robot::subsystems::intake::IntakeSubsystem, 412

GET\_LEFT\_WING\_STATE\_NAME  
    wisco::robot::subsystems::wings::WingsSubsystem, 518

GET\_POSITION\_STATE  
    wisco::user::elevator::ElevatorOperator, 549

GET\_POSITION\_STATE\_NAME  
    wisco::robot::subsystems::elevator::ElevatorSubsystem, 350  
    wisco::robot::subsystems::position::PositionSubsystem, 491

GET\_RADIUS\_STATE\_NAME  
    wisco::robot::subsystems::drive::DifferentialDriveSubsystem, 306

GET\_RIGHT\_WING\_STATE\_NAME  
    wisco::robot::subsystems::wings::WingsSubsystem, 518

GET\_VELOCITY\_STATE\_NAME  
    wisco::robot::subsystems::drive::DifferentialDriveSubsystem, 306  
    wisco::robot::subsystems::intake::IntakeSubsystem, 412

getAcceleration  
    wisco::robot::subsystems::drive::CurveVelocityProfile, 300  
    wisco::robot::subsystems::drive::IVelocityProfile, 321

getAnalog  
    pros\_adapters::ProsController, 47  
    wisco::user::IController, 563

getAnalogControlMapping  
    wisco::IProfile, 256  
    wisco::profiles::HenryProfile, 284  
    wisco::profiles::JohnProfile, 289

getAngle  
    pros\_adapters::ProsRotation, 74  
    wisco::io::IRotationSensor, 254

getAngularVelocity  
    pros\_adapters::ProsEXPMotor, 60  
    pros\_adapters::ProsV5Motor, 81  
    wisco::hal::MotorGroup, 227  
    wisco::io::IMotor, 250

getAngularVelocityConstant  
    pros\_adapters::ProsEXPMotor, 59  
    pros\_adapters::ProsV5Motor, 80  
    wisco::hal::MotorGroup, 227

wisco::io::IMotor, 249  
**getBallAngle**  
 wisco::robot::subsystems::intake::DistanceVisionBallDetector, 401  
 wisco::robot::subsystems::intake::IBallDetector, 405  
**getBallDistance**  
 wisco::robot::subsystems::intake::DistanceVisionBallDetector, 401  
 wisco::robot::subsystems::intake::IBallDetector, 405  
**getCapDistance**  
 wisco::user::elevator::ElevatorOperator, 542  
**getControlMode**  
 wisco::IProfile, 255  
 wisco::profiles::HenryProfile, 283  
 wisco::profiles::JohnProfile, 288  
**getControlValue**  
 wisco::control::PID, 218  
**getDigital**  
 pros\_adapters::ProsController, 48  
 wisco::user::IController, 563  
**getDigitalControlMapping**  
 wisco::IProfile, 256  
 wisco::profiles::HenryProfile, 285  
 wisco::profiles::JohnProfile, 289  
**getDistance**  
 pros\_adapters::ProsDistance, 56  
 wisco::hal::TrackingWheel, 234  
 wisco::io::IDistanceSensor, 244  
 wisco::io::IDistanceTrackingSensor, 245  
**getDriveRadius**  
 wisco::control::motion::PDTurn, 187  
**getElevatorPosition**  
 wisco::user::elevator::ElevatorOperator, 542  
**getGearRatio**  
 pros\_adapters::ProsEXPMotor, 60  
 pros\_adapters::ProsV5Motor, 81  
 wisco::hal::MotorGroup, 227  
 wisco::io::IMotor, 250  
**getHangArmUp**  
 wisco::user::elevator::ElevatorOperator, 542  
**getHeading**  
 pros\_adapters::ProsHeading, 64  
 wisco::io::IHeadingSensor, 247  
**getLeftWing**  
 wisco::robot::subsystems::wings::IWings, 506  
 wisco::robot::subsystems::wings::PistonWings, 509  
**getName**  
 wisco::alliances::BlueAlliance, 85  
 wisco::alliances::RedAlliance, 87  
 wisco::alliances::SkillsAlliance, 88  
 wisco::autons::BlueMatchAuton, 98  
 wisco::autons::BlueSkillsAuton, 103  
 wisco::autons::OrangeMatchAuton, 106  
 wisco::autons::OrangeSkillsAuton, 108  
 wisco::configs::BlueConfiguration, 114  
 wisco::configs::OrangeConfiguration, 142  
 wisco::control::AControl, 146  
**getNewDigital**  
 pros\_adapters::ProsController, 48  
 wisco::user::IController, 564  
**getOdometryPosition**  
 wisco::control::boomerang::PIDBoomerang, 158  
 wisco::control::motion::PDTurn, 188  
**getPosition**  
 pros\_adapters::ProsV5Motor, 81  
 wisco::hal::MotorGroup, 228  
 wisco::io::IMotor, 250  
 wisco::robot::subsystems::elevator::IElevator, 352  
 wisco::robot::subsystems::elevator::PDElevator, 356  
 wisco::robot::subsystems::loader::PIDLoader, 438  
 wisco::robot::subsystems::position::InertialOdometry, 467  
 wisco::robot::subsystems::position::IPositionTracker, 485  
**getRadius**  
 wisco::robot::subsystems::drive::DirectDifferentialDrive, 310  
 wisco::robot::subsystems::drive::IDifferentialDrive, 320  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive, 328  
**getResetX**  
 wisco::robot::subsystems::position::DistancePositionResetter, 455  
 wisco::robot::subsystems::position::IPositionResetter, 483  
**getResetY**  
 wisco::robot::subsystems::position::DistancePositionResetter, 455  
 wisco::robot::subsystems::position::IPositionResetter, 483  
**getResistance**  
 pros\_adapters::ProsEXPMotor, 59  
 pros\_adapters::ProsV5Motor, 80  
 wisco::hal::MotorGroup, 226  
 wisco::io::IMotor, 249  
**getRightWing**  
 wisco::robot::subsystems::wings::IWings, 506  
 wisco::robot::subsystems::wings::PistonWings, 509  
**getRotation**  
 pros\_adapters::ProsHeading, 65  
 pros\_adapters::ProsRotation, 73  
 wisco::io::IHeadingSensor, 247  
 wisco::io::IRotationSensor, 254

getSelection  
    wisco::menu::LvglMenu, 268

getState  
    wisco::control::ControlSystem, 175  
    wisco::hal::PistonGroup, 231  
    wisco::robot::Robot, 298

getSystemConfiguration  
    wisco::IMenu, 242  
    wisco::menu::MenuAdapter, 275

getTime  
    pros\_adapters::ProsClock, 43  
    wisco::rtos::IClock, 519

getTorqueConstant  
    pros\_adapters::ProsEXPMotor, 59  
    pros\_adapters::ProsV5Motor, 80  
    wisco::hal::MotorGroup, 226  
    wisco::io::IMotor, 249

getValue  
    wisco::hal::DistanceBooleanSensor, 223  
    wisco::io::IBooleanSensor, 243

getVelocity  
    wisco::robot::subsystems::drive::DirectDifferentialDrive, 308  
    wisco::robot::subsystems::drive::IDifferentialDrive, 319  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive, 326  
    wisco::robot::subsystems::intake::IIntake, 407  
    wisco::robot::subsystems::intake::PIDIntake, 416

getX  
    wisco::control::path::Point, 205

getY  
    wisco::control::path::Point, 206

give  
    pros\_adapters::ProsMutex, 67  
    wisco::rtos::IMutex, 522

GO\_TO\_POSITION\_COMMAND\_NAME  
    wisco::control::boomerang::BoomerangControl, 151

goToPosition  
    wisco::control::boomerang::IBoomerang, 154  
    wisco::control::boomerang::PIDBoomerang, 160

HANG\_ARM\_PISTON\_1\_EXTENDED\_STATE  
    wisco::configs::BlueConfiguration, 136

HANG\_ARM\_PISTON\_1\_PORT  
    wisco::configs::BlueConfiguration, 136

HANG\_ARM\_UP\_STATE  
    wisco::configs::BlueConfiguration, 136

HANG\_ARM\_UP\_STATE\_NAME  
    wisco::user::elevator::ElevatorOperator, 550

HANG\_CLAW\_CLOSED\_STATE  
    wisco::configs::BlueConfiguration, 136

HANG\_CLAW\_PISTON\_1\_EXTENDED\_STATE  
    wisco::configs::BlueConfiguration, 135

HANG\_CLAW\_PISTON\_1\_PORT  
    wisco::configs::BlueConfiguration, 135

HANG\_SUBSYSTEM\_NAME  
    wisco::user::elevator::ElevatorOperator, 549

wisco::user::hang::HangOperator, 560

HANG\_WINCH\_ENGAGED\_STATE  
    wisco::configs::BlueConfiguration, 137

HANG\_WINCH\_PISTON\_1\_EXTENDED\_STATE  
    wisco::configs::BlueConfiguration, 137

HANG\_WINCH\_PISTON\_1\_PORT  
    wisco::configs::BlueConfiguration, 136

HangOperator  
    wisco::user::hang::HangOperator, 554

HangSubsystem  
    wisco::robot::subsystems::hang::HangSubsystem, 369

hardware\_configurations  
    wisco::menu::MenuAdapter, 277

HEADING\_TO\_RADIANS  
    wisco::testing::pros\_testing::DriveTest, 530

IN\_POSITION  
    wisco::user::elevator::ElevatorOperator, 550

INCHES\_TO\_METERS  
    wisco::testing::pros\_testing::DriveTest, 530

INERTIAL\_PORT  
    wisco::testing::TestFactory, 533

initialize  
    pros\_adapters::ProsController, 47  
    pros\_adapters::ProsDistance, 55  
    pros\_adapters::ProsEXPMotor, 59  
    pros\_adapters::ProsHeading, 64  
    pros\_adapters::ProsRotation, 73  
    pros\_adapters::ProsV5Motor, 80  
    wisco::autons::BlueMatchAuton, 98  
    wisco::autons::BlueSkillsAuton, 103  
    wisco::autons::OrangeMatchAuton, 106  
    wisco::autons::OrangeSkillsAuton, 108  
    wisco::control::AControl, 146  
    wisco::control::boomerang::BoomerangControl, 150  
    wisco::control::boomerang::IBoomerang, 153  
    wisco::control::boomerang::PIDBoomerang, 160  
    wisco::control::ControlSystem, 175  
    wisco::control::motion::ITurn, 177  
    wisco::control::motion::MotionControl, 181  
    wisco::control::motion::PIDTurn, 189  
    wisco::hal::DistanceBooleanSensor, 223  
    wisco::hal::MotorGroup, 226  
    wisco::hal::TrackingWheel, 233  
    wisco::IAutonomous, 237  
    wisco::io::IBooleanSensor, 243  
    wisco::io::IDistanceSensor, 244  
    wisco::io::IDistanceTrackingSensor, 245  
    wisco::io::IHeadingSensor, 247  
    wisco::io::IMotor, 249  
    wisco::io::IRotationSensor, 253  
    wisco::MatchController, 258  
    wisco::robot::ASubsystem, 293  
    wisco::robot::Robot, 297  
    wisco::robot::subsystems::drive::DifferentialDriveSubsystem, 304

wisco::robot::subsystems::drive::DirectDifferentialDrive, wisco::OPControlManager, 280  
 308  
 initializeStyles  
 wisco::robot::subsystems::drive::IDifferentialDrive, wisco::menu::LvglMenu, 263  
 319  
 INTAKE\_KD  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive wisco::configs::BlueConfiguration, 129  
 325  
 INTAKE\_KI  
 wisco::robot::subsystems::elevator::ElevatorSubsystem, wisco::configs::BlueConfiguration, 129  
 349  
 INTAKE\_KP  
 wisco::configs::BlueConfiguration, 129  
 INTAKE\_MOTOR\_1\_GEARSET  
 wisco::configs::BlueConfiguration, 130  
 INTAKE\_MOTOR\_1\_PORT  
 wisco::configs::BlueConfiguration, 130  
 INTAKE\_MOTOR\_2\_GEARSET  
 wisco::configs::BlueConfiguration, 130  
 INTAKE\_MOTOR\_2\_PORT  
 wisco::configs::BlueConfiguration, 130  
 INTAKE\_ROLLER\_RADIUS  
 wisco::configs::BlueConfiguration, 130  
 INTAKE\_SUBSYSTEM\_NAME  
 wisco::user::intake::IntakeOperator, 568  
 IntakeOperator  
 wisco::user::intake::IntakeOperator, 566  
 IntakeSubsystem  
 wisco::robot::subsystems::intake::IntakeSubsystem,  
 409  
 IS\_IN\_STATE\_NAME  
 wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
 504  
 IS\_LOADED\_STATE\_NAME  
 wisco::robot::subsystems::loader::LoaderSubsystem,  
 434  
 IS\_OUT\_STATE\_NAME  
 wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
 504  
 IS\_READY\_STATE\_NAME  
 wisco::robot::subsystems::loader::LoaderSubsystem,  
 434  
 isClosed  
 wisco::robot::subsystems::hang::IClaw, 377  
 wisco::robot::subsystems::hang::PistonClaw, 383  
 isDisengaged  
 wisco::robot::subsystems::hang::IWinch, 381  
 wisco::robot::subsystems::hang::PistonWinch, 396  
 isDown  
 wisco::robot::subsystems::hang::IToggleArm, 379  
 wisco::robot::subsystems::hang::PistonToggleArm,  
 390  
 isEngaged  
 wisco::robot::subsystems::hang::IWinch, 381  
 wisco::robot::subsystems::hang::PistonWinch, 396  
 isExtended  
 pros\_adapters::ProsPiston, 70  
 wisco::io::IPiston, 252  
 isIn

wisco::robot::subsystems::umbrella::IUmbrella, 493  
wisco::robot::subsystems::umbrella::PistonUmbrella, 496  
isLoaded  
    wisco::robot::subsystems::loader::ILoader, 429  
    wisco::robot::subsystems::loader::PIDLoader, 440  
    wisco::user::loader::LoaderOperator, 572  
isOpen  
    wisco::robot::subsystems::hang::IClaw, 377  
    wisco::robot::subsystems::hang::PistonClaw, 383  
isOut  
    wisco::robot::subsystems::umbrella::IUmbrella, 494  
    wisco::robot::subsystems::umbrella::PistonUmbrella, 496  
isReady  
    wisco::robot::subsystems::loader::ILoader, 429  
    wisco::robot::subsystems::loader::PIDLoader, 440  
    wisco::user::loader::LoaderOperator, 572  
isRetracted  
    pros\_adapters::ProsPiston, 70  
    wisco::io::IPiston, 252  
isStarted  
    wisco::IMenu, 241  
    wisco::menu::MenuAdapter, 274  
isUp  
    wisco::robot::subsystems::hang::IToggleArm, 379  
    wisco::robot::subsystems::hang::PistonToggleArm, 390  
join  
    pros\_adapters::ProsTask, 77  
    wisco::rtos::ITask, 524  
k0  
    wisco::control::path::QuinticBezier, 213  
k1  
    wisco::control::path::QuinticBezier, 214  
last\_error  
    wisco::control::PID, 220  
last\_heading  
    wisco::robot::subsystems::position::InertialOdometry, 473  
last\_linear\_distance  
    wisco::robot::subsystems::position::InertialOdometry, 474  
last\_rumble\_refresh  
    pros\_adapters::ProsController, 51  
last\_strafe\_distance  
    wisco::robot::subsystems::position::InertialOdometry, 474  
last\_time  
    wisco::control::PID, 220  
    wisco::robot::subsystems::drive::CurveVelocityProfile, 302  
    wisco::robot::subsystems::position::InertialOdometry, 474  
LEFT\_DRIVE\_PORTS  
    wisco::testing::TestFactory, 533  
left\_toggle\_state  
    wisco::user::wings::WingsOperator, 589  
left\_velocity  
    wisco::robot::subsystems::drive::Velocity, 346  
LEFT\_WING\_PISTON\_1\_EXTENDED\_STATE  
    wisco::configs::BlueConfiguration, 140  
LEFT\_WING\_PISTON\_1\_PORT  
    wisco::configs::BlueConfiguration, 140  
LINEAR\_COUNTS\_PER\_INCH  
    wisco::testing::TestFactory, 534  
LINEAR\_FILE\_NAME  
    wisco::testing::pros\_testing::DriveTest, 529  
LINEAR\_TRACKING\_PORT  
    wisco::testing::TestFactory, 533  
LOADER\_KD  
    wisco::configs::BlueConfiguration, 137  
LOADER\_KI  
    wisco::configs::BlueConfiguration, 137  
LOADER\_KP  
    wisco::configs::BlueConfiguration, 137  
LOADER\_LOADED\_POSITION  
    wisco::configs::BlueConfiguration, 138  
LOADER\_MOTOR\_1\_GEARSET  
    wisco::configs::BlueConfiguration, 138  
LOADER\_MOTOR\_1\_PORT  
    wisco::configs::BlueConfiguration, 138  
LOADER\_POSITION\_TOLERANCE  
    wisco::configs::BlueConfiguration, 138  
LOADER\_READY\_POSITION  
    wisco::configs::BlueConfiguration, 138  
LoaderOperator  
    wisco::user::loader::LoaderOperator, 571  
LoaderSubsystem  
    wisco::robot::subsystems::loader::LoaderSubsystem, 431  
LOWER\_ARM\_COMMAND\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem, 373  
    wisco::user::hang::HangOperator, 560  
lowerArm  
    wisco::user::hang::HangOperator, 555  
lvgl\_menu  
    wisco::menu::MenuAdapter, 277  
m\_autonomous  
    wisco::AutonomousManager, 91  
m\_ball\_detector  
    wisco::robot::subsystems::intake::IntakeSubsystem, 413  
m\_boomerang  
    wisco::control::boomerang::BoomerangControl, 152  
m\_claw  
    wisco::robot::subsystems::hang::HangSubsystem, 375  
m\_clock  
    wisco::AutonomousManager, 91

wisco::control::PID, 219  
wisco::MatchController, 260  
wisco::OPControlManager, 281  
wisco::robot::subsystems::drive::CurveVelocityProfile, 301  
wisco::robot::subsystems::elevator::PIDElevator, 359  
wisco::robot::subsystems::elevator::PIDElevatorBuilder, 366  
wisco::robot::subsystems::intake::PIDIntake, 420  
wisco::robot::subsystems::intake::PIDIntakeBuilder, 426  
wisco::robot::subsystems::loader::PIDLoader, 443  
wisco::robot::subsystems::loader::PIDLoaderBuilder, 451  
wisco::robot::subsystems::position::InertialOdometry, 471  
wisco::robot::subsystems::position::InertialOdometryBuilder, 480  
m\_closed\_state  
  wisco::robot::subsystems::hang::PistonClaw, 385  
  wisco::robot::subsystems::hang::PistonClawBuilder, 388  
m\_controller  
  pros\_adapters::ProsController, 51  
  wisco::user::drive::DifferentialDriveOperator, 538  
  wisco::user::elevator::ElevatorOperator, 551  
  wisco::user::hang::HangOperator, 561  
  wisco::user::intake::IntakeOperator, 569  
  wisco::user::loader::LoaderOperator, 576  
  wisco::user::umbrella::UmbrellaOperator, 581  
  wisco::user::wings::WingsOperator, 589  
m\_current\_acceleration  
  wisco::robot::subsystems::drive::CurveVelocityProfile, 302  
m\_delayer  
  wisco::AutonomousManager, 91  
  wisco::control::boomerang::PIDBoomerang, 165  
  wisco::control::boomerang::PIDBoomerangBuilder, 172  
  wisco::control::motion::PIDTurn, 195  
  wisco::control::motion::PIDTurnBuilder, 202  
  wisco::MatchController, 260  
  wisco::OPControlManager, 281  
  wisco::robot::subsystems::drive::KinematicDifferentialDrive, 332  
  wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 343  
  wisco::robot::subsystems::elevator::PIDElevator, 359  
  wisco::robot::subsystems::elevator::PIDElevatorBuilder, 366  
  wisco::robot::subsystems::intake::PIDIntake, 421  
  wisco::robot::subsystems::intake::PIDIntakeBuilder, 426  
  wisco::robot::subsystems::loader::PIDLoader, 443  
  wisco::robot::subsystems::loader::PIDLoaderBuilder, 451  
m\_digital\_out  
  pros\_adapters::ProsPiston, 71  
m\_distance\_sensor  
  wisco::hal::DistanceBooleanSensor, 224  
  wisco::robot::subsystems::elevator::ElevatorSubsystem, 351  
  wisco::robot::subsystems::intake::DistanceVisionBallDetector, 402  
  wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder, 404  
  wisco::robot::subsystems::position::DistancePositionResetter, 457  
  wisco::robot::subsystems::position::DistancePositionResetterBuilder, 461  
m\_elevator  
  wisco::robot::subsystems::elevator::ElevatorSubsystem, 351  
m\_engaged\_state  
  wisco::robot::subsystems::hang::PistonWinch, 397  
  wisco::robot::subsystems::hang::PistonWinchBuilder, 399  
m\_extended\_value  
  pros\_adapters::ProsPiston, 71  
m\_gear\_ratio  
  wisco::robot::subsystems::drive::DirectDifferentialDrive, 312  
  wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder, 317  
  wisco::robot::subsystems::drive::KinematicDifferentialDrive, 334  
  wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 345  
m\_heading\_sensor  
  wisco::robot::subsystems::position::InertialOdometry, 472  
  wisco::robot::subsystems::position::InertialOdometryBuilder, 481  
m\_inches\_per\_radian  
  wisco::robot::subsystems::elevator::PIDElevator, 361  
  wisco::robot::subsystems::elevator::PIDElevatorBuilder, 367  
  wisco::robot::subsystems::intake::IntakeSubsystem, 412  
  wisco::robot::subsystems::intake::IntakeSubsystemBuilder, 421  
m\_jerk\_rate  
  wisco::robot::subsystems::drive::CurveVelocityProfile, 301  
m\_kd  
  wisco::control::PID, 220

m\_ki  
    wisco::control::PID, 219

m\_kp  
    wisco::control::PID, 219

m\_lead  
    wisco::control::boomerang::PIDBoomerang, 166  
    wisco::control::boomerang::PIDBoomerangBuilder,  
        173

m\_left\_drive\_motors  
    wisco::testing::pros\_testing::DriveTest, 531

m\_left\_motors  
    wisco::robot::subsystems::drive::DirectDifferentialDrive,  
        312  
    wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder, 345  
        317

wisco::robot::subsystems::drive::KinematicDifferentialDrive  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 333  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuild  
        344

m\_left\_pistons  
    wisco::robot::subsystems::wings::PistonWings,  
        511  
    wisco::robot::subsystems::wings::PistonWingsBuilder,  
        514

m\_left\_velocity\_profile  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive  
        333  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuild  
        344

m\_linear\_counts\_per\_inch  
    wisco::testing::pros\_testing::DriveTest, 531

m\_linear\_distance\_tracking\_offset  
    wisco::robot::subsystems::position::InertialOdometry,  
        473

wisco::robot::subsystems::position::InertialOdometryBuilder  
    wisco::robot::subsystems::elevator::PIDElevator,  
        481

m\_linear\_distance\_tracking\_sensor  
    wisco::robot::subsystems::position::InertialOdometry,  
        472  
    wisco::robot::subsystems::position::InertialOdometryBuilder  
        481

m\_linear\_pid  
    wisco::control::boomerang::PIDBoomerang, 166  
    wisco::control::boomerang::PIDBoomerangBuilder,  
        172

m\_linear\_sensor  
    wisco::testing::pros\_testing::DriveTest, 531

m\_loader  
    wisco::robot::subsystems::loader::LoaderSubsystem,  
        434

m\_local\_theta  
    wisco::robot::subsystems::position::DistancePositionResetter,  
        458

wisco::robot::subsystems::position::DistancePositionResetterBu  
    462

m\_local\_x  
    wisco::robot::subsystems::position::DistancePositionResetter  
        458

wisco::robot::subsystems::position::DistancePositionResetterBuilder  
    461

m\_local\_y  
    wisco::robot::subsystems::position::DistancePositionResetter,  
        458

wisco::robot::subsystems::position::DistancePositionResetterBuilder,  
    461

m\_lower\_threshold  
    wisco::hal::DistanceBooleanSensor, 224

m\_mass  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
        334  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
        345

m\_match\_load\_position  
    wisco::robot::subsystems::loader::PIDLoader, 444  
    wisco::robot::subsystems::loader::PIDLoaderBuilder,

m\_max\_acceleration  
    wisco::robot::subsystems::drive::CurveVelocityProfile,  
        301

m\_menu  
    wisco::MatchController, 260

m\_mode  
    wisco::hal::DistanceBooleanSensor, 224

m\_movement\_of\_inertia  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
        333  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuild  
        344

wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
    345

m\_motor  
    pros\_adapters::ProsEXPMotor, 62  
    pros\_adapters::ProsV5Motor, 84

m\_motors  
    wisco::robot::subsystems::elevator::PIDElevator,  
        360  
    wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
        367

wisco::robot::subsystems::intake::PIDIntake, 421  
    wisco::robot::subsystems::intake::PIDIntakeBuilder,  
        427

wisco::robot::subsystems::loader::PIDLoader, 444  
    wisco::robot::subsystems::loader::PIDLoaderBuilder,  
        451

m\_mutex  
    wisco::control::boomerang::PIDBoomerang, 165  
    wisco::control::boomerang::PIDBoomerangBuilder,  
        172

wisco::control::motion::PITurn, 196  
    wisco::control::motion::PITurnBuilder, 202

wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
    wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
        448

wisco::robot::subsystems::elevator::PIDElevator,  
    360

wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
    366

wisco::robot::subsystems::intake::PIDIntake, 421  
wisco::robot::subsystems::intake::PIDIntakeBuilder, 427  
wisco::robot::subsystems::loader::PIDLoader, 444  
wisco::robot::subsystems::loader::PIDLoaderBuilder, m\_position\_tracker  
451  
wisco::robot::subsystems::position::InertialOdometry,  
472  
wisco::robot::subsystems::position::InertialOdometryBuilder, wisco::OPControlManager, 282  
480  
m\_name  
wisco::control::AControl, 148  
wisco::robot::ASubsystem, 295  
m\_out\_state  
wisco::robot::subsystems::umbrella::PistonUmbrella,  
497  
wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder, wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
499  
wisco::robot::subsystems::wings::PistonWings,  
511  
wisco::robot::subsystems::wings::PistonWingsBuilder,  
514  
m\_pid  
wisco::control::motion::PIDTurn, 196  
wisco::control::motion::PIDTurnBuilder, 202  
wisco::robot::subsystems::elevator::PIDElevator,  
360  
wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
366  
wisco::robot::subsystems::intake::PIDIntake, 421  
wisco::robot::subsystems::intake::PIDIntakeBuilder,  
427  
wisco::robot::subsystems::loader::PIDLoader, 444  
wisco::robot::subsystems::loader::PIDLoaderBuilder, m\_right\_pistons  
451  
m\_pistons  
wisco::robot::subsystems::hang::PistonClaw, 385  
wisco::robot::subsystems::hang::PistonClawBuilder,  
388  
wisco::robot::subsystems::hang::PistonToggleArm,  
391  
wisco::robot::subsystems::hang::PistonToggleArmBuilder, wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
393  
wisco::robot::subsystems::hang::PistonWinch, 397  
wisco::robot::subsystems::hang::PistonWinchBuilder,  
399  
wisco::robot::subsystems::umbrella::PistonUmbrella,  
497  
wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder, wisco::user::loader::LoaderOperator, 576  
499  
m\_position  
wisco::robot::subsystems::elevator::PIDElevator,  
361  
wisco::robot::subsystems::position::InertialOdometry,  
473  
m\_position\_resetter  
wisco::robot::subsystems::position::PositionSubsystem,  
492  
m\_position\_tolerance  
wisco::robot::subsystems::loader::PIDLoader, 445  
wisco::robot::subsystems::loader::PIDLoaderBuilder,  
452  
wisco::robot::subsystems::position::PositionSubsystem,  
491  
m\_profile  
m\_radius  
wisco::robot::subsystems::drive::DirectDifferentialDrive,  
313  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
318  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
334  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
345  
m\_ready\_position  
wisco::robot::subsystems::loader::PIDLoader, 444  
wisco::robot::subsystems::loader::PIDLoaderBuilder,  
452  
m\_right\_drive\_motors  
wisco::testing::pros\_testing::DriveTest, 531  
m\_right\_motors  
wisco::robot::subsystems::drive::DirectDifferentialDrive,  
312  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
317  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
334  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
344  
m\_right\_velocity\_profile  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
333  
wisco::user::drive::DifferentialDriveOperator, 539  
wisco::user::elevator::ElevatorOperator, 552  
wisco::user::hang::HangOperator, 561  
wisco::user::intake::IntakeOperator, 569  
wisco::user::loader::LoaderOperator, 576  
wisco::user::umbrella::UmbrellaOperator, 581  
wisco::user::wings::WingsOperator, 589  
m\_roller\_radius  
wisco::robot::subsystems::intake::PIDIntake, 421  
wisco::robot::subsystems::intake::PIDIntakeBuilder,  
427  
m\_rotation\_sensor  
wisco::robot::subsystems::elevator::PIDElevator,  
360

wisco::robot::subsystems::elevator::PIDElevatorBuilder, 375  
367  
m\_rotational\_pid  
wisco::control::boomerang::PIDBoomerang, 166  
wisco::control::boomerang::PIDBoomerangBuilder, 173  
m\_sensor  
pros\_adapters::ProsDistance, 56  
pros\_adapters::ProsHeading, 66  
pros\_adapters::ProsRotation, 75  
wisco::hal::TrackingWheel, 234  
m\_state  
wisco::hal::PistonGroup, 231  
m\_strafe\_distance\_tracking\_offset  
wisco::robot::subsystems::position::InertialOdometry, 473  
wisco::robot::subsystems::position::InertialOdometryBuilder, 481  
m\_strafe\_distance\_tracking\_sensor  
wisco::robot::subsystems::position::InertialOdometry, 473  
wisco::robot::subsystems::position::InertialOdometryBuilder, 481  
m\_target\_tolerance  
wisco::control::boomerang::PIDBoomerang, 166  
wisco::control::boomerang::PIDBoomerangBuilder, 173  
wisco::control::motion::PIDTurn, 196  
wisco::control::motion::PIDTurnBuilder, 202  
m\_target\_velocity  
wisco::control::motion::PIDTurn, 196  
wisco::control::motion::PIDTurnBuilder, 202  
m\_task  
wisco::control::boomerang::PIDBoomerang, 166  
wisco::control::boomerang::PIDBoomerangBuilder, 172  
wisco::control::motion::PIDTurn, 196  
wisco::control::motion::PIDTurnBuilder, 202  
wisco::robot::subsystems::drive::KinematicDifferentialDrive, 333  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 344  
wisco::robot::subsystems::elevator::PIDElevator, 360  
wisco::robot::subsystems::elevator::PIDElevatorBuilder, 366  
wisco::robot::subsystems::intake::PIDIntake, 421  
wisco::robot::subsystems::intake::PIDIntakeBuilder, 427  
wisco::robot::subsystems::loader::PIDLoader, 444  
wisco::robot::subsystems::loader::PIDLoaderBuilder, 451  
wisco::robot::subsystems::position::InertialOdometry, 472  
wisco::robot::subsystems::position::InertialOdometryBuilder, 480  
m\_toggle\_arm  
wisco::robot::subsystems::hang::HangSubsystem, 367  
m\_tuning\_constant  
pros\_adapters::ProsDistance, 56  
pros\_adapters::ProsHeading, 66  
m\_tuning\_offset  
pros\_adapters::ProsDistance, 57  
m\_turn  
wisco::control::motion::MotionControl, 183  
m\_umbrella  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem, 504  
m\_up\_state  
wisco::robot::subsystems::hang::PistonToggleArm, 391  
wisco::robot::subsystems::hang::PistonToggleArmBuilder, 393  
m\_upper\_threshold  
wisco::hal::DistanceBooleanSensor, 224  
m\_velocity  
wisco::robot::subsystems::drive::KinematicDifferentialDrive, 336  
m\_velocity\_to\_voltage  
wisco::robot::subsystems::drive::DirectDifferentialDrive, 312  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder, 317  
m\_wheel\_radius  
wisco::hal::TrackingWheel, 235  
wisco::robot::subsystems::drive::DirectDifferentialDrive, 313  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder, 318  
wisco::robot::subsystems::drive::KinematicDifferentialDrive, 335  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder, 345  
m\_winch  
wisco::robot::subsystems::hang::HangSubsystem, 375  
DriveBuilder, 344  
wisco::robot::subsystems::wings::WingsSubsystem, 518  
m\_x  
wisco::control::path::Point, 210  
m\_y  
wisco::control::path::Point, 210  
manual\_input  
wisco::user::elevator::ElevatorOperator, 552  
MATCH\_LOAD\_POSITION  
wisco::user::elevator::ElevatorOperator, 550  
MatchController  
wisco::MatchController, 258  
MatchControllerFactory, 41  
createMatchController, 41  
MAX\_MILLIVOLTS  
pros\_adapters::ProsV5Motor, 84  
MAX\_RUMBLE\_LENGTH

pros\_adapters::ProsController, 50  
**MENU\_DELAY**  
 wisco::MatchController, 260  
**MILLIS\_TO\_S**  
 wisco::testing::pros\_testing::DriveTest, 529  
**MOTION\_CONTROL\_NAME**  
 wisco::autons::BlueMatchAuton, 101  
**MOTION\_PAUSE\_TURN\_COMMAND\_NAME**  
 wisco::autons::BlueMatchAuton, 102  
**MOTION\_RESUME\_TURN\_COMMAND\_NAME**  
 wisco::autons::BlueMatchAuton, 102  
**MOTION\_TURN\_TARGET\_REACHED\_STATE\_NAME**  
 wisco::autons::BlueMatchAuton, 102  
**MOTION\_TURN\_TO\_ANGLE\_COMMAND\_NAME**  
 wisco::autons::BlueMatchAuton, 101  
**MOTION\_TURN\_TO\_POINT\_COMMAND\_NAME**  
 wisco::autons::BlueMatchAuton, 101  
**motion\_velocity**  
 wisco::control::boomerang::PIDBoomerang, 167  
**MotionControl**  
 wisco::control::motion::MotionControl, 180  
**motionPauseTurn**  
 wisco::autons::BlueMatchAuton, 97  
**motionResumeTurn**  
 wisco::autons::BlueMatchAuton, 97  
**motionTurnTargetReached**  
 wisco::autons::BlueMatchAuton, 98  
**motionTurnToAngle**  
 wisco::autons::BlueMatchAuton, 96  
**motionTurnToPoint**  
 wisco::autons::BlueMatchAuton, 96  
**motors**  
 wisco::hal::MotorGroup, 229  
**mutex**  
 pros\_adapters::ProsController, 51  
 pros\_adapters::ProsMutex, 68  
**name**  
 wisco::menu::Option, 278  
**NEAR\_WALL**  
 wisco::robot::subsystems::position::DistancePositionResetter  
 457  
**new\_rumble\_pattern**  
 pros\_adapters::ProsController, 51  
**NO\_CARTRIDGE**  
 pros\_adapters::ProsV5Motor, 83  
**ODOMETRY\_GET\_POSITION\_STATE\_NAME**  
 wisco::autons::BlueMatchAuton, 101  
 wisco::control::boomerang::PIDBoomerang, 165  
 wisco::control::motion::PIDTurn, 195  
**ODOMETRY\_HEADING\_PORT**  
 wisco::configs::BlueConfiguration, 121  
**ODOMETRY\_HEADING\_TUNING\_CONSTANT**  
 wisco::configs::BlueConfiguration, 121  
**ODOMETRY\_LINEAR\_OFFSET**  
 wisco::configs::BlueConfiguration, 122  
**ODOMETRY\_LINEAR\_PORT**  
 wisco::configs::BlueConfiguration, 122  
**ODOMETRY\_LINEAR\_RADIUS**  
 wisco::configs::BlueConfiguration, 122  
**ODOMETRY\_SET\_POSITION\_COMMAND\_NAME**  
 wisco::autons::BlueMatchAuton, 100  
**ODOMETRY\_STRAFE\_OFFSET**  
 wisco::configs::BlueConfiguration, 123  
**ODOMETRY\_STRAFE\_PORT**  
 wisco::configs::BlueConfiguration, 122  
**ODOMETRY\_STRAFE\_RADIUS**  
 wisco::configs::BlueConfiguration, 122  
**ODOMETRY\_SUBSYSTEM\_NAME**  
 wisco::autons::BlueMatchAuton, 100  
 wisco::control::boomerang::PIDBoomerang, 165  
 wisco::control::motion::PITurn, 194  
**odometryGetPosition**  
 wisco::autons::BlueMatchAuton, 95  
**odometrySetPosition**  
 wisco::autons::BlueMatchAuton, 95  
**opcontrol\_manager**  
 wisco::MatchController, 261  
**OPControlManager**  
 wisco::OPControlManager, 279  
**open**  
 wisco::robot::subsystems::hang::IClaw, 377  
 wisco::robot::subsystems::hang::PistonClaw, 383  
**OPEN\_CLAW\_COMMAND\_NAME**  
 wisco::robot::subsystems::hang::HangSubsystem,  
 372  
 wisco::user::hang::HangOperator, 560  
**openClaw**  
 wisco::user::hang::HangOperator, 555  
**operator+**  
 wisco::control::path::Point, 206  
**operator+=**  
 wisco::control::path::Point, 207  
**operator-**  
 wisco::control::path::Point, 206  
**operator-=**  
 wisco::control::path::Point, 208  
**operator/**  
 wisco::control::path::Point, 207  
**operator/=**  
 wisco::control::path::Point, 209  
**operator=**  
 wisco::control::AControl, 147  
 wisco::control::path::Point, 209  
 wisco::control::path::QuinticBezier, 212, 213  
 wisco::control::PID, 218, 219  
 wisco::hal::MotorGroup, 229  
 wisco::hal::PistonGroup, 231  
 wisco::robot::ASubsystem, 295  
**operator\***  
 wisco::control::path, 19  
 wisco::control::path::Point, 207  
**operator\*=?**  
 wisco::control::path::Point, 208  
**operatorControl**  
 wisco::MatchController, 259

options  
  wisco::menu::LvglMenu, 271

PARTNER\_HANG\_POSITION  
  wisco::user::elevator::ElevatorOperator, 551

pause  
  wisco::control::boomerang::IBoomerang, 154  
  wisco::control::boomerang::PIDBoomerang, 161  
  wisco::control::motion::ITurn, 178  
  wisco::control::motion::PIDTurn, 191

PAUSE\_COMMAND\_NAME  
  wisco::control::boomerang::BoomerangControl, 152

PAUSE\_TURN\_COMMAND\_NAME  
  wisco::control::motion::MotionControl, 183

paused  
  wisco::control::boomerang::PIDBoomerang, 167  
  wisco::control::motion::PIDTurn, 197

PID  
  wisco::control::PID, 217

pistons  
  wisco::hal::PistonGroup, 231

Point  
  wisco::control::path::Point, 204

POLE\_HANG\_DISTANCE  
  wisco::user::elevator::ElevatorOperator, 551

POLE\_HANG\_POSITION  
  wisco::user::elevator::ElevatorOperator, 551

POSITION\_CONVERSION  
  pros\_adapters::ProsV5Motor, 84

PositionSubsystem  
  wisco::robot::subsystems::position::PositionSubsystem, 489

profile  
  wisco::SystemConfiguration, 525

PROFILE\_NAME  
  wisco::profiles::HenryProfile, 285  
  wisco::profiles::JohnProfile, 290

PROFILE\_OPTION\_NAME  
  wisco::menu::MenuAdapter, 276

pros\_adapters, 13

pros\_adapters::ProsClock, 42  
  clone, 43  
  getTime, 43

pros\_adapters::ProsController, 44  
  ANALOG\_CONVERSION, 49  
  ANALOG\_MAP, 50  
  DIGITAL\_MAP, 50  
  getAnalog, 47  
  getDigital, 48  
  getNewDigital, 48  
  initialize, 47  
  last\_rumble\_refresh, 51  
  m\_controller, 51  
  MAX\_RUMBLE\_LENGTH, 50  
  mutex, 51  
  new\_rumble\_pattern, 51  
  ProsController, 45  
  rumble, 49

rumble\_pattern, 51  
RUMBLE\_REFRESH\_RATE, 49  
run, 47  
TASK\_DELAY, 49  
taskLoop, 46  
taskUpdate, 46  
updateRumble, 46

pros\_adapters::ProsDelayer, 52  
  clone, 53  
  delay, 53  
  delayUntil, 53

pros\_adapters::ProsDistance, 54  
  getDistance, 56  
  initialize, 55  
  m\_sensor, 56  
  m\_tuning\_constant, 56  
  m\_tuning\_offset, 57  
  ProsDistance, 55  
  reset, 55  
  UNIT\_CONVERTER, 56

pros\_adapters::ProsEXPMotor, 57  
  ANGULAR\_VELOCITY\_CONSTANT, 61  
  GEAR\_RATIO, 61  
  getAngularVelocity, 60  
  getAngularVelocityConstant, 59  
  getGearRatio, 60  
  getResistance, 59  
  getTorqueConstant, 59  
  initialize, 59  
  m\_motor, 62  
  ProsEXPMotor, 58  
  RESISTANCE, 61  
  setVoltage, 60  
  TORQUE\_CONSTANT, 61  
  VELOCITY\_CONVERSION, 61  
  VOLTAGE\_CONVERSION, 62

pros\_adapters::ProsHeading, 62  
  getHeading, 64  
  getRotation, 65  
  initialize, 64  
  m\_sensor, 66  
  m\_tuning\_constant, 66  
  ProsHeading, 63  
  reset, 64  
  setHeading, 64  
  setRotation, 65  
  UNIT\_CONVERTER, 66

pros\_adapters::ProsMutex, 66  
  give, 67  
  mutex, 68  
  take, 67

pros\_adapters::ProsPiston, 68  
  extend, 69  
  extended, 71  
  isExtended, 70  
  isRetracted, 70  
  m\_digital\_out, 71  
  m\_extended\_value, 71

ProsPiston, 69  
 retract, 69  
 toggle, 70  
**pros\_adapters::ProsRotation**, 72  
 getAngle, 74  
 getRotation, 73  
 initialize, 73  
 m\_sensor, 75  
**ProsRotation**, 73  
 reset, 73  
 setRotation, 74  
**UNIT\_CONVERSION**, 75  
**pros\_adapters::ProsTask**, 75  
 join, 77  
 remove, 76  
 resume, 77  
 start, 76  
 suspend, 76  
 task, 77  
**pros\_adapters::ProsV5Motor**, 78  
 ANGULAR\_VELOCITY\_CONSTANT, 83  
 cartridge\_map, 82  
 getAngularVelocity, 81  
 getAngularVelocityConstant, 80  
 getGearRatio, 81  
 getPosition, 81  
 getResistance, 80  
 getTorqueConstant, 80  
 initialize, 80  
 m\_motor, 84  
 MAX\_MILLIVOLTS, 84  
 NO\_CARTRIDGE, 83  
 POSITION\_CONVERSION, 84  
**ProsV5Motor**, 79  
 RESISTANCE, 83  
 setVoltage, 82  
 TORQUE\_CONSTANT, 83  
 VELOCITY\_CONVERSION, 83  
 VOLTAGE\_CONVERSION, 84  
**ProsController**  
 pros\_adapters::ProsController, 45  
**ProsDistance**  
 pros\_adapters::ProsDistance, 55  
**ProsEXPMotor**  
 pros\_adapters::ProsEXPMotor, 58  
**ProsHeading**  
 pros\_adapters::ProsHeading, 63  
**ProsPiston**  
 pros\_adapters::ProsPiston, 69  
**ProsRotation**  
 pros\_adapters::ProsRotation, 73  
**ProsV5Motor**  
 pros\_adapters::ProsV5Motor, 79  
**QuinticBezier**  
 wisco::control::path::QuinticBezier, 211, 212  
**RAISE\_ARM\_COMMAND\_NAME**  
 wisco::robot::subsystems::hang::HangSubsystem, 373  
 wisco::user::hang::HangOperator, 560  
**raiseArm**  
 wisco::user::hang::HangOperator, 555  
**readConfiguration**  
 wisco::menu::LvgIMenu, 267  
**remove**  
 pros\_adapters::ProsTask, 76  
 wisco::rtos::ITask, 524  
**removeControl**  
 wisco::control::ControlSystem, 174  
**removeOption**  
 wisco::menu::LvgIMenu, 264  
**removeSubsystem**  
 wisco::robot::Robot, 297  
**reset**  
 pros\_adapters::ProsDistance, 55  
 pros\_adapters::ProsHeading, 64  
 pros\_adapters::ProsRotation, 73  
 wisco::control::PID, 218  
 wisco::hal::DistanceBooleanSensor, 223  
 wisco::hal::TrackingWheel, 233  
 wisco::io::IBooleanSensor, 243  
 wisco::io::IDistanceSensor, 244  
 wisco::io::IDistanceTrackingSensor, 245  
 wisco::io::IHeadingSensor, 247  
 wisco::io::IRotationSensor, 253  
**RESET\_X\_COMMAND\_NAME**  
 wisco::robot::subsystems::position::PositionSubsystem, 491  
**RESET\_Y\_COMMAND\_NAME**  
 wisco::robot::subsystems::position::PositionSubsystem, 491  
**RESETTER\_DISTANCE\_CONSTANT**  
 wisco::configs::BlueConfiguration, 123  
**RESETTER\_DISTANCE\_OFFSET**  
 wisco::configs::BlueConfiguration, 123  
**RESETTER\_DISTANCE\_PORT**  
 wisco::configs::BlueConfiguration, 123  
**RESETTER\_OFFSET\_THETA**  
 wisco::configs::BlueConfiguration, 124  
**RESETTER\_OFFSET\_X**  
 wisco::configs::BlueConfiguration, 123  
**RESETTER\_OFFSET\_Y**  
 wisco::configs::BlueConfiguration, 124  
**RESISTANCE**  
 pros\_adapters::ProsEXPMotor, 61  
 pros\_adapters::ProsV5Motor, 83  
**resume**  
 pros\_adapters::ProsTask, 77  
 wisco::control::boomerang::IBoomerang, 154  
 wisco::control::boomerang::PIDBoomerang, 161  
 wisco::control::motion::ITurn, 178  
 wisco::control::motion::PIDTurn, 191  
 wisco::rtos::ITask, 524  
**RESUME\_COMMAND\_NAME**

wisco::control::boomerang::BoomerangControl, 152  
RESUME\_TURN\_COMMAND\_NAME  
wisco::control::motion::MotionControl, 183  
retract  
pros\_adapters::ProsPiston, 69  
wisco::io::IPiston, 252  
RIGHT\_DRIVE\_PORTS  
wisco::testing::TestFactory, 533  
right\_toggle\_state  
wisco::user::wings::WingsOperator, 590  
right\_velocity  
wisco::robot::subsystems::drive::Velocity, 346  
RIGHT\_WING\_PISTON\_1\_EXTENDED\_STATE  
wisco::configs::BlueConfiguration, 141  
RIGHT\_WING\_PISTON\_1\_PORT  
wisco::configs::BlueConfiguration, 141  
robot  
wisco::MatchController, 261  
rumble  
pros\_adapters::ProsController, 49  
wisco::user::IController, 564  
rumble\_pattern  
pros\_adapters::ProsController, 51  
RUMBLE\_REFRESH\_RATE  
pros\_adapters::ProsController, 49  
run  
pros\_adapters::ProsController, 47  
wisco::autons::BlueMatchAuton, 99  
wisco::autons::BlueSkillsAuton, 104  
wisco::autons::OrangeMatchAuton, 106  
wisco::autons::OrangeSkillsAuton, 109  
wisco::control::AControl, 146  
wisco::control::boomerang::BoomerangControl, 150  
wisco::control::boomerang::IBoomerang, 153  
wisco::control::boomerang::PIDBoomerang, 160  
wisco::control::motion::ITurn, 177  
wisco::control::motion::MotionControl, 181  
wisco::control::motion::PIDTurn, 189  
wisco::IAutonomous, 237  
wisco::robot::ASubsystem, 294  
wisco::robot::subsystems::drive::DifferentialDriveSubsystems, 304  
wisco::robot::subsystems::drive::DirectDifferentialDrive, 308  
wisco::robot::subsystems::drive::IDifferentialDrive, 319  
wisco::robot::subsystems::drive::KinematicDifferentialDrive, 326  
wisco::robot::subsystems::elevator::ElevatorSubsystem, 349  
wisco::robot::subsystems::elevator::IElevator, 352  
wisco::robot::subsystems::elevator::PIDElevator, 355  
wisco::robot::subsystems::hang::HangSubsystem, 370  
wisco::robot::subsystems::hang::IClaw, 376  
wisco::robot::subsystems::hang::IToggleArm, 378  
wisco::robot::subsystems::hang::IWinch, 380  
wisco::robot::subsystems::hang::PistonClaw, 382  
wisco::robot::subsystems::hang::PistonToggleArm, 389  
wisco::robot::subsystems::hang::PistonWinch, 395  
wisco::robot::subsystems::intake::DistanceVisionBallDetector, 401  
wisco::robot::subsystems::intake::IBallDetector, 405  
wisco::robot::subsystems::intake::IIntake, 406  
wisco::robot::subsystems::intake::IntakeSubsystem, 410  
wisco::robot::subsystems::intake::PIDIntake, 416  
wisco::robot::subsystems::loader::ILoader, 429  
wisco::robot::subsystems::loader::LoaderSubsystem, 432  
wisco::robot::subsystems::loader::PIDLoader, 439  
wisco::robot::subsystems::position::DistancePositionResetter, 454  
wisco::robot::subsystems::position::InertialOdometry, 466  
wisco::robot::subsystems::position::IPositionResetter, 483  
wisco::robot::subsystems::position::IPositionTracker, 484  
wisco::robot::subsystems::position::PositionSubsystem, 489  
wisco::robot::subsystems::umbrella::IUmbrella, 493  
wisco::robot::subsystems::umbrella::PistonUmbrella, 495  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem, 502  
wisco::robot::subsystems::wings::IWings, 505  
wisco::robot::subsystems::wings::PistonWings, 508  
wisco::robot::subsystems::wings::WingsSubsystem, 516  
wisco::user::IController, 563  
runAutonomous  
wisco::AutonomousManager, 90  
runInfraredTest  
wisco::testing::pros\_testing::DriveTest, 528  
runOpcontrol  
wisco::OPControlManager, 280  
runTurningTest  
wisco::testing::pros\_testing::DriveTest, 528  
selected  
wisco::menu::Option, 278  
selectionComplete  
wisco::menu::LvglMenu, 268  
sendCommand  
wisco::control::ControlSystem, 175  
wisco::robot::Robot, 297  
SET\_IN\_COMMAND  
wisco::user::umbrella::UmbrellaOperator, 581  
SET\_IN\_COMMAND\_NAME

wisco::robot::subsystems::umbrella::UmbrellaSubsystem, wisco::profiles::JohnProfile, 288  
**503**

**SET\_LEFT\_WING\_COMMAND**  
wisco::user::wings::WingsOperator, 589

**SET\_LEFT\_WING\_COMMAND\_NAME**  
wisco::robot::subsystems::wings::WingsSubsystem,  
**517**

**SET\_OUT\_COMMAND**  
wisco::user::umbrella::UmbrellaOperator, 581

**SET\_OUT\_COMMAND\_NAME**  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem, wisco::robot::subsystems::position::InertialOdometry,  
**503**

**SET\_POSITION\_COMMAND**  
wisco::user::elevator::ElevatorOperator, 549

**SET\_POSITION\_COMMAND\_NAME**  
wisco::robot::subsystems::elevator::ElevatorSubsystem  
**350**

wisco::robot::subsystems::position::PositionSubsystem,  
**491**

**SET\_RIGHT\_WING\_COMMAND**  
wisco::user::wings::WingsOperator, 589

**SET\_RIGHT\_WING\_COMMAND\_NAME**  
wisco::robot::subsystems::wings::WingsSubsystem,  
**518**

**SET\_VELOCITY\_COMMAND\_NAME**  
wisco::robot::subsystems::drive::DifferentialDriveSubsystem  
**305**

wisco::robot::subsystems::intake::IntakeSubsystem, **411**

**SET\_VOLTAGE\_COMMAND**  
wisco::user::drive::DifferentialDriveOperator, 538

wisco::user::intake::IntakeOperator, 569

**SET\_VOLTAGE\_COMMAND\_NAME**  
wisco::robot::subsystems::drive::DifferentialDriveSubsystem  
**306**

wisco::robot::subsystems::intake::IntakeSubsystem,  
**412**

**setAcceleration**  
wisco::robot::subsystems::drive::CurveVelocityProfile, **301**

wisco::robot::subsystems::drive::IVelocityProfile,  
**321**

**setAutonomous**  
wisco::AutonomousManager, 90

**setClock**  
wisco::robot::subsystems::elevator::PIDElevator,  
**357**

wisco::robot::subsystems::intake::PIDIntake, 417

wisco::robot::subsystems::loader::PIDLoader, 440

wisco::robot::subsystems::position::InertialOdometry,  
**467**

**setClosedState**  
wisco::robot::subsystems::hang::PistonClaw, 384

**setComplete**  
wisco::menu::LvglMenu, 267

**setControlMode**  
wisco::IProfile, 256

wisco::profiles::HenryProfile, 284

**setDelay**  
wisco::control::boomerang::PIDBoomerang, 162

wisco::control::motion::PIDTurn, 192

wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
**328**

wisco::robot::subsystems::elevator::PIDElevator,  
**357**

wisco::robot::subsystems::intake::PIDIntake, 417

wisco::robot::subsystems::loader::PIDLoader, 441

wisco::robot::subsystems::position::InertialOdometry,  
**467**

**setDistance**  
wisco::hal::TrackingWheel, 234

wisco::io::IDistanceTrackingSensor, 245

**setDistanceSensor**  
wisco::robot::subsystems::intake::DistanceVisionBallDetector,  
**402**

wisco::robot::subsystems::position::DistancePositionResetter,  
**456**

**setDown**  
wisco::robot::subsystems::hang::IToggleArm, 378

wisco::robot::subsystems::hang::PistonToggleArm,  
**389**

**setDriveVelocity**  
wisco::user::drive::DifferentialDriveOperator, 537

**setElevatorPosition**  
wisco::user::elevator::ElevatorOperator, 548

**setEngagedState**  
wisco::robot::subsystems::hang::PistonWinch, 397

**setGearRatio**  
wisco::robot::subsystems::drive::DirectDifferentialDrive,  
**311**

wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
**332**

**setGrabbedState**  
wisco::user::hang::HangOperator, 556

**setHangState**  
wisco::user::hang::HangOperator, 559

**setHeading**  
pros\_adapters::ProsHeading, 64

wisco::io::IHeadingSensor, 247

**setHeadingSensor**  
wisco::robot::subsystems::position::InertialOdometry,  
**468**

**setHungState**  
wisco::user::hang::HangOperator, 556

**setIn**  
wisco::robot::subsystems::umbrella::IUmbrella,  
**493**

wisco::robot::subsystems::umbrella::PistonUmbrella,  
**495**

wisco::user::umbrella::UmbrellaOperator, 579

**setInactiveState**  
wisco::user::hang::HangOperator, 556

setInchesPerRadian  
    wisco::robot::subsystems::elevator::PIDElevator,  
        359

setIntakeVoltage  
    wisco::user::intake::IntakeOperator, 568

setLead  
    wisco::control::boomerang::PIDBoomerang, 164

setLeftMotors  
    wisco::robot::subsystems::drive::DirectDifferentialDrive,  
        310

    wisco::robot::subsystems::drive::KinematicDifferentialDrive  
        wisco::user::umbrella::UmbrellaOperator, 579

    329

setLeftPistons  
    wisco::robot::subsystems::wings::PistonWings,  
        509

setLeftWing  
    wisco::robot::subsystems::wings::IWings, 505

    wisco::robot::subsystems::wings::PistonWings,  
        508

    wisco::user::wings::WingsOperator, 584

setLinearDistanceTrackingOffset  
    wisco::robot::subsystems::position::InertialOdometry,  
        470

setLinearDistanceTrackingSensor  
    wisco::robot::subsystems::position::InertialOdometry,  
        470

setLinearPID  
    wisco::control::boomerang::PIDBoomerang, 163

setLoaderPosition  
    wisco::user::loader::LoaderOperator, 575

setLocalTheta  
    wisco::robot::subsystems::position::DistancePositionResetter  
        457

setLocalX  
    wisco::robot::subsystems::position::DistancePositionResetter  
        456

setLocalY  
    wisco::robot::subsystems::position::DistancePositionResetter  
        456

setMass  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive  
        330

setMatchLoadPosition  
    wisco::robot::subsystems::loader::PIDLoader, 442

setMomentOfInertia  
    wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
        330

setMotors  
    wisco::robot::subsystems::elevator::PIDElevator,  
        358

    wisco::robot::subsystems::intake::PIDIntake, 420

    wisco::robot::subsystems::loader::PIDLoader, 442

setMutex  
    wisco::control::boomerang::PIDBoomerang, 162

    wisco::control::motion::PIDTurn, 192

    wisco::robot::subsystems::drive::KinematicDifferentialDrive  
        328

    wisco::robot::subsystems::elevator::PIDElevator,

setOut  
    wisco::robot::subsystems::umbrella::IUmbrella,  
        493

setOutState  
    wisco::robot::subsystems::umbrella::PistonUmbrella,  
        496

    wisco::robot::subsystems::wings::PistonWings,  
        510

setPID  
    wisco::control::motion::PIDTurn, 193

    wisco::robot::subsystems::elevator::PIDElevator,  
        358

    wisco::robot::subsystems::intake::PIDIntake, 418

    wisco::robot::subsystems::loader::PIDLoader, 442

setPistons  
    wisco::robot::subsystems::hang::PistonClaw, 384

    wisco::robot::subsystems::hang::PistonToggleArm,  
        390

    wisco::robot::subsystems::hang::PistonWinch, 396

    wisco::robot::subsystems::umbrella::PistonUmbrella,  
        496

setPosition  
    wisco::robot::subsystems::elevator::IElevator, 352

    wisco::robot::subsystems::elevator::PIDElevator,  
        356

    wisco::robot::subsystems::position::InertialOdometry,  
        466

    wisco::robot::subsystems::position::IPositionTracker,  
        485

setPositionTolerance  
    wisco::robot::subsystems::loader::PIDLoader, 443

setProfile  
    wisco::OPControlManager, 280

setRadius  
    wisco::robot::subsystems::drive::DirectDifferentialDrive,  
        311

    wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
        330

setRaisedState  
    wisco::user::hang::HangOperator, 556

setReadyPosition  
    wisco::robot::subsystems::loader::PIDLoader, 442

setRightMotors  
    wisco::robot::subsystems::drive::DirectDifferentialDrive,  
        310

    wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
        329

setRightPistons  
    wisco::robot::subsystems::wings::PistonWings,  
        510

setRightWing  
 wisco::robot::subsystems::wings::IWings, 506  
 wisco::robot::subsystems::wings::PistonWings,  
 509  
 wisco::user::wings::WingsOperator, 584

setRollerRadius  
 wisco::robot::subsystems::intake::PIDIntake, 420

setRotation  
 pros\_adapters::ProsHeading, 65  
 pros\_adapters::ProsRotation, 74  
 wisco::io::IHeadingSensor, 248  
 wisco::io::IRotationSensor, 254

setRotationalPID  
 wisco::control::boomerang::PIDBoomerang, 163

setRotationSensor  
 wisco::robot::subsystems::elevator::PIDElevator,  
 358

setState  
 wisco::hal::PistonGroup, 230

setStrafeDistanceTrackingOffset  
 wisco::robot::subsystems::position::InertialOdometry,  
 471

setStrafeDistanceTrackingSensor  
 wisco::robot::subsystems::position::InertialOdometry,  
 470

setTargetTolerance  
 wisco::control::boomerang::PIDBoomerang, 164  
 wisco::control::motion::PIDTurn, 193

setTargetVelocity  
 wisco::control::motion::PIDTurn, 194

setTask  
 wisco::control::boomerang::PIDBoomerang, 163  
 wisco::control::motion::PIDTurn, 193  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive  
 328  
 wisco::robot::subsystems::elevator::PIDElevator,  
 358  
 wisco::robot::subsystems::intake::PIDIntake, 418  
 wisco::robot::subsystems::loader::PIDLoader, 441  
 wisco::robot::subsystems::position::InertialOdometry,  
 468

settingsBackButtonEventHandler  
 wisco::menu, 22

settingsButtonEventHandler  
 wisco::menu, 22

settingsButtonMatrixEventHandler  
 wisco::menu, 23

setUmbrellaPosition  
 wisco::user::umbrella::UmbrellaOperator, 580

setUp  
 wisco::robot::subsystems::hang::IToggleArm, 379  
 wisco::robot::subsystems::hang::PistonToggleArm,  
 390

setUpState  
 wisco::robot::subsystems::hang::PistonToggleArm,  
 391

setVelocity

wisco::robot::subsystems::drive::DirectDifferentialDrive,  
 309  
 wisco::robot::subsystems::drive::IDifferentialDrive,  
 319  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
 327  
 wisco::robot::subsystems::intake::IIntake, 407  
 wisco::robot::subsystems::intake::PIDIntake, 416

setVelocityProfiles  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
 329

setVelocityToVoltage  
 wisco::robot::subsystems::drive::DirectDifferentialDrive,  
 311

setVoltage  
 pros\_adapters::ProsEXPMotor, 60  
 pros\_adapters::ProsV5Motor, 82  
 wisco::hal::MotorGroup, 228  
 wisco::io::IMotor, 250  
 wisco::robot::subsystems::drive::DirectDifferentialDrive,  
 309  
 wisco::robot::subsystems::drive::IDifferentialDrive,  
 320  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
 327  
 wisco::robot::subsystems::intake::IIntake, 407  
 wisco::robot::subsystems::intake::PIDIntake, 417

setWheelRadius  
 wisco::robot::subsystems::drive::DirectDifferentialDrive,  
 311  
 wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
 332

setWings  
 wisco::user::wings::WingsOperator, 584

setWingsPosition  
 wisco::user::wings::WingsOperator, 588

setX  
 wisco::control::path::Point, 205

setY  
 wisco::control::path::Point, 205

start  
 pros\_adapters::ProsTask, 76  
 wisco::rtos::ITask, 523

startButtonEventHandler  
 wisco::menu, 21

state  
 wisco::control::AControl, 147  
 wisco::control::boomerang::BoomerangControl,  
 151  
 wisco::control::motion::MotionControl, 182  
 wisco::robot::ASubsystem, 294  
 wisco::robot::subsystems::drive::DifferentialDriveSubsystem,  
 305  
 wisco::robot::subsystems::elevator::ElevatorSubsystem,  
 349  
 wisco::robot::subsystems::hang::HangSubsystem,  
 371  
 wisco::robot::subsystems::intake::IntakeSubsystem,

411  
wisco::robot::subsystems::loader::LoaderSubsystem, task  
433  
wisco::robot::subsystems::loader::PIDLoader, 445  
wisco::robot::subsystems::position::PositionSubsystem,  
490  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
502  
wisco::robot::subsystems::wings::WingsSubsystem,  
517  
styles\_initialized  
wisco::menu::LvglMenu, 270  
SUBSYSTEM\_NAME  
wisco::robot::subsystems::drive::DifferentialDriveSubsystem  
wisco::robot::subsystems::position::InertialOdometry,  
305  
wisco::robot::subsystems::elevator::ElevatorSubsystem  
taskLoop  
350  
wisco::robot::subsystems::hang::HangSubsystem,  
372  
wisco::robot::subsystems::intake::IntakeSubsystem,  
411  
wisco::robot::subsystems::loader::LoaderSubsystem,  
433  
wisco::robot::subsystems::position::PositionSubsystem,  
491  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
503  
wisco::robot::subsystems::wings::WingsSubsystem, taskUpdate  
517  
wisco::user::loader::LoaderOperator, 575  
subsystems  
wisco::robot::Robot, 298  
suspend  
pros\_adapters::ProsTask, 76  
wisco::rtos::ITask, 524  
take  
pros\_adapters::ProsMutex, 67  
wisco::rtos::IMutex, 522  
target\_position  
wisco::robot::subsystems::loader::PIDLoader, 445  
target\_reached  
wisco::control::boomerang::PIDBoomerang, 167  
wisco::control::motion::PIDTurn, 197  
TARGET\_REACHED\_STATE\_NAME  
wisco::control::boomerang::BoomerangControl,  
152  
target\_theta  
wisco::control::boomerang::PIDBoomerang, 167  
target\_x  
wisco::control::boomerang::PIDBoomerang, 167  
wisco::control::motion::PIDTurn, 197  
target\_y  
wisco::control::boomerang::PIDBoomerang, 167  
wisco::control::motion::PIDTurn, 197  
targetReached  
wisco::control::boomerang::IBoomerang, 154  
wisco::control::boomerang::PIDBoomerang, 162  
wisco::control::motion::ITurn, 178  
wisco::control::motion::PIDTurn, 192  
pros\_adapters::ProsTask, 77  
TASK\_DELAY  
pros\_adapters::ProsController, 49  
wisco::control::boomerang::PIDBoomerang, 164  
wisco::control::motion::PIDTurn, 194  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
332  
wisco::robot::subsystems::elevator::PDElevator,  
359  
wisco::robot::subsystems::intake::PIDIntake, 420  
wisco::robot::subsystems::loader::PIDLoader, 443  
wisco::robot::subsystems::position::InertialOdometry,  
471  
pros\_adapters::ProsController, 46  
wisco::control::boomerang::PIDBoomerang, 157  
wisco::control::motion::PIDTurn, 186  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
324  
wisco::robot::subsystems::elevator::PDElevator,  
354  
wisco::robot::subsystems::intake::PIDIntake, 415  
wisco::robot::subsystems::loader::PIDLoader, 437  
wisco::robot::subsystems::position::InertialOdometry,  
464  
pros\_adapters::ProsController, 46  
wisco::control::boomerang::PIDBoomerang, 157  
wisco::control::motion::PIDTurn, 186  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
325  
wisco::robot::subsystems::elevator::PDElevator,  
355  
wisco::robot::subsystems::intake::PIDIntake, 415  
wisco::robot::subsystems::loader::PIDLoader, 437  
wisco::robot::subsystems::position::InertialOdometry,  
465  
TEST\_DURATION  
wisco::testing::pros\_testing::DriveTest, 530  
TEST\_V  
wisco::testing::pros\_testing::DriveTest, 530  
theta  
wisco::robot::subsystems::position::Position, 486  
thetaV  
wisco::robot::subsystems::position::Position, 487  
TIME\_UNIT\_CONVERTER  
wisco::robot::subsystems::position::InertialOdometry,  
471  
toggle  
pros\_adapters::ProsPiston, 70  
wisco::io::IPiston, 252  
toggle\_state  
wisco::user::elevator::ElevatorOperator, 552  
wisco::user::hang::HangOperator, 561  
wisco::user::intake::IntakeOperator, 569  
wisco::user::loader::LoaderOperator, 576

wisco::user::umbrella::UmbrellaOperator, 582  
wisco::user::wings::WingsOperator, 589  
**TORQUE\_CONSTANT**  
pros\_adapters::ProsEXPMotor, 61  
pros\_adapters::ProsV5Motor, 83  
**TrackingWheel**  
wisco::hal::TrackingWheel, 233  
**turn\_direction**  
wisco::control::motion::PIDTurn, 197  
**TURN\_KD**  
wisco::configs::BlueConfiguration, 139  
**TURN\_KI**  
wisco::configs::BlueConfiguration, 139  
**TURN\_KP**  
wisco::configs::BlueConfiguration, 139  
**TURN\_TARGET\_REACHED\_STATE\_NAME**  
wisco::control::motion::MotionControl, 183  
**TURN\_TARGET\_TOLERANCE**  
wisco::configs::BlueConfiguration, 139  
**TURN\_TARGET\_VELOCITY**  
wisco::configs::BlueConfiguration, 139  
**TURN\_TO\_ANGLE\_COMMAND\_NAME**  
wisco::control::motion::MotionControl, 182  
**TURN\_TO\_ANGLE\_DISTANCE**  
wisco::control::motion::PIDTurn, 195  
**TURN\_TO\_POINT\_COMMAND\_NAME**  
wisco::control::motion::MotionControl, 183  
**turn\_velocity**  
wisco::control::motion::PIDTurn, 197  
**TURNING\_FILE\_NAME**  
wisco::testing::pros\_testing::DriveTest, 529  
**turnToAngle**  
wisco::control::motion::ITurn, 177  
wisco::control::motion::PIDTurn, 190  
**turnToPoint**  
wisco::control::motion::ITurn, 178  
wisco::control::motion::PIDTurn, 190  
**UMBRELLA\_OUT\_STATE**  
wisco::configs::BlueConfiguration, 140  
**UMBRELLA\_PISTON\_1\_EXTENDED\_STATE**  
wisco::configs::BlueConfiguration, 140  
**UMBRELLA\_PISTON\_1\_PORT**  
wisco::configs::BlueConfiguration, 140  
**UMBRELLA\_SUBSYSTEM\_NAME**  
wisco::user::umbrella::UmbrellaOperator, 581  
**UmbrellaOperator**  
wisco::user::umbrella::UmbrellaOperator, 578  
**UmbrellaSubsystem**  
wisco::robot::subsystems::umbrella::UmbrellaSubsystem, 501  
**UNIT\_CONVERSION**  
pros\_adapters::ProsRotation, 75  
**UNIT\_CONVERTER**  
pros\_adapters::ProsDistance, 56  
pros\_adapters::ProsHeading, 66  
**updateAcceleration**  
wisco::robot::subsystems::drive::KinematicDifferentialDrive, 325  
**updateArcade**  
wisco::user::drive::DifferentialDriveOperator, 536  
**updateDriveVoltage**  
wisco::user::drive::DifferentialDriveOperator, 536  
**updateDualHold**  
wisco::user::wings::WingsOperator, 585  
**updateDualSplit**  
wisco::user::wings::WingsOperator, 585  
**updateDualToggle**  
wisco::user::wings::WingsOperator, 585  
**updateElevatorPosition**  
wisco::user::elevator::ElevatorOperator, 543  
**updateHold**  
wisco::user::loader::LoaderOperator, 573  
wisco::user::umbrella::UmbrellaOperator, 579  
**updateIntakeVoltage**  
wisco::user::intake::IntakeOperator, 566  
**updateMacro**  
wisco::user::loader::LoaderOperator, 573  
**updateManual**  
wisco::user::elevator::ElevatorOperator, 543  
**updatePoleHangPosition**  
wisco::user::elevator::ElevatorOperator, 543  
**updatePosition**  
wisco::robot::subsystems::elevator::PDElevator, 355  
wisco::robot::subsystems::loader::PIDLoader, 438  
wisco::robot::subsystems::position::InertialOdometry, 465  
**updatePresetLadder**  
wisco::user::elevator::ElevatorOperator, 546  
wisco::user::hang::HangOperator, 558  
**updatePresetLadderIntake**  
wisco::user::elevator::ElevatorOperator, 547  
**updatePresetReset**  
wisco::user::hang::HangOperator, 559  
**updatePresetSplit**  
wisco::user::elevator::ElevatorOperator, 545  
wisco::user::hang::HangOperator, 557  
**updatePresetToggle**  
wisco::user::elevator::ElevatorOperator, 546  
wisco::user::hang::HangOperator, 557  
**updateRumble**  
pros\_adapters::ProsController, 46  
**updateSingleArcadeLeft**  
wisco::user::drive::DifferentialDriveOperator, 536  
**updateSingleArcadeRight**  
wisco::user::drive::DifferentialDriveOperator, 536  
**updateSingleHold**  
wisco::user::wings::WingsOperator, 586  
**updateSingleSplit**  
wisco::user::wings::WingsOperator, 586  
**updateSingleToggle**  
wisco::user::intake::IntakeOperator, 567  
wisco::user::loader::LoaderOperator, 574  
wisco::user::wings::WingsOperator, 587  
**updateSplit**  
wisco::user::umbrella::UmbrellaOperator, 579

updateSplitArcadeLeft  
    wisco::user::drive::DifferentialDriveOperator, 537  
updateSplitArcadeRight  
    wisco::user::drive::DifferentialDriveOperator, 537  
updateSplitHold  
    wisco::user::intake::IntakeOperator, 567  
updateSplitToggle  
    wisco::user::intake::IntakeOperator, 567  
    wisco::user::loader::LoaderOperator, 574  
updateState  
    wisco::robot::subsystems::loader::PIDLoader, 438  
updateTank  
    wisco::user::drive::DifferentialDriveOperator, 537  
updateToggle  
    wisco::user::umbrella::UmbrellaOperator, 580  
updateToggleVoltage  
    wisco::user::intake::IntakeOperator, 566  
updateVelocity  
    wisco::control::boomerang::PIDBoomerang, 159  
    wisco::robot::subsystems::intake::PIDIntake, 415

V\_TO\_MV  
    wisco::testing::pros\_testing::DriveTest, 530

value  
    wisco::hal::DistanceBooleanSensor, 225

velocity\_control  
    wisco::robot::subsystems::intake::PIDIntake, 422

VELOCITY\_CONVERSION  
    pros\_adapters::ProsEXPMotor, 61  
    pros\_adapters::ProsV5Motor, 83

VOLTAGE\_CONVERSION  
    pros\_adapters::ProsEXPMotor, 62  
    pros\_adapters::ProsV5Motor, 84  
    wisco::user::drive::DifferentialDriveOperator, 538

VOLTAGE\_SETTING  
    wisco::user::intake::IntakeOperator, 569

WINCH\_DISENGAGED\_STATE\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem,  
        374

WINCH\_ENGAGED\_STATE\_NAME  
    wisco::robot::subsystems::hang::HangSubsystem,  
        374

WINGS\_OUT\_STATE  
    wisco::configs::BlueConfiguration, 141

WINGS\_SUBSYSTEM\_NAME  
    wisco::user::wings::WingsOperator, 588

WingsOperator  
    wisco::user::wings::WingsOperator, 583

WingsSubsystem  
    wisco::robot::subsystems::wings::WingsSubsystem,  
        515

wisco, 14

wisco::alliances, 15

wisco::alliances::BlueAlliance, 85  
    ALLIANCE\_NAME, 86  
    getName, 85

wisco::alliances::RedAlliance, 86  
    ALLIANCE\_NAME, 87

    getName, 87

wisco::alliances::SkillsAlliance, 87  
    ALLIANCE\_NAME, 88  
    getName, 88

wisco::AutonomousManager, 89  
    AutonomousManager, 89  
    initializeAutonomous, 90  
    m\_autonomous, 91  
    m\_clock, 91  
    m\_delayer, 91  
    runAutonomous, 90  
    setAutonomous, 90

wisco::autons, 15

wisco::autons::BlueMatchAuton, 91  
    AUTONOMOUS\_NAME, 99  
    BOOMERANG\_CONTROL\_NAME, 99  
    BOOMERANG\_GO\_TO\_POSITION\_COMMAND\_NAME,  
        100  
    BOOMERANG\_PAUSE\_COMMAND\_NAME, 100  
    BOOMERANG\_RESUME\_COMMAND\_NAME,  
        100  
    BOOMERANG\_TARGET\_REACHED\_STATE\_NAME,  
        101  
    boomerangGoToPoint, 93  
    boomerangPause, 94  
    boomerangResume, 94  
    boomerangTargetReached, 94  
    getName, 98  
    initialize, 98  
    MOTION\_CONTROL\_NAME, 101  
    MOTION\_PAUSE\_TURN\_COMMAND\_NAME, 102  
    MOTION\_RESUME\_TURN\_COMMAND\_NAME,  
        102  
    MOTION\_TURN\_TARGET\_REACHED\_STATE\_NAME,  
        102  
    MOTION\_TURN\_TO\_ANGLE\_COMMAND\_NAME,  
        101  
    MOTION\_TURN\_TO\_POINT\_COMMAND\_NAME,  
        101  
    motionPauseTurn, 97  
    motionResumeTurn, 97  
    motionTurnTargetReached, 98  
    motionTurnToAngle, 96  
    motionTurnToPoint, 96  
    ODOMETRY\_GET\_POSITION\_STATE\_NAME,  
        101  
    ODOMETRY\_SET\_POSITION\_COMMAND\_NAME,  
        100  
    ODOMETRY\_SUBSYSTEM\_NAME, 100  
    odometryGetPosition, 95  
    odometrySetPosition, 95  
    run, 99

wisco::autons::BlueSkillsAuton, 102  
    AUTONOMOUS\_NAME, 104  
    getName, 103  
    initialize, 103  
    run, 104

wisco::autons::OrangeMatchAuton, 105

AUTONOMOUS\_NAME, 107  
 getName, 106  
 initialize, 106  
 run, 106  
 wisco::autons::OrangeSkillsAuton, 107  
   AUTONOMOUS\_NAME, 109  
   getName, 108  
   initialize, 108  
   run, 109  
 wisco::configs, 16  
 wisco::configs::BlueConfiguration, 109  
   BALL\_DETECTOR\_DISTANCE\_CONSTANT, 131  
   BALL\_DETECTOR\_DISTANCE\_OFFSET, 131  
   BALL\_DETECTOR\_DISTANCE\_PORT, 131  
   BOOMERANG\_LEAD, 132  
   BOOMERANG\_LINEAR\_KD, 132  
   BOOMERANG\_LINEAR\_KI, 131  
   BOOMERANG\_LINEAR\_KP, 131  
   BOOMERANG\_ROTATIONAL\_KD, 132  
   BOOMERANG\_ROTATIONAL\_KI, 132  
   BOOMERANG\_ROTATIONAL\_KP, 132  
   BOOMERANG\_TARGET\_TOLERANCE, 133  
   buildController, 115  
   buildControlSystem, 114  
   buildRobot, 116  
   CONFIGURATION\_NAME, 121  
   DRIVE\_GEAR\_RATIO, 129  
   DRIVE\_KINEMATIC, 124  
   DRIVE\_LEFT\_MOTOR\_1\_GEARSET, 125  
   DRIVE\_LEFT\_MOTOR\_1\_PORT, 125  
   DRIVE\_LEFT\_MOTOR\_2\_GEARSET, 125  
   DRIVE\_LEFT\_MOTOR\_2\_PORT, 125  
   DRIVE\_LEFT\_MOTOR\_3\_GEARSET, 126  
   DRIVE\_LEFT\_MOTOR\_3\_PORT, 125  
   DRIVE\_LEFT\_MOTOR\_4\_GEARSET, 126  
   DRIVE\_LEFT\_MOTOR\_4\_PORT, 126  
   DRIVE\_MASS, 128  
   DRIVE\_MOMENT\_OF\_INERTIA, 128  
   DRIVE\_RADIUS, 128  
   DRIVE\_RIGHT\_MOTOR\_1\_GEARSET, 126  
   DRIVE\_RIGHT\_MOTOR\_1\_PORT, 126  
   DRIVE\_RIGHT\_MOTOR\_2\_GEARSET, 127  
   DRIVE\_RIGHT\_MOTOR\_2\_PORT, 127  
   DRIVE\_RIGHT\_MOTOR\_3\_GEARSET, 127  
   DRIVE\_RIGHT\_MOTOR\_3\_PORT, 127  
   DRIVE\_RIGHT\_MOTOR\_4\_GEARSET, 128  
   DRIVE\_RIGHT\_MOTOR\_4\_PORT, 127  
   DRIVE\_VELOCITY\_PROFILE\_JERK\_RATE, 124  
   DRIVE\_VELOCITY\_PROFILE\_MAX\_ACCELERATION,  
     124  
   DRIVE\_VELOCITY\_TO\_VOLTAGE, 128  
   DRIVE\_WHEEL\_RADIUS, 129  
   ELEVATOR\_DISTANCE\_CONSTANT, 135  
   ELEVATOR\_DISTANCE\_OFFSET, 135  
   ELEVATOR\_DISTANCE\_PORT, 135  
   ELEVATOR\_INCHES\_PER\_RADIAN, 134  
   ELEVATOR\_KD, 133  
   ELEVATOR\_KI, 133  
   ELEVATOR\_KP, 133  
   ELEVATOR\_MOTOR\_1\_GEARSET, 134  
   ELEVATOR\_MOTOR\_1\_PORT, 133  
   ELEVATOR\_MOTOR\_2\_GEARSET, 134  
   ELEVATOR\_MOTOR\_2\_PORT, 134  
   ELEVATOR\_ROTATION\_SENSOR\_PORT, 134  
   getName, 114  
   HANG\_ARM\_PISTON\_1\_EXTENDED\_STATE,  
     136  
   HANG\_ARM\_PISTON\_1\_PORT, 136  
   HANG\_ARM\_UP\_STATE, 136  
   HANG\_CLAW\_CLOSED\_STATE, 136  
   HANG\_CLAW\_PISTON\_1\_EXTENDED\_STATE,  
     135  
   HANG\_CLAW\_PISTON\_1\_PORT, 135  
   HANG\_WINCH\_ENGAGED\_STATE, 137  
   HANG\_WINCH\_PISTON\_1\_EXTENDED\_STATE,  
     137  
   HANG\_WINCH\_PISTON\_1\_PORT, 136  
   INTAKE\_KD, 129  
   INTAKE\_KI, 129  
   INTAKE\_KP, 129  
   INTAKE\_MOTOR\_1\_GEARSET, 130  
   INTAKE\_MOTOR\_1\_PORT, 130  
   INTAKE\_MOTOR\_2\_GEARSET, 130  
   INTAKE\_MOTOR\_2\_PORT, 130  
   INTAKE\_ROLLER\_RADIUS, 130  
   LEFT\_WING\_PISTON\_1\_EXTENDED\_STATE,  
     140  
   LEFT\_WING\_PISTON\_1\_PORT, 140  
   LOADER\_KD, 137  
   LOADER\_KI, 137  
   LOADER\_KP, 137  
   LOADER\_LOADED\_POSITION, 138  
   LOADER\_MOTOR\_1\_GEARSET, 138  
   LOADER\_MOTOR\_1\_PORT, 138  
   LOADER\_POSITION\_TOLERANCE, 138  
   LOADER\_READY\_POSITION, 138  
   ODOMETRY\_HEADING\_PORT, 121  
   ODOMETRY\_HEADING\_TUNING\_CONSTANT,  
     121  
   ODOMETRY\_LINEAR\_OFFSET, 122  
   ODOMETRY\_LINEAR\_PORT, 122  
   ODOMETRY\_LINEAR\_RADIUS, 122  
   ODOMETRY\_STRAFE\_OFFSET, 123  
   ODOMETRY\_STRAFE\_PORT, 122  
   ODOMETRY\_STRAFE\_RADIUS, 122  
   RESETTER\_DISTANCE\_CONSTANT, 123  
   RESETTER\_DISTANCE\_OFFSET, 123  
   RESETTER\_DISTANCE\_PORT, 123  
   RESETTER\_OFFSET\_THETA, 124  
   RESETTER\_OFFSET\_X, 123  
   RESETTER\_OFFSET\_Y, 124  
   RIGHT\_WING\_PISTON\_1\_EXTENDED\_STATE,  
     141  
   RIGHT\_WING\_PISTON\_1\_PORT, 141  
   TURN\_KD, 139  
   TURN\_KI, 139

TURN\_KP, 139  
TURN\_TARGET\_TOLERANCE, 139  
TURN\_TARGET\_VELOCITY, 139  
UMBRELLA\_OUT\_STATE, 140  
UMBRELLA\_PIKE\_1\_EXTENDED\_STATE, 140  
UMBRELLA\_PIKE\_1\_PORT, 140  
WINGS\_OUT\_STATE, 141  
wisco::configs::OrangeConfiguration, 141  
buildController, 143  
buildControlSystem, 142  
buildRobot, 143  
CONFIGURATION\_NAME, 144  
getName, 142  
wisco::control, 16  
wisco::control::AControl, 144  
AControl, 145  
command, 146  
getName, 146  
initialize, 146  
m\_name, 148  
operator=, 147  
run, 146  
state, 147  
wisco::control::boomerang, 17  
wisco::control::boomerang::BoomerangControl, 148  
BoomerangControl, 149  
command, 150  
CONTROL\_NAME, 151  
GO\_TO\_POSITION\_COMMAND\_NAME, 151  
initialize, 150  
m\_boomerang, 152  
PAUSE\_COMMAND\_NAME, 152  
RESUME\_COMMAND\_NAME, 152  
run, 150  
state, 151  
TARGET\_REACHED\_STATE\_NAME, 152  
wisco::control::boomerang::IBoomerang, 153  
goToPosition, 154  
initialize, 153  
pause, 154  
resume, 154  
run, 153  
targetReached, 154  
wisco::control::boomerang::PIDBoomerang, 155  
calculateCarrotPoint, 159  
calculateDistanceToTarget, 159  
control\_robot, 166  
DRIVE\_SET\_VELOCITY\_COMMAND\_NAME, 165  
DRIVE\_SUBSYSTEM\_NAME, 164  
getOdometryPosition, 158  
goToPosition, 160  
initialize, 160  
m\_delayer, 165  
m\_lead, 166  
m\_linear\_pid, 166  
m\_mutex, 165  
m\_rotational\_pid, 166  
m\_target\_tolerance, 166  
m\_task, 166  
motion\_velocity, 167  
ODOMETRY\_GET\_POSITION\_STATE\_NAME, 165  
ODOMETRY\_SUBSYSTEM\_NAME, 165  
pause, 161  
paused, 167  
resume, 161  
run, 160  
setDelayer, 162  
setDriveVelocity, 158  
setLead, 164  
setLinearPID, 163  
setMutex, 162  
setRotationalPID, 163  
setTargetTolerance, 164  
setTask, 163  
target\_reached, 167  
target\_theta, 167  
target\_x, 167  
target\_y, 167  
targetReached, 162  
TASK\_DELAY, 164  
taskLoop, 157  
taskUpdate, 157  
updateVelocity, 159  
wisco::control::boomerang::PIDBoomerangBuilder, 168  
build, 172  
m\_delayer, 172  
m\_lead, 173  
m\_linear\_pid, 172  
m\_mutex, 172  
m\_rotational\_pid, 173  
m\_target\_tolerance, 173  
m\_task, 172  
withDelayer, 169  
withLead, 171  
withLinearPID, 170  
withMutex, 169  
withRotationalPID, 170  
withTargetTolerance, 171  
withTask, 170  
wisco::control::ControlSystem, 173  
addControl, 174  
controls, 176  
getState, 175  
initialize, 175  
removeControl, 174  
sendCommand, 175  
wisco::control::motion, 17  
ETurnDirection, 18  
wisco::control::motion::ITurn, 176  
initialize, 177  
pause, 178  
resume, 178  
run, 177  
targetReached, 178  
turnToAngle, 177

turnToPoint, 178  
**wisco::control::motion::MotionControl**, 179  
 command, 181  
 CONTROL\_NAME, 182  
 initialize, 181  
 m\_turn, 183  
 MotionControl, 180  
 PAUSE\_TURN\_COMMAND\_NAME, 183  
 RESUME\_TURN\_COMMAND\_NAME, 183  
 run, 181  
 state, 182  
 TURN\_TARGET\_REACHED\_STATE\_NAME, 183  
 TURN\_TO\_ANGLE\_COMMAND\_NAME, 182  
 TURN\_TO\_POINT\_COMMAND\_NAME, 183  
**wisco::control::motion::PIDTurn**, 184  
 calculateAngleToTarget, 188  
 calculateDriveVelocity, 189  
 control\_robot, 196  
 DRIVE\_GET\_RADIUS\_STATE\_NAME, 195  
 DRIVE\_SET\_VELOCITY\_COMMAND\_NAME, 195  
 DRIVE\_SUBSYSTEM\_NAME, 194  
 forced\_direction\_reached, 198  
 getDriveRadius, 187  
 getOdometryPosition, 188  
 initialize, 189  
 m\_delayer, 195  
 m\_mutex, 196  
 m\_pid, 196  
 m\_target\_tolerance, 196  
 m\_target\_velocity, 196  
 m\_task, 196  
 ODOMETRY\_GET\_POSITION\_STATE\_NAME, 195  
 ODOMETRY\_SUBSYSTEM\_NAME, 194  
 pause, 191  
 paused, 197  
 resume, 191  
 run, 189  
 setDelayer, 192  
 setDriveVelocity, 187  
 setMutex, 192  
 setPID, 193  
 setTargetTolerance, 193  
 setTargetVelocity, 194  
 setTask, 193  
 target\_reached, 197  
 target\_x, 197  
 target\_y, 197  
 targetReached, 192  
 TASK\_DELAY, 194  
 taskLoop, 186  
 taskUpdate, 186  
 turn\_direction, 197  
 TURN\_TO\_ANGLE\_DISTANCE, 195  
 turn\_velocity, 197  
 turnToAngle, 190  
 turnToPoint, 190  
**wisco::control::motion::PIDTurnBuilder**, 198  
 build, 201  
 m\_delayer, 202  
 m\_mutex, 202  
 m\_pid, 202  
 m\_target\_tolerance, 202  
 m\_target\_velocity, 202  
 m\_task, 202  
 withDelayer, 199  
 withMutex, 199  
 withPID, 200  
 withTargetTolerance, 200  
 withTargetVelocity, 201  
 withTask, 199  
**wisco::control::path**, 18  
 operator\*, 19  
**wisco::control::path::Point**, 203  
 getX, 205  
 getY, 206  
 m\_x, 210  
 m\_y, 210  
 operator+, 206  
 operator+=, 207  
 operator-, 206  
 operator-=, 208  
 operator/, 207  
 operator/=, 209  
 operator=, 209  
 operator\*, 207  
 operator\*=, 208  
 Point, 204  
 setX, 205  
 setY, 205  
**wisco::control::path::QuinticBezier**, 210  
 c0, 213  
 c1, 213  
 c2, 214  
 c3, 214  
 calculatePoint, 212  
 k0, 213  
 k1, 214  
 operator=, 212, 213  
 QuinticBezier, 211, 212  
**wisco::control::path::QuinticBezierSpline**, 214  
 calculate, 215  
**wisco::control::PID**, 216  
 accumulated\_error, 220  
 getControlValue, 218  
 last\_error, 220  
 last\_time, 220  
 m\_clock, 219  
 m\_kd, 220  
 m\_ki, 219  
 m\_kp, 219  
 operator=, 218, 219  
 PID, 217  
 reset, 218  
**wisco::hal**, 19  
 DistanceBooleanMode, 20

wisco::hal::DistanceBooleanSensor, 221  
    DistanceBooleanSensor, 222  
    getValue, 223  
    initialize, 223  
    m\_distance\_sensor, 224  
    m\_lower\_threshold, 224  
    m\_mode, 224  
    m\_upper\_threshold, 224  
    reset, 223  
    value, 225  
wisco::hal::MotorGroup, 225  
    addMotor, 226  
    getAngularVelocity, 227  
    getAngularVelocityConstant, 227  
    getGearRatio, 227  
    getPosition, 228  
    getResistance, 226  
    getTorqueConstant, 226  
    initialize, 226  
    motors, 229  
    operator=, 229  
    setVoltage, 228  
wisco::hal::PistonGroup, 229  
    addPiston, 230  
    getState, 231  
    m\_state, 231  
    operator=, 231  
    pistons, 231  
    setState, 230  
wisco::hal::TrackingWheel, 232  
    getDistance, 234  
    initialize, 233  
    m\_sensor, 234  
    m\_wheel\_radius, 235  
    reset, 233  
    setDistance, 234  
    TrackingWheel, 233  
wisco::IAlliance, 235  
    getName, 236  
wisco::IAutonomous, 236  
    getName, 237  
    initialize, 237  
    run, 237  
wisco::IConfiguration, 238  
    buildController, 239  
    buildControlSystem, 238  
    buildRobot, 239  
    getName, 238  
wisco::IMenu, 239  
    addAlliance, 240  
    addAutonomous, 240  
    addConfiguration, 241  
    addProfile, 241  
    display, 241  
    getSystemConfiguration, 242  
    isStarted, 241  
wisco::io, 20  
wisco::io::IBooleanSensor, 242  
    getValue, 243  
    initialize, 243  
    reset, 243  
wisco::io::IDistanceSensor, 243  
    getDistance, 244  
    initialize, 244  
    reset, 244  
wisco::io::IDistanceTrackingSensor, 244  
    getDistance, 245  
    initialize, 245  
    reset, 245  
    setDistance, 245  
wisco::io::IHeadingSensor, 246  
    getHeading, 247  
    getRotation, 247  
    initialize, 247  
    reset, 247  
    setHeading, 247  
    setRotation, 248  
wisco::io::IMotor, 248  
    getAngularVelocity, 250  
    getAngularVelocityConstant, 249  
    getGearRatio, 250  
    getPosition, 250  
    getResistance, 249  
    getTorqueConstant, 249  
    initialize, 249  
    setVoltage, 250  
wisco::io::IPiston, 251  
    extend, 252  
    isExtended, 252  
    isRetracted, 252  
    retract, 252  
    toggle, 252  
wisco::io::IRotationSensor, 253  
    getAngle, 254  
    getRotation, 254  
    initialize, 253  
    reset, 253  
    setRotation, 254  
wisco::IProfile, 255  
    getAnalogControlMapping, 256  
    getControlMode, 255  
    getDigitalControlMapping, 256  
    getName, 255  
    setControlMode, 256  
wisco::MatchController, 257  
    autonomous, 259  
    autonomous\_manager, 260  
    competitionInitialize, 259  
    control\_system, 261  
    controller, 261  
    disabled, 259  
    initialize, 258  
    m\_clock, 260  
    m\_delayer, 260  
    m\_menu, 260  
    MatchController, 258

MENU\_DELAY, 260  
 opcontrol\_manager, 261  
 operatorControl, 259  
 robot, 261  
 wisco::menu, 21  
     settingsBackButtonEventHandler, 22  
     settingsButtonEventHandler, 22  
     settingsButtonMatrixEventHandler, 23  
     startButtonEventHandler, 21  
 wisco::menu::LvglMenu, 261  
     addOption, 264  
     button\_default\_style, 269  
     button\_matrix\_items\_style, 270  
     button\_matrix\_main\_style, 270  
     button\_pressed\_style, 270  
     BUTTONS\_PER\_LINE, 269  
     COLUMN\_WIDTH, 269  
     complete, 271  
     CONFIGURATION\_FILE, 269  
     container\_default\_style, 270  
     container\_pressed\_style, 270  
     displayMenu, 268  
     drawMainMenu, 265  
     drawSettingsMenu, 266  
     getSelection, 268  
     initializeStyles, 263  
     options, 271  
     readConfiguration, 267  
     removeOption, 264  
     selectionComplete, 268  
     setComplete, 267  
     styles\_initialized, 270  
     writeConfiguration, 268  
 wisco::menu::MenuAdapter, 271  
     addAlliance, 272  
     addAutonomous, 273  
     addConfiguration, 273  
     addProfile, 274  
     ALLIANCE\_OPTION\_NAME, 276  
     alliances, 276  
     AUTONOMOUS\_OPTION\_NAME, 276  
     autonomous\_routines, 276  
     CONFIGURATION\_OPTION\_NAME, 276  
     display, 274  
     driver\_profiles, 277  
     getSystemConfiguration, 275  
     hardware\_configurations, 277  
     isStarted, 274  
     lvgl\_menu, 277  
     PROFILE\_OPTION\_NAME, 276  
 wisco::menu::Option, 277  
     choices, 278  
     name, 278  
     selected, 278  
 wisco::OPControlManager, 278  
     CONTROL\_DELAY, 281  
     initializeOpcontrol, 280  
     m\_clock, 281  
     m\_delayer, 281  
     m\_profile, 282  
     OPControlManager, 279  
     runOpcontrol, 280  
     setProfile, 280  
 wisco::profiles, 23  
 wisco::profiles::HenryProfile, 282  
     ANALOG\_CONTROL\_MAP, 286  
     CONTROL\_MODE\_MAP, 285  
     DIGITAL\_CONTROL\_MAP, 286  
     getAnalogControlMapping, 284  
     getControlMode, 283  
     getDigitalControlMapping, 285  
     getName, 283  
     PROFILE\_NAME, 285  
     setControlMode, 284  
 wisco::profiles::JohnProfile, 287  
     ANALOG\_CONTROL\_MAP, 290  
     CONTROL\_MODE\_MAP, 290  
     DIGITAL\_CONTROL\_MAP, 291  
     getAnalogControlMapping, 289  
     getControlMode, 288  
     getDigitalControlMapping, 289  
     getName, 288  
     PROFILE\_NAME, 290  
     setControlMode, 288  
 wisco::robot, 23  
 wisco::robot::ASubsystem, 291  
     ASubsystem, 292, 293  
     command, 294  
     getName, 293  
     initialize, 293  
     m\_name, 295  
     operator=, 295  
     run, 294  
     state, 294  
 wisco::robot::Robot, 296  
     addSubsystem, 296  
     getState, 298  
     initialize, 297  
     removeSubsystem, 297  
     sendCommand, 297  
     subsystems, 298  
 wisco::robot::subsystems, 24  
 wisco::robot::subsystems::drive, 25  
 wisco::robot::subsystems::drive::CurveVelocityProfile,  
     299  
     CurveVelocityProfile, 300  
     getAcceleration, 300  
     last\_time, 302  
     m\_clock, 301  
     m\_current\_acceleration, 302  
     m\_jerk\_rate, 301  
     m\_max\_acceleration, 301  
     setAcceleration, 301  
 wisco::robot::subsystems::drive::DifferentialDriveSubsystem,  
     302  
     command, 304

DifferentialDriveSubsystem, 303  
GET\_RADIUS\_STATE\_NAME, 306  
GET\_VELOCITY\_STATE\_NAME, 306  
initialize, 304  
m\_differential\_drive, 306  
run, 304  
SET\_VELOCITY\_COMMAND\_NAME, 305  
SET\_VOLTAGE\_COMMAND\_NAME, 306  
state, 305  
SUBSYSTEM\_NAME, 305  
wisco::robot::subsystems::drive::DirectDifferentialDrive,  
    307  
    getRadius, 310  
    getVelocity, 308  
    initialize, 308  
    m\_gear\_ratio, 312  
    m\_left\_motors, 312  
    m\_radius, 313  
    m\_right\_motors, 312  
    m\_velocity\_to\_voltage, 312  
    m\_wheel\_radius, 313  
    run, 308  
    setGearRatio, 311  
    setLeftMotors, 310  
    setRadius, 311  
    setRightMotors, 310  
    setVelocity, 309  
    setVelocityToVoltage, 311  
    setVoltage, 309  
    setWheelRadius, 311  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
    313  
    build, 317  
    m\_gear\_ratio, 317  
    m\_left\_motors, 317  
    m\_radius, 318  
    m\_right\_motors, 317  
    m\_velocity\_to\_voltage, 317  
    m\_wheel\_radius, 318  
    withGearRatio, 315  
    withLeftMotor, 314  
    withRadius, 316  
    withRightMotor, 314  
    withVelocityToVoltage, 315  
    withWheelRadius, 316  
wisco::robot::subsystems::drive::IDifferentialDrive, 318  
    getRadius, 320  
    getVelocity, 319  
    initialize, 319  
    run, 319  
    setVelocity, 319  
    setVoltage, 320  
wisco::robot::subsystems::drive::IVelocityProfile, 320  
    getAcceleration, 321  
    setAcceleration, 321  
wisco::robot::subsystems::drive::KinematicDifferentialDrive,  
    322  
    c1, 335  
    c2, 335  
    c3, 335  
    c4, 335  
    c5, 335  
    c6, 336  
    c7, 336  
    getRadius, 328  
    getVelocity, 326  
    initialize, 325  
    m\_delayer, 332  
    m\_gear\_ratio, 334  
    m\_left\_motors, 333  
    m\_left\_velocity\_profile, 333  
    m\_mass, 334  
    m\_moment\_of\_inertia, 334  
    m\_mutex, 333  
    m\_radius, 334  
    m\_right\_motors, 334  
    m\_right\_velocity\_profile, 333  
    m\_task, 333  
    m\_velocity, 336  
    m\_wheel\_radius, 335  
    run, 326  
    setDelayer, 328  
    setGearRatio, 332  
    setLeftMotors, 329  
    setMass, 330  
    setMomentOfInertia, 330  
    setMutex, 328  
    setRadius, 330  
    setRightMotors, 329  
    setTask, 328  
    setVelocity, 327  
    setVelocityProfiles, 329  
    setVoltage, 327  
    setWheelRadius, 332  
    TASK\_DELAY, 332  
    taskLoop, 324  
    taskUpdate, 325  
    updateAcceleration, 325  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
    336  
    build, 343  
    m\_delayer, 343  
    m\_gear\_ratio, 345  
    m\_left\_motors, 344  
    m\_left\_velocity\_profile, 344  
    m\_mass, 345  
    m\_moment\_of\_inertia, 345  
    m\_mutex, 343  
    m\_radius, 345  
    m\_right\_motors, 344  
    m\_right\_velocity\_profile, 344  
    m\_task, 344  
    m\_wheel\_radius, 345  
    withDelayer, 338  
    withGearRatio, 342  
    withLeftMotor, 340

withLeftVelocityProfile, 339  
 withMass, 341  
 withMomentOfInertia, 342  
 withMutex, 338  
 withRadius, 341  
 withRightMotor, 340  
 withRightVelocityProfile, 340  
 withTask, 339  
 withWheelRadius, 342  
**wisco::robot::subsystems::drive::Velocity**, 346  
 left\_velocity, 346  
 right\_velocity, 346  
**wisco::robot::subsystems::elevator**, 25  
**wisco::robot::subsystems::elevator::ElevatorSubsystem**,  
 347  
 CAP\_DISTANCE\_STATE\_NAME, 351  
 command, 349  
 ElevatorSubsystem, 348  
 GET\_POSITION\_STATE\_NAME, 350  
 initialize, 349  
 m\_distance\_sensor, 351  
 m\_elevator, 351  
 run, 349  
 SET\_POSITION\_COMMAND\_NAME, 350  
 state, 349  
 SUBSYSTEM\_NAME, 350  
**wisco::robot::subsystems::elevator::IElevator**, 351  
 getPosition, 352  
 initialize, 352  
 run, 352  
 setPosition, 352  
**wisco::robot::subsystems::elevator::PIDElevator**, 353  
 getPosition, 356  
 initialize, 355  
 m\_clock, 359  
 m\_delayer, 359  
 m\_inches\_per\_radian, 361  
 m\_motors, 360  
 m\_mutex, 360  
 m\_pid, 360  
 m\_position, 361  
 m\_rotation\_sensor, 360  
 m\_task, 360  
 run, 355  
 setClock, 357  
 setDelayer, 357  
 setInchesPerRadian, 359  
 setMotors, 358  
 setMutex, 357  
 setPID, 358  
 setPosition, 356  
 setRotationSensor, 358  
 setTask, 358  
 TASK\_DELAY, 359  
 taskLoop, 354  
 taskUpdate, 355  
 updatePosition, 355  
**wisco::robot::subsystems::elevator::PIDElevatorBuilder**,  
 361  
 build, 365  
 m\_clock, 366  
 m\_delayer, 366  
 m\_inches\_per\_radian, 367  
 m\_motors, 367  
 m\_mutex, 366  
 m\_pid, 366  
 m\_rotation\_sensor, 367  
 m\_task, 366  
 withClock, 362  
 withDelayer, 362  
 withInchesPerRadian, 365  
 withMotor, 364  
 withMutex, 363  
 withPID, 364  
 withRotationSensor, 364  
 withTask, 363  
**wisco::robot::subsystems::hang**, 26  
**wisco::robot::subsystems::hang::HangSubsystem**, 367  
 ARM\_DOWN\_STATE\_NAME, 374  
 ARM\_UP\_STATE\_NAME, 374  
 CLAW\_CLOSED\_STATE\_NAME, 373  
 CLAW\_OPEN\_STATE\_NAME, 374  
 CLOSE\_CLAW\_COMMAND\_NAME, 372  
 command, 370  
 DISENGAGE\_WINCH\_COMMAND\_NAME, 373  
 ENGAGE\_WINCH\_COMMAND\_NAME, 373  
 HangSubsystem, 369  
 initialize, 370  
 LOWER\_ARM\_COMMAND\_NAME, 373  
 m\_claw, 375  
 m\_toggle\_arm, 375  
 m\_winch, 375  
 OPEN\_CLAW\_COMMAND\_NAME, 372  
 RAISE\_ARM\_COMMAND\_NAME, 373  
 run, 370  
 state, 371  
 SUBSYSTEM\_NAME, 372  
 WINCH\_DISENGAGED\_STATE\_NAME, 374  
 WINCH\_ENGAGED\_STATE\_NAME, 374  
**wisco::robot::subsystems::hang::IClaw**, 375  
 close, 376  
 initialize, 376  
 isClosed, 377  
 isOpen, 377  
 open, 377  
 run, 376  
**wisco::robot::subsystems::hang::IToggleArm**, 377  
 initialize, 378  
 isDown, 379  
 isUp, 379  
 run, 378  
 setDown, 378  
 setUp, 379  
**wisco::robot::subsystems::hang::IWinch**, 379  
 disengage, 381

engage, 380  
initialize, 380  
isDisengaged, 381  
isEngaged, 381  
run, 380  
wisco::robot::subsystems::hang::PistonClaw, 381  
close, 383  
initialize, 382  
isClosed, 383  
isOpen, 383  
m\_closed\_state, 385  
m\_pistons, 385  
open, 383  
run, 382  
setClosedState, 384  
setPistons, 384  
wisco::robot::subsystems::hang::PistonClawBuilder,  
385  
build, 387  
m\_closed\_state, 388  
m\_pistons, 388  
withClosedState, 387  
withPiston, 386  
wisco::robot::subsystems::hang::PistonToggleArm, 388  
initialize, 389  
isDown, 390  
isUp, 390  
m\_pistons, 391  
m\_up\_state, 391  
run, 389  
setDown, 389  
setPistons, 390  
setUp, 390  
setUpState, 391  
wisco::robot::subsystems::hang::PistonToggleArmBuilder,  
391  
build, 393  
m\_pistons, 393  
m\_up\_state, 393  
withPiston, 392  
withUpState, 392  
wisco::robot::subsystems::hang::PistonWinch, 394  
disengage, 395  
engage, 395  
initialize, 395  
isDisengaged, 396  
isEngaged, 396  
m\_engaged\_state, 397  
m\_pistons, 397  
run, 395  
setEngagedState, 397  
setPistons, 396  
wisco::robot::subsystems::hang::PistonWinchBuilder,  
397  
build, 399  
m\_engaged\_state, 399  
m\_pistons, 399  
withEngagedState, 398  
withPiston, 398  
wisco::robot::subsystems::intake, 27  
wisco::robot::subsystems::intake::DistanceVisionBallDetector,  
400  
getBallAngle, 401  
getBallDistance, 401  
initialize, 401  
m\_distance\_sensor, 402  
run, 401  
setDistanceSensor, 402  
wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder,  
402  
build, 403  
m\_distance\_sensor, 404  
withDistanceSensor, 403  
wisco::robot::subsystems::intake::IBallDetector, 404  
getBallAngle, 405  
getBallDistance, 405  
initialize, 405  
run, 405  
wisco::robot::subsystems::intake::IIntake, 406  
getVelocity, 407  
initialize, 406  
run, 406  
setVelocity, 407  
setVoltage, 407  
wisco::robot::subsystems::intake::IntakeSubsystem,  
408  
command, 410  
GET BALL\_ANGLE\_STATE\_NAME, 412  
GET BALL\_DISTANCE\_STATE\_NAME, 412  
GET VELOCITY\_STATE\_NAME, 412  
initialize, 410  
IntakeSubsystem, 409  
m\_ball\_detector, 413  
m\_intake, 412  
run, 410  
SET VELOCITY\_COMMAND\_NAME, 411  
SET VOLTAGE\_COMMAND\_NAME, 412  
state, 411  
SUBSYSTEM\_NAME, 411  
wisco::robot::subsystems::intake::PIDIntake, 413  
getVelocity, 416  
initialize, 415  
m\_clock, 420  
m\_delayer, 421  
m\_motors, 421  
m\_mutex, 421  
m\_pid, 421  
m\_roller\_radius, 421  
m\_task, 421  
m\_velocity, 422  
run, 416  
setClock, 417  
setDelayer, 417  
setMotors, 420  
setMutex, 418  
setPID, 418

setRollerRadius, 420  
 setTask, 418  
 setVelocity, 416  
 setVoltage, 417  
 TASK\_DELAY, 420  
 taskLoop, 415  
 taskUpdate, 415  
 updateVelocity, 415  
 velocity\_control, 422  
**wisco::robot::subsystems::intake::PIDIntakeBuilder**, 422  
 build, 426  
 m\_clock, 426  
 m\_delayer, 426  
 m\_motors, 427  
 m\_mutex, 427  
 m\_pid, 427  
 m\_roller\_radius, 427  
 m\_task, 427  
 withClock, 423  
 withDelay, 423  
 withMotor, 425  
 withMutex, 424  
 withPID, 425  
 withRollerRadius, 425  
 withTask, 424  
**wisco::robot::subsystems::loader**, 27  
**wisco::robot::subsystems::loader::ILoader**, 428  
 doLoad, 429  
 doReady, 429  
 initialize, 429  
 isLoaded, 429  
 isReady, 429  
 run, 429  
**wisco::robot::subsystems::loader::LoaderSubsystem**,  
 430  
 command, 432  
 DO\_LOAD\_COMMAND\_NAME, 433  
 DO\_READY\_COMMAND\_NAME, 434  
 initialize, 432  
 IS\_LOADED\_STATE\_NAME, 434  
 IS\_READY\_STATE\_NAME, 434  
 LoaderSubsystem, 431  
 m\_loader, 434  
 run, 432  
 state, 433  
 SUBSYSTEM\_NAME, 433  
**wisco::robot::subsystems::loader::PIDLoader**, 435  
 doLoad, 439  
 doReady, 439  
 EState, 437  
 getPosition, 438  
 initialize, 438  
 isLoaded, 440  
 isReady, 440  
 m\_clock, 443  
 m\_delayer, 443  
 m\_match\_load\_position, 444  
 m\_motors, 444  
 m\_mutex, 444  
 m\_pid, 444  
 m\_position\_tolerance, 445  
 m\_ready\_position, 444  
 m\_task, 444  
 run, 439  
 setClock, 440  
 setDelay, 441  
 setMatchLoadPosition, 442  
 setMotors, 442  
 setMutex, 441  
 setPID, 442  
 setPositionTolerance, 443  
 setReadyPosition, 442  
 setTask, 441  
 state, 445  
 target\_position, 445  
 TASK\_DELAY, 443  
 taskLoop, 437  
 taskUpdate, 437  
 updatePosition, 438  
 updateState, 438  
**wisco::robot::subsystems::loader::PIDLoaderBuilder**,  
 445  
 build, 450  
 m\_clock, 451  
 m\_delayer, 451  
 m\_match\_load\_position, 452  
 m\_motors, 451  
 m\_mutex, 451  
 m\_pid, 451  
 m\_position\_tolerance, 452  
 m\_ready\_position, 452  
 m\_task, 451  
 withClock, 447  
 withDelay, 447  
 withMatchLoadPosition, 449  
 withMotor, 449  
 withMutex, 447  
 withPID, 448  
 withPositionTolerance, 450  
 withReadyPosition, 449  
 withTask, 448  
**wisco::robot::subsystems::position**, 28  
**wisco::robot::subsystems::position::DistancePositionResetter**,  
 452  
 bindRadians, 454  
 FAR\_WALL, 457  
 getResetX, 455  
 getResetY, 455  
 initialize, 454  
 m\_distance\_sensor, 457  
 m\_local\_theta, 458  
 m\_local\_x, 458  
 m\_local\_y, 458  
 NEAR\_WALL, 457  
 run, 454  
 setDistanceSensor, 456

setLocalTheta, 457  
setLocalX, 456  
setLocalY, 456  
wisco::robot::subsystems::position::DistancePositionResetterBuilder, 458  
build, 461  
m\_distance\_sensor, 461  
m\_local\_theta, 462  
m\_local\_x, 461  
m\_local\_y, 461  
withDistanceSensor, 459  
withLocalTheta, 460  
withLocalX, 459  
withLocalY, 460  
wisco::robot::subsystems::position::InertialOdometry, 462  
getPosition, 467  
initialize, 466  
last\_heading, 473  
last\_linear\_distance, 474  
last\_strafe\_distance, 474  
last\_time, 474  
m\_clock, 471  
m\_delayer, 472  
m\_heading\_sensor, 472  
m\_linear\_distance\_tracking\_offset, 473  
m\_linear\_distance\_tracking\_sensor, 472  
m\_mutex, 472  
m\_position, 473  
m\_strafe\_distance\_tracking\_offset, 473  
m\_strafe\_distance\_tracking\_sensor, 473  
m\_task, 472  
run, 466  
setClock, 467  
setDelayer, 467  
setHeadingSensor, 468  
setLinearDistanceTrackingOffset, 470  
setLinearDistanceTrackingSensor, 470  
setMutex, 468  
setPosition, 466  
setStrafeDistanceTrackingOffset, 471  
setStrafeDistanceTrackingSensor, 470  
setTask, 468  
TASK\_DELAY, 471  
taskLoop, 464  
taskUpdate, 465  
TIME\_UNIT\_CONVERTER, 471  
updatePosition, 465  
wisco::robot::subsystems::position::InertialOdometryBuilder, 474  
build, 479  
m\_clock, 480  
m\_delayer, 480  
m\_heading\_sensor, 481  
m\_linear\_distance\_tracking\_offset, 481  
m\_linear\_distance\_tracking\_sensor, 481  
m\_mutex, 480  
m\_strafe\_distance\_tracking\_offset, 481  
m\_strafe\_distance\_tracking\_sensor, 481  
m\_task, 481  
m\_strafe\_distance\_tracking\_sensor, 481  
m\_task, 480  
withClock, 476  
WithDelayer, 476  
withHeadingSensor, 477  
withLinearDistanceTrackingOffset, 478  
withLinearDistanceTrackingSensor, 478  
withMutex, 477  
withStrafeDistanceTrackingOffset, 479  
withStrafeDistanceTrackingSensor, 479  
withTask, 477  
wisco::robot::subsystems::position::IPositionResetter, 482  
getResetX, 483  
getResetY, 483  
initialize, 483  
run, 483  
wisco::robot::subsystems::position::IPositionTracker, 484  
getPosition, 485  
initialize, 484  
run, 484  
setPosition, 485  
wisco::robot::subsystems::position::Position, 485  
theta, 486  
thetaV, 487  
x, 486  
xV, 486  
y, 486  
yV, 486  
wisco::robot::subsystems::position::PositionSubsystem, 487  
command, 489  
GET\_POSITION\_STATE\_NAME, 491  
initialize, 489  
m\_position\_resetter, 492  
m\_position\_tracker, 491  
PositionSubsystem, 489  
RESET\_X\_COMMAND\_NAME, 491  
RESET\_Y\_COMMAND\_NAME, 491  
run, 489  
SET\_POSITION\_COMMAND\_NAME, 491  
state, 490  
SUBSYSTEM\_NAME, 491  
wisco::robot::subsystems::umbrella, 29  
wisco::robot::subsystems::umbrella::IUmbrella, 492  
initialize, 493  
isIn, 493  
isOut, 494  
run, 493  
setIn, 493  
setOut, 493  
wisco::robot::subsystems::umbrella::PistonUmbrella, 494  
initialize, 495  
isIn, 496  
isOut, 496  
m\_out\_state, 497

m\_pistons, 497  
 run, 495  
 setIn, 495  
 setOut, 496  
 setOutState, 497  
 setPistons, 496  
 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder,  
     497  
 build, 499  
 m\_out\_state, 499  
 m\_pistons, 499  
 withOutState, 498  
 withPiston, 498  
 wisco::robot::subsystems::umbrella::UmbrellaSubsystem,  
     500  
 command, 502  
 initialize, 502  
 IS\_IN\_STATE\_NAME, 504  
 IS\_OUT\_STATE\_NAME, 504  
 m\_umbrella, 504  
 run, 502  
 SET\_IN\_COMMAND\_NAME, 503  
 SET\_OUT\_COMMAND\_NAME, 503  
 state, 502  
 SUBSYSTEM\_NAME, 503  
 UmbrellaSubsystem, 501  
 wisco::robot::subsystems::wings, 29  
 wisco::robot::subsystems::wings::IWings, 504  
     getLeftWing, 506  
     getRightWing, 506  
     initialize, 505  
     run, 505  
     setLeftWing, 505  
     setRightWing, 506  
 wisco::robot::subsystems::wings::PistonWings, 507  
     getLeftWing, 509  
     getRightWing, 509  
     initialize, 508  
     m\_left\_pistons, 511  
     m\_out\_state, 511  
     m\_right\_pistons, 511  
     run, 508  
     setLeftPistons, 509  
     setLeftWing, 508  
     setOutState, 510  
     setRightPistons, 510  
     setRightWing, 509  
 wisco::robot::subsystems::wings::PistonWingsBuilder,  
     511  
     build, 513  
     m\_left\_pistons, 514  
     m\_out\_state, 514  
     m\_right\_pistons, 514  
     withLeftPiston, 512  
     withOutState, 513  
     withRightPiston, 512  
 wisco::robot::subsystems::wings::WingsSubsystem, 514  
     command, 516  
     GET\_LEFT\_WING\_STATE\_NAME, 518  
     GET\_RIGHT\_WING\_STATE\_NAME, 518  
     initialize, 516  
     m\_wings, 518  
     run, 516  
     SET\_LEFT\_WING\_COMMAND\_NAME, 517  
     SET\_RIGHT\_WING\_COMMAND\_NAME, 518  
     state, 517  
     SUBSYSTEM\_NAME, 517  
     WingsSubsystem, 515  
 wisco::rtos, 30  
 wisco::rtos::IClock, 519  
     clone, 519  
     getTime, 519  
 wisco::rtos::IDelayLayer, 520  
     clone, 521  
     delay, 521  
     delayUntil, 521  
 wisco::rtos::IMutex, 521  
     give, 522  
     take, 522  
 wisco::rtos::ITask, 523  
     join, 524  
     remove, 524  
     resume, 524  
     start, 523  
     suspend, 524  
 wisco::SystemConfiguration, 524  
     alliance, 525  
     autonomous, 525  
     configuration, 525  
     profile, 525  
 wisco::testing, 30  
 wisco::testing::pros\_testing, 31  
     FILE\_PATH, 31  
 wisco::testing::pros\_testing::DriveTest, 526  
     DriveTest, 527  
     HEADING\_TO\_RADIANS, 530  
     INCHES\_TO\_METERS, 530  
     initialize, 528  
     LINEAR\_FILE\_NAME, 529  
     m\_heading\_sensor, 531  
     m\_left\_drive\_motors, 531  
     m\_linear\_counts\_per\_inch, 531  
     m\_linear\_sensor, 531  
     m\_right\_drive\_motors, 531  
     MILLIS\_TO\_S, 529  
     runLinearTest, 528  
     runTurningTest, 528  
     TEST\_DURATION, 530  
     TEST\_V, 530  
     TURNING\_FILE\_NAME, 529  
     V\_TO\_MV, 530  
 wisco::testing::TestFactory, 532  
     createDriveTest, 533  
     INERTIAL\_PORT, 533  
     LEFT\_DRIVE\_PORTS, 533  
     LINEAR\_COUNTS\_PER\_INCH, 534

LINEAR\_TRACKING\_PORT, 533  
RIGHT\_DRIVE\_PORTS, 533  
wisco::user, 31  
  EControl, 33  
  EControllerAnalog, 33  
  EControllerDigital, 34  
  EControlType, 34  
wisco::user::drive, 35  
  EChassisControlMode, 35  
wisco::user::drive::DifferentialDriveOperator, 534  
  DIFFERENTIAL\_DRIVE\_SUBSYSTEM\_NAME, 538  
    DifferentialDriveOperator, 535  
    m\_controller, 538  
    m\_robot, 539  
    SET\_VOLTAGE\_COMMAND, 538  
    setDriveVoltage, 537  
    updateArcade, 536  
    updateDriveVoltage, 536  
    updateSingleArcadeLeft, 536  
    updateSingleArcadeRight, 536  
    updateSplitArcadeLeft, 537  
    updateSplitArcadeRight, 537  
    updateTank, 537  
    VOLTAGE\_CONVERSION, 538  
wisco::user::elevator, 35  
  EElevatorControlMode, 36  
wisco::user::elevator::ElevatorOperator, 539  
  CAP\_DETECTED\_DISTANCE, 551  
  CAP\_DISTANCE\_STATE\_NAME, 550  
  ELEVATOR\_SUBSYSTEM\_NAME, 549  
  ElevatorOperator, 541  
  EToggleState, 541  
  FIELD\_POSITION, 550  
  GET\_POSITION\_STATE, 549  
  getCapDistance, 542  
  getElevatorPosition, 542  
  getHangArmUp, 542  
  HANG\_ARM\_UP\_STATE\_NAME, 550  
  HANG\_SUBSYSTEM\_NAME, 549  
  IN\_POSITION, 550  
  m\_controller, 551  
  m\_robot, 552  
  manual\_input, 552  
  MATCH\_LOAD\_POSITION, 550  
  PARTNER\_HANG\_POSITION, 551  
  POLE\_HANG\_DISTANCE, 551  
  POLE\_HANG\_POSITION, 551  
  SET\_POSITION\_COMMAND, 549  
  setElevatorPosition, 548  
  toggle\_state, 552  
  updateElevatorPosition, 543  
  updateManual, 543  
  updatePoleHangPosition, 543  
  updatePresetLadder, 546  
  updatePresetLadderIntake, 547  
  updatePresetSplit, 545  
  updatePresetToggle, 546  
wisco::user::hang, 36  
  EHangControlMode, 37  
wisco::user::hang::HangOperator, 552  
  CLOSE\_CLAW\_COMMAND\_NAME, 560  
  closeClaw, 555  
  DISENGAGE\_WINCH\_COMMAND\_NAME, 561  
  disengageWinch, 556  
  ENGAGE\_WINCH\_COMMAND\_NAME, 561  
  engageWinch, 555  
  EToggleState, 554  
  HANG\_SUBSYSTEM\_NAME, 560  
  HangOperator, 554  
  LOWER\_ARM\_COMMAND\_NAME, 560  
  lowerArm, 555  
  m\_controller, 561  
  m\_robot, 561  
  OPEN\_CLAW\_COMMAND\_NAME, 560  
  openClaw, 555  
  RAISE\_ARM\_COMMAND\_NAME, 560  
  raiseArm, 555  
  setGrabbedState, 556  
  setHangState, 559  
  setHungState, 556  
  setInactiveState, 556  
  setRaisedState, 556  
  toggle\_state, 561  
  updatePresetLadder, 558  
  updatePresetReset, 559  
  updatePresetSplit, 557  
  updatePresetToggle, 557  
wisco::user::IController, 562  
  getAnalog, 563  
  getDigital, 563  
  getNewDigital, 564  
  initialize, 563  
  rumble, 564  
  run, 563  
wisco::user::intake, 37  
  EIntakeControlMode, 38  
wisco::user::intake::IntakeOperator, 564  
  EToggleState, 566  
  INTAKE\_SUBSYSTEM\_NAME, 568  
  IntakeOperator, 566  
  m\_controller, 569  
  m\_robot, 569  
  SET\_VOLTAGE\_COMMAND, 569  
  setIntakeVoltage, 568  
  toggle\_state, 569  
  updateIntakeVoltage, 566  
  updateSingleToggle, 567  
  updateSplitHold, 567  
  updateSplitToggle, 567  
  updateToggleVoltage, 566  
  VOLTAGE\_SETTING, 569  
wisco::user::loader, 38  
  ELoaderControlMode, 39  
wisco::user::loader::LoaderOperator, 570  
  DO\_LOAD\_COMMAND\_NAME, 575

DO\_READY\_COMMAND\_NAME, 575  
doLoad, 572  
doReady, 572  
EToggleState, 571  
IS\_LOADED\_STATE\_NAME, 576  
IS\_READY\_STATE\_NAME, 576  
isLoaded, 572  
isReady, 572  
LoaderOperator, 571  
m\_controller, 576  
m\_robot, 576  
setLoaderPosition, 575  
SUBSYSTEM\_NAME, 575  
toggle\_state, 576  
updateHold, 573  
updateMacro, 573  
updateSingleToggle, 574  
updateSplitToggle, 574  
wisco::user::umbrella, 39  
EUmbrellaControlMode, 40  
wisco::user::umbrella::UmbrellaOperator, 577  
EToggleState, 578  
m\_controller, 581  
m\_robot, 581  
SET\_IN\_COMMAND, 581  
SET\_OUT\_COMMAND, 581  
setIn, 579  
setOut, 579  
setUmbrellaPosition, 580  
toggle\_state, 582  
UMBRELLA\_SUBSYSTEM\_NAME, 581  
UmbrellaOperator, 578  
updateHold, 579  
updateSplit, 579  
updateToggle, 580  
wisco::user::wings, 40  
EWingsControlMode, 40  
wisco::user::wings::WingsOperator, 582  
EToggleState, 583  
left\_toggle\_state, 589  
m\_controller, 589  
m\_robot, 589  
right\_toggle\_state, 590  
SET\_LEFT\_WING\_COMMAND, 589  
SET\_RIGHT\_WING\_COMMAND, 589  
setLeftWing, 584  
setRightWing, 584  
setWings, 584  
setWingsPosition, 588  
toggle\_state, 589  
updateDualHold, 585  
updateDualSplit, 585  
updateDualToggle, 585  
updateSingleHold, 586  
updateSingleSplit, 586  
updateSingleToggle, 587  
WINGS\_SUBSYSTEM\_NAME, 588  
WingsOperator, 583

withClock  
wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
362  
wisco::robot::subsystems::intake::PIDIntakeBuilder,  
423  
wisco::robot::subsystems::loader::PIDLoaderBuilder,  
447  
wisco::robot::subsystems::position::InertialOdometryBuilder,  
476  
withClosedState  
wisco::robot::subsystems::hang::PistonClawBuilder,  
387  
withDelay  
wisco::control::boomerang::PIDBoomerangBuilder,  
169  
wisco::control::motion::PIDTurnBuilder, 199  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
338  
wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
362  
wisco::robot::subsystems::intake::PIDIntakeBuilder,  
423  
wisco::robot::subsystems::loader::PIDLoaderBuilder,  
447  
wisco::robot::subsystems::position::InertialOdometryBuilder,  
476  
withDistanceSensor  
wisco::robot::subsystems::intake::DistanceVisionBallDetectorBuilder,  
403  
wisco::robot::subsystems::position::DistancePositionResetterBuilder,  
459  
withEngagedState  
wisco::robot::subsystems::hang::PistonWinchBuilder,  
398  
withGearRatio  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
315  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
342  
withHeadingSensor  
wisco::robot::subsystems::position::InertialOdometryBuilder,  
477  
withInchesPerRadian  
wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
365  
withLead  
wisco::control::boomerang::PIDBoomerangBuilder,  
171  
withLeftMotor  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
314  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
340  
withLeftPiston  
wisco::robot::subsystems::wings::PistonWingsBuilder,  
512  
withLeftVelocityProfile  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,

**339**  
 withLinearDistanceTrackingOffset  
 wisco::robot::subsystems::position::InertialOdometryBuilder  
**478**  
 withLinearDistanceTrackingSensor  
 wisco::robot::subsystems::position::InertialOdometryBuilder  
**478**  
 withLinearPID  
 wisco::control::boomerang::PIDBoomerangBuilder,  
**170**  
 withLocalTheta  
 wisco::robot::subsystems::position::DistancePositionResetter  
**460**  
 withLocalX  
 wisco::robot::subsystems::position::DistancePositionResetter  
**459**  
 withLocalY  
 wisco::robot::subsystems::position::DistancePositionResetter  
**460**  
 withMass  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
**341**  
 withMatchLoadPosition  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**449**  
 withMomentOfInertia  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder  
**342**  
 withMotor  
 wisco::robot::subsystems::elevator::PIDElevatorBuilder  
**364**  
 wisco::robot::subsystems::intake::PIDIntakeBuilder,  
**425**  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**449**  
 withMutex  
 wisco::control::boomerang::PIDBoomerangBuilder,  
**169**  
 wisco::control::motion::PDTurnBuilder,  
**199**  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder  
**338**  
 wisco::robot::subsystems::elevator::PIDElevatorBuilder  
**363**  
 wisco::robot::subsystems::intake::PIDIntakeBuilder,  
**424**  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**447**  
 wisco::robot::subsystems::position::InertialOdometryBuilder  
**477**  
 withOutState  
 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder  
**498**  
 wisco::robot::subsystems::wings::PistonWingsBuilder,  
**513**  
 withPID  
 wisco::control::motion::PDTurnBuilder,  
**200**  
 wisco::robot::subsystems::elevator::PIDElevatorBuilder  
**364**  
 wisco::robot::subsystems::intake::PIDIntakeBuilder,  
**425**  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**448**  
 withPiston  
 wisco::robot::subsystems::hang::PistonClawBuilder,  
**386**  
 wisco::robot::subsystems::hang::PistonToggleArmBuilder,  
**392**  
 wisco::robot::subsystems::hang::PistonWinchBuilder,  
**398**  
 wisco::robot::subsystems::umbrella::PistonUmbrellaBuilder,  
**498**  
 withPositionTolerance  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**450**  
 withRadius  
 wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
**316**  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
**341**  
 withReadyPosition  
 wisco::robot::subsystems::loader::PIDLoaderBuilder,  
**449**  
 withRightMotor  
 wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
**314**  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
**340**  
 withRightPiston  
 wisco::robot::subsystems::wings::PistonWingsBuilder,  
**512**  
 withRightVelocityProfile  
 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
**340**  
 withRollerRadius  
 wisco::robot::subsystems::intake::PIDIntakeBuilder,  
**425**  
 withRotationalPID  
 wisco::control::boomerang::PIDBoomerangBuilder,  
**170**  
 withRotationSensor  
 wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
**364**  
 withStrafeDistanceTrackingOffset  
 wisco::robot::subsystems::position::InertialOdometryBuilder,  
**479**  
 withStrafeDistanceTrackingSensor  
 wisco::robot::subsystems::position::InertialOdometryBuilder,  
**479**  
 withTargetTolerance  
 wisco::control::boomerang::PIDBoomerangBuilder,  
**171**  
 withTask  
 wisco::control::motion::PDTurnBuilder,  
**200**  
 withTargetVelocity  
 wisco::control::motion::PDTurnBuilder,  
**201**  
 wisco::control::boomerang::PIDBoomerangBuilder,

170  
wisco::control::motion::PIDTurnBuilder, 199  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
339  
wisco::robot::subsystems::elevator::PIDElevatorBuilder,  
363  
wisco::robot::subsystems::intake::PIDIntakeBuilder,  
424  
wisco::robot::subsystems::loader::PIDLoaderBuilder,  
448  
wisco::robot::subsystems::position::InertialOdometryBuilder,  
477  
withUpState  
wisco::robot::subsystems::hang::PistonToggleArmBuilder,  
392  
withVelocityToVoltage  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
315  
withWheelRadius  
wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder,  
316  
wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder,  
342  
writeConfiguration  
wisco::menu::LvglMenu, 268

x  
wisco::robot::subsystems::position::Position, 486  
xV  
wisco::robot::subsystems::position::Position, 486

y  
wisco::robot::subsystems::position::Position, 486  
yV  
wisco::robot::subsystems::position::Position, 486