WISCO_OverUnder

Generated by Doxygen 1.10.0

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

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 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.

Classes

- class PID
  A general-purpose PID controller.

4.6.1 Detailed Description

Namespace for control algorithms.

Author

Nathan Sandvig

4.7 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 TrackingWheel

  A tracking wheel sensor.

Enumerations

- enum class DistanceBooleanMode { **ABOVE_THRESHOLD** , **BELOW_THRESHOLD** , **BETWEEN_↩ THRESHOLD** }

  The modes of a distance boolean sensor.

4.7.1 Detailed Description

The namespace for the hardware abstraction layer.

Author

Nathan Sandvig

4.7.2 Enumeration Type Documentation

4.7.2.1 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 };
```

4.8 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 IRotationSensor

  Interface for rotation sensors.

4.8.1 Detailed Description

Namespace for the io types.

Author

Nathan Sandvig

4.9 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.

4.9.1 Detailed Description

Interface for the menu system.

Program data related to the menu system.

Author

Nathan Sandvig

4.9.2 Function Documentation

4.9.2.1 startButtonEventHandler()

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

Event handler function for the start button.

Parameters

event	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 }
```

4.9.2.2 settingsButtonEventHandler()

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

Event handler function for the settings button.

Parameters

event	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 }
```

4.9.2.3 settingsBackButtonEventHandler()

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

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

Parameters

event	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 }
```

4.9.2.4 settingsButtonMatrixEventHandler()

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

Event handler function for the button matrices in settings.

Parameters

event	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 }
```

4.10 wisco::profiles Namespace Reference

Namespace for the available driver profiles.

Classes

- class HenryProfile

  Driver profile for Henry.
- class JohnProfile

  Driver profile for John.

4.10.1 Detailed Description

Namespace for the available driver profiles.

Author

Nathan Sandvig

4.11 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.

4.11.1 Detailed Description

The namespace that holds all robot classes.

Author

Nathan Sandvig

4.12 wisco::robot::subsystems Namespace Reference

Namespace for all robot subsystems.

Namespaces

- namespace drive

  Namespace for drive classes.
- namespace elevator

  Namespace for elevator classes.
- namespace intake

  Namespace for intake classes.
- namespace position

  Namespace for all position subsystem classes.

4.12.1 Detailed Description

Namespace for all robot subsystems.

Author

Nathan Sandvig

4.13 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.

4.13.1 Detailed Description

Namespace for drive classes.

Author

Nathan Sandvig

4.14 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.

4.14.1 Detailed Description

Namespace for elevator classes.

Author

Nathan Sandvig

4.15 wisco::robot::subsystems::intake Namespace Reference

Namespace for intake classes.

Classes

- 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.

4.15.1 Detailed Description

Namespace for intake classes.

Author

Nathan Sandvig

4.16 wisco::robot::subsystems::position Namespace Reference

Namespace for all position subsystem classes.

Classes

- 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 IPositionTracker

  Interface for position tracking subsystems.
- struct Position

  Holds a robot position.
- class PositionSubsystem

  Adapter from a position tracker to a robot subsystem.

4.16.1 Detailed Description

Namespace for all position subsystem classes.

Author

Nathan Sandvig

4.17 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.

4.17.1 Detailed Description

Namespace for the rtos interface of the library.

Author

Nathan Sandvig

4.18 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.

4.18.1 Detailed Description

Namespace for all testing functions.

Author

Nathan Sandvig

4.19 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.

4.19.1 Detailed Description

Namespace for pros-based testing functions.

Author

Nathan Sandvig

4.19.2 Variable Documentation

4.19.2.1 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/"};
```

4.20 wisco::user Namespace Reference

Namespace for all user interactive components.

Classes

- class DifferentialDriveOperator

  Runs the operator-controlled differential drive voltage settings.
- class ElevatorOperator

  Runs the operator-controlled elevator position settings.
- class IController

  Interface for a controller.
- class IntakeOperator

  Runs the operator-controlled intake 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.
- enum class EControl {
  ELEVATOR_IN , **ELEVATOR_FIELD** , **ELEVATOR_MATCH_LOAD** , **ELEVATOR_OUT** ,
  ELEVATOR_TOGGLE , **INTAKE_IN** , **INTAKE_OUT** , **INTAKE_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** , **INTAKE** }

  Defines all different control types.
- enum class EElevatorControlMode { **MANUAL** , **PRESET_SPLIT** , **PRESET_TOGGLE_SINGLE** , **PRESET_TOGGLE_LADDER** }

  Defines all different elevator control formats.
- enum class EIntakeControlMode { **SINGLE_TOGGLE** , **SPLIT_HOLD** , **SPLIT_TOGGLE** }

  Defines all different intake control formats.

4.20.1 Detailed Description

Namespace for all user interactive components.

Author

Nathan Sandvig

4.20.2 Enumeration Type Documentation

4.20.2.1 EChassisControlMode

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

Defines all different chassis control formats.

Author

Nathan Sandvig

Definition at line 25 of file EChassisControlMode.hpp.

```
00026 {
00027     SINGLE_ARCADE_LEFT,
00028     SINGLE_ARCADE_RIGHT,
00029     SPLIT_ARCADE_LEFT,
00030     SPLIT_ARCADE_RIGHT,
00031     TANK
00032 };
```

4.20.2.2 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_OUT,
00031     ELEVATOR_TOGGLE,
00032     INTAKE_IN,
00033     INTAKE_OUT,
00034     INTAKE_TOGGLE
00035 };
```

4.20.2.3 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 };
```

4.20.2.4 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 };
```

4.20.2.5 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     INTAKE
00030 };
```

4.20.2.6 EElevatorControlMode

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

Defines all different elevator control formats.

Author

Nathan Sandvig

Definition at line 25 of file EElevatorControlMode.hpp.

```
00026 {
00027     MANUAL,
00028     PRESET_SPLIT,
00029     PRESET_TOGGLE_SINGLE,
00030     PRESET_TOGGLE_LADDER
00031 };
```

4.20.2.7 EIntakeControlMode

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

Defines all different intake control formats.

Author

Nathan Sandvig

Definition at line 25 of file EIntakeControlMode.hpp.

```
00026 {
00027     SINGLE_TOGGLE,
00028     SPLIT_HOLD,
00029     SPLIT_TOGGLE
00030 };
```

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

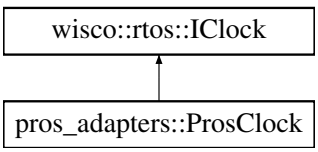
Definition at line 3 of file MatchControllerFactory.cpp.

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

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.

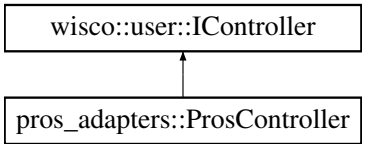
Definition at line 10 of file ProsClock.cpp.

```
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

controller	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

params	

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

analog_channel	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

digital_channel	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

digital_channel	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

pattern	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]
```

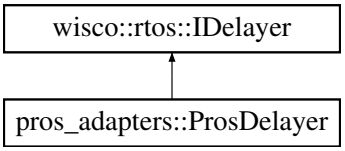
The last time the rumble was refreshed.

Definition at line 122 of file ProsController.hpp.
```
00122 {};
```

5.4 pros_adapters::ProsDelayer Class Reference

Pros rtos delay adapter for the wisco rtos IDelayer interface.

Inheritance diagram for pros_adapters::ProsDelayer:



Public Member Functions

- std::unique_ptr< wisco::rtos::IDelayer > clone () const override

  Clones the IDelayer 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::IDelayer

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

  Destroy the IDelayer object.

5.4.1 Detailed Description

Pros rtos delay adapter for the wisco rtos IDelayer interface.

Author

Nathan Sandvig

Definition at line 20 of file ProsDelayer.hpp.

5.4.2 Member Function Documentation

5.4.2.1 clone()

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

Clones the IDelayer object.

Returns

std::unique_ptr<IDelayer> The cloned IDelayer object

Implements wisco::rtos::IDelayer.

Definition at line 5 of file ProsDelayer.cpp.

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

5.4.2.2 delay()

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

Delays the rtos system for a number of milliseconds.

Parameters

millis	The number of milliseconds to delay

Implements wisco::rtos::IDelayer.

Definition at line 10 of file ProsDelayer.cpp.

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

5.4.2.3 delayUntil()

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

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

Parameters

time	The time in milliseconds to delay until

Implements wisco::rtos::IDelayer.

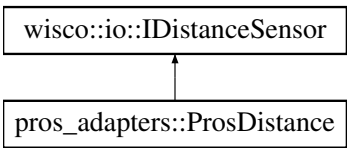
Definition at line 15 of file ProsDelayer.cpp.

```
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

sensor	The sensor being adapted
tuning_constant	The tuning constant multiplier for the sensor
tuning_offset	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]
```

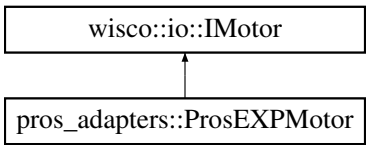
The tuning offset for the sensor.

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

motor	The motor being adapted

Definition at line 5 of file ProsEXPMotor.cpp.

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

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

volts	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]
```

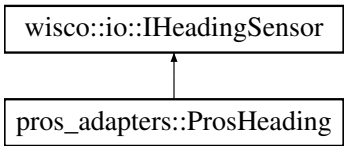
The motor being adapted.

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

sensor	The sensor being adapted
tuning_constant	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

heading	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

rotation	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]
```

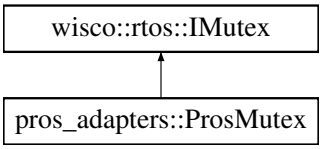
The tuning constant for the sensor.

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]
```

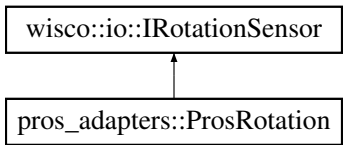
The mutex being adapted.

Definition at line 29 of file ProsMutex.hpp.
```
00029 {};
```

5.9 pros_adapters::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.

5.9.1 Detailed Description

Pros rotation sensor adapter for the wisco IRotationSensor interface.

Author

Nathan Sandvig

Definition at line 23 of file ProsRotation.hpp.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 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                                                   : m_sensor{std::move(sensor)}
00006 {
00007
00008 }
```

5.9.3 Member Function Documentation

5.9.3.1 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 }
```

5.9.3.2 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 }
```

5.9.3.3 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 }
```

5.9.3.4 setRotation()

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

Set the rotation of the sensor in radians.

Parameters

rotation	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 }
```

5.9.3.5 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 }
```

5.9.4 Member Data Documentation

5.9.4.1 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};
```

5.9.4.2 m_sensor

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

The rotation sensor being adapted.

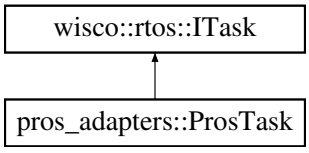
Definition at line 36 of file ProsRotation.hpp.

```
00036 {};
```

5.10 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.

5.10.1 Detailed Description

Pros rtos task adapter for the wisco rtos ITask interface.

Author

Nathan Sandvig

Definition at line 22 of file ProsTask.hpp.

5.10.2 Member Function Documentation

5.10.2.1 start()

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

Starts the task.

Parameters

function	The function to run in the task
parameters	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 }
```

5.10.2.2 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 }
```

5.10.2.3 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 }
```

5.10.2.4 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 }
```

5.10.2.5 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 }
```

5.10.3 Member Data Documentation

5.10.3.1 task

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

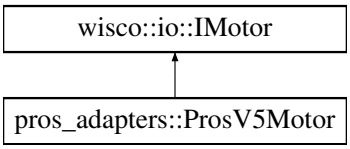
The pros task being adapted.

Definition at line 29 of file ProsTask.hpp.
```
00029 {};
```

5.11 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.

5.11.1 Detailed Description

Pros v5 smart motor adapter for the wisco IMotor interface.

Author

Nathan Sandvig

Definition at line 24 of file ProsV5Motor.hpp.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ProsV5Motor()

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

Construct a new Pros V5 Motor object.

Parameters

motor	The motor being adapted

Definition at line 5 of file ProsV5Motor.cpp.

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

5.11.3 Member Function Documentation

5.11.3.1 initialize()

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

Initializes the motor.

Implements wisco::io::IMotor.

Definition at line 10 of file ProsV5Motor.cpp.

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

5.11.3.2 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 19 of file ProsV5Motor.cpp.

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

5.11.3.3 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 24 of file ProsV5Motor.cpp.

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

5.11.3.4 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 29 of file ProsV5Motor.cpp.

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

5.11.3.5 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 34 of file ProsV5Motor.cpp.

```
00035 {
00036     double ratio{};
00037
00038     if (m_motor)
00039     {
00040         pros::MotorGears gearing{m_motor->get_gearing()};
00041         if (cartridge_map.contains(gearing))
00042             ratio = cartridge_map.at(gearing);
00043         else
00044             ratio = NO_CARTRIDGE;
00045     }
00046
00047     return ratio;
00048 }
```

5.11.3.6 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 50 of file ProsV5Motor.cpp.

```
00051 {
00052     double angular_velocity{};
00053
00054     if (m_motor)
00055         angular_velocity = m_motor->get_actual_velocity() * VELOCITY_CONVERSION;
00056
00057     return angular_velocity;
00058 }
```

5.11.3.7 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 60 of file ProsV5Motor.cpp.

```
00061 {
00062     double angular_velocity{};
00063
00064     if (m_motor)
00065         angular_velocity = m_motor->get_position() * (POSITION_CONVERSION / getGearRatio());
00066
00067     return angular_velocity;
00068 }
```

5.11.3.8 setVoltage()

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

Set the voltage input to the motor in Volts.

Parameters

volts	The voltage input in Volts

Implements wisco::io::IMotor.

Definition at line 70 of file ProsV5Motor.cpp.

```
00071 {
00072     int millivolts{static_cast<int>(volts * VOLTAGE_CONVERSION)};
00073     millivolts = std::min(millivolts, MAX_MILLIVOLTS);
00074     millivolts = std::max(millivolts, -MAX_MILLIVOLTS);
00075
00076     if (m_motor)
00077         m_motor->move_voltage(millivolts);
00078 }
```

5.11.4 Member Data Documentation

5.11.4.1 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     };
```

5.11.4.2 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};
```

5.11.4.3 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};
```

5.11.4.4 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};
```

5.11.4.5 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};
```

5.11.4.6 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};
```

5.11.4.7 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};
```

5.11.4.8 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};
```

5.11.4.9 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};
```

5.11.4.10 m_motor

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

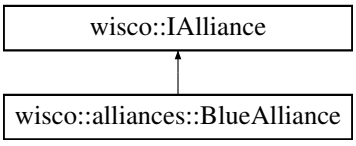
The motor being adapted.

Definition at line 90 of file ProsV5Motor.hpp.
```
00090 {};
```

5.12 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.

5.12.1 Detailed Description

The blue match alliance.

Author

Nathan Sandvig

Definition at line 28 of file BlueAlliance.hpp.

5.12.2 Member Function Documentation

5.12.2.1 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 }
```

5.12.3 Member Data Documentation

5.12.3.1 ALLIANCE_NAME

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

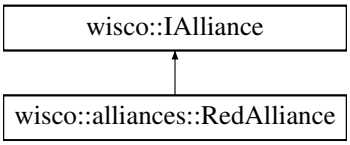
The name of the alliance.

Definition at line 35 of file BlueAlliance.hpp.
```
00035 {"BLUE"};
```

5.13 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.

5.13.1 Detailed Description

The red match alliance.

Author

Nathan Sandvig

Definition at line 28 of file RedAlliance.hpp.

5.13.2 Member Function Documentation

5.13.2.1 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 }
```

5.13.3 Member Data Documentation

5.13.3.1 ALLIANCE_NAME

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

The name of the alliance.

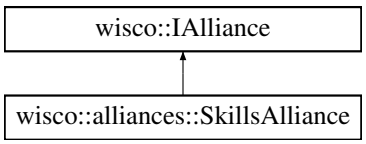
Definition at line 35 of file RedAlliance.hpp.

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

5.14 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.

5.14.1 Detailed Description

The skills alliance.

Author

Nathan Sandvig

Definition at line 28 of file SkillsAlliance.hpp.

5.14.2 Member Function Documentation

5.14.2.1 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 }
```

5.14.3 Member Data Documentation

5.14.3.1 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"};
```

5.15 wisco::AutonomousManager Class Reference

Manages the execution of the autonomous routine.

Public Member Functions

- void setAutonomous (std::unique_ptr< IAutonomous > &autonomous)
  Set the autonomous routine.
- void initializeAutonomous (std::shared_ptr< robot::Robot > robot)
  Initialize the autonomous routine.
- void runAutonomous (std::shared_ptr< robot::Robot > robot)
  Run the autonomous routine.

Private Attributes

- std::unique_ptr< IAutonomous > m_autonomous {}
  The autonomous routine.

5.15.1 Detailed Description

Manages the execution of the autonomous routine.

Author

Nathan Sandvig

Definition at line 21 of file AutonomousManager.hpp.

5.15.2 Member Function Documentation

5.15.2.1 setAutonomous()

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

Set the autonomous routine.

Parameters

autonomous	The autonomous routine

Definition at line 6 of file AutonomousManager.cpp.

```
00007 {
00008     m_autonomous = std::move(autonomous);
00009 }
```

5.15.2.2 initializeAutonomous()

```
void wisco::AutonomousManager::initializeAutonomous (
            std::shared_ptr< robot::Robot > robot )
```

Initialize the autonomous routine.

Parameters

robot	The robot being controlled

Definition at line 11 of file AutonomousManager.cpp.

```
00012 {
00013
00014 }
```

5.15.2.3 runAutonomous()

```
void wisco::AutonomousManager::runAutonomous (
            std::shared_ptr< robot::Robot > robot )
```

Run the autonomous routine.

Parameters

robot	The robot being controlled

Definition at line 16 of file AutonomousManager.cpp.

```
00017 {
00018
00019 }
```

5.15.3 Member Data Documentation

5.15.3.1 m_autonomous

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

The autonomous routine.

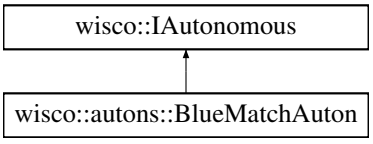
Definition at line 28 of file AutonomousManager.hpp.

```
00028 {};
```

5.16 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< robot::Robot > robot) override

  Initialize the autonomous.
- void run (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_MATCH"}

  The name of the autonomous.

5.16.1 Detailed Description

The auton for the blue robot in matches.

Author

Nathan Sandvig

Definition at line 26 of file BlueMatchAuton.hpp.

5.16.2 Member Function Documentation

5.16.2.1 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 7 of file BlueMatchAuton.cpp.

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

5.16.2.2 initialize()

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

Initialize the autonomous.

Implements wisco::IAutonomous.

Definition at line 12 of file BlueMatchAuton.cpp.

```
00013 {
00014
00015 }
```

5.16.2.3 run()

```
void wisco::autons::BlueMatchAuton::run (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Run the autonomous.

Implements wisco::IAutonomous.

Definition at line 17 of file BlueMatchAuton.cpp.

```
00018 {
00019
00020 }
```

5.16.3 Member Data Documentation

5.16.3.1 AUTONOMOUS_NAME

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

The name of the autonomous.

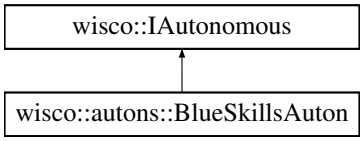
Definition at line 33 of file BlueMatchAuton.hpp.

```
00033 {"B_MATCH"};
```

5.17 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< robot::Robot > robot) override

  Initialize the autonomous.
- void run (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.

5.17.1 Detailed Description

The auton for the blue robot in skills.

Author

Nathan Sandvig

Definition at line 26 of file BlueSkillsAuton.hpp.

5.17.2 Member Function Documentation

5.17.2.1 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;
00010 }
```

5.17.2.2 initialize()

```
void wisco::autons::BlueSkillsAuton::initialize (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Initialize the autonomous.

Implements wisco::IAutonomous.

Definition at line 12 of file BlueSkillsAuton.cpp.

```
00013 {
00014
00015 }
```

5.17.2.3 run()

```
void wisco::autons::BlueSkillsAuton::run (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Run the autonomous.

Implements wisco::IAutonomous.

Definition at line 17 of file BlueSkillsAuton.cpp.

```
00018 {
00019
00020 }
```

5.17.3 Member Data Documentation

5.17.3.1 AUTONOMOUS_NAME

```
constexpr char wisco::autons::BlueSkillsAuton::AUTONOMOUS_NAME[] {"B_SKILLS"}  [static], [constexpr],
[private]
```

The name of the autonomous.

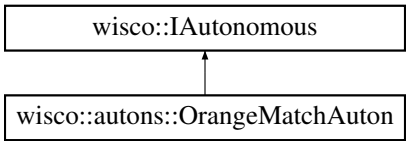
Definition at line 33 of file BlueSkillsAuton.hpp.

```
00033 {"B_SKILLS"};
```

5.18 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< robot::Robot > robot) override

  Initialize the autonomous.
- void run (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.

5.18.1 Detailed Description

The auton for the orange robot in matches.

Author

Nathan Sandvig

Definition at line 26 of file OrangeMatchAuton.hpp.

5.18.2 Member Function Documentation

5.18.2.1 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 }
```

5.18.2.2 initialize()

```
void wisco::autons::OrangeMatchAuton::initialize (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Initialize the autonomous.

Implements wisco::IAutonomous.

Definition at line 12 of file OrangeMatchAuton.cpp.

```
00013 {
00014
00015 }
```

5.18.2.3 run()

```
void wisco::autons::OrangeMatchAuton::run (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Run the autonomous.

Implements wisco::IAutonomous.

Definition at line 17 of file OrangeMatchAuton.cpp.

```
00018 {
00019
00020 }
```

5.18.3 Member Data Documentation

5.18.3.1 AUTONOMOUS_NAME

```
constexpr char wisco::autons::OrangeMatchAuton::AUTONOMOUS_NAME[] {"O_MATCH"}  [static], [constexpr],
[private]
```

The name of the autonomous.

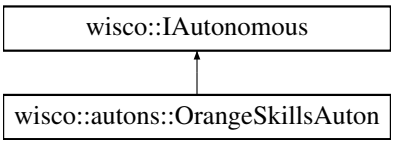
Definition at line 33 of file OrangeMatchAuton.hpp.

```
00033 {"O_MATCH"};
```

5.19 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< robot::Robot > robot) override

  Initialize the autonomous.
- void run (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.

5.19.1 Detailed Description

The auton for the orange robot in skills.

Author

Nathan Sandvig

Definition at line 26 of file OrangeSkillsAuton.hpp.

5.19.2 Member Function Documentation

5.19.2.1 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 }
```

5.19.2.2 initialize()

```
void wisco::autons::OrangeSkillsAuton::initialize (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Initialize the autonomous.

Implements wisco::IAutonomous.

Definition at line 12 of file OrangeSkillsAuton.cpp.

```
00013 {
00014
00015 }
```

5.19.2.3 run()

```
void wisco::autons::OrangeSkillsAuton::run (
            std::shared_ptr< robot::Robot > robot )  [override], [virtual]
```

Run the autonomous.

Implements wisco::IAutonomous.

Definition at line 17 of file OrangeSkillsAuton.cpp.

```
00018 {
00019
00020 }
```

5.19.3 Member Data Documentation

5.19.3.1 AUTONOMOUS_NAME

```
constexpr char wisco::autons::OrangeSkillsAuton::AUTONOMOUS_NAME[] {"R_SKILLS"}  [static],
[constexpr], [private]
```

The name of the autonomous.

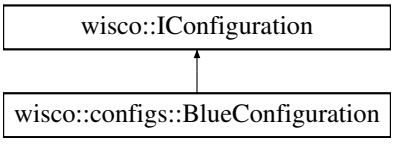
Definition at line 33 of file OrangeSkillsAuton.hpp.

```
00033 {"R_SKILLS"};
```

5.20 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< 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 {9}

  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 uint8_t ODOMETRY_LINEAR_PORT {8}

  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 uint8_t ODOMETRY_STRAFE_PORT {2}

  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 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 {11}

  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 {12}

  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 {-13}

  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 {-14}

  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 {17}

  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 {18}

  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 {-19}

  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 {-20}

  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 / 1.43}

  The conversion from velocity to voltage on the drive Current calculation = 12 volts to 1.43 meters per second.
- static constexpr double DRIVE_MASS {9.89}

  The mass of the drive.
- static constexpr double DRIVE_RADIUS {6.5 ∗ 2.54 / 100}

  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 {}

  The KP for the intake PID.
- static constexpr double INTAKE_KI {}

  The KI for the intake PID.
- static constexpr double INTAKE_KD {}

  The KD for the intake PID.
- static constexpr int8_t INTAKE_MOTOR_1_PORT {}

  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 {}

  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 {}

  The radius of the intake roller.
- static constexpr double ELEVATOR_KP {}

  The KP for the elevator PID.
- static constexpr double ELEVATOR_KI {}

  The KI for the elevator PID.
- static constexpr double ELEVATOR_KD {}

  The KD for the elevator PID.
- static constexpr int8_t ELEVATOR_MOTOR_1_PORT {}

  The first elevator motor port.
- static constexpr pros::v5::MotorGears ELEVATOR_MOTOR_1_GEARSET {pros::E_MOTOR_GEARSET_18}

  The first elevator motor gearset.
- static constexpr int8_t ELEVATOR_MOTOR_2_PORT {}

  The second elevator motor port.
- static constexpr pros::v5::MotorGears ELEVATOR_MOTOR_2_GEARSET {pros::E_MOTOR_GEARSET_18}

  The second elevator motor gearset.
- static constexpr int8_t ELEVATOR_ROTATION_SENSOR_PORT {}

  The elevator rotation sensor port.
- static constexpr double ELEVATOR_INCHES_PER_RADIAN {}

  The number of inches moved per radian on the elevator.

5.20.1 Detailed Description

The hardware configuration of the blue robot.

Author

Nathan Sandvig

Definition at line 54 of file BlueConfiguration.hpp.

5.20.2 Member Function Documentation

5.20.2.1 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 }
```

5.20.2.2 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 12 of file BlueConfiguration.cpp.

```
00013 {
00014     std::unique_ptr<pros::Controller>
      pros_controller{std::make_unique<pros::Controller>(pros::E_CONTROLLER_MASTER)};
00015     std::shared_ptr<user::IController>
      pros_controller_controller{std::make_shared<pros_adapters::ProsController>(pros_controller)};
00016     return pros_controller_controller;
00017 }
```

5.20.2.3 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 19 of file BlueConfiguration.cpp.

```
00020 {
00021     std::shared_ptr<robot::Robot> robot{std::make_unique<robot::Robot>()};
00022
00023     // Odometry creation
00024     robot::subsystems::position::InertialOdometryBuilder inertial_odometry_builder{};
00025     std::unique_ptr<wisco::rtos::IClock>
      odometry_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00026     std::unique_ptr<wisco::rtos::IDelayer>
      odometry_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00027     std::unique_ptr<wisco::rtos::IMutex>
      odometry_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00028     std::unique_ptr<wisco::rtos::ITask>
      odometry_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00029     std::unique_ptr<pros::Imu>
      odometry_pros_heading{std::make_unique<pros::Imu>(ODOMETRY_HEADING_PORT)};
00030     std::unique_ptr<wisco::io::IHeadingSensor>
      odometry_pros_heading_sensor{std::make_unique<pros_adapters::ProsHeading>(odometry_pros_heading,
      ODOMETRY_HEADING_TUNING_CONSTANT)};
00031     std::unique_ptr<pros::Rotation>
      odometry_pros_linear_rotation{std::make_unique<pros::Rotation>(ODOMETRY_LINEAR_PORT)};
00032     std::unique_ptr<wisco::io::IRotationSensor>
      odometry_pros_linear_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(odometry_pros_linear_rotation)};
00033     std::unique_ptr<wisco::io::IDistanceTrackingSensor>
      odometry_linear_tracking_wheel{std::make_unique<wisco::hal::TrackingWheel>(odometry_pros_linear_rotation_sensor,
      ODOMETRY_LINEAR_RADIUS)};
00034     std::unique_ptr<pros::Rotation>
      odometry_pros_strafe_rotation{std::make_unique<pros::Rotation>(ODOMETRY_STRAFE_PORT)};
00035     std::unique_ptr<wisco::io::IRotationSensor>
      odometry_pros_strafe_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(odometry_pros_strafe_rotation)};
00036     std::unique_ptr<wisco::io::IDistanceTrackingSensor>
      odometry_strafe_tracking_wheel{std::make_unique<wisco::hal::TrackingWheel>(odometry_pros_strafe_rotation_sensor,
      ODOMETRY_STRAFE_RADIUS)};
```

```
00037     std::unique_ptr<wisco::robot::subsystems::position::IPositionTracker> inertial_odometry
00038     {
00039         inertial_odometry_builder.
00040         withClock(odometry_pros_clock)->
00041         withDelayer(odometry_pros_delayer)->
00042         withMutex(odometry_pros_mutex)->
00043         withTask(odometry_pros_task)->
00044         withHeadingSensor(odometry_pros_heading_sensor)->
00045         withLinearDistanceTrackingSensor(odometry_linear_tracking_wheel)->
00046         withLinearDistanceTrackingOffset(ODOMETRY_LINEAR_OFFSET)->
00047         withStrafeDistanceTrackingSensor(odometry_strafe_tracking_wheel)->
00048         withStrafeDistanceTrackingOffset(ODOMETRY_STRAFE_OFFSET)->
00049         build()
00050     };
00051     std::unique_ptr<wisco::robot::ASubsystem>
      odometry_subsystem{std::make_unique<wisco::robot::subsystems::position::PositionSubsystem>(inertial_odometry)};
00052     robot->addSubsystem(odometry_subsystem);
00053
00054     // Drive creation
00055     if (DRIVE_KINEMATIC)
00056     {
00057         wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder
      kinematic_differential_drive_builder{};
00058         std::unique_ptr<wisco::rtos::IDelayer>
      drive_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00059         std::unique_ptr<wisco::rtos::IMutex>
      drive_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00060         std::unique_ptr<wisco::rtos::ITask>
      drive_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00061         std::unique_ptr<wisco::rtos::IClock>
      drive_left_velocity_profile_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00062         std::unique_ptr<wisco::robot::subsystems::drive::IVelocityProfile>
      drive_left_velocity_profile{std::make_unique<wisco::robot::subsystems::drive::CurveVelocityProfile>(drive_left_velocity_
      DRIVE_VELOCITY_PROFILE_JERK_RATE, DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION)};
00063         std::unique_ptr<wisco::rtos::IClock>
      drive_right_velocity_profile_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00064         std::unique_ptr<wisco::robot::subsystems::drive::IVelocityProfile>
      drive_right_velocity_profile{std::make_unique<wisco::robot::subsystems::drive::CurveVelocityProfile>(drive_right_velocit
      DRIVE_VELOCITY_PROFILE_JERK_RATE, DRIVE_VELOCITY_PROFILE_MAX_ACCELERATION)};
00065         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_1{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_1_PORT,
      DRIVE_LEFT_MOTOR_1_GEARSET)};
00066         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_1)};
00067         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_2{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_2_PORT,
      DRIVE_LEFT_MOTOR_2_GEARSET)};
00068         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_2)};
00069         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_3{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_3_PORT,
      DRIVE_LEFT_MOTOR_3_GEARSET)};
00070         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_3)};
00071         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_4{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_4_PORT,
      DRIVE_LEFT_MOTOR_4_GEARSET)};
00072         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_4)};
00073         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_1{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_1_PORT,
      DRIVE_RIGHT_MOTOR_1_GEARSET)};
00074         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_1)};
00075         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_2{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_2_PORT,
      DRIVE_RIGHT_MOTOR_2_GEARSET)};
00076         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_2)};
00077         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_3{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_3_PORT,
      DRIVE_RIGHT_MOTOR_3_GEARSET)};
00078         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_3)};
00079         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_4{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_4_PORT,
      DRIVE_RIGHT_MOTOR_4_GEARSET)};
00080         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_4)};
00081         std::unique_ptr<wisco::robot::subsystems::drive::IDifferentialDrive> differential_drive
00082         {
00083             kinematic_differential_drive_builder.
00084             withDelayer(drive_pros_delayer)->
00085             withMutex(drive_pros_mutex)->
00086             withTask(drive_pros_task)->
00087             withLeftVelocityProfile(drive_left_velocity_profile)->
00088             withRightVelocityProfile(drive_right_velocity_profile)->
```

```
00089             withLeftMotor(drive_pros_left_motor_1_motor)->
00090             withLeftMotor(drive_pros_left_motor_2_motor)->
00091             withLeftMotor(drive_pros_left_motor_3_motor)->
00092             withLeftMotor(drive_pros_left_motor_4_motor)->
00093             withRightMotor(drive_pros_right_motor_1_motor)->
00094             withRightMotor(drive_pros_right_motor_2_motor)->
00095             withRightMotor(drive_pros_right_motor_3_motor)->
00096             withRightMotor(drive_pros_right_motor_4_motor)->
00097             withMass(DRIVE_MASS)->
00098             withRadius(DRIVE_RADIUS)->
00099             withMomentOfInertia(DRIVE_MOMENT_OF_INERTIA)->
00100             withGearRatio(DRIVE_GEAR_RATIO)->
00101             withWheelRadius(DRIVE_WHEEL_RADIUS)->
00102             build()
00103         };
00104         std::unique_ptr<wisco::robot::ASubsystem>
      drive_subsystem{std::make_unique<wisco::robot::subsystems::drive::DifferentialDriveSubsystem>(differential_drive)};
00105         robot->addSubsystem(drive_subsystem);
00106     }
00107     else
00108     {
00109         wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder
      direct_differential_drive_builder{};
00110         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_1{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_1_PORT,
      DRIVE_LEFT_MOTOR_1_GEARSET)};
00111         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_1)};
00112         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_2{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_2_PORT,
      DRIVE_LEFT_MOTOR_2_GEARSET)};
00113         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_2)};
00114         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_3{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_3_PORT,
      DRIVE_LEFT_MOTOR_3_GEARSET)};
00115         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_3)};
00116         std::unique_ptr<pros::Motor>
      drive_pros_left_motor_4{std::make_unique<pros::Motor>(DRIVE_LEFT_MOTOR_4_PORT,
      DRIVE_LEFT_MOTOR_4_GEARSET)};
00117         std::unique_ptr<wisco::io::IMotor>
      drive_pros_left_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_left_motor_4)};
00118         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_1{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_1_PORT,
      DRIVE_RIGHT_MOTOR_1_GEARSET)};
00119         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_1)};
00120         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_2{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_2_PORT,
      DRIVE_RIGHT_MOTOR_2_GEARSET)};
00121         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_2)};
00122         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_3{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_3_PORT,
      DRIVE_RIGHT_MOTOR_3_GEARSET)};
00123         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_3_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_3)};
00124         std::unique_ptr<pros::Motor>
      drive_pros_right_motor_4{std::make_unique<pros::Motor>(DRIVE_RIGHT_MOTOR_4_PORT,
      DRIVE_RIGHT_MOTOR_4_GEARSET)};
00125         std::unique_ptr<wisco::io::IMotor>
      drive_pros_right_motor_4_motor{std::make_unique<pros_adapters::ProsV5Motor>(drive_pros_right_motor_4)};
00126         std::unique_ptr<wisco::robot::subsystems::drive::IDifferentialDrive> differential_drive
00127         {
00128             direct_differential_drive_builder.
00129             withLeftMotor(drive_pros_left_motor_1_motor)->
00130             withLeftMotor(drive_pros_left_motor_2_motor)->
00131             withLeftMotor(drive_pros_left_motor_3_motor)->
00132             withLeftMotor(drive_pros_left_motor_4_motor)->
00133             withRightMotor(drive_pros_right_motor_1_motor)->
00134             withRightMotor(drive_pros_right_motor_2_motor)->
00135             withRightMotor(drive_pros_right_motor_3_motor)->
00136             withRightMotor(drive_pros_right_motor_4_motor)->
00137             withVelocityToVoltage(DRIVE_VELOCITY_TO_VOLTAGE)->
00138             withGearRatio(DRIVE_GEAR_RATIO)->
00139             withWheelRadius(DRIVE_WHEEL_RADIUS)->
00140             build()
00141         };
00142         std::unique_ptr<wisco::robot::ASubsystem>
      drive_subsystem{std::make_unique<wisco::robot::subsystems::drive::DifferentialDriveSubsystem>(differential_drive)};
00143         robot->addSubsystem(drive_subsystem);
00144     }
00145
00146     // Intake creation
00147     wisco::robot::subsystems::intake::PIDIntakeBuilder pid_intake_builder{};
00148     std::unique_ptr<wisco::rtos::IClock>
```

```
    intake_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00149     std::unique_ptr<wisco::rtos::IDelayer>
    intake_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00150     std::unique_ptr<wisco::rtos::IMutex>
    intake_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00151     std::unique_ptr<wisco::rtos::ITask> intake_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00152     wisco::control::PID intake_pid{intake_pros_clock, INTAKE_KP, INTAKE_KI, INTAKE_KD};
00153     std::unique_ptr<pros::Motor>
    intake_pros_motor_1{std::make_unique<pros::Motor>(INTAKE_MOTOR_1_PORT, INTAKE_MOTOR_1_GEARSET)};
00154     std::unique_ptr<wisco::io::IMotor>
    intake_pros_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(intake_pros_motor_1)};
00155     std::unique_ptr<pros::Motor>
    intake_pros_motor_2{std::make_unique<pros::Motor>(INTAKE_MOTOR_2_PORT, INTAKE_MOTOR_2_GEARSET)};
00156     std::unique_ptr<wisco::io::IMotor>
    intake_pros_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(intake_pros_motor_2)};
00157     std::unique_ptr<wisco::robot::subsystems::intake::IIntake> pid_intake
00158     {
00159         pid_intake_builder.
00160         withClock(intake_pros_clock)->
00161         withDelayer(intake_pros_delayer)->
00162         withMutex(intake_pros_mutex)->
00163         withTask(intake_pros_task)->
00164         withPID(intake_pid)->
00165         withMotor(intake_pros_motor_1_motor)->
00166         withMotor(intake_pros_motor_2_motor)->
00167         withRollerRadius(INTAKE_ROLLER_RADIUS)->
00168         build()
00169     };
00170     std::unique_ptr<wisco::robot::ASubsystem>
    intake_subsystem{std::make_unique<wisco::robot::subsystems::intake::IntakeSubsystem>(pid_intake)};
00171     robot->addSubsystem(intake_subsystem);
00172
00173     // Elevator creation
00174     wisco::robot::subsystems::elevator::PIDElevatorBuilder pid_elevator_builder{};
00175     std::unique_ptr<wisco::rtos::IClock>
    elevator_pros_clock{std::make_unique<pros_adapters::ProsClock>()};
00176     std::unique_ptr<wisco::rtos::IDelayer>
    elevator_pros_delayer{std::make_unique<pros_adapters::ProsDelayer>()};
00177     std::unique_ptr<wisco::rtos::IMutex>
    elevator_pros_mutex{std::make_unique<pros_adapters::ProsMutex>()};
00178     std::unique_ptr<wisco::rtos::ITask>
    elevator_pros_task{std::make_unique<pros_adapters::ProsTask>()};
00179     wisco::control::PID elevator_pid{elevator_pros_clock, ELEVATOR_KP, ELEVATOR_KI, ELEVATOR_KD};
00180     std::unique_ptr<pros::Motor>
    elevator_pros_motor_1{std::make_unique<pros::Motor>(ELEVATOR_MOTOR_1_PORT, ELEVATOR_MOTOR_1_GEARSET)};
00181     std::unique_ptr<wisco::io::IMotor>
    elevator_pros_motor_1_motor{std::make_unique<pros_adapters::ProsV5Motor>(elevator_pros_motor_1)};
00182     std::unique_ptr<pros::Motor>
    elevator_pros_motor_2{std::make_unique<pros::Motor>(ELEVATOR_MOTOR_2_PORT, ELEVATOR_MOTOR_2_GEARSET)};
00183     std::unique_ptr<wisco::io::IMotor>
    elevator_pros_motor_2_motor{std::make_unique<pros_adapters::ProsV5Motor>(elevator_pros_motor_2)};
00184     if (ELEVATOR_ROTATION_SENSOR_PORT)
00185     {
00186         std::unique_ptr<pros::Rotation>
    elevator_pros_rotation{std::make_unique<pros::Rotation>(ELEVATOR_ROTATION_SENSOR_PORT)};
00187         std::unique_ptr<wisco::io::IRotationSensor>
    elevator_pros_rotation_sensor{std::make_unique<pros_adapters::ProsRotation>(elevator_pros_rotation)};
00188         pid_elevator_builder.withRotationSensor(elevator_pros_rotation_sensor);
00189     }
00190     std::unique_ptr<wisco::robot::subsystems::elevator::IElevator> pid_elevator
00191     {
00192         pid_elevator_builder.
00193         withClock(elevator_pros_clock)->
00194         withDelayer(elevator_pros_delayer)->
00195         withMutex(elevator_pros_mutex)->
00196         withTask(elevator_pros_task)->
00197         withPID(elevator_pid)->
00198         withMotor(elevator_pros_motor_1_motor)->
00199         withMotor(elevator_pros_motor_2_motor)->
00200         withInchesPerRadian(ELEVATOR_INCHES_PER_RADIAN)->
00201         build()
00202     };
00203     std::unique_ptr<wisco::robot::ASubsystem>
    elevator_subsystem{std::make_unique<wisco::robot::subsystems::elevator::ElevatorSubsystem>(pid_elevator)};
00204     robot->addSubsystem(elevator_subsystem);
00205
00206     return robot;
00207 }
```

5.20.3 Member Data Documentation

5.20.3.1 CONFIGURATION_NAME

```
constexpr char wisco::configs::BlueConfiguration::CONFIGURATION_NAME[] {"BLUE"}  [static],
[constexpr], [private]
```

The name of the configuration.

Definition at line 61 of file BlueConfiguration.hpp.
```
00061 {"BLUE"};
```

5.20.3.2 ODOMETRY_HEADING_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ODOMETRY_HEADING_PORT {9}  [static], [constexpr],
[private]
```

The port for the odometry heading sensor.

Definition at line 67 of file BlueConfiguration.hpp.
```
00067 {9};
```

5.20.3.3 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 73 of file BlueConfiguration.hpp.
```
00073 {1.014};
```

5.20.3.4 ODOMETRY_LINEAR_PORT

```
constexpr uint8_t wisco::configs::BlueConfiguration::ODOMETRY_LINEAR_PORT {8}  [static], [constexpr],
[private]
```

The port for the odometry linear distance tracking sensor.

Definition at line 79 of file BlueConfiguration.hpp.
```
00079 {8};
```

5.20.3.5 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 85 of file BlueConfiguration.hpp.
```
00085 {1.22};
```

5.20.3.6 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 91 of file BlueConfiguration.hpp.
```
00091 {3.35};
```

5.20.3.7 ODOMETRY_STRAFE_PORT

```
constexpr uint8_t wisco::configs::BlueConfiguration::ODOMETRY_STRAFE_PORT {2}  [static], [constexpr],
[private]
```

The port for the odometry strafe distance tracking sensor.

Definition at line 97 of file BlueConfiguration.hpp.
```
00097 {2};
```

5.20.3.8 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 103 of file BlueConfiguration.hpp.
```
00103 {1.22};
```

5.20.3.9 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 109 of file BlueConfiguration.hpp.
```
00109 {4.6};
```

5.20.3.10 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 115 of file BlueConfiguration.hpp.
```
00115 {false};
```

5.20.3.11 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 121 of file BlueConfiguration.hpp.
```
00121 {20.0};
```

5.20.3.12 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 127 of file BlueConfiguration.hpp.
```
00127 {5.0};
```

5.20.3.13 DRIVE_LEFT_MOTOR_1_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_1_PORT {11}  [static],
[constexpr], [private]
```

The first left drive motor port.

Definition at line 133 of file BlueConfiguration.hpp.
```
00133 {11};
```

5.20.3.14 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 139 of file BlueConfiguration.hpp.
```
00139 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.15 DRIVE_LEFT_MOTOR_2_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_2_PORT {12}  [static],
[constexpr], [private]
```

The second left drive motor port.

Definition at line 145 of file BlueConfiguration.hpp.
```
00145 {12};
```

5.20.3.16 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 151 of file BlueConfiguration.hpp.
```
00151 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.17 DRIVE_LEFT_MOTOR_3_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_3_PORT {-13}  [static],
[constexpr], [private]
```

The third left drive motor port.

Definition at line 157 of file BlueConfiguration.hpp.
```
00157 {-13};
```

5.20.3.18 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 163 of file BlueConfiguration.hpp.
```
00163 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.19 DRIVE_LEFT_MOTOR_4_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_LEFT_MOTOR_4_PORT {-14}  [static],
[constexpr], [private]
```

The fourth left drive motor port.

Definition at line 169 of file BlueConfiguration.hpp.
```
00169 {-14};
```

5.20.3.20 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 175 of file BlueConfiguration.hpp.
```
00175 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.21 DRIVE_RIGHT_MOTOR_1_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_1_PORT {17}  [static],
[constexpr], [private]
```

The first right drive motor port.

Definition at line 181 of file BlueConfiguration.hpp.
```
00181 {17};
```

5.20.3.22 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 187 of file BlueConfiguration.hpp.
```
00187 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.23 DRIVE_RIGHT_MOTOR_2_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_2_PORT {18}  [static],
[constexpr], [private]
```

The second right drive motor port.

Definition at line 193 of file BlueConfiguration.hpp.
```
00193 {18};
```

5.20.3.24 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 199 of file BlueConfiguration.hpp.
```
00199 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.25 DRIVE_RIGHT_MOTOR_3_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_3_PORT {-19}  [static],
[constexpr], [private]
```

The third right drive motor port.

Definition at line 205 of file BlueConfiguration.hpp.
```
00205 {-19};
```

5.20.3.26 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 211 of file BlueConfiguration.hpp.
```
00211 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.27 DRIVE_RIGHT_MOTOR_4_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::DRIVE_RIGHT_MOTOR_4_PORT {-20}  [static],
[constexpr], [private]
```

The fourth right drive motor port.

Definition at line 217 of file BlueConfiguration.hpp.
```
00217 {-20};
```

5.20.3.28 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 223 of file BlueConfiguration.hpp.
```
00223 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.29 DRIVE_VELOCITY_TO_VOLTAGE

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_VELOCITY_TO_VOLTAGE {12.0 / 1.43}
[static], [constexpr], [private]
```

The conversion from velocity to voltage on the drive Current calculation = 12 volts to 1.43 meters per second.

Definition at line 230 of file BlueConfiguration.hpp.
```
00230 {12.0 / 1.43};
```

5.20.3.30 DRIVE_MASS

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_MASS {9.89}  [static], [constexpr],
[private]
```

The mass of the drive.

Definition at line 236 of file BlueConfiguration.hpp.
```
00236 {9.89};
```

5.20.3.31 DRIVE_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::DRIVE_RADIUS {6.5 * 2.54 / 100}  [static],
[constexpr], [private]
```

The radius of the drive.

Definition at line 242 of file BlueConfiguration.hpp.
```
00242 {6.5 * 2.54 / 100};
```

5.20.3.32 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 248 of file BlueConfiguration.hpp.
```
00248 {19.887 * DRIVE_RADIUS * DRIVE_MASS};
```

5.20.3.33 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 254 of file BlueConfiguration.hpp.
```
00254 {600.0 / 331.4};
```

5.20.3.34 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 260 of file BlueConfiguration.hpp.
```
00260 {3.25 * 2.54 / 100};
```

5.20.3.35 INTAKE_KP

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KP {}  [static], [constexpr], [private]
```

The KP for the intake PID.

Definition at line 266 of file BlueConfiguration.hpp.
```
00266 {};
```

5.20.3.36 INTAKE_KI

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KI {}  [static], [constexpr], [private]
```

The KI for the intake PID.

Definition at line 272 of file BlueConfiguration.hpp.
```
00272 {};
```

5.20.3.37 INTAKE_KD

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_KD {}  [static], [constexpr], [private]
```

The KD for the intake PID.

Definition at line 278 of file BlueConfiguration.hpp.
```
00278 {};
```

5.20.3.38 INTAKE_MOTOR_1_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::INTAKE_MOTOR_1_PORT {}  [static], [constexpr],
[private]
```

The first intake motor port.

Definition at line 284 of file BlueConfiguration.hpp.
```
00284 {};
```

5.20.3.39 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 290 of file BlueConfiguration.hpp.
```
00290 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.40 INTAKE_MOTOR_2_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::INTAKE_MOTOR_2_PORT {}  [static], [constexpr],
[private]
```

The second intake motor port.

Definition at line 296 of file BlueConfiguration.hpp.
```
00296 {};
```

5.20.3.41 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 302 of file BlueConfiguration.hpp.
```
00302 {pros::E_MOTOR_GEARSET_06};
```

5.20.3.42 INTAKE_ROLLER_RADIUS

```
constexpr double wisco::configs::BlueConfiguration::INTAKE_ROLLER_RADIUS {}  [static], [constexpr],
[private]
```

The radius of the intake roller.

Definition at line 308 of file BlueConfiguration.hpp.
```
00308 {};
```

5.20.3.43 ELEVATOR_KP

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KP {}  [static], [constexpr],
[private]
```

The KP for the elevator PID.

Definition at line 314 of file BlueConfiguration.hpp.
```
00314 {};
```

5.20.3.44 ELEVATOR_KI

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KI {}  [static], [constexpr],
[private]
```

The KI for the elevator PID.

Definition at line 320 of file BlueConfiguration.hpp.
```
00320 {};
```

5.20.3.45 ELEVATOR_KD

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_KD {}  [static], [constexpr],
[private]
```

The KD for the elevator PID.

Definition at line 326 of file BlueConfiguration.hpp.
```
00326 {};
```

5.20.3.46 ELEVATOR_MOTOR_1_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_1_PORT {}  [static], [constexpr],
[private]
```

The first elevator motor port.

Definition at line 332 of file BlueConfiguration.hpp.
```
00332 {};
```

5.20.3.47 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 338 of file BlueConfiguration.hpp.
```
00338 {pros::E_MOTOR_GEARSET_18};
```

5.20.3.48 ELEVATOR_MOTOR_2_PORT

```
constexpr int8_t wisco::configs::BlueConfiguration::ELEVATOR_MOTOR_2_PORT {}  [static], [constexpr],
[private]
```

The second elevator motor port.

Definition at line 344 of file BlueConfiguration.hpp.
```
00344 {};
```

5.20.3.49 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 350 of file BlueConfiguration.hpp.
```
00350 {pros::E_MOTOR_GEARSET_18};
```

5.20.3.50 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 356 of file BlueConfiguration.hpp.
```
00356 {};
```

5.20.3.51 ELEVATOR_INCHES_PER_RADIAN

```
constexpr double wisco::configs::BlueConfiguration::ELEVATOR_INCHES_PER_RADIAN {}  [static],
[constexpr], [private]
```

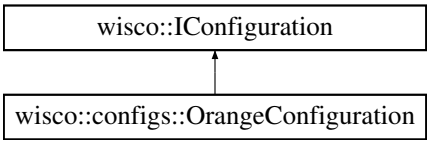
The number of inches moved per radian on the elevator.

Definition at line 362 of file BlueConfiguration.hpp.
```
00362 {};
```

5.21 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< 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.

5.21.1 Detailed Description

The hardware configuration of the orange robot.

Author

Nathan Sandvig

Definition at line 26 of file OrangeConfiguration.hpp.

5.21.2 Member Function Documentation

5.21.2.1 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;
00010 }
```

5.21.2.2 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 12 of file OrangeConfiguration.cpp.

```
00013 {
00014     return std::shared_ptr<user::IController>{};
00015 }
```

5.21.2.3 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 17 of file OrangeConfiguration.cpp.

```
00018 {
00019     return std::shared_ptr<robot::Robot>{};
00020 }
```

5.21.3 Member Data Documentation

5.21.3.1 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"};
```

5.22 wisco::control::PID Class Reference

A general-purpose PID controller.

Public Member Functions

- **PID** ()=default
  Construct a new PID object.
- PID (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.

5.22.1 Detailed Description

A general-purpose PID controller.

Author

Nathan Sandvig

Definition at line 29 of file PID.hpp.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 PID() [1/3]

```
wisco::control::PID::PID (
            std::unique_ptr< rtos::IClock > & clock,
            double kp,
            double ki,
            double kd )
```

Construct a new PID object.

Parameters

clock	The system clock
kp	The proportional constant
ki	The integral constant
kd	The derivative constant

Definition at line 7 of file PID.cpp.

```
00008     : m_clock{std::move(clock)}, m_kp{kp}, m_ki{ki}, m_kd{kd}
00009 {
00010
00011 }
```

5.22.2.2 PID() [2/3]

```
wisco::control::PID::PID (
            const PID & copy )
```

Construct a new PID object.

Parameters

copy	The PID 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 }
```

5.22.2.3 PID() [3/3]

```
wisco::control::PID::PID (
            PID && move )  [default]
```

Construct a new PID object.

Parameters

move	The PID object being moved

5.22.3 Member Function Documentation

5.22.3.1 getControlValue()

```
double wisco::control::PID::getControlValue (
            double current,
            double target )
```

Get the control value output of the PID controller.

Parameters

current	The current system value
target	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
00035     double error{target - current};
00036     accumulated_error += (error * time_change);
00037     double error_change{(error - last_error) / time_change};
00038     last_error = error;
00039
00040     return (m_kp * error) + (m_ki * accumulated_error) + (m_kd * error_change);
00041 }
```

5.22.3.2 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 }
```

5.22.3.3 operator=() [1/2]

```
PID & wisco::control::PID::operator= (
            const PID & rhs )
```

Copy assignment operator.

Parameters

rhs	The PID 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 }
```

5.22.3.4 operator=() [2/2]

```
PID & wisco::control::PID::operator= (
            PID && rhs )  [default]
```

Move assignment operator.

Parameters

rhs	The PID object being moved

Returns

PID& This PID object with the assigned values

5.22.4 Member Data Documentation

5.22.4.1 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 {};
```

5.22.4.2 m_kp

```
double wisco::control::PID::m_kp {}  [private]
```

The proportional constant.

Definition at line 42 of file PID.hpp.
```
00042 {};
```

5.22.4.3 m_ki

```
double wisco::control::PID::m_ki {}  [private]
```

The integral constant.

Definition at line 48 of file PID.hpp.
```
00048 {};
```

5.22.4.4 m_kd

```
double wisco::control::PID::m_kd {}  [private]
```

The derivative constant.

Definition at line 54 of file PID.hpp.
```
00054 {};
```

5.22.4.5 accumulated_error

```
double wisco::control::PID::accumulated_error {}  [private]
```

The accumulated error.

Definition at line 60 of file PID.hpp.
```
00060 {};
```

5.22.4.6 last_error

```
double wisco::control::PID::last_error {}  [private]
```

The error during the last timestep.

Definition at line 66 of file PID.hpp.
```
00066 {};
```

5.22.4.7 last_time

```
double wisco::control::PID::last_time {}  [private]
```

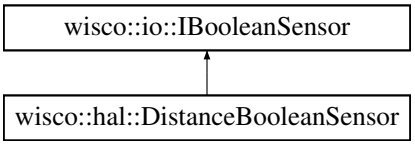
The system clock time during the last timestep.

Definition at line 72 of file PID.hpp.
```
00072 {};
```

5.23 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.

5.23.1 Detailed Description

A distance sensor used to create boolean outputs.

Author

Nathan Sandvig

Definition at line 32 of file DistanceBooleanSensor.hpp.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 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

distance_sensor	The distance sensor
mode	The mode of the sensor
lower_threshold	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},
      m_upper_threshold{lower_threshold}
00011 {
00012
00013 }
```

5.23.2.2 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

distance_sensor	The distance sensor
mode	The mode of the sensor
lower_threshold	The lower threshold of the sensor
upper_threshold	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},
      m_upper_threshold{upper_threshold}
00020 {
00021
00022 }
```

5.23.3 Member Function Documentation

5.23.3.1 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 }
```

5.23.3.2 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 }
```

5.23.3.3 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 }
```

5.23.4 Member Data Documentation

5.23.4.1 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 {};
```

5.23.4.2 m_mode

```
DistanceBooleanMode wisco::hal::DistanceBooleanSensor::m_mode {}  [private]
```

The mode of the sensor.

Definition at line 45 of file DistanceBooleanSensor.hpp.

```
00045 {};
```

5.23.4.3 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 {};
```

5.23.4.4 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 {};
```

5.23.4.5 value

```
bool wisco::hal::DistanceBooleanSensor::value {}  [private]
```

The current boolean value.

Definition at line 63 of file DistanceBooleanSensor.hpp.
```
00063 {};
```

5.24 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.

5.24.1 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.

5.24.2 Member Function Documentation

5.24.2.1 addMotor()

```
void wisco::hal::MotorGroup::addMotor (
            std::unique_ptr< io::IMotor > & motor )
```

Adds a motor to the motor group.

Parameters

motor	The motor being added to the group

Definition at line 7 of file MotorGroup.cpp.

```
00008 {
00009     motors.push_back(std::move(motor));
00010 }
```

5.24.2.2 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 }
```

5.24.2.3 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 }
```

5.24.2.4 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 }
```

5.24.2.5 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 }
```

5.24.2.6 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 }
```

5.24.2.7 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 }
```

5.24.2.8 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 }
```

5.24.2.9 setVoltage()

```
void wisco::hal::MotorGroup::setVoltage (
            double volts )
```

Set the voltage input to the motors in Volts.

Parameters

volts	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 }
```

5.24.2.10 operator=()

```
MotorGroup & wisco::hal::MotorGroup::operator= (
            MotorGroup & rhs )
```

Override for the assignment operator for MotorGroup.

Parameters

rhs	The MotorGroup 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 }
```

5.24.3 Member Data Documentation

5.24.3.1 motors

```
std::vector<std::unique_ptr<io::IMotor> > wisco::hal::MotorGroup::motors {}  [private]
```

The motors in the group.

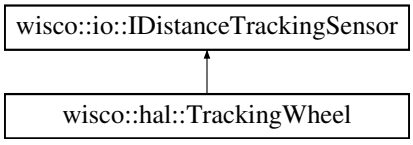
Definition at line 38 of file MotorGroup.hpp.

```
00038 {};
```

5.25 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.

5.25.1 Detailed Description

A tracking wheel sensor.

Author

Nathan Sandvig

Definition at line 28 of file TrackingWheel.hpp.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 TrackingWheel()

```
wisco::hal::TrackingWheel::TrackingWheel (
            std::unique_ptr< io::IRotationSensor > & sensor,
            double wheel_radius )
```

Construct a new Tracking Wheel object.

Parameters

sensor	The rotation sensor on the tracking wheel
wheel_radius	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 }
```

5.25.3 Member Function Documentation

5.25.3.1 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 }
```

5.25.3.2 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 }
```

5.25.3.3 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 }
```

5.25.3.4 setDistance()

```
void wisco::hal::TrackingWheel::setDistance (
              double distance ) [override], [virtual]
```

Set the distance tracked by the sensor in inches.

Parameters

distance	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 }
```

5.25.4 Member Data Documentation

5.25.4.1 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 {};
```

5.25.4.2 m_wheel_radius

```
double wisco::hal::TrackingWheel::m_wheel_radius {}  [private]
```

The radius of the wheel in inches.

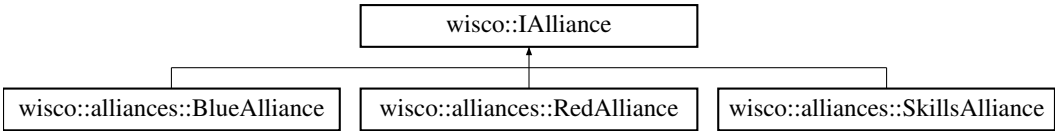
Definition at line 41 of file TrackingWheel.hpp.

```
00041 {};
```

5.26 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.

5.26.1 Detailed Description

Interface for the alliances for the robot.

Author

Nathan Sandvig

Definition at line 19 of file IAlliance.hpp.

5.26.2 Member Function Documentation

5.26.2.1 getName()

```
virtual std::string wisco::IAlliance::getName ( )  [pure virtual]
```

Get the name of the alliance.

Returns

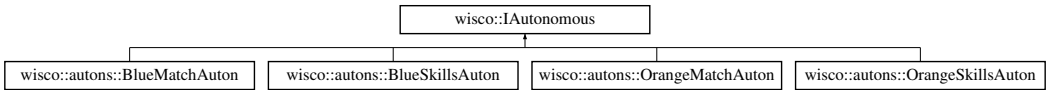
std::string The name of the alliance

Implemented in wisco::alliances::BlueAlliance, wisco::alliances::RedAlliance, and wisco::alliances::SkillsAlliance.

5.27 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< robot::Robot > robot)=0
  Initialize the autonomous.
- virtual void run (std::shared_ptr< robot::Robot > robot)=0
  Run the autonomous.

5.27.1 Detailed Description

Interface for the autonomous routines in the system.

Author

Nathan Sandvig

Definition at line 22 of file IAutonomous.hpp.

5.27.2 Member Function Documentation

5.27.2.1 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.

5.27.2.2 initialize()

```
virtual void wisco::IAutonomous::initialize (
            std::shared_ptr< robot::Robot > robot )  [pure virtual]
```

Initialize the autonomous.

Implemented in wisco::autons::BlueMatchAuton, wisco::autons::BlueSkillsAuton, wisco::autons::OrangeMatchAuton, and wisco::autons::OrangeSkillsAuton.

5.27.2.3 run()

```
virtual void wisco::IAutonomous::run (
            std::shared_ptr< robot::Robot > robot )  [pure virtual]
```

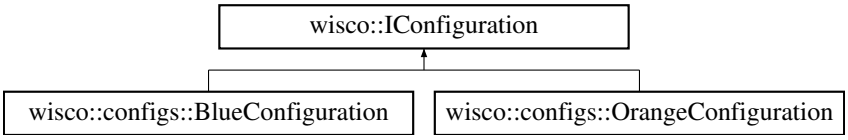
Run the autonomous.

Implemented in wisco::autons::BlueMatchAuton, wisco::autons::BlueSkillsAuton, wisco::autons::OrangeMatchAuton, and wisco::autons::OrangeSkillsAuton.

5.28 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< user::IController > buildController ()=0

  Build a controller using this configuration.
- virtual std::shared_ptr< robot::Robot > buildRobot ()=0

  Build a robot using this configuration.

5.28.1 Detailed Description

Interface for the configurations in the system.

Author

Nathan Sandvig

Definition at line 23 of file IConfiguration.hpp.

5.28.2 Member Function Documentation

5.28.2.1 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.

5.28.2.2 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.

5.28.2.3 buildRobot()

```
virtual std::shared_ptr< robot::Robot > wisco::IConfiguration::buildRobot ( )  [pure virtual]
```

Build a robot using this configuration.

Returns

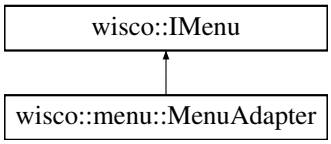
robot::Robot The robot built by this configuration

Implemented in wisco::configs::BlueConfiguration, and wisco::configs::OrangeConfiguration.

5.29 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.

5.29.1 Detailed Description

Interface for the menu system.

Author

Nathan Sandvig

Definition at line 19 of file IMenu.hpp.

5.29.2 Member Function Documentation

5.29.2.1 addAlliance()

```
virtual void wisco::IMenu::addAlliance (
            std::unique_ptr< IAlliance > & alliance )  [pure virtual]
```

Adds an alliance to the menu system.

Parameters

alliance	The new alliance

Implemented in wisco::menu::MenuAdapter.

5.29.2.2 addAutonomous()

```
virtual void wisco::IMenu::addAutonomous (
            std::unique_ptr< IAutonomous > & autonomous )  [pure virtual]
```

Adds an autonomous routine to the menu system.

Parameters

autonomous	The new autonomous routine

Implemented in wisco::menu::MenuAdapter.

5.29.2.3 addConfiguration()

```
virtual void wisco::IMenu::addConfiguration (
            std::unique_ptr< IConfiguration > & configuration )  [pure virtual]
```

Adds a hardware configuration to the menu system.

Parameters

configuration	The new hardware configuration

Implemented in wisco::menu::MenuAdapter.

5.29.2.4 addProfile()

```
virtual void wisco::IMenu::addProfile (
            std::unique_ptr< IProfile > & profile )  [pure virtual]
```

Adds a driver profile to the menu system.

Parameters

profile	The new driver profile

Implemented in wisco::menu::MenuAdapter.

5.29.2.5 display()

```
virtual void wisco::IMenu::display ( )  [pure virtual]
```

Display the menu.

Implemented in wisco::menu::MenuAdapter.

5.29.2.6 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.

5.29.2.7 getSystemConfiguration()

```
virtual SystemConfiguration wisco::IMenu::getSystemConfiguration ( )  [pure virtual]
```

Get the system configuration information.

Returns

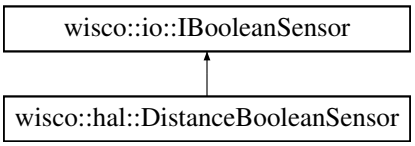
SystemConfiguration The system configuration information

Implemented in wisco::menu::MenuAdapter.

5.30 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.

5.30.1 Detailed Description

Interface for sensors that generate a boolean value.

Author

Nathan Sandvig

Definition at line 25 of file IBooleanSensor.hpp.

5.30.2 Member Function Documentation

5.30.2.1 initialize()

```
virtual void wisco::io::IBooleanSensor::initialize ( )  [pure virtual]
```

Initializes the sensor.

Implemented in wisco::hal::DistanceBooleanSensor.

5.30.2.2 reset()

```
virtual void wisco::io::IBooleanSensor::reset ( )  [pure virtual]
```

Resets the sensor.

Implemented in wisco::hal::DistanceBooleanSensor.

5.30.2.3 getValue()

```
virtual bool wisco::io::IBooleanSensor::getValue ( )  [pure virtual]
```

Get the boolean value of the sensor.

Returns

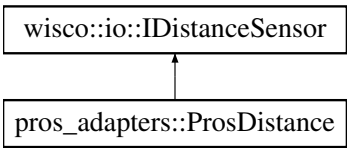
bool The value of the sensor

Implemented in wisco::hal::DistanceBooleanSensor.

5.31 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.

5.31.1 Detailed Description

Interface for distance tracking sensors.

Author

Nathan Sandvig

Definition at line 25 of file IDistanceSensor.hpp.

5.31.2 Member Function Documentation

5.31.2.1 initialize()

```
virtual void wisco::io::IDistanceSensor::initialize ( )  [pure virtual]
```

Initializes the sensor.

Implemented in pros_adapters::ProsDistance.

5.31.2.2 reset()

```
virtual void wisco::io::IDistanceSensor::reset ( )  [pure virtual]
```

Resets the sensor.

Implemented in pros_adapters::ProsDistance.

5.31.2.3 getDistance()

```
virtual double wisco::io::IDistanceSensor::getDistance ( )  [pure virtual]
```

Get the distance detected by the sensor in inches.

Returns

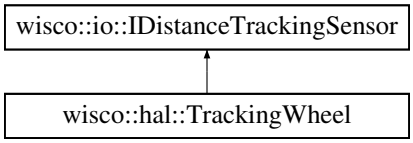
double The distance detected by the sensor

Implemented in pros_adapters::ProsDistance.

5.32 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.

5.32.1 Detailed Description

Interface for distance tracking sensors.

Author

Nathan Sandvig

Definition at line 23 of file IDistanceTrackingSensor.hpp.

5.32.2 Member Function Documentation

5.32.2.1 initialize()

```
virtual void wisco::io::IDistanceTrackingSensor::initialize ( )  [pure virtual]
```

Initializes the sensor.

Implemented in wisco::hal::TrackingWheel.

5.32.2.2 reset()

```
virtual void wisco::io::IDistanceTrackingSensor::reset ( )  [pure virtual]
```

Resets the sensor.

Implemented in wisco::hal::TrackingWheel.

5.32.2.3 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.

5.32.2.4 setDistance()

```
virtual void wisco::io::IDistanceTrackingSensor::setDistance (
            double distance )  [pure virtual]
```

Set the distance tracked by the sensor in inches.

Parameters

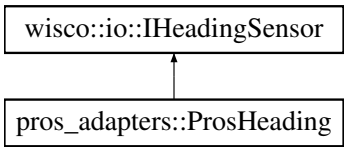
distance	The new distance tracked value

Implemented in wisco::hal::TrackingWheel.

5.33 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.

5.33.1 Detailed Description

Interface for heading sensors.

Author

Nathan Sandvig

Definition at line 23 of file IHeadingSensor.hpp.

5.33.2 Member Function Documentation

5.33.2.1 initialize()

```
virtual void wisco::io::IHeadingSensor::initialize ( )  [pure virtual]
```

Initializes the sensor.

Implemented in pros_adapters::ProsHeading.

5.33.2.2 reset()

```
virtual void wisco::io::IHeadingSensor::reset ( )  [pure virtual]
```

Resets the sensor.

Implemented in pros_adapters::ProsHeading.

5.33.2.3 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.

5.33.2.4 setHeading()

```
virtual void wisco::io::IHeadingSensor::setHeading (
            double heading )  [pure virtual]
```

Set the heading of the sensor in radians.

Parameters

heading	The heading in radians

Implemented in pros_adapters::ProsHeading.

5.33.2.5 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.

5.33.2.6 setRotation()

```
virtual void wisco::io::IHeadingSensor::setRotation (
            double rotation )  [pure virtual]
```

Set the rotation of the sensor in radians.

Parameters

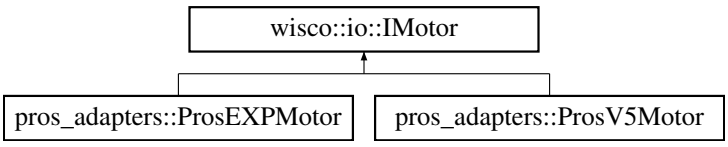
rotation	The rotation in radians

Implemented in pros_adapters::ProsHeading.

5.34 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.

5.34.1 Detailed Description

Interface for electric motors controlled by voltage.

Author

Nathan Sandvig

Definition at line 24 of file IMotor.hpp.

5.34.2 Member Function Documentation

5.34.2.1 initialize()

```
virtual void wisco::io::IMotor::initialize ( )  [pure virtual]
```

Initializes the motor.

Implemented in pros_adapters::ProsEXPMotor, and pros_adapters::ProsV5Motor.

5.34.2.2 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.

5.34.2.3 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.

5.34.2.4 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.

5.34.2.5 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.

5.34.2.6 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.

5.34.2.7 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.

5.34.2.8 setVoltage()

```
virtual void wisco::io::IMotor::setVoltage (
            double volts )  [pure virtual]
```

Set the voltage input to the motor in Volts.

Parameters

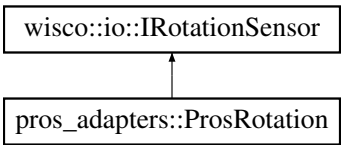
volts	The voltage input in Volts

Implemented in pros_adapters::ProsEXPMotor, and pros_adapters::ProsV5Motor.

5.35 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.

5.35.1 Detailed Description

Interface for rotation sensors.

Author

Nathan Sandvig

Definition at line 23 of file IRotationSensor.hpp.

5.35.2 Member Function Documentation

5.35.2.1 initialize()

```
virtual void wisco::io::IRotationSensor::initialize ( )  [pure virtual]
```

Initializes the rotation sensor.

Implemented in pros_adapters::ProsRotation.

5.35.2.2 reset()

```
virtual void wisco::io::IRotationSensor::reset ( )  [pure virtual]
```

Resets the rotation sensor.

Implemented in pros_adapters::ProsRotation.

5.35.2.3 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.

5.35.2.4 setRotation()

```
virtual void wisco::io::IRotationSensor::setRotation (
            double rotation )  [pure virtual]
```

Set the rotation of the sensor in radians.

Parameters

rotation	The number of radians of rotation

Implemented in pros_adapters::ProsRotation.

5.35.2.5 getAngle()

```
virtual double wisco::io::IRotationSensor::getAngle ( )  [pure virtual]
```

Get the angle of the sensor in radians.

Returns

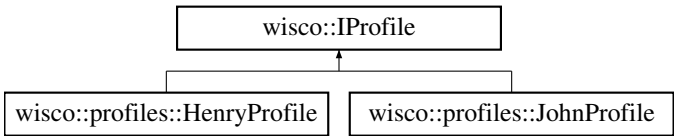
double The angle in radians

Implemented in pros_adapters::ProsRotation.

5.36 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 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.

5.36.1 Detailed Description

Interface for the profiles in the system.

Author

Nathan Sandvig

Definition at line 24 of file IProfile.hpp.

5.36.2 Member Function Documentation

5.36.2.1 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.

5.36.2.2 getControlMode()

```
virtual int wisco::IProfile::getControlMode (
            user::EControlType control_type ) const  [pure virtual]
```

Get the control mode for a specific control type.

Parameters

control_type	The control type

Returns

int The control mode

Implemented in wisco::profiles::HenryProfile, and wisco::profiles::JohnProfile.

5.36.2.3 getAnalogControlMapping()

```
virtual user::EControllerAnalog wisco::IProfile::getAnalogControlMapping (
            user::EControl control ) const  [pure virtual]
```

Get the mapping of a control to analog inputs.

Parameters

control	The control

Returns

user::EControllerAnalog The mapping of this control to a analog input

Implemented in wisco::profiles::HenryProfile, and wisco::profiles::JohnProfile.

5.36.2.4 getDigitalControlMapping()

```
virtual user::EControllerDigital wisco::IProfile::getDigitalControlMapping (
            user::EControl control ) const  [pure virtual]
```

Get the mapping of a control to digital inputs.

Parameters

control	The control

Returns

user::EControllerDigital The mapping of this control to a digital input

Implemented in wisco::profiles::HenryProfile, and wisco::profiles::JohnProfile.

5.37 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::IDelayer > &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::IDelayer > m_delayer {}

  The rtos delayer.
- AutonomousManager autonomous_manager {}

  The autonomous management object.
- OPControlManager opcontrol_manager {m_clock, m_delayer}

  The opcontrol management object.
- 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.

5.37.1 Detailed Description

Handles the field controller inputs during a match.

Author

Nathan Sandvig

Definition at line 25 of file MatchController.hpp.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 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

menu	The menu to use in the match controller
clock	The rtos clock to use in the match controller
delayer	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)}, opcontrol_manager{clock,
      delayer}
00007 {
00008
00009 }
```

5.37.3 Member Function Documentation

5.37.3.1 initialize()

```
void wisco::MatchController::initialize ( )
```

Runs the robot initialization code.

Definition at line 11 of file MatchController.cpp.

```
00012 {
00013     if (m_menu)
00014     {
00015         m_menu->display();
00016         while (m_delayer && !m_menu->isStarted())
00017             m_delayer->delay(MENU_DELAY);
00018     }
00019
00020     SystemConfiguration system_configuration{};
00021     if (m_menu)
00022         system_configuration = m_menu->getSystemConfiguration();
```

```
00023     autonomous_manager.setAutonomous(system_configuration.autonomous);
00024     opcontrol_manager.setProfile(system_configuration.profile);
00025     controller = system_configuration.configuration->buildController();
00026     robot = system_configuration.configuration->buildRobot();
00027     if (robot)
00028     {
00029         robot->initialize();
00030         autonomous_manager.initializeAutonomous(robot);
00031         if (controller)
00032         {
00033             controller->initialize();
00034             controller->run();
00035             opcontrol_manager.initializeOpcontrol(controller, robot);
00036         }
00037     }
00038 }
```

5.37.3.2 disabled()

```
void wisco::MatchController::disabled ( )
```

Runs the robot disablement code.

Definition at line 40 of file MatchController.cpp.

```
00041 {
00042
00043 }
```

5.37.3.3 competitionInitialize()

```
void wisco::MatchController::competitionInitialize ( )
```

Runs the robot competition initialization code.

Definition at line 45 of file MatchController.cpp.

```
00046 {
00047
00048 }
```

5.37.3.4 autonomous()

```
void wisco::MatchController::autonomous ( )
```

Runs the robot autonomous code.

Definition at line 50 of file MatchController.cpp.

```
00051 {
00052     if (robot)
00053         autonomous_manager.runAutonomous(robot);
00054 }
```

5.37.3.5 operatorControl()

```
void wisco::MatchController::operatorControl ( )
```

Runs the robot operator control code.

Definition at line 56 of file MatchController.cpp.

```
00057 {
00058     if (controller && robot)
00059         opcontrol_manager.runOpcontrol(controller, robot);
00060 }
```

5.37.4 Member Data Documentation

5.37.4.1 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 32 of file MatchController.hpp.
```
00032 {10};
```

5.37.4.2 m_menu

```
std::unique_ptr<IMenu> wisco::MatchController::m_menu {}  [private]
```

The menu system.

Definition at line 38 of file MatchController.hpp.
```
00038 {};
```

5.37.4.3 m_clock

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

The rtos clock.

Definition at line 44 of file MatchController.hpp.
```
00044 {};
```

5.37.4.4 m_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::MatchController::m_delayer {}  [private]
```

The rtos delayer.

Definition at line 50 of file MatchController.hpp.
```
00050 {};
```

5.37.4.5 autonomous_manager

```
AutonomousManager wisco::MatchController::autonomous_manager {}  [private]
```

The autonomous management object.

Definition at line 56 of file MatchController.hpp.
```
00056 {};
```

5.37.4.6 opcontrol_manager

```
OPControlManager wisco::MatchController::opcontrol_manager {m_clock, m_delayer}  [private]
```

The opcontrol management object.

Definition at line 62 of file MatchController.hpp.
```
00062 {m_clock, m_delayer};
```

5.37.4.7 controller

```
std::shared_ptr<user::IController> wisco::MatchController::controller {}  [private]
```

The user input controller.

Definition at line 68 of file MatchController.hpp.
```
00068 {};
```

5.37.4.8 robot

```
std::shared_ptr<robot::Robot> wisco::MatchController::robot {}  [private]
```

The robot being controlled.

Definition at line 74 of file MatchController.hpp.
```
00074 {};
```

5.38 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 [ ] {"/usd/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.

5.38.1 Detailed Description

Controls an lvgl-based menu selection system.

Author

Nathan Sandvig

Definition at line 60 of file LvglMenu.hpp.

5.38.2 Member Function Documentation

5.38.2.1 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),
      8));
00073     lv_style_set_border_opa(&button_default_style, LV_OPA_100);
00074     lv_style_set_border_width(&button_default_style, 2);
00075     lv_style_set_border_color(&button_default_style, lv_color_black());
00076     lv_style_set_text_color(&button_default_style, lv_color_black());
00077     lv_style_set_text_font(&button_default_style, &lv_font_montserrat_20);
00078
00079     // Create the pressed button style
00080     lv_style_init(&button_pressed_style);
00081     lv_style_set_radius(&button_pressed_style, 5);
00082     lv_style_set_bg_opa(&button_pressed_style, LV_OPA_100);
00083     lv_style_set_translate_y(&button_pressed_style, 3);
00084     lv_style_set_shadow_ofs_y(&button_pressed_style, 3);
00085     lv_style_set_bg_color(&button_pressed_style, lv_color_darken(lv_color_make(192, 192, 192), 16));
00086     lv_style_set_bg_grad_color(&button_pressed_style, lv_color_darken(lv_color_make(192, 192, 192),
      24));
00087     lv_style_set_border_opa(&button_pressed_style, LV_OPA_100);
00088     lv_style_set_border_width(&button_pressed_style, 2);
00089     lv_style_set_border_color(&button_pressed_style, lv_color_black());
00090     lv_style_set_text_color(&button_pressed_style, lv_color_black());
00091     lv_style_set_text_font(&button_pressed_style, &lv_font_montserrat_20);
00092
00093     // Create the default container style
00094     lv_style_init(&container_default_style);
00095     lv_style_set_radius(&container_default_style, 0);
00096     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 }
```

5.38.2.2 addOption()

```
void wisco::menu::LvglMenu::addOption (
            Option option )
```

Adds an option to the menu system.

Parameters

option	The option being added

Definition at line 125 of file LvglMenu.cpp.

```
00126 {
00127     options.push_back(option);
00128 }
```

5.38.2.3 removeOption()

```
void wisco::menu::LvglMenu::removeOption (
            const std::string & option_name )
```

Removes an option from the menu system.

Parameters

option_name	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 }
```

5.38.2.4 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 }
```

5.38.2.5 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),
      0);
00264     lv_obj_t* section{lv_menu_section_create(root_page)};
00265     lv_menu_set_sidebar_page(menu, root_page);
00266 
00267     // Create the back button
00268     lv_obj_t* back_btn{lv_menu_get_sidebar_header_back_btn(menu)};
00269     lv_obj_remove_style_all(back_btn);
00270     lv_obj_add_style(back_btn, &button_default_style, 0);
00271     lv_obj_add_style(back_btn, &button_pressed_style, LV_STATE_PRESSED);
00272     lv_obj_set_style_text_font(back_btn, &lv_font_montserrat_14, 0);
00273     lv_obj_set_style_text_font(back_btn, &lv_font_montserrat_14, LV_STATE_PRESSED);
00274     lv_obj_set_style_pad_all(back_btn, 3, 0);
00275     lv_obj_set_style_pad_all(back_btn, 3, LV_STATE_PRESSED);
00276     lv_obj_set_style_translate_y(back_btn, 0, LV_STATE_PRESSED);
00277     lv_obj_set_style_shadow_ofs_y(back_btn, 0, LV_STATE_PRESSED);
00278     lv_obj_t* back_btn_label{lv_label_create(back_btn)};
00279     lv_label_set_text(back_btn_label, "   Back");
00280     static void* back_user_data[]{ nullptr, nullptr };
00281     back_user_data[0] = menu;
00282     back_user_data[1] = this;
00283     lv_obj_add_event_cb(menu, settingsBackButtonEventHandler, LV_EVENT_CLICKED, back_user_data);
00284 
00285     static std::vector<std::shared_ptr<std::vector<const char*>> option_button_matrix_maps{};
00286     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),
      0), 0);
00291         lv_menu_separator_create(option_page);
00292 
00293         std::shared_ptr<std::vector<const char*>
      option_button_matrix_map{std::make_shared<std::vector<const char*>>()};
00294         uint8_t line_counter{};
00295         for (std::string& choice : option.choices)
00296         {
00297             if (line_counter >= BUTTONS_PER_LINE)
00298             {
00299                 option_button_matrix_map->push_back("\n");
00300                 line_counter = 0;
00301             }
00302             option_button_matrix_map->push_back(choice.c_str());
00303             ++line_counter;
00304         }
00305         option_button_matrix_map->push_back("");
00306         option_button_matrix_maps.push_back(option_button_matrix_map);
00307 
00308         lv_obj_t* option_button_matrix{lv_btnmatrix_create(option_page)};
00309         lv_btnmatrix_set_map(option_button_matrix, option_button_matrix_map->data());
00310         lv_btnmatrix_set_one_checked(option_button_matrix, true);
00311         lv_btnmatrix_set_btn_ctrl_all(option_button_matrix, LV_BTNMATRIX_CTRL_CHECKABLE);
00312         lv_btnmatrix_set_btn_ctrl(option_button_matrix, option.selected, LV_BTNMATRIX_CTRL_CHECKED);
00313         lv_obj_add_style(option_button_matrix, &button_matrix_main_style, LV_PART_MAIN);
00314         lv_obj_set_size(option_button_matrix, 300, 220);
00315 
00316         void* option_user_data[]{ nullptr };
00317         option_user_data[0] = &option;
00318         option_button_matrix_user_data.push_back(option_user_data);
00319         lv_obj_add_event_cb(option_button_matrix, settingsButtonMatrixEventHandler,
      LV_EVENT_VALUE_CHANGED, &option);
00320 
00321         lv_obj_t* option_menu_container{lv_menu_cont_create(section)};
00322         lv_obj_remove_style_all(option_menu_container);
00323         lv_obj_add_style(option_menu_container, &container_default_style, 0);
00324         lv_obj_add_style(option_menu_container, &container_pressed_style, LV_STATE_CHECKED);
```

```
00325         lv_obj_t* option_menu_container_label{lv_label_create(option_menu_container)};
00326         lv_label_set_text(option_menu_container_label, option.name.c_str());
00327         lv_menu_set_load_page_event(menu, option_menu_container, option_page);
00328     }
00329
00330     lv_event_send(lv_obj_get_child(lv_obj_get_child(lv_menu_get_cur_sidebar_page(menu), 0), 0),
      LV_EVENT_CLICKED, NULL);
00331 }
```

5.38.2.6 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 }
```

5.38.2.7 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
00366     configuration_file.close();
00367 }
```

5.38.2.8 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 }
```

5.38.2.9 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 }
```

5.38.2.10 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 }
```

5.38.2.11 getSelection()

```
std::string wisco::menu::LvglMenu::getSelection (
            const std::string & option_name )
```

Get the selection for an option.

Parameters

option_name	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 }
```

5.38.3 Member Data Documentation

5.38.3.1 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"};
```

5.38.3.2 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};
```

5.38.3.3 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};
```

5.38.3.4 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.

5.38.3.5 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.

5.38.3.6 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.

5.38.3.7 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.

5.38.3.8 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.

5.38.3.9 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.

5.38.3.10 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.

5.38.3.11 options

```
std::vector<Option> wisco::menu::LvglMenu::options {}  [private]
```

The options available in the menu.

Definition at line 127 of file LvglMenu.hpp.
00127 {};

5.38.3.12 complete

```
bool wisco::menu::LvglMenu::complete {false}  [private]
```

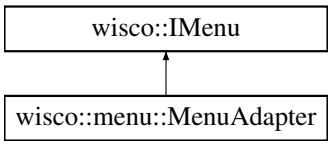
Whether or not selection is complete.

Definition at line 133 of file LvglMenu.hpp.
```
00133 {false};
```

5.39 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.

5.39.1 Detailed Description

This class adapts the menu system to the IMenu interface.

Author

Nathan Sandvig

Definition at line 29 of file MenuAdapter.hpp.

5.39.2 Member Function Documentation

5.39.2.1 addAlliance()

```
void wisco::menu::MenuAdapter::addAlliance (
            std::unique_ptr< IAlliance > & alliance )  [override], [virtual]
```

Adds an alliance to the menu system.

Parameters

alliance	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 }
```

5.39.2.2 addAutonomous()

```
void wisco::menu::MenuAdapter::addAutonomous (
            std::unique_ptr< IAutonomous > & autonomous )  [override], [virtual]
```

Adds an autonomous routine to the menu system.

Parameters

autonomous	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 }
```

5.39.2.3 addConfiguration()

```
void wisco::menu::MenuAdapter::addConfiguration (
            std::unique_ptr< IConfiguration > & configuration )  [override], [virtual]
```

Adds a hardware configuration to the menu system.

Parameters

configuration	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 }
```

5.39.2.4 addProfile()

```
void wisco::menu::MenuAdapter::addProfile (
            std::unique_ptr< IProfile > & profile )  [override], [virtual]
```

Adds a driver profile to the menu system.

Parameters

profile	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 }
```

5.39.2.5 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 }
```

5.39.2.6 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 }
```

5.39.2.7 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 }
```

5.39.3 Member Data Documentation

5.39.3.1 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"};
```

5.39.3.2 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"};
```

5.39.3.3 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"};
```

5.39.3.4 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"};
```

5.39.3.5 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 {};
```

5.39.3.6 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 {};
```

5.39.3.7 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 {};
```

5.39.3.8 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 {};
```

5.39.3.9 lvgl_menu

```
LvglMenu wisco::menu::MenuAdapter::lvgl_menu {}  [private]
```

The lvgl menu being adapted.

Definition at line 84 of file MenuAdapter.hpp.
```
00084 {};
```

5.40 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.

5.40.1 Detailed Description

An option in the menu system.

Author

Nathan Sandvig

Definition at line 26 of file Option.hpp.

5.40.2 Member Data Documentation

5.40.2.1 name

```
std::string wisco::menu::Option::name {}
```

The name of the option.

Definition at line 32 of file Option.hpp.
```
00032 {};
```

5.40.2.2 choices

```
std::vector<std::string> wisco::menu::Option::choices {}
```

The choices available for that option.

Definition at line 38 of file Option.hpp.
```
00038 {};
```

5.40.2.3 selected

```
int wisco::menu::Option::selected {}
```

The index of the selected choice.

Definition at line 44 of file Option.hpp.
```
00044 {};
```

5.41 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::IDelayer > &delayer)

  Construct a new OPControlManager object.
- void setProfile (std::unique_ptr< IProfile > &profile)

  Set the operator profile.
- void initializeOpcontrol (std::shared_ptr< user::IController > controller, std::shared_ptr< robot::Robot > robot)

  Initialize the operator control.
- void runOpcontrol (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::IDelayer > m_delayer {}

  The rtos delayer 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.

5.41.1 Detailed Description

Manages the execution of the operator control.

Author

Nathan Sandvig

Definition at line 29 of file OpcontrolManager.hpp.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 OPControlManager()

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

Construct a new OPControlManager object.

Parameters

clock	The rtos clock
delayer	The rtos delayer

Definition at line 5 of file OPControlManager.cpp.

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

5.41.3 Member Function Documentation

5.41.3.1 setProfile()

```
void wisco::OPControlManager::setProfile (
            std::unique_ptr< IProfile > & profile )
```

Set the operator profile.

Parameters

profile	The operator profile

Definition at line 11 of file OPControlManager.cpp.

```
00012 {
00013     m_profile = std::move(profile);
00014 }
```

5.41.3.2 initializeOpcontrol()

```
void wisco::OPControlManager::initializeOpcontrol (
            std::shared_ptr< user::IController > controller,
            std::shared_ptr< robot::Robot > robot )
```

Initialize the operator control.

Parameters

controller	The controller for the robot
robot	The robot being controlled

Definition at line 16 of file OPControlManager.cpp.

```
00017 {
00018
00019 }
```

5.41.3.3 runOpcontrol()

```
void wisco::OPControlManager::runOpcontrol (
            std::shared_ptr< user::IController > controller,
            std::shared_ptr< robot::Robot > robot )
```

Run the operator control.

Parameters

controller	The controller for the robot
robot	The robot being controlled

Definition at line 21 of file OPControlManager.cpp.

```
00022 {
00023     user::DifferentialDriveOperator drive_operator{controller, robot};
00024     user::ElevatorOperator elevator_operator{controller, robot};
00025     user::IntakeOperator intake_operator{controller, robot};
00026     uint32_t current_time{};
00027     while (true)
00028     {
00029         current_time = m_clock->getTime();
00030
00031         //TODO user control code
00032
      drive_operator.setDriveVoltage(static_cast<user::EChassisControlMode>(m_profile->getControlMode(user::EControlType::DRIV
00033         elevator_operator.setElevatorPosition(m_profile);
00034         intake_operator.setIntakeVoltage(m_profile);
00035
00036         m_delayer->delayUntil(current_time + CONTROL_DELAY);
00037     }
00038 }
```

5.41.4 Member Data Documentation

5.41.4.1 CONTROL_DELAY

```
constexpr uint32_t wisco::OPControlManager::CONTROL_DELAY {10}  [static], [constexpr], [private]
```

The loop delay for control inputs.

Definition at line 36 of file OpcontrolManager.hpp.

```
00036 {10};
```

5.41.4.2 m_clock

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

The rtos clock for the control loop.

Definition at line 42 of file OpcontrolManager.hpp.

```
00042 {};
```

5.41.4.3 m_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::OPControlManager::m_delayer {}  [private]
```

The rtos delayer for the control loop.

Definition at line 48 of file OpcontrolManager.hpp.

```
00048 {};
```

5.41.4.4 m_profile

```
std::unique_ptr<IProfile> wisco::OPControlManager::m_profile {}  [private]
```

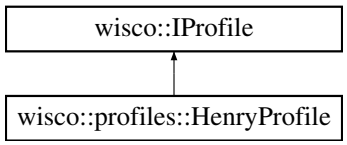
The driver profile.

Definition at line 54 of file OpcontrolManager.hpp.
```
00054 {};
```

5.42 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.
- 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

- const 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.

5.42.1 Detailed Description

Driver profile for Henry.

Author

Nathan Sandvig

Definition at line 29 of file HenryProfile.hpp.

5.42.2 Member Function Documentation

5.42.2.1 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 }
```

5.42.2.2 getControlMode()

```
int wisco::profiles::HenryProfile::getControlMode (
            user::EControlType control_type ) const  [override], [virtual]
```

Get the control mode for a specific control type.

Parameters

control_type	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 }
```

5.42.2.3 getAnalogControlMapping()

```
user::EControllerAnalog wisco::profiles::HenryProfile::getAnalogControlMapping (
            user::EControl control ) const  [override], [virtual]
```

Get the mapping of a control to analog inputs.

Parameters

control	The control

Returns

user::EControllerAnalog The mapping of this control to a analog input

Implements wisco::IProfile.

Definition at line 20 of file HenryProfile.cpp.

```
00021 {
00022     user::EControllerAnalog analog{user::EControllerAnalog::NONE};
00023     if (ANALOG_CONTROL_MAP.contains(control))
00024         analog = ANALOG_CONTROL_MAP.at(control);
00025     return analog;
00026 }
```

5.42.2.4 getDigitalControlMapping()

```
user::EControllerDigital wisco::profiles::HenryProfile::getDigitalControlMapping (
            user::EControl control ) const  [override], [virtual]
```

Get the mapping of a control to digital inputs.

Parameters

control	The control

Returns

user::EControllerDigital The mapping of this control to a digital input

Implements wisco::IProfile.

Definition at line 28 of file HenryProfile.cpp.

```
00029 {
00030     user::EControllerDigital digital{user::EControllerDigital::NONE};
00031     if (DIGITAL_CONTROL_MAP.contains(control))
00032         digital = DIGITAL_CONTROL_MAP.at(control);
00033     return digital;
00034 }
```

5.42.3 Member Data Documentation

5.42.3.1 PROFILE_NAME

```
constexpr char wisco::profiles::HenryProfile::PROFILE_NAME[] {"HENRY"}  [static], [constexpr],
[private]
```

The name of the profile.

Definition at line 36 of file HenryProfile.hpp.
```
00036 {"HENRY"};
```

5.42.3.2 CONTROL_MODE_MAP

```
const std::map<user::EControlType, int> wisco::profiles::HenryProfile::CONTROL_MODE_MAP  [private]
```

Initial value:
```
{
        {user::EControlType::DRIVE, static_cast<int>(user::EChassisControlMode::SPLIT_ARCADE_LEFT)},
        {user::EControlType::INTAKE, static_cast<int>(user::EIntakeControlMode::SPLIT_HOLD)}
    }
```

The control modes for the profile.

Definition at line 42 of file HenryProfile.hpp.
```
00043     {
00044         {user::EControlType::DRIVE, static_cast<int>(user::EChassisControlMode::SPLIT_ARCADE_LEFT)},
00045         {user::EControlType::INTAKE, static_cast<int>(user::EIntakeControlMode::SPLIT_HOLD)}
00046     };
```

5.42.3.3 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 52 of file HenryProfile.hpp.
```
00052 {};
```

5.42.3.4 DIGITAL_CONTROL_MAP

```
const std::map<user::EControl, user::EControllerDigital> wisco::profiles::HenryProfile::↩
DIGITAL_CONTROL_MAP  [private]
```

Initial value:
```
{
        {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
        {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM}
    }
```

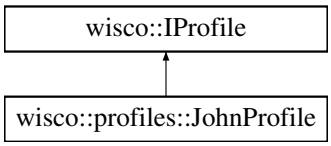
The mapping of the controls to the digital inputs.

Definition at line 58 of file HenryProfile.hpp.
```
00059     {
00060         {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
00061         {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM}
00062     };
```

5.43 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.
- 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

- const 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.

5.43.1 Detailed Description

Driver profile for John.

Author

Nathan Sandvig

Definition at line 29 of file JohnProfile.hpp.

5.43.2 Member Function Documentation

5.43.2.1 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 }
```

5.43.2.2 getControlMode()

```
int wisco::profiles::JohnProfile::getControlMode (
            user::EControlType control_type ) const  [override], [virtual]
```

Get the control mode for a specific control type.

Parameters

control_type	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 }
```

5.43.2.3 getAnalogControlMapping()

```
user::EControllerAnalog wisco::profiles::JohnProfile::getAnalogControlMapping (
            user::EControl control ) const  [override], [virtual]
```

Get the mapping of a control to analog inputs.

Parameters

control	The control

Returns

user::EControllerAnalog The mapping of this control to a analog input

Implements wisco::IProfile.

Definition at line 21 of file JohnProfile.cpp.

```
00022 {
00023     user::EControllerAnalog analog{user::EControllerAnalog::NONE};
00024     if (ANALOG_CONTROL_MAP.contains(control))
00025         analog = ANALOG_CONTROL_MAP.at(control);
00026     return analog;
00027 }
```

5.43.2.4 getDigitalControlMapping()

```
user::EControllerDigital wisco::profiles::JohnProfile::getDigitalControlMapping (
            user::EControl control ) const  [override], [virtual]
```

Get the mapping of a control to digital inputs.

Parameters

control	The control

Returns

user::EControllerDigital The mapping of this control to a digital input

Implements wisco::IProfile.

Definition at line 29 of file JohnProfile.cpp.

```
00030 {
00031     user::EControllerDigital digital{user::EControllerDigital::NONE};
00032     if (DIGITAL_CONTROL_MAP.contains(control))
00033         digital = DIGITAL_CONTROL_MAP.at(control);
00034     return digital;
00035 }
```

5.43.3 Member Data Documentation

5.43.3.1 PROFILE_NAME

```
constexpr char wisco::profiles::JohnProfile::PROFILE_NAME[] {"JOHN"}  [static], [constexpr],
[private]
```

The name of the profile.

Definition at line 36 of file JohnProfile.hpp.

```
00036 {"JOHN"};
```

5.43.3.2 CONTROL_MODE_MAP

```
const std::map<user::EControlType, int> wisco::profiles::JohnProfile::CONTROL_MODE_MAP  [private]
```

Initial value:

```
{
        {user::EControlType::DRIVE, static_cast<int>(user::EChassisControlMode::TANK)},
        {user::EControlType::INTAKE, static_cast<int>(user::EIntakeControlMode::SPLIT_HOLD)}
    }
```

The control modes for the profile.

Definition at line 42 of file JohnProfile.hpp.

```
00043     {
00044         {user::EControlType::DRIVE, static_cast<int>(user::EChassisControlMode::TANK)},
00045         {user::EControlType::INTAKE, static_cast<int>(user::EIntakeControlMode::SPLIT_HOLD)}
00046     };
```

5.43.3.3 ANALOG_CONTROL_MAP

```
const std::map<user::EControl, user::EControllerAnalog> wisco::profiles::JohnProfile::ANALOG↩
_CONTROL_MAP {}  [private]
```

The mapping of the controls to the analog inputs.

Definition at line 52 of file JohnProfile.hpp.

```
00052 {};
```

5.43.3.4 DIGITAL_CONTROL_MAP

```
const std::map<user::EControl, user::EControllerDigital> wisco::profiles::JohnProfile::↩
DIGITAL_CONTROL_MAP  [private]
```

Initial value:

```
{
        {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
        {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM}
    }
```

The mapping of the controls to the digital inputs.

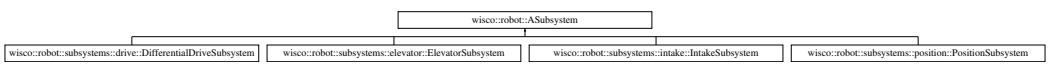
Definition at line 58 of file JohnProfile.hpp.

```
00059     {
00060         {user::EControl::INTAKE_IN, user::EControllerDigital::TRIGGER_LEFT_TOP},
00061         {user::EControl::INTAKE_OUT, user::EControllerDigital::TRIGGER_LEFT_BOTTOM}
00062     };
```

5.44 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.

5.44.1 Detailed Description

An abstract class for robot subsystems.

Author

Nathan Sandvig

Definition at line 28 of file ASubsystem.hpp.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 ASubsystem() [1/3]

```
wisco::robot::ASubsystem::ASubsystem (
            const ASubsystem & other )  [default]
```

Construct a new ASubsystem object.

Parameters

other	The ASubsystem object being copied

5.44.2.2 ASubsystem() [2/3]

```
wisco::robot::ASubsystem::ASubsystem (
            ASubsystem && other )  [default]
```

Construct a new ASubsystem object.

Parameters

other	The ASubsystem object being moved

5.44.2.3 ASubsystem() [3/3]

```
wisco::robot::ASubsystem::ASubsystem (
            std::string name )  [inline]
```

Construct a new ASubsystem object.

Parameters

name	The name of the subsystem

Definition at line 63 of file ASubsystem.hpp.

```
00063 : m_name{name} {}
```

5.44.3 Member Function Documentation

5.44.3.1 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     }
```

5.44.3.2 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::intake::IntakeSubsystem, and wisco::robot::subsystems::position::PositionSubsystem.

5.44.3.3 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::intake::IntakeSubsystem, and wisco::robot::subsystems::position::PositionSubsystem.

5.44.3.4 command()

```
virtual void wisco::robot::ASubsystem::command (
            std::string command_name,
            va_list & args )  [pure virtual]
```

Runs a command for the subsystem.

Parameters

command_name	The name of the command to run
args	The parameters for the command

Implemented in wisco::robot::subsystems::drive::DifferentialDriveSubsystem, wisco::robot::subsystems::elevator::ElevatorSubsystem, wisco::robot::subsystems::intake::IntakeSubsystem, and wisco::robot::subsystems::position::PositionSubsystem.

5.44.3.5 state()

```
virtual void * wisco::robot::ASubsystem::state (
            std::string state_name )  [pure virtual]
```

Gets a state of the subsystem.

Parameters

state_name	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::intake::IntakeSubsystem, and wisco::robot::subsystems::position::PositionSubsystem.

5.44.3.6 operator=() [1/2]

```
ASubsystem & wisco::robot::ASubsystem::operator= (
            const ASubsystem & rhs )  [default]
```

Copy assignment operator for ASubsystem.

Parameters

rhs	The ASubsystem on the right hand side of the operator

Returns

ASubsystem& This ASubsystem with the copied values

5.44.3.7 operator=() [2/2]

```
ASubsystem & wisco::robot::ASubsystem::operator= (
            ASubsystem && rhs )  [default]
```

Move assignment operator for ASubsystem.

Parameters

rhs	The ASubsystem value on the right hand side of the operator

Returns

ASubsystem& This ASubsystem with the moved values

5.44.4 Member Data Documentation

5.44.4.1 m_name

```
std::string wisco::robot::ASubsystem::m_name {}  [private]
```

The name of the subsystem.

Definition at line 35 of file ASubsystem.hpp.

```
00035 {};
```

5.45 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.

5.45.1 Detailed Description

A container class for subsystems.

Author

Nathan Sandvig

Definition at line 30 of file Robot.hpp.

5.45.2 Member Function Documentation

5.45.2.1 addSubsystem()

```
void wisco::robot::Robot::addSubsystem (
            std::unique_ptr< ASubsystem > & subsystem )
```

Adds a subsystem to the robot.

Parameters

subsystem	The subsystem being added to the robot

Definition at line 7 of file Robot.cpp.

```
00008 {
00009     subsystems.push_back(std::move(subsystem));
00010 }
```

5.45.2.2 removeSubsystem()

```
bool wisco::robot::Robot::removeSubsystem (
```

```
            std::string subsystem_name )
```

Removes a subsystem from the robot.

Parameters

subsystem_name	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 }
```

5.45.2.3 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 }
```

5.45.2.4 sendCommand()

```
void wisco::robot::Robot::sendCommand (
            std::string subsystem_name,
            std::string command_name,
              ...  )
```

Sends a command to a subsystem.

Parameters

subsystem_name	The name of the subsystem
command_name	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 }
```

5.45.2.5 getState()

```
void * wisco::robot::Robot::getState (
            std::string subsystem_name,
            std::string state_name )
```

Gets a state of a subsystem.

Parameters

subsystem_name	The name of the subsystem
state_name	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 }
```

5.45.3 Member Data Documentation

5.45.3.1 subsystems

```
std::vector<std::unique_ptr<ASubsystem> > wisco::robot::Robot::subsystems {}  [private]
```

The subsystems contained in the robot.

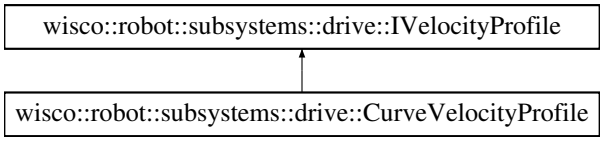
Definition at line 37 of file Robot.hpp.

```
00037 {};
```

5.46 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.

5.46.1 Detailed Description

An s-curve velocity profile for the drive.

Author

Nathan Sandvig

Definition at line 48 of file CurveVelocityProfile.hpp.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 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

clock	The system clock
jerk_rate	The jerk rate for the velocity profile
max_acceleration	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 }
```

5.46.3 Member Function Documentation

5.46.3.1 getAcceleration()

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::getAcceleration (
            double current_velocity,
            double target_velocity )  [override], [virtual]
```

Get the target acceleration from the profile.

Parameters

current_velocity	The current velocity
target_velocity	The target velocity

Returns

double The acceleration in m/s^2

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
00029     // 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 }
```

5.46.3.2 setAcceleration()

```
void wisco::robot::subsystems::drive::CurveVelocityProfile::setAcceleration (
            double acceleration )  [override], [virtual]
```

Set the current acceleration.

Parameters

acceleration	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 }
```

5.46.4 Member Data Documentation

5.46.4.1 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 {};
```

5.46.4.2 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 {};
```

5.46.4.3 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 {};
```

5.46.4.4 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 {};
```

5.46.4.5 last_time

```
double wisco::robot::subsystems::drive::CurveVelocityProfile::last_time  [private]
```

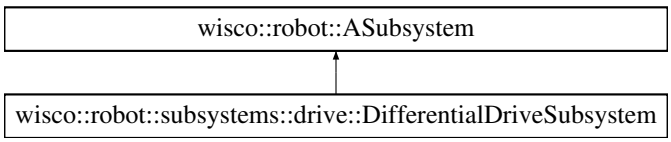
The last timestamp during execution.

Definition at line 79 of file CurveVelocityProfile.hpp.

5.47 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.

5.47.1 Detailed Description

The subsystem adapter for differential drives.

Author

Nathan Sandvig

Definition at line 47 of file DifferentialDriveSubsystem.hpp.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 DifferentialDriveSubsystem()

```
wisco::robot::subsystems::drive::DifferentialDriveSubsystem::DifferentialDriveSubsystem (
            std::unique_ptr< IDifferentialDrive > & differential_drive )
```

Construct a new Differential Drive Subsystem object.

Parameters

differential_drive	The differential drive being adapted

Definition at line 12 of file DifferentialDriveSubsystem.cpp.

```
00013     : ASubsystem{SUBSYSTEM_NAME}, m_differential_drive{std::move(differential_drive)}
00014 {
00015
00016 }
```

5.47.3 Member Function Documentation

5.47.3.1 initialize()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::initialize ( )  [override],
[virtual]
```

Initializes the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 18 of file DifferentialDriveSubsystem.cpp.

```
00019 {
00020     m_differential_drive->initialize();
00021 }
```

5.47.3.2 run()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::run ( )  [override], [virtual]
```

Runs the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 23 of file DifferentialDriveSubsystem.cpp.

```
00024 {
00025     m_differential_drive->run();
00026 }
```

5.47.3.3 command()

```
void wisco::robot::subsystems::drive::DifferentialDriveSubsystem::command (
            std::string command_name,
            va_list & args )  [override], [virtual]
```

Runs a command for the subsystem.

Parameters

command_name	The name of the command to run
args	The parameters for the command

Implements wisco::robot::ASubsystem.

Definition at line 28 of file DifferentialDriveSubsystem.cpp.

```
00029 {
00030     if (command_name == SET_VELOCITY_COMMAND_NAME)
00031     {
00032         double left_velocity{va_arg(args, double)};
00033         double right_velocity{va_arg(args, double)};
00034         Velocity velocity{left_velocity, right_velocity};
00035         m_differential_drive->setVelocity(velocity);
00036     }
00037     else if (command_name == SET_VOLTAGE_COMMAND_NAME)
00038     {
00039         double left_voltage{va_arg(args, double)};
00040         double right_voltage{va_arg(args, double)};
00041         m_differential_drive->setVoltage(left_voltage, right_voltage);
00042     }
00043 }
```

5.47.3.4 state()

```
void * wisco::robot::subsystems::drive::DifferentialDriveSubsystem::state (
            std::string state_name )  [override], [virtual]
```

Gets a state of the subsystem.

Parameters

state_name	The name of the state to get

Returns

void* The current value of that state

Implements wisco::robot::ASubsystem.

Definition at line 45 of file DifferentialDriveSubsystem.cpp.

```
00046 {
00047     void* result{nullptr};
00048
00049     if (state_name == GET_VELOCITY_STATE_NAME)
00050     {
00051         Velocity* velocity{new Velocity{m_differential_drive->getVelocity()}};
00052         result = velocity;
00053     }
00054
00055     return result;
00056 }
```

5.47.4 Member Data Documentation

5.47.4.1 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"};
```

5.47.4.2 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"};
```

5.47.4.3 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"};
```

5.47.4.4 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"};
```

5.47.4.5 m_differential_drive

```
std::unique_ptr<IDifferentialDrive> wisco::robot::subsystems::drive::DifferentialDriveSubsystem↩
::m_differential_drive {}  [private]
```

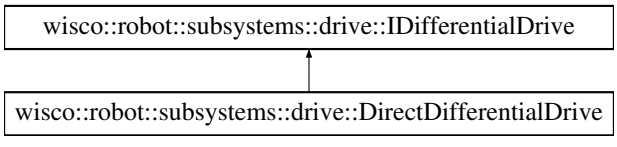
The differential drive being adapted.

Definition at line 78 of file DifferentialDriveSubsystem.hpp.
```
00078 {};
```

5.48 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.
- 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.

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.

5.48.1 Detailed Description

A direct drive controller with independent left and right wheelsets.

Author

Nathan Sandvig

Definition at line 45 of file DirectDifferentialDrive.hpp.

5.48.2 Member Function Documentation

5.48.2.1 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 }
```

5.48.2.2 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 }
```

5.48.2.3 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 }
```

5.48.2.4 setVelocity()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVelocity (
            Velocity velocity )  [override], [virtual]
```

Set the velocity values of the drive.

Parameters

velocity	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 }
```

5.48.2.5 setVoltage()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVoltage (
            double left_voltage,
            double right_voltage )  [override], [virtual]
```

Set the voltages of the drive directly.

Parameters

left_voltage	The voltage for the left side of the drive
right_voltage	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 }
```

5.48.2.6 setLeftMotors()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setLeftMotors (
            hal::MotorGroup & left_motors )
```

Set the left drive motors.

Parameters

left_motors	The motors on the left side of the drive

Definition at line 44 of file DirectDifferentialDrive.cpp.

```
00045 {
00046     m_left_motors = left_motors;
00047 }
```

5.48.2.7 setRightMotors()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setRightMotors (
            hal::MotorGroup & right_motors )
```

Set the right drive motors.

Parameters

right_motors	The motors on the right side of the drive

Definition at line 49 of file DirectDifferentialDrive.cpp.

```
00050 {
00051     m_right_motors = right_motors;
00052 }
```

5.48.2.8 setVelocityToVoltage()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setVelocityToVoltage (
            double velocity_to_voltage )
```

Set the velocity to voltage conversion constant.

Parameters

velocity_to_voltage	The velocity to voltage conversion constant

Definition at line 54 of file DirectDifferentialDrive.cpp.

```
00055 {
00056     m_velocity_to_voltage = velocity_to_voltage;
00057 }
```

5.48.2.9 setGearRatio()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setGearRatio (
            double gear_ratio )
```

Set the gear ratio.

Parameters

gear_ratio	The gear ratio of the drive

Definition at line 59 of file DirectDifferentialDrive.cpp.

```
00060 {
00061     m_gear_ratio = gear_ratio;
00062 }
```

5.48.2.10 setWheelRadius()

```
void wisco::robot::subsystems::drive::DirectDifferentialDrive::setWheelRadius (
            double wheel_radius )
```

Set the wheel radius.

Parameters

wheel_radius	The wheel radius of the drive

Definition at line 64 of file DirectDifferentialDrive.cpp.

```
00065 {
00066     m_wheel_radius = wheel_radius;
00067 }
```

5.48.3 Member Data Documentation

5.48.3.1 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 {};
```

5.48.3.2 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 {};
```

5.48.3.3 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};
```

5.48.3.4 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 {};
```

5.48.3.5 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 {};
```

5.49 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.
- 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.

5.49.1 Detailed Description

Builder class for the direct differential drive class.

Author

Nathan Sandvig

Definition at line 45 of file DirectDifferentialDriveBuilder.hpp.

5.49.2 Member Function Documentation

5.49.2.1 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

left_motor	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 }
```

5.49.2.2 withRightMotor()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDrive↩
Builder::withRightMotor (
            std::unique_ptr< io::IMotor > & right_motor )
```

Add a right drive motor to the build.

Parameters

right_motor	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 }
```

5.49.2.3 withVelocityToVoltage()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDrive↩
Builder::withVelocityToVoltage (
            double velocity_to_voltage )
```

Add the velocity to voltage conversion constant to the build.

Parameters

velocity_to_voltage	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 }
```

5.49.2.4 withGearRatio()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDrive↩
Builder::withGearRatio (
            double gear_ratio )
```

Add the gear ratio to the build.

Parameters

gear_ratio	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 }
```

5.49.2.5 withWheelRadius()

```
DirectDifferentialDriveBuilder * wisco::robot::subsystems::drive::DirectDifferentialDrive↩
Builder::withWheelRadius (
            double wheel_radius )
```

Add the wheel radius to the build.

Parameters

wheel_radius	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 }
```

5.49.2.6 build()

```
std::unique_ptr< IDifferentialDrive > wisco::robot::subsystems::drive::DirectDifferential↩
DriveBuilder::build ( )
```

Builds the differential drive system.

Returns

std::unique_ptr<IDifferentialDrive> The differential drive system as a differential drive interface

Definition at line 41 of file DirectDifferentialDriveBuilder.cpp.

```
00042 {
00043     std::unique_ptr<DirectDifferentialDrive>
      differential_drive{std::make_unique<DirectDifferentialDrive>()};
00044     differential_drive->setLeftMotors(m_left_motors);
00045     differential_drive->setRightMotors(m_right_motors);
00046     differential_drive->setVelocityToVoltage(m_velocity_to_voltage);
00047     return differential_drive;
00048 }
```

5.49.3 Member Data Documentation

5.49.3.1 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 {};
```

5.49.3.2 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 {};
```

5.49.3.3 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};
```

5.49.3.4 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 {};
```

5.49.3.5 m_wheel_radius

```
double wisco::robot::subsystems::drive::DirectDifferentialDriveBuilder::m_wheel_radius {}
[private]
```

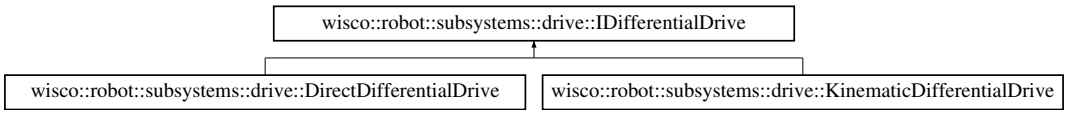
The radius of the drive wheels.

Definition at line 76 of file DirectDifferentialDriveBuilder.hpp.
```
00076 {};
```

5.50 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.

5.50.1 Detailed Description

Interface for differential drivetrains.

Author

Nathan Sandvig

Definition at line 43 of file IDifferentialDrive.hpp.

5.50.2 Member Function Documentation

5.50.2.1 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::KinematicDifferentialDri

5.50.2.2 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::KinematicDifferentialDri

5.50.2.3 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::KinematicDifferentialDri

5.50.2.4 setVelocity()

```
virtual void wisco::robot::subsystems::drive::IDifferentialDrive::setVelocity (
            Velocity velocity )  [pure virtual]
```

Set the velocity values of the drive.

Parameters

velocity	The velocity values for the drive

Implemented in wisco::robot::subsystems::drive::DirectDifferentialDrive, and wisco::robot::subsystems::drive::KinematicDifferentialDri

5.50.2.5 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

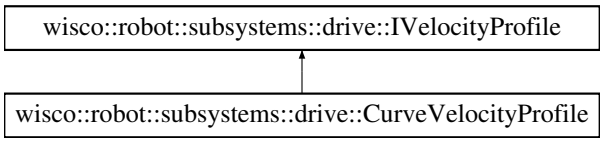
left_voltage	The voltage for the left side of the drive
right_voltage	The voltage for the right side of the drive

Implemented in wisco::robot::subsystems::drive::DirectDifferentialDrive, and wisco::robot::subsystems::drive::KinematicDifferentialDri

5.51 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.

5.51.1 Detailed Description

Interface for drive velocity profiles.

Author

Nathan Sandvig

Definition at line 41 of file IVelocityProfile.hpp.

5.51.2 Member Function Documentation

5.51.2.1 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

current_velocity	The current velocity
target_velocity	The target velocity

Returns

double The acceleration in m/s^2

Implemented in wisco::robot::subsystems::drive::CurveVelocityProfile.

5.51.2.2 setAcceleration()

```
virtual void wisco::robot::subsystems::drive::IVelocityProfile::setAcceleration (
            double acceleration )  [pure virtual]
```

Set the current acceleration.

Parameters

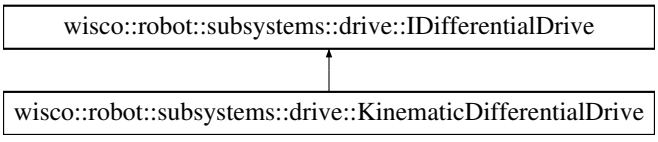
acceleration	The current acceleration

Implemented in wisco::robot::subsystems::drive::CurveVelocityProfile.

5.52 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.
- 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 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 {}

  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.
- 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.

5.52.1 Detailed Description

A kinematic drive controller with independent left and right wheelsets.

Author

Nathan Sandvig

Definition at line 52 of file KinematicDifferentialDrive.hpp.

5.52.2 Member Function Documentation

5.52.2.1 taskLoop()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::taskLoop (
            void * params )  [static], [private]
```

The task loop function for background updates.

Parameters

params	

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 }
```

5.52.2.2 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 }
```

5.52.2.3 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,
      m_velocity.left_velocity)};
00036     double right_acceleration{m_right_velocity_profile->getAcceleration(velocity.right_velocity,
      m_velocity.right_velocity)};
00037
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 }
```

5.52.2.4 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) *
      m_left_motors.getTorqueConstant())
00065         / (m_left_motors.getAngularVelocityConstant() * m_left_motors.getResistance() *
      std::pow(m_wheel_radius, 2));
00066
00067     c2 = (m_left_motors.getGearRatio() * m_gear_ratio * m_left_motors.getTorqueConstant())
00068         / (m_left_motors.getResistance() * m_wheel_radius);
00069
00070     c3 = (-1 * std::pow(m_right_motors.getGearRatio() * m_gear_ratio, 2) *
      m_right_motors.getTorqueConstant())
00071         / (m_right_motors.getAngularVelocityConstant() * m_right_motors.getResistance() *
      std::pow(m_wheel_radius, 2));
00072
00073     c4 = (m_right_motors.getGearRatio() * m_gear_ratio * m_right_motors.getTorqueConstant())
00074         / (m_right_motors.getResistance() * m_wheel_radius);
00075
00076     c5 = (1 / m_mass) + (std::pow(m_radius, 2) / m_moment_of_inertia);
00077
00078     c6 = (1 / m_mass) - (std::pow(m_radius, 2) / m_moment_of_inertia);
00079
00080     c7 = std::pow(c5, 2) - std::pow(c6, 2);
00081 }
```

5.52.2.5 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 }
```

5.52.2.6 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 }
```

5.52.2.7 setVelocity()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setVelocity (
            Velocity velocity )  [override], [virtual]
```

Set the velocity values of the drive.

Parameters

velocity	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 }
```

5.52.2.8 setVoltage()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setVoltage (
            double left_voltage,
            double right_voltage )  [override], [virtual]
```

Set the voltages of the drive directly.

Parameters

left_voltage	The voltage for the left side of the drive
right_voltage	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 }
```

5.52.2.9 setDelayer()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setDelayer (
            std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

Parameters

delayer	The rtos delayer

Definition at line 120 of file KinematicDifferentialDrive.cpp.

```
00121 {
00122     m_delayer = std::move(delayer);
00123 }
```

5.52.2.10 setMutex()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Set the os mutex.

Parameters

mutex	The os mutex

Definition at line 125 of file KinematicDifferentialDrive.cpp.

```
00126 {
00127     m_mutex = std::move(mutex);
00128 }
```

5.52.2.11 setTask()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setTask (
            std::unique_ptr< rtos::ITask > & task )
```

Set the rtos task handler.

Parameters

task	The rtos task handler

Definition at line 130 of file KinematicDifferentialDrive.cpp.

```
00131 {
00132     m_task = std::move(task);
00133 }
```

5.52.2.12 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

left_velocity_profile	The velocity profile for the left side
right_velocity_profile	The velocity profile for the right side

Definition at line 135 of file KinematicDifferentialDrive.cpp.

```
00136 {
00137     m_left_velocity_profile = std::move(left_velocity_profile);
00138     m_right_velocity_profile = std::move(right_velocity_profile);
00139 }
```

5.52.2.13 setLeftMotors()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setLeftMotors (
            hal::MotorGroup & left_motors )
```

Set the left drive motors.

Parameters

left_motors	The motors on the left side of the drive

Definition at line 141 of file KinematicDifferentialDrive.cpp.

```
00142 {
00143     m_left_motors = left_motors;
00144 }
```

5.52.2.14 setRightMotors()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setRightMotors (
            hal::MotorGroup & right_motors )
```

Set the right drive motors.

Parameters

right_motors	The motors on the right side of the drive

Definition at line 146 of file KinematicDifferentialDrive.cpp.

```
00147 {
00148     m_right_motors = right_motors;
00149 }
```

5.52.2.15 setMass()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMass (
            double mass )
```

Set the mass.

Parameters

mass	The mass of the drive

Definition at line 151 of file KinematicDifferentialDrive.cpp.

```
00152 {
00153     m_mass = mass;
00154 }
```

5.52.2.16 setRadius()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setRadius (
            double radius )
```

Set the radius.

Parameters

radius	The radius of the drive

Definition at line 156 of file KinematicDifferentialDrive.cpp.

```
00157 {
00158     m_radius = radius;
00159 }
```

5.52.2.17 setMomentOfInertia()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setMomentOfInertia (
            double moment_of_inertia )
```

Set the moment of inertia.

Parameters

moment_of_inertia	The moment of inertia of the drive

Definition at line 161 of file KinematicDifferentialDrive.cpp.

```
00162 {
00163     m_moment_of_inertia = moment_of_inertia;
00164 }
```

5.52.2.18 setGearRatio()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setGearRatio (
            double gear_ratio )
```

Set the gear ratio.

Parameters

gear_ratio	The gear ratio of the drive

Definition at line 166 of file KinematicDifferentialDrive.cpp.

```
00167 {
00168     m_gear_ratio = gear_ratio;
00169 }
```

5.52.2.19 setWheelRadius()

```
void wisco::robot::subsystems::drive::KinematicDifferentialDrive::setWheelRadius (
            double wheel_radius )
```

Set the wheel radius.

Parameters

wheel_radius	The wheel radius of the drive

Definition at line 171 of file KinematicDifferentialDrive.cpp.

```
00172 {
00173     m_wheel_radius = wheel_radius;
00174 }
```

5.52.3 Member Data Documentation

5.52.3.1 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};
```

5.52.3.2 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 {};
```

5.52.3.3 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 {};
```

5.52.3.4 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 {};
```

5.52.3.5 m_left_velocity_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferential↩
Drive::m_left_velocity_profile {}  [private]
```

The left velocity profile.

Definition at line 90 of file KinematicDifferentialDrive.hpp.
```
00090 {};
```

5.52.3.6 m_right_velocity_profile

```
std::unique_ptr<IVelocityProfile> wisco::robot::subsystems::drive::KinematicDifferential↩
Drive::m_right_velocity_profile {}  [private]
```

The right velocity profile.

Definition at line 96 of file KinematicDifferentialDrive.hpp.
```
00096 {};
```

5.52.3.7 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 {};
```

5.52.3.8 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 {};
```

5.52.3.9 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 {};
```

5.52.3.10 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 {};
```

5.52.3.11 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 {};
```

5.52.3.12 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 {};
```

5.52.3.13 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 {};
```

5.52.3.14 c1

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c1 {}  [private]
```

The first kinematic constant.

Definition at line 144 of file KinematicDifferentialDrive.hpp.
```
00144 {};
```

5.52.3.15 c2

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c2 {}  [private]
```

The second kinematic constant.

Definition at line 150 of file KinematicDifferentialDrive.hpp.
```
00150 {};
```

5.52.3.16 c3

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c3 {}  [private]
```

The third kinematic constant.

Definition at line 156 of file KinematicDifferentialDrive.hpp.
```
00156 {};
```

5.52.3.17 c4

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c4 {}  [private]
```

The fourth kinematic constant.

Definition at line 162 of file KinematicDifferentialDrive.hpp.
```
00162 {};
```

5.52.3.18 c5

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c5 {}  [private]
```

The fifth kinematic constant.

Definition at line 168 of file KinematicDifferentialDrive.hpp.
```
00168 {};
```

5.52.3.19 c6

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c6 {}  [private]
```

The sixth kinematic constant.

Definition at line 174 of file KinematicDifferentialDrive.hpp.
```
00174 {};
```

5.52.3.20 c7

```
double wisco::robot::subsystems::drive::KinematicDifferentialDrive::c7 {}  [private]
```

The seventh kinematic constant.

Definition at line 180 of file KinematicDifferentialDrive.hpp.
```
00180 {};
```

5.52.3.21 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 {};
```

5.53 wisco::robot::subsystems::drive::KinematicDifferentialDriveBuilder Class Reference

Builder class for the kinematic differential drive class.

Public Member Functions

- KinematicDifferentialDriveBuilder * withDelayer (std::unique_ptr< rtos::IDelayer > &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::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 {}

  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.

5.53.1 Detailed Description

Builder class for the kinematic differential drive class.

Author

Nathan Sandvig

Definition at line 45 of file KinematicDifferentialDriveBuilder.hpp.

5.53.2 Member Function Documentation

5.53.2.1 withDelayer()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withDelayer (
            std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

Parameters

delayer	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 }
```

5.53.2.2 withMutex()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Add an os mutex to the build.

Parameters

mutex	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 }
```

5.53.2.3 withTask()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withTask (
            std::unique_ptr< rtos::ITask > & task )
```

Add an rtos task handler to the build.

Parameters

task	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 }
```

5.53.2.4 withLeftVelocityProfile()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withLeftVelocityProfile (
            std::unique_ptr< IVelocityProfile > & left_velocity_profile )
```

Adds a left velocity profile to the build.

Parameters

left_velocity_profile	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 }
```

5.53.2.5 withRightVelocityProfile()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withRightVelocityProfile (
            std::unique_ptr< IVelocityProfile > & right_velocity_profile )
```

Adds a right velocity profile to the build.

Parameters

right_velocity_profile	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 }
```

5.53.2.6 withLeftMotor()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withLeftMotor (
            std::unique_ptr< io::IMotor > & left_motor )
```

Add a left drive motor to the build.

Parameters

left_motor	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 }
```

5.53.2.7 withRightMotor()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withRightMotor (
            std::unique_ptr< io::IMotor > & right_motor )
```

Add a right drive motor to the build.

Parameters

right_motor	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 }
```

5.53.2.8 withMass()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withMass (
            double mass )
```

Add the mass to the build.

Parameters

mass	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 }
```

5.53.2.9 withRadius()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withRadius (
            double radius )
```

Add the radius to the build.

Parameters

radius	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 }
```

5.53.2.10 withMomentOfInertia()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withMomentOfInertia (
            double moment_of_inertia )
```

Add the moment of inertia to the build.

Parameters

moment_of_inertia	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 }
```

5.53.2.11 withGearRatio()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withGearRatio (
            double gear_ratio )
```

Add the gear ratio to the build.

Parameters

gear_ratio	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 }
```

5.53.2.12 withWheelRadius()

```
KinematicDifferentialDriveBuilder * wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::withWheelRadius (
            double wheel_radius )
```

Add the wheel radius to the build.

Parameters

wheel_radius	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 }
```

5.53.2.13 build()

```
std::unique_ptr< IDifferentialDrive > wisco::robot::subsystems::drive::KinematicDifferential↩
DriveBuilder::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>
      differential_drive{std::make_unique<KinematicDifferentialDrive>()};
00086     differential_drive->setDelayer(m_delayer);
00087     differential_drive->setMutex(m_mutex);
00088     differential_drive->setTask(m_task);
00089     differential_drive->setVelocityProfiles(m_left_velocity_profile, m_right_velocity_profile);
00090     differential_drive->setLeftMotors(m_left_motors);
00091     differential_drive->setRightMotors(m_right_motors);
00092     differential_drive->setMass(m_mass);
00093     differential_drive->setRadius(m_radius);
00094     differential_drive->setMomentOfInertia(m_moment_of_inertia);
00095     differential_drive->setGearRatio(m_gear_ratio);
00096     differential_drive->setWheelRadius(m_wheel_radius);
00097     return differential_drive;
00098 }
```

5.53.3 Member Data Documentation

5.53.3.1 m_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::drive::KinematicDifferentialDrive↩
Builder::m_delayer {}  [private]
```

The system delayer.

Definition at line 52 of file KinematicDifferentialDriveBuilder.hpp.

```
00052 {};
```

5.53.3.2 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 {};
```

5.53.3.3 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 {};
```

5.53.3.4 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 {};
```

5.53.3.5 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 {};
```

5.53.3.6 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 {};
```

5.53.3.7 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 {};
```

5.53.3.8 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 {};
```

5.53.3.9 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 {};
```

5.53.3.10 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 {};
```

5.53.3.11 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 {};
```

5.53.3.12 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 {};
```

5.54 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.

5.54.1 Detailed Description

Holds the velocity values for the drive.

Author

Nathan Sandvig

Definition at line 41 of file Velocity.hpp.

5.54.2 Member Data Documentation

5.54.2.1 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 {};
```

5.54.2.2 right_velocity

```
double wisco::robot::subsystems::drive::Velocity::right_velocity {}
```

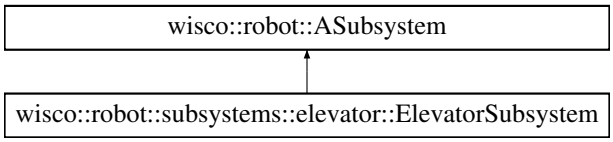
The velocity for the right side of the drive.

Definition at line 53 of file Velocity.hpp.
```
00053 {};
```

5.55 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)

  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.

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.

5.55.1 Detailed Description

The subsystem adapter for elevators.

Author

Nathan Sandvig

Definition at line 46 of file ElevatorSubsystem.hpp.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 ElevatorSubsystem()

```
wisco::robot::subsystems::elevator::ElevatorSubsystem::ElevatorSubsystem (
            std::unique_ptr< IElevator > & elevator )
```

Construct a new ELEVATOR Subsystem object.

Parameters

elevator	The elevator being adapted

Definition at line 11 of file ElevatorSubsystem.cpp.

```
00011                                                                   :
      m_elevator{std::move(elevator)}
00012 {
00013
00014 }
```

5.55.3 Member Function Documentation

5.55.3.1 initialize()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::initialize ( )  [override], [virtual]
```

Initializes the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 16 of file ElevatorSubsystem.cpp.

```
00017 {
00018     if (m_elevator)
00019         m_elevator->initialize();
00020 }
```

5.55.3.2 run()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::run ( )  [override], [virtual]
```

Runs the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 22 of file ElevatorSubsystem.cpp.

```
00023 {
00024     if (m_elevator)
00025         m_elevator->run();
00026 }
```

5.55.3.3 command()

```
void wisco::robot::subsystems::elevator::ElevatorSubsystem::command (
            std::string command_name,
            va_list & args )  [override], [virtual]
```

Runs a command for the subsystem.

Parameters

command_name	The name of the command to run
args	The parameters for the command

Implements wisco::robot::ASubsystem.

Definition at line 28 of file ElevatorSubsystem.cpp.

```
00029 {
00030     if (command_name == SET_POSITION_COMMAND_NAME)
00031     {
00032         double position{va_arg(args, double)};
00033         m_elevator->setPosition(position);
00034     }
00035 }
```

5.55.3.4 state()

```
void * wisco::robot::subsystems::elevator::ElevatorSubsystem::state (
            std::string state_name )  [override], [virtual]
```

Gets a state of the subsystem.

Parameters

state_name	The name of the state to get

Returns

void∗ The current value of that state

Implements wisco::robot::ASubsystem.

Definition at line 37 of file ElevatorSubsystem.cpp.

```
00038 {
00039     void* result{nullptr};
00040
00041     if (state_name == GET_POSITION_STATE_NAME)
00042     {
00043         double* position{new double{m_elevator->getPosition()}};
00044         result = position;
00045     }
00046
00047     return result;
00048 }
```

5.55.4 Member Data Documentation

5.55.4.1 SUBSYSTEM_NAME

```
constexpr char wisco::robot::subsystems::elevator::ElevatorSubsystem::SUBSYSTEM_NAME[] {"ELEVATOR"}
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file ElevatorSubsystem.hpp.

```
00053 {"ELEVATOR"};
```

5.55.4.2 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 59 of file ElevatorSubsystem.hpp.

```
00059 {"SET POSITION"};
```

5.55.4.3 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 65 of file ElevatorSubsystem.hpp.

```
00065 {"GET POSITION"};
```

5.55.4.4 m_elevator

```
std::unique_ptr<IElevator> wisco::robot::subsystems::elevator::ElevatorSubsystem::m_elevator
{}  [private]
```

The elevator being adapted.

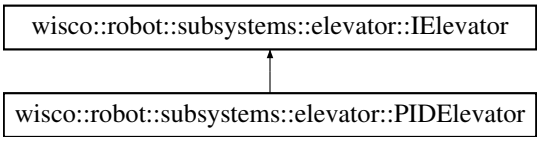
Definition at line 71 of file ElevatorSubsystem.hpp.

```
00071 {};
```

5.56 wisco::robot::subsystems::elevator::IElevator Class Reference

Interface for elevators.

Inheritance diagram for wisco::robot::subsystems::elevator::IElevator:



Public Member Functions

- virtual ~**IElevator** ()=default
  Destroy the IElevator 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.

5.56.1 Detailed Description

Interface for elevators.

Author

Nathan Sandvig

Definition at line 41 of file IElevator.hpp.

5.56.2 Member Function Documentation

5.56.2.1 initialize()

```
virtual void wisco::robot::subsystems::elevator::IElevator::initialize ( )  [pure virtual]
```

Initializes the elevator.

Implemented in wisco::robot::subsystems::elevator::PIDElevator.

5.56.2.2 run()

```
virtual void wisco::robot::subsystems::elevator::IElevator::run ( )  [pure virtual]
```

Runs the elevator.

Implemented in wisco::robot::subsystems::elevator::PIDElevator.

5.56.2.3 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.

5.56.2.4 setPosition()

```
virtual void wisco::robot::subsystems::elevator::IElevator::setPosition (
            double position )  [pure virtual]
```

Set the position of the elevator in inches.

Parameters

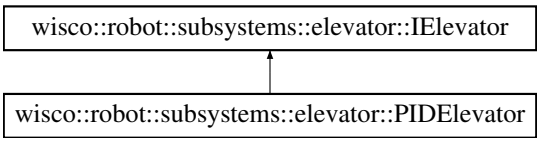
position	The elevator position in inches

Implemented in wisco::robot::subsystems::elevator::PIDElevator.

5.57 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::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 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.

5.57.1 Detailed Description

An elevator controller with PID position control.

Author

Nathan Sandvig

Definition at line 53 of file PIDElevator.hpp.

5.57.2 Member Function Documentation

5.57.2.1 taskLoop()

```
void wisco::robot::subsystems::elevator::PIDElevator::taskLoop (
            void * params )  [static], [private]
```

The task loop function for background updates.

Parameters

params	

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 }
```

5.57.2.2 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 }
```

5.57.2.3 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 voltage{m_pid.getControlValue(getPosition(), m_position)};
00034     m_motors.setVoltage(voltage);
00035
00036     if (m_mutex)
00037         m_mutex->give();
00038 }
```

5.57.2.4 initialize()

```
void wisco::robot::subsystems::elevator::PIDElevator::initialize ( )  [override], [virtual]
```

Initializes the intake.

Implements wisco::robot::subsystems::elevator::IElevator.

Definition at line 40 of file PIDElevator.cpp.

```
00041 {
00042     m_pid.reset();
00043     m_motors.initialize();
00044     if (m_rotation_sensor)
00045         m_rotation_sensor->initialize();
00046 }
```

5.57.2.5 run()

```
void wisco::robot::subsystems::elevator::PIDElevator::run ( )  [override], [virtual]
```

Runs the intake.

Implements wisco::robot::subsystems::elevator::IElevator.

Definition at line 48 of file PIDElevator.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(&PIDElevator::taskLoop, params);
00055     }
00056 }
```

5.57.2.6 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 58 of file PIDElevator.cpp.

```
00059 {
00060     double position{};
00061     if (m_rotation_sensor)
00062     {
00063         position = m_rotation_sensor->getRotation() * m_inches_per_radian;
00064     }
00065     else
00066     {
00067         position = m_motors.getPosition() * m_inches_per_radian;
00068     }
00069     return position;
00070 }
```

5.57.2.7 setPosition()

```
void wisco::robot::subsystems::elevator::PIDElevator::setPosition (
            double position )  [override], [virtual]
```

Set the position of the elevator in inches.

Parameters

position	The position of the elevator

Implements wisco::robot::subsystems::elevator::IElevator.

Definition at line 72 of file PIDElevator.cpp.

```
00073 {
00074     if (m_mutex)
00075         m_mutex->take();
00076
00077     m_position = position;
00078
00079     if (m_mutex)
00080         m_mutex->give();
00081 }
```

5.57.2.8 setClock()

```
void wisco::robot::subsystems::elevator::PIDElevator::setClock (
            const std::unique_ptr< rtos::IClock > & clock )
```

Set the rtos clock.

Parameters

clock	The rtos clock

Definition at line 83 of file PIDElevator.cpp.

```
00084 {
00085     m_clock = clock->clone();
00086 }
```

5.57.2.9 setDelayer()

```
void wisco::robot::subsystems::elevator::PIDElevator::setDelayer (
            const std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

Parameters

delayer	The rtos delayer

Definition at line 88 of file PIDElevator.cpp.

```
00089 {
00090     m_delayer = delayer->clone();
00091 }
```

5.57.2.10 setMutex()

```
void wisco::robot::subsystems::elevator::PIDElevator::setMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Set the thread mutex.

Parameters

mutex	The thread mutex

Definition at line 93 of file PIDElevator.cpp.

```
00094 {
00095     m_mutex = std::move(mutex);
00096 }
```

5.57.2.11 setTask()

```
void wisco::robot::subsystems::elevator::PIDElevator::setTask (
            std::unique_ptr< rtos::ITask > & task )
```

Set the task handler.

Parameters

task	The task handler

Definition at line 98 of file PIDElevator.cpp.

```
00099 {
00100     m_task = std::move(task);
00101 }
```

5.57.2.12 setPID()

```
void wisco::robot::subsystems::elevator::PIDElevator::setPID (
            control::PID pid )
```

Set the PID controller.

Parameters

pid	The PID controller

Definition at line 103 of file PIDElevator.cpp.

```
00104 {
00105     m_pid = pid;
00106 }
```

5.57.2.13 setMotors()

```
void wisco::robot::subsystems::elevator::PIDElevator::setMotors (
            hal::MotorGroup & motors )
```

Set the motors.

Parameters

motors	The motors

Definition at line 108 of file PIDElevator.cpp.

```
00109 {
00110     m_motors = motors;
00111 }
```

5.57.2.14 setRotationSensor()

```
void wisco::robot::subsystems::elevator::PIDElevator::setRotationSensor (
            std::unique_ptr< io::IRotationSensor > & rotation_sensor )
```

Set the rotation sensor.

Parameters

rotation_sensor	The rotation sensor

Definition at line 113 of file PIDElevator.cpp.

```
00114 {
00115     m_rotation_sensor = std::move(rotation_sensor);
00116 }
```

5.57.2.15 setInchesPerRadian()

```
void wisco::robot::subsystems::elevator::PIDElevator::setInchesPerRadian (
            double inches_per_radian )
```

Set the inches per radian of the elevator.

Parameters

inches_per_radian	The inches per radian of the elevator

Definition at line 118 of file PIDElevator.cpp.

```
00119 {
00120     m_inches_per_radian = inches_per_radian;
00121 }
```

5.57.3 Member Data Documentation

5.57.3.1 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};
```

5.57.3.2 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 {};
```

5.57.3.3 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 {};
```

5.57.3.4 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 {};
```

5.57.3.5 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 {};
```

5.57.3.6 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 {};
```

5.57.3.7 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 {};
```

5.57.3.8 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 {};
```

5.57.3.9 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 {};
```

5.57.3.10 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 {};
```

5.58 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::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 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.

5.58.1 Detailed Description

Builder class for a pid-based elevator system.

Author

Nathan Sandvig

Definition at line 45 of file PIDElevatorBuilder.hpp.

5.58.2 Member Function Documentation

5.58.2.1 withClock()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withClock (
            const std::unique_ptr< rtos::IClock > & clock )
```

Add an rtos clock to the build.

Parameters

clock	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 }
```

5.58.2.2 withDelayer()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withDelayer (
            const std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

Parameters

delayer	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 }
```

5.58.2.3 withMutex()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Add a thread mutex to the build.

Parameters

mutex	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 }
```

5.58.2.4 withTask()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withTask (
            std::unique_ptr< rtos::ITask > & task )
```

Add a task handler to the build.

Parameters

task	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 }
```

5.58.2.5 withPID()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withPID (
            control::PID pid )
```

Add a PID controller to the build.

Parameters

pid	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 }
```

5.58.2.6 withMotor()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withMotor (
            std::unique_ptr< io::IMotor > & motor )
```

Add a motor to the build.

Parameters

motor	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 }
```

5.58.2.7 withRotationSensor()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withRotation↩
Sensor (
            std::unique_ptr< io::IRotationSensor > & rotation_sensor )
```

Add a rotation sensor to the build.

Parameters

rotation_sensor	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 }
```

5.58.2.8 withInchesPerRadian()

```
PIDElevatorBuilder * wisco::robot::subsystems::elevator::PIDElevatorBuilder::withInchesPer↩
Radian (
            double inches_per_radian )
```

Add an inches per radian constant to the build.

Parameters

inches_per_radian	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 }
```

5.58.2.9 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 }
```

5.58.3 Member Data Documentation

5.58.3.1 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 {};
```

5.58.3.2 m_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_↩
delayer {}  [private]
```

The rtos delayer.

Definition at line 58 of file PIDElevatorBuilder.hpp.

```
00058 {};
```

5.58.3.3 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 {};
```

5.58.3.4 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 {};
```

5.58.3.5 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 {};
```

5.58.3.6 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 {};
```

5.58.3.7 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 {};
```

5.58.3.8 m_inches_per_radian

```
double wisco::robot::subsystems::elevator::PIDElevatorBuilder::m_inches_per_radian {}  [private]
```

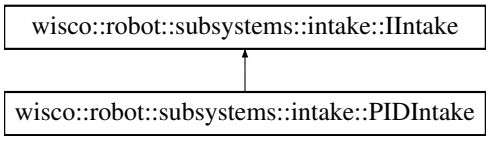
The number of movement inches per radian.

Definition at line 94 of file PIDElevatorBuilder.hpp.
```
00094 {};
```

5.59 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.

5.59.1 Detailed Description

Interface for intakes.

Author

Nathan Sandvig

Definition at line 41 of file IIntake.hpp.

5.59.2 Member Function Documentation

5.59.2.1 initialize()

```
virtual void wisco::robot::subsystems::intake::IIntake::initialize ( )  [pure virtual]
```

Initializes the intake.

Implemented in wisco::robot::subsystems::intake::PIDIntake.

5.59.2.2 run()

```
virtual void wisco::robot::subsystems::intake::IIntake::run ( )  [pure virtual]
```

Runs the intake.

Implemented in wisco::robot::subsystems::intake::PIDIntake.

5.59.2.3 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.

5.59.2.4 setVelocity()

```
virtual void wisco::robot::subsystems::intake::IIntake::setVelocity (
            double velocity )  [pure virtual]
```

Set the velocity of the intake.

Parameters

velocity	The velocity of the intake in in/s

Implemented in wisco::robot::subsystems::intake::PIDIntake.

5.59.2.5 setVoltage()

```
virtual void wisco::robot::subsystems::intake::IIntake::setVoltage (
            double voltage )  [pure virtual]
```

Set the voltage of the intake directly.

Parameters

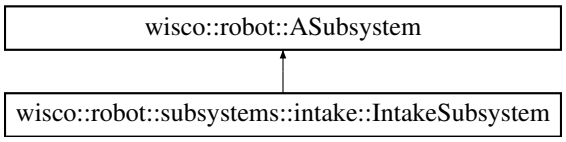
voltage	The voltage for the intake

Implemented in wisco::robot::subsystems::intake::PIDIntake.

5.60 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)
  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.

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.

5.60.1 Detailed Description

The subsystem adapter for intakes.

Author

Nathan Sandvig

Definition at line 46 of file IntakeSubsystem.hpp.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 IntakeSubsystem()

```
wisco::robot::subsystems::intake::IntakeSubsystem::IntakeSubsystem (
            std::unique_ptr< IIntake > & intake )
```

Construct a new Intake Subsystem object.

Parameters

intake	The intake being adapted

Definition at line 11 of file IntakeSubsystem.cpp.

```
00011                                                                   : m_intake{std::move(intake)}
00012 {
00013
00014 }
```

5.60.3 Member Function Documentation

5.60.3.1 initialize()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::initialize ( )  [override], [virtual]
```

Initializes the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 16 of file IntakeSubsystem.cpp.

```
00017 {
00018     if (m_intake)
00019         m_intake->initialize();
00020 }
```

5.60.3.2 run()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::run ( )  [override], [virtual]
```

Runs the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 22 of file IntakeSubsystem.cpp.

```
00023 {
00024     if (m_intake)
00025         m_intake->run();
00026 }
```

5.60.3.3 command()

```
void wisco::robot::subsystems::intake::IntakeSubsystem::command (
            std::string command_name,
            va_list & args )  [override], [virtual]
```

Runs a command for the subsystem.

Parameters

command_name	The name of the command to run
args	The parameters for the command

Implements wisco::robot::ASubsystem.

Definition at line 28 of file IntakeSubsystem.cpp.

```
00029 {
00030     if (command_name == SET_VELOCITY_COMMAND_NAME)
00031     {
00032         double velocity{va_arg(args, double)};
00033         m_intake->setVelocity(velocity);
00034     }
00035     else if (command_name == SET_VOLTAGE_COMMAND_NAME)
00036     {
00037         double voltage{va_arg(args, double)};
00038         m_intake->setVoltage(voltage);
00039     }
00040 }
```

5.60.3.4 state()

```
void * wisco::robot::subsystems::intake::IntakeSubsystem::state (
            std::string state_name )  [override], [virtual]
```

Gets a state of the subsystem.

Parameters

state_name	The name of the state to get

Returns

void* The current value of that state

Implements wisco::robot::ASubsystem.

Definition at line 42 of file IntakeSubsystem.cpp.

```
00043 {
00044     void* result{nullptr};
00045
00046     if (state_name == GET_VELOCITY_STATE_NAME)
00047     {
00048         double* velocity{new double{m_intake->getVelocity()}};
00049         result = velocity;
00050     }
00051
00052     return result;
00053 }
```

5.60.4 Member Data Documentation

5.60.4.1 SUBSYSTEM_NAME

```
constexpr char wisco::robot::subsystems::intake::IntakeSubsystem::SUBSYSTEM_NAME[] {"INTAKE"}
[static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file IntakeSubsystem.hpp.

```
00053 {"INTAKE"};
```

5.60.4.2 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 59 of file IntakeSubsystem.hpp.
```
00059 {"SET VELOCITY"};
```

5.60.4.3 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 65 of file IntakeSubsystem.hpp.
```
00065 {"SET VOLTAGE"};
```

5.60.4.4 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 71 of file IntakeSubsystem.hpp.
```
00071 {"GET VELOCITY"};
```

5.60.4.5 m_intake

```
std::unique_ptr<IIntake> wisco::robot::subsystems::intake::IntakeSubsystem::m_intake {}  [private]
```

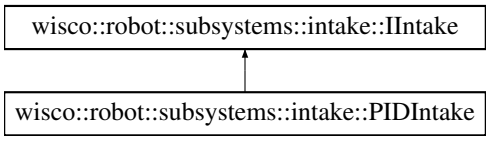
The intake being adapted.

Definition at line 77 of file IntakeSubsystem.hpp.
```
00077 {};
```

5.61 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::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 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.

5.61.1 Detailed Description

An intake controller with PID velocity control.

Author

Nathan Sandvig

Definition at line 52 of file PIDIntake.hpp.

5.61.2 Member Function Documentation

5.61.2.1 taskLoop()

```
void wisco::robot::subsystems::intake::PIDIntake::taskLoop (
            void * params )  [static], [private]
```

The task loop function for background updates.

Parameters

params	

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 }
```

5.61.2.2 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 }
```

5.61.2.3 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 }
```

5.61.2.4 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 }
```

5.61.2.5 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 }
```

5.61.2.6 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 }
```

5.61.2.7 setVelocity()

```
void wisco::robot::subsystems::intake::PIDIntake::setVelocity (
            double velocity )  [override], [virtual]
```

Set the velocity of the intake.

Parameters

velocity	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 }
```

5.61.2.8 setVoltage()

```
void wisco::robot::subsystems::intake::PIDIntake::setVoltage (
            double voltage )  [override], [virtual]
```

Set the voltage of the intake directly.

Parameters

voltage	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 }
```

5.61.2.9 setClock()

```
void wisco::robot::subsystems::intake::PIDIntake::setClock (
            const std::unique_ptr< rtos::IClock > & clock )
```

Set the rtos clock.

Parameters

clock	The rtos clock

Definition at line 88 of file PIDIntake.cpp.

```
00089 {
00090     m_clock = clock->clone();
00091 }
```

5.61.2.10 setDelayer()

```
void wisco::robot::subsystems::intake::PIDIntake::setDelayer (
            const std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

Parameters

delayer	The rtos delayer

Definition at line 93 of file PIDIntake.cpp.

```
00094 {
00095     m_delayer = delayer->clone();
00096 }
```

5.61.2.11 setMutex()

```
void wisco::robot::subsystems::intake::PIDIntake::setMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Set the thread mutex.

Parameters

mutex	The thread mutex

Definition at line 98 of file PIDIntake.cpp.

```
00099 {
00100     m_mutex = std::move(mutex);
00101 }
```

5.61.2.12 setTask()

```
void wisco::robot::subsystems::intake::PIDIntake::setTask (
            std::unique_ptr< rtos::ITask > & task )
```

Set the task handler.

Parameters

task	The task handler

Definition at line 103 of file PIDIntake.cpp.

```
00104 {
00105     m_task = std::move(task);
00106 }
```

5.61.2.13 setPID()

```
void wisco::robot::subsystems::intake::PIDIntake::setPID (
            control::PID pid )
```

Set the PID controller.

Parameters

pid	The PID controller

Definition at line 108 of file PIDIntake.cpp.

```
00109 {
00110     m_pid = pid;
00111 }
```

5.61.2.14 setMotors()

```
void wisco::robot::subsystems::intake::PIDIntake::setMotors (
            hal::MotorGroup & motors )
```

Set the motors.

Parameters

motors	The motors

Definition at line 113 of file PIDIntake.cpp.

```
00114 {
00115     m_motors = motors;
00116 }
```

5.61.2.15 setRollerRadius()

```
void wisco::robot::subsystems::intake::PIDIntake::setRollerRadius (
            double roller_radius )
```

Set the radius of the roller.

Parameters

roller_radius	The radius of the roller

Definition at line 118 of file PIDIntake.cpp.

```
00119 {
00120     m_roller_radius = roller_radius;
00121 }
```

5.61.3 Member Data Documentation

5.61.3.1 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};
```

5.61.3.2 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 {};
```

5.61.3.3 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 {};
```

5.61.3.4 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 {};
```

5.61.3.5 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 {};
```

5.61.3.6 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 {};
```

5.61.3.7 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 {};
```

5.61.3.8 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 {};
```

5.61.3.9 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 {};
```

5.61.3.10 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 {};
```

5.62 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::IDelayer > &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::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.

5.62.1 Detailed Description

A builder class for a PID-based intake subsystem.

Author

Nathan Sandvig

Definition at line 43 of file PIDIntakeBuilder.hpp.

5.62.2 Member Function Documentation

5.62.2.1 withClock()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withClock (
            const std::unique_ptr< rtos::IClock > & clock )
```

Add an rtos clock to the build.

Parameters

clock	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 }
```

5.62.2.2 withDelayer()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withDelayer (
            const std::unique_ptr< rtos::IDelayer > & delayer )
```

Add an rtos delayer to the build.

Parameters

delayer	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 }
```

5.62.2.3 withMutex()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Add a thread mutex to the build.

Parameters

mutex	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 }
```

5.62.2.4 withTask()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withTask (
            std::unique_ptr< rtos::ITask > & task )
```

Add a task handler to the build.

Parameters

task	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 }
```

5.62.2.5 withPID()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withPID (
            control::PID pid )
```

Add a PID controller to the build.

Parameters

pid	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 }
```

5.62.2.6 withMotor()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withMotor (
            std::unique_ptr< io::IMotor > & motor )
```

Add a motor to the build.

Parameters

motor	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 }
```

5.62.2.7 withRollerRadius()

```
PIDIntakeBuilder * wisco::robot::subsystems::intake::PIDIntakeBuilder::withRollerRadius (
            double roller_radius )
```

Add a roller radius to the build.

Parameters

roller_radius	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 }
```

5.62.2.8 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 }
```

5.62.3 Member Data Documentation

5.62.3.1 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 {};
```

5.62.3.2 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 {};
```

5.62.3.3 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 {};
```

5.62.3.4 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 {};
```

5.62.3.5 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 {};
```

5.62.3.6 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 {};
```

5.62.3.7 m_roller_radius

```
double wisco::robot::subsystems::intake::PIDIntakeBuilder::m_roller_radius {}  [private]
```

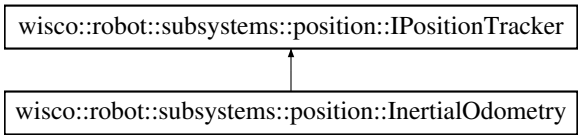
The radius of the intake roller.

Definition at line 86 of file PIDIntakeBuilder.hpp.
```
00086 {};
```

5.63 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::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< 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.

5.63.1 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.

5.63.2 Member Function Documentation

5.63.2.1 taskLoop()

```
void wisco::robot::subsystems::position::InertialOdometry::taskLoop (
            void * params )  [static], [private]
```

The task loop function for background updates.

Parameters

params	

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 }
```

5.63.2.2 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 }
```

5.63.2.3 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 }
```

5.63.2.4 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 }
```

5.63.2.5 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 }
```

5.63.2.6 setPosition()

```
void wisco::robot::subsystems::position::InertialOdometry::setPosition (
            Position position )  [override], [virtual]
```

Set the position of the position tracking system.

Parameters

position	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_mutex)
00131         m_mutex->give();
00132 }
```

5.63.2.7 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 134 of file InertialOdometry.cpp.

```
00135 {
00136     Position position{};
00137     if (m_mutex)
00138         m_mutex->take();
00139     position = m_position;
00140     if (m_mutex)
00141         m_mutex->give();
00142     return position;
00143 }
```

5.63.2.8 setClock()

```
void wisco::robot::subsystems::position::InertialOdometry::setClock (
            std::unique_ptr< rtos::IClock > & clock )
```

Set the system clock.

Parameters

clock	The system clock

Definition at line 145 of file InertialOdometry.cpp.

```
00146 {
00147     m_clock = std::move(clock);
00148 }
```

5.63.2.9 setDelayer()

```
void wisco::robot::subsystems::position::InertialOdometry::setDelayer (
            std::unique_ptr< rtos::IDelayer > & delayer )
```

Set the rtos delayer.

Parameters

delayer	The rtos delayer

Definition at line 150 of file InertialOdometry.cpp.

```
00151 {
00152     m_delayer = std::move(delayer);
00153 }
```

5.63.2.10 setMutex()

```
void wisco::robot::subsystems::position::InertialOdometry::setMutex (
            std::unique_ptr< rtos::IMutex > & mutex )
```

Set the os mutex.

Parameters

mutex	The os mutex

Definition at line 155 of file InertialOdometry.cpp.

```
00156 {
00157     m_mutex = std::move(mutex);
00158 }
```

5.63.2.11 setTask()

```
void wisco::robot::subsystems::position::InertialOdometry::setTask (
            std::unique_ptr< rtos::ITask > & task )
```

Set the rtos task handler.

Parameters

task	The rtos task handler

Definition at line 160 of file InertialOdometry.cpp.

```
00161 {
00162     m_task = std::move(task);
00163 }
```

5.63.2.12 setHeadingSensor()

```
void wisco::robot::subsystems::position::InertialOdometry::setHeadingSensor (
            std::unique_ptr< io::IHeadingSensor > & heading_sensor )
```

Set the heading sensor.

Parameters

heading_sensor	The heading sensor

Definition at line 165 of file InertialOdometry.cpp.

```
00166 {
00167     m_heading_sensor = std::move(heading_sensor);
00168 }
```

5.63.2.13 setLinearDistanceTrackingSensor()

```
void wisco::robot::subsystems::position::InertialOdometry::setLinearDistanceTrackingSensor (
            std::unique_ptr< io::IDistanceTrackingSensor > & linear_distance_tracking_sensor
)
```

Set the linear distance tracking sensor.

Parameters

linear_distance_tracking_sensor	The linear distance tracking sensor

Definition at line 170 of file InertialOdometry.cpp.

```
00171 {
00172     m_linear_distance_tracking_sensor = std::move(linear_distance_tracking_sensor);
00173 }
```

5.63.2.14 setLinearDistanceTrackingOffset()

```
void wisco::robot::subsystems::position::InertialOdometry::setLinearDistanceTrackingOffset (
            double linear_distance_tracking_offset )
```

Set the linear distance tracking offset.

Parameters

linear_distance_tracking_offset	The linear distance tracking offset

Definition at line 175 of file InertialOdometry.cpp.

```
00176 {
00177     m_linear_distance_tracking_offset = linear_distance_tracking_offset;
00178 }
```

5.63.2.15 setStrafeDistanceTrackingSensor()

```
void wisco::robot::subsystems::position::InertialOdometry::setStrafeDistanceTrackingSensor (
            std::unique_ptr< io::IDistanceTrackingSensor > & strafe_distance_tracking_sensor
)
```

Set the strafe distance tracking sensor.

Parameters

strafe_distance_tracking_sensor	The strafe distance tracking sensor

Definition at line 180 of file InertialOdometry.cpp.

```
00181 {
00182     m_strafe_distance_tracking_sensor = std::move(strafe_distance_tracking_sensor);
00183 }
```

5.63.2.16 setStrafeDistanceTrackingOffset()

```
void wisco::robot::subsystems::position::InertialOdometry::setStrafeDistanceTrackingOffset (
            double strafe_distance_tracking_offset )
```

Set the strafe distance tracking offset.

Parameters

strafe_distance_tracking_offset	The strafe distance tracking offset

Definition at line 185 of file InertialOdometry.cpp.

```
00186 {
00187     m_strafe_distance_tracking_offset = strafe_distance_tracking_offset;
00188 }
```

5.63.3 Member Data Documentation

5.63.3.1 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};
```

5.63.3.2 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};
```

5.63.3.3 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 {};
```

5.63.3.4 m_delayer

```
std::unique_ptr<rtos::IDelayer> wisco::robot::subsystems::position::InertialOdometry::m_↩
delayer {}  [private]
```

The system delayer.

Definition at line 85 of file InertialOdometry.hpp.

```
00085 {};
```

5.63.3.5 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 {};
```

5.63.3.6 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 {};
```

5.63.3.7 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 {};
```

5.63.3.8 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 {};
```

5.63.3.9 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 {};
```

5.63.3.10 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 {};
```

5.63.3.11 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 {};
```

5.63.3.12 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 {};
```

5.63.3.13 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 {};
```

5.63.3.14 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 {};
```

5.63.3.15 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 {};
```

5.63.3.16 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 {};
```

5.64 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::IDelayer > &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::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< 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.

5.64.1 Detailed Description

Builder class for the inertial odometry class.

Author

Nathan Sandvig

Definition at line 43 of file InertialOdometryBuilder.hpp.

5.64.2 Member Function Documentation

5.64.2.1 withClock()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
Clock (
            std::unique_ptr< wisco::rtos::IClock > & clock )
```

Adds a system clock to the builder.

Parameters

clock	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 }
```

5.64.2.2 withDelayer()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
Delayer (
            std::unique_ptr< wisco::rtos::IDelayer > & delayer )
```

Adds a system delayer to the builder.

Parameters

delayer	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 }
```

5.64.2.3 withMutex()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
Mutex (
            std::unique_ptr< wisco::rtos::IMutex > & mutex )
```

Adds a system mutex to the builder.

Parameters

mutex	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 }
```

5.64.2.4 withTask()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
Task (
            std::unique_ptr< wisco::rtos::ITask > & task )
```

Adds a system task to the builder.

Parameters

task	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 }
```

5.64.2.5 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

heading_sensor	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 }
```

5.64.2.6 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

linear_distance_tracking_sensor	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 }
```

5.64.2.7 withLinearDistanceTrackingOffset()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
LinearDistanceTrackingOffset (
            double linear_distance_tracking_offset )
```

Adds a linear distance tracking offset to the builder.

Parameters

linear_distance_tracking_offset	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 }
```

5.64.2.8 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

strafe_distance_tracking_sensor	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 }
```

5.64.2.9 withStrafeDistanceTrackingOffset()

```
InertialOdometryBuilder * wisco::robot::subsystems::position::InertialOdometryBuilder::with↩
StrafeDistanceTrackingOffset (
            double strafe_distance_tracking_offset )
```

Adds a strafe distance tracking offset to the builder.

Parameters

strafe_distance_tracking_offset	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 }
```

5.64.2.10 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 }
```

5.64.3 Member Data Documentation

5.64.3.1 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 {};
```

5.64.3.2 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 {};
```

5.64.3.3 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 {};
```

5.64.3.4 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 {};
```

5.64.3.5 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 {};
```

5.64.3.6 m_linear_distance_tracking_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::Inertial↩
OdometryBuilder::m_linear_distance_tracking_sensor {}  [private]
```

The sensor to track the robot's linear movement.

Definition at line 80 of file InertialOdometryBuilder.hpp.

```
00080 {};
```

5.64.3.7 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 {};
```

5.64.3.8 m_strafe_distance_tracking_sensor

```
std::unique_ptr<io::IDistanceTrackingSensor> wisco::robot::subsystems::position::Inertial↩
OdometryBuilder::m_strafe_distance_tracking_sensor {}  [private]
```

The sensor to track the robot's strafe movement.

Definition at line 93 of file InertialOdometryBuilder.hpp.

```
00093 {};
```

5.64.3.9 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.

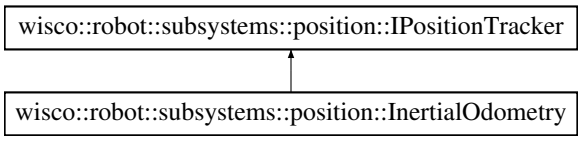
Definition at line 100 of file InertialOdometryBuilder.hpp.

```
00100 {};
```

5.65 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.

5.65.1 Detailed Description

Interface for position tracking subsystems.

Author

Nathan Sandvig

Definition at line 43 of file IPositionTracker.hpp.

5.65.2 Member Function Documentation

5.65.2.1 initialize()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::initialize ( )  [pure virtual]
```

Initializes the position tracking system.

Implemented in wisco::robot::subsystems::position::InertialOdometry.

5.65.2.2 run()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::run ( )  [pure virtual]
```

Runs the position tracking system.

Implemented in wisco::robot::subsystems::position::InertialOdometry.

5.65.2.3 setPosition()

```
virtual void wisco::robot::subsystems::position::IPositionTracker::setPosition (
            Position position )  [pure virtual]
```

Set the position of the position tracking system.

Parameters

position	The new position

Implemented in wisco::robot::subsystems::position::InertialOdometry.

5.65.2.4 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.

5.66 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.

5.66.1 Detailed Description

Holds a robot position.

Author

Nathan Sandvig

Definition at line 41 of file Position.hpp.

5.66.2 Member Data Documentation

5.66.2.1 x

```
double wisco::robot::subsystems::position::Position::x {}
```

The X-coordinate.

Definition at line 47 of file Position.hpp.
```
00047 {};
```

5.66.2.2 y

```
double wisco::robot::subsystems::position::Position::y {}
```

The Y-coordinate.

Definition at line 53 of file Position.hpp.
```
00053 {};
```

5.66.2.3 theta

```
double wisco::robot::subsystems::position::Position::theta {}
```

The angle.

Definition at line 59 of file Position.hpp.
```
00059 {};
```

5.66.2.4 xV

```
double wisco::robot::subsystems::position::Position::xV {}
```

The X-velocity.

Definition at line 65 of file Position.hpp.
```
00065 {};
```

5.66.2.5 yV

```
double wisco::robot::subsystems::position::Position::yV {}
```

The Y-velocity.

Definition at line 71 of file Position.hpp.
```
00071 {};
```

5.66.2.6 thetaV

```
double wisco::robot::subsystems::position::Position::thetaV {}
```

The angular velocity.

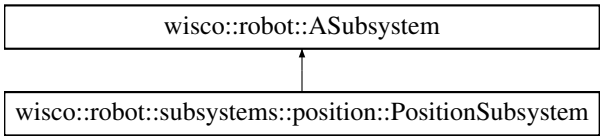
Definition at line 77 of file Position.hpp.

```
00077 {};
```

5.67 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)
  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.

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 GET_POSITION_STATE_NAME [ ] {"GET POSITION"}
  The name of the get position command.

5.67.1 Detailed Description

Adapter from a position tracker to a robot subsystem.

Author

Nathan Sandvig

Definition at line 46 of file PositionSubsystem.hpp.

5.67.2 Constructor & Destructor Documentation

5.67.2.1 PositionSubsystem()

```
wisco::robot::subsystems::position::PositionSubsystem::PositionSubsystem (
            std::unique_ptr< IPositionTracker > & position_tracker )
```

Construct a new Position Subsystem object.

Parameters

position_tracker	The position tracker being adapted

Definition at line 12 of file PositionSubsystem.cpp.

```
00013     : ASubsystem{SUBSYSTEM_NAME}, m_position_tracker{std::move(position_tracker)}
00014 {
00015
00016 }
```

5.67.3 Member Function Documentation

5.67.3.1 initialize()

```
void wisco::robot::subsystems::position::PositionSubsystem::initialize ( )  [override], [virtual]
```

Initializes the subsystem.

Implements wisco::robot::ASubsystem.

Definition at line 18 of file PositionSubsystem.cpp.

```
00019 {
00020     m_position_tracker->initialize();
00021 }
```

5.67.3.2 run()

```
void wisco::robot::subsystems::position::PositionSubsystem::run ( )  [override], [virtual]
```

Runs the subsystems.

Implements wisco::robot::ASubsystem.

Definition at line 23 of file PositionSubsystem.cpp.

```
00024 {
00025     m_position_tracker->run();
00026 }
```

5.67.3.3 command()

```
void wisco::robot::subsystems::position::PositionSubsystem::command (
            std::string command_name,
            va_list & args )  [override], [virtual]
```

Runs a command for the subsystem.

Parameters

command_name	The name of the command to run
args	The parameters for the command

Implements wisco::robot::ASubsystem.

Definition at line 28 of file PositionSubsystem.cpp.

```
00029 {
00030     if (command_name == SET_POSITION_COMMAND_NAME)
00031     {
00032         Position position{va_arg(args, double), va_arg(args, double), va_arg(args, double)};
00033         m_position_tracker->setPosition(position);
00034     }
00035 }
```

5.67.3.4 state()

```
void * wisco::robot::subsystems::position::PositionSubsystem::state (
            std::string state_name )  [override], [virtual]
```

Gets a state of the subsystem.

Parameters

state_name	The name of the state to get

Returns

void* The current value of that state

Implements wisco::robot::ASubsystem.

Definition at line 37 of file PositionSubsystem.cpp.

```
00038 {
00039     void* result{};
00040
00041     if (state_name == GET_POSITION_STATE_NAME)
00042     {
00043         Position* position{new Position{m_position_tracker->getPosition()}};
00044         result = position;
00045     }
00046
00047     return result;
00048 }
```

5.67.4 Member Data Documentation

5.67.4.1 SUBSYSTEM_NAME

```
constexpr char wisco::robot::subsystems::position::PositionSubsystem::SUBSYSTEM_NAME[] {"POSITION
TRACKER"}  [static], [constexpr], [private]
```

The name of the subsystem.

Definition at line 53 of file PositionSubsystem.hpp.

```
00053 {"POSITION TRACKER"};
```

5.67.4.2 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 59 of file PositionSubsystem.hpp.

```
00059 {"SET POSITION"};
```

5.67.4.3 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 65 of file PositionSubsystem.hpp.

```
00065 {"GET POSITION"};
```

5.67.4.4 m_position_tracker

```
std::unique_ptr<IPositionTracker> wisco::robot::subsystems::position::PositionSubsystem::m_↩
position_tracker {}  [private]
```

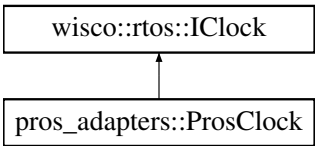
The position tracker being adapted.

Definition at line 71 of file PositionSubsystem.hpp.
```
00071 {};
```

5.68 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.

5.68.1 Detailed Description

Interface for an rtos system clock.

Author

Nathan Sandvig

Definition at line 28 of file IClock.hpp.

5.68.2 Member Function Documentation

5.68.2.1 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.

5.68.2.2 getTime()

```
virtual uint32_t wisco::rtos::IClock::getTime ( )  [pure virtual]
```

Get the clock time in milliseconds.

Returns

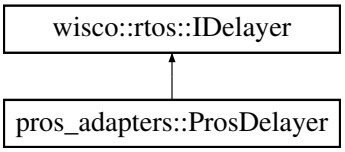
uint32_t The clock time in milliseconds

Implemented in pros_adapters::ProsClock.

5.69 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.

5.69.1 Detailed Description

Interface for rtos delay systems.

Author

Nathan Sandvig

Definition at line 28 of file IDelayer.hpp.

5.69.2 Member Function Documentation

5.69.2.1 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.

5.69.2.2 delay()

```
virtual void wisco::rtos::IDelayer::delay (
            uint32_t millis )  [pure virtual]
```

Delays the rtos system for a number of milliseconds.

Parameters

millis	The number of milliseconds to delay

Implemented in pros_adapters::ProsDelayer.

5.69.2.3 delayUntil()

```
virtual void wisco::rtos::IDelayer::delayUntil (
            uint32_t time )  [pure virtual]
```

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

Parameters

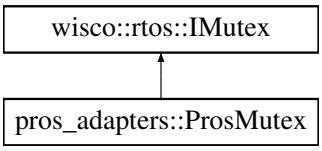
time	The time in milliseconds to delay until

Implemented in pros_adapters::ProsDelayer.

5.70 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.

5.70.1 Detailed Description

Interface for rtos mutexes.

Author

Nathan Sandvig

Definition at line 25 of file IMutex.hpp.

5.70.2 Member Function Documentation

5.70.2.1 take()

```
virtual void wisco::rtos::IMutex::take ( )  [pure virtual]
```

Takes the mutex and locks it.

Implemented in pros_adapters::ProsMutex.

5.70.2.2 give()

```
virtual void wisco::rtos::IMutex::give ( )  [pure virtual]
```

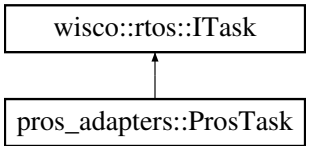
Gives the mutex and unlocks it.

Implemented in pros_adapters::ProsMutex.

5.71 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.

5.71.1 Detailed Description

Interface for an rtos task system.

Author

Nathan Sandvig

Definition at line 25 of file ITask.hpp.

5.71.2 Member Function Documentation

5.71.2.1 start()

```
virtual void wisco::rtos::ITask::start (
            void(*)(void *) function,
            void * parameters )  [pure virtual]
```

Starts the task.

Parameters

function	The function to run in the task
parameters	The parameters for the function

Implemented in pros_adapters::ProsTask.

5.71.2.2 remove()

```
virtual void wisco::rtos::ITask::remove ( )  [pure virtual]
```

Removes the task from the system.

Implemented in pros_adapters::ProsTask.

5.71.2.3 suspend()

```
virtual void wisco::rtos::ITask::suspend ( )  [pure virtual]
```

Suspends the task in the scheduler.

Implemented in pros_adapters::ProsTask.

5.71.2.4 resume()

```
virtual void wisco::rtos::ITask::resume ( )  [pure virtual]
```

Resumes the task in the scheduler.

Implemented in pros_adapters::ProsTask.

5.71.2.5 join()

```
virtual void wisco::rtos::ITask::join ( )  [pure virtual]
```

Waits for the task to finish.

Implemented in pros_adapters::ProsTask.

5.72 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.

5.72.1 Detailed Description

Holds the system configuration information.

Author

Nathan Sandvig

Definition at line 24 of file SystemConfiguration.hpp.

5.72.2 Member Data Documentation

5.72.2.1 alliance

```
std::unique_ptr<IAlliance> wisco::SystemConfiguration::alliance {}
```

The system alliance.

Definition at line 30 of file SystemConfiguration.hpp.
```
00030 {};
```

5.72.2.2 autonomous

```
std::unique_ptr<IAutonomous> wisco::SystemConfiguration::autonomous {}
```

The system autonomous.

Definition at line 36 of file SystemConfiguration.hpp.
```
00036 {};
```

5.72.2.3 configuration

```
std::unique_ptr<IConfiguration> wisco::SystemConfiguration::configuration {}
```

The system configuration.

Definition at line 42 of file SystemConfiguration.hpp.
```
00042 {};
```

5.72.2.4 profile

```
std::unique_ptr<IProfile> wisco::SystemConfiguration::profile {}
```

The system profile.

Definition at line 48 of file SystemConfiguration.hpp.
```
00048 {};
```

5.73 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.

5.73.1 Detailed Description

Tests a pros-based drive.

Author

Nathan Sandvig

Definition at line 48 of file DriveTest.hpp.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 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

left_drive_motors	The left drive motors
right_drive_motors	The right drive motors
heading_sensor	The heading sensor
linear_sensor	The linear distance tracking sensor
linear_counts_per_inch	The lienar 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 }
```

5.73.3 Member Function Documentation

5.73.3.1 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 }
```

5.73.3.2 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 *
      INCHES_TO_METERS};
00049         double last_time{pros::millis() * MILLIS_TO_S};
00050         while (pros::millis() - start_time < TEST_DURATION)
00051         {
00052             double current_position{m_linear_sensor->get_position() / m_linear_counts_per_inch *
      INCHES_TO_METERS};
00053             if (current_position != last_position)
00054             {
00055                 double position_change{current_position - last_position};
00056                 last_position = current_position;
00057
00058                 double current_time{pros::millis() * MILLIS_TO_S};
00059                 double time_change{current_time - last_time};
00060                 last_time = current_time;
00061
00062                 double velocity{position_change / time_change};
00063                 test_output_file « velocity « ',' « current_time « '\n';
00064                 test_output_file.flush();
00065             }
00066             pros::delay(1);
00067         }
00068
00069         m_left_drive_motors->move_voltage(0);
00070         m_right_drive_motors->move_voltage(0);
00071     }
00072
00073     test_output_file.close();
00074 }
```

5.73.3.3 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 }
```

5.73.4 Member Data Documentation

5.73.4.1 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"};
```

5.73.4.2 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"};
```

5.73.4.3 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};
```

5.73.4.4 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};
```

5.73.4.5 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};
```

5.73.4.6 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};
```

5.73.4.7 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};
```

5.73.4.8 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};
```

5.73.4.9 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 {};
```

5.73.4.10 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 {};
```

5.73.4.11 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 {};
```

5.73.4.12 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 {};
```

5.73.4.13 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.

5.74 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.

5.74.1 Detailed Description

Factory to build test classes.

Author

Nathan Sandvig

Definition at line 30 of file TestFactory.hpp.

5.74.2 Member Function Documentation

5.74.2.1 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>
      left_drive_motors{std::make_unique<pros::MotorGroup>(LEFT_DRIVE_PORTS)};
00013     std::unique_ptr<pros::MotorGroup>
      right_drive_motors{std::make_unique<pros::MotorGroup>(RIGHT_DRIVE_PORTS)};
00014     std::unique_ptr<pros::Imu> heading_sensor{std::make_unique<pros::Imu>(INERTIAL_PORT)};
00015     std::unique_ptr<pros::Rotation>
      linear_tracking_sensor{std::make_unique<pros::Rotation>(LINEAR_TRACKING_PORT)};
00016
00017     return std::make_unique<pros_testing::DriveTest>(left_drive_motors, right_drive_motors,
      heading_sensor, linear_tracking_sensor, LINEAR_COUNTS_PER_INCH);
00018 }
```

5.74.3 Member Data Documentation

5.74.3.1 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.

5.74.3.2 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.

5.74.3.3 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.

```
00049 {9};
```

5.74.3.4 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};
```

5.74.3.5 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};
```

5.75 wisco::user::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.

5.75.1 Detailed Description

Runs the operator-controlled differential drive voltage settings.

Author

Nathan Sandvig

Definition at line 31 of file DifferentialDriveOperator.hpp.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 DifferentialDriveOperator()

```
wisco::user::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 56 of file DifferentialDriveOperator.cpp.

```
00058     : m_controller{controller}, m_robot{robot}
00059 {
00060
00061 }
```

5.75.3 Member Function Documentation

5.75.3.1 updateDriveVoltage()

```
void wisco::user::DifferentialDriveOperator::updateDriveVoltage (
            double left_voltage,
            double right_voltage )  [private]
```

Updates the voltage of the drive subsystem.

Parameters

left_voltage	The left drive voltage
right_voltage	The right drive voltage

Definition at line 7 of file DifferentialDriveOperator.cpp.

```
00008 {
00009     if (m_robot)
00010     {
00011         m_robot->sendCommand(DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME, SET_VOLTAGE_COMMAND, left_voltage,
      right_voltage);
00012     }
00013 }
```

5.75.3.2 updateArcade()

```
void wisco::user::DifferentialDriveOperator::updateArcade (
            double forward,
            double turn )  [private]
```

Updates the drive using arcade inputs.

Parameters

forward	The forward voltage
turn	The turn voltage (positive to the right)

Definition at line 15 of file DifferentialDriveOperator.cpp.

```
00016 {
00017     double left_voltage{(forward + turn) * VOLTAGE_CONVERSION};
00018     double right_voltage{(forward - turn) * VOLTAGE_CONVERSION};
00019     updateDriveVoltage(left_voltage, right_voltage);
00020 }
```

5.75.3.3 updateSingleArcadeLeft()

```
void wisco::user::DifferentialDriveOperator::updateSingleArcadeLeft ( )  [private]
```

Update the drive voltage for single left stick arcade drive.

Definition at line 22 of file DifferentialDriveOperator.cpp.

```
00023 {
00024     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y)};
00025     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_X)};
00026     updateArcade(forward, turn);
00027 }
```

5.75.3.4 updateSingleArcadeRight()

```
void wisco::user::DifferentialDriveOperator::updateSingleArcadeRight ( )  [private]
```

Update the drive voltage for single right stick arcade drive.

Definition at line 29 of file DifferentialDriveOperator.cpp.

```
00030 {
00031     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y)};
00032     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_X)};
00033     updateArcade(forward, turn);
00034 }
```

5.75.3.5 updateSplitArcadeLeft()

```
void wisco::user::DifferentialDriveOperator::updateSplitArcadeLeft ( )  [private]
```

Update the drive voltage for split stick arcade with left stick forward control.

Definition at line 36 of file DifferentialDriveOperator.cpp.

```
00037 {
00038     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y)};
00039     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_X)};
00040     updateArcade(forward, turn);
00041 }
```

5.75.3.6 updateSplitArcadeRight()

```
void wisco::user::DifferentialDriveOperator::updateSplitArcadeRight ( )  [private]
```

Update the drive voltage for split stick arcade with right stick forward control.

Definition at line 43 of file DifferentialDriveOperator.cpp.

```
00044 {
00045     double forward{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y)};
00046     double turn{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_X)};
00047     updateArcade(forward, turn);
00048 }
```

5.75.3.7 updateTank()

```
void wisco::user::DifferentialDriveOperator::updateTank ( )  [private]
```

Update the drive voltage for tank control.

Definition at line 50 of file DifferentialDriveOperator.cpp.

```
00051 {
00052     double left_voltage{m_controller->getAnalog(EControllerAnalog::JOYSTICK_LEFT_Y) *
      VOLTAGE_CONVERSION};
00053     double right_voltage{m_controller->getAnalog(EControllerAnalog::JOYSTICK_RIGHT_Y) *
      VOLTAGE_CONVERSION};
00054 }
```

5.75.3.8 setDriveVoltage()

```
void wisco::user::DifferentialDriveOperator::setDriveVoltage (
            EChassisControlMode control_mode )
```

Set the drive voltage.

Parameters

control_mode	The control mode of the drive

Definition at line 63 of file DifferentialDriveOperator.cpp.

```
00064 {
00065     if (!m_controller)
00066     {
00067         updateDriveVoltage(0, 0);
00068         return;
00069     }
00070
00071     switch (control_mode)
00072     {
00073         case EChassisControlMode::SINGLE_ARCADE_LEFT:
00074             updateSingleArcadeLeft();
00075             break;
00076         case EChassisControlMode::SINGLE_ARCADE_RIGHT:
00077             updateSingleArcadeRight();
00078             break;
00079         case EChassisControlMode::SPLIT_ARCADE_LEFT:
00080             updateSplitArcadeLeft();
00081             break;
00082         case EChassisControlMode::SPLIT_ARCADE_RIGHT:
00083             updateSplitArcadeRight();
00084             break;
00085         case EChassisControlMode::TANK:
00086             updateTank();
00087             break;
00088     }
00089 }
```

5.75.4 Member Data Documentation

5.75.4.1 DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME

```
constexpr char wisco::user::DifferentialDriveOperator::DIFFERENTIAL_DRIVE_SUBSYSTEM_NAME[]
{"DIFFERENTIAL DRIVE"}  [static], [constexpr], [private]
```

The name of the differential drive subsystem.

Definition at line 38 of file DifferentialDriveOperator.hpp.

```
00038 {"DIFFERENTIAL DRIVE"};
```

5.75.4.2 SET_VOLTAGE_COMMAND

```
constexpr char wisco::user::DifferentialDriveOperator::SET_VOLTAGE_COMMAND[] {"SET VOLTAGE"}
[static], [constexpr], [private]
```

The command to set drive voltage.

Definition at line 44 of file DifferentialDriveOperator.hpp.

```
00044 {"SET VOLTAGE"};
```

5.75.4.3 VOLTAGE_CONVERSION

```
constexpr double wisco::user::DifferentialDriveOperator::VOLTAGE_CONVERSION {12.0}  [static],
[constexpr], [private]
```

Converts controller input to voltage.

Definition at line 50 of file DifferentialDriveOperator.hpp.

```
00050 {12.0};
```

5.75.4.4 m_controller

```
std::shared_ptr<user::IController> wisco::user::DifferentialDriveOperator::m_controller {}
[private]
```

The user input controller.

Definition at line 56 of file DifferentialDriveOperator.hpp.
```
00056 {};
```

5.75.4.5 m_robot

```
std::shared_ptr<robot::Robot> wisco::user::DifferentialDriveOperator::m_robot {}  [private]
```

The robot being controlled.

Definition at line 62 of file DifferentialDriveOperator.hpp.
```
00062 {};
```

5.76 wisco::user::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** , **OUT** }

  The available states for elevator toggles.

Private Member Functions

- double getElevatorPosition ()

  Gets the current elevator position.
- void updateElevatorPosition (double position)

  Updates the position of the elevator subsystem.
- void updateManual (EControllerDigital in, EControllerDigital out)

  Updates the elevator position based on manual control.
- void updatePresetSplit (EControllerDigital in, EControllerDigital field, EControllerDigital match_load, EControllerDigital out)

  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.

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 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 double IN_POSITION {0.0}

  The in position for the elevator.
- static constexpr double FIELD_POSITION {4.0}

  The field position for the elevator.
- static constexpr double MATCH_LOAD_POSITION {7.0}

  The match loading position for the elevator.
- static constexpr double OUT_POSITION {20.0}

  The out position for the elevator.

5.76.1 Detailed Description

Runs the operator-controlled elevator position settings.

Author

Nathan Sandvig

Definition at line 32 of file ElevatorOperator.hpp.

5.76.2 Member Enumeration Documentation

5.76.2.1 EToggleState

```
enum class wisco::user::ElevatorOperator::EToggleState  [strong], [private]
```

The available states for elevator toggles.

Definition at line 39 of file ElevatorOperator.hpp.

```
00040     {
00041         IN,
00042         FIELD,
00043         MATCH_LOAD,
00044         OUT
00045     };
```

5.76.3 Constructor & Destructor Documentation

5.76.3.1 ElevatorOperator()

```
wisco::user::ElevatorOperator::ElevatorOperator (
            const std::shared_ptr< user::IController > & controller,
            const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Elevator Operator object.

Parameters

controller	The user input controller
robot	The robot to control

Definition at line 8 of file ElevatorOperator.cpp.

```
00010     : m_controller{controller}, m_robot{robot}
00011 {
00012
00013 }
```

5.76.4 Member Function Documentation

5.76.4.1 getElevatorPosition()

```
double wisco::user::ElevatorOperator::getElevatorPosition ( )  [private]
```

Gets the current elevator position.

Returns

double The current elevator position

Definition at line 15 of file ElevatorOperator.cpp.

```
00016 {
00017     double* result{static_cast<double*>(m_robot->getState(ELEVATOR_SUBSYSTEM_NAME,
      GET_POSITION_STATE))};
00018     double position{*result};
00019     delete result;
00020     return position;
00021 }
```

5.76.4.2 updateElevatorPosition()

```
void wisco::user::ElevatorOperator::updateElevatorPosition (
            double position )  [private]
```

Updates the position of the elevator subsystem.

Parameters

position	The elevator position

Definition at line 23 of file ElevatorOperator.cpp.

```
00024 {
00025     m_robot->sendCommand(ELEVATOR_SUBSYSTEM_NAME, SET_POSITION_COMMAND, position);
00026 }
```

5.76.4.3 updateManual()

```
void wisco::user::ElevatorOperator::updateManual (
            EControllerDigital in,
            EControllerDigital out )  [private]
```

Updates the elevator position based on manual control.

Parameters

in	The digital control for moving the elevator in
out	The digital control for moving the elevator out

Definition at line 28 of file ElevatorOperator.cpp.

```
00029 {
00030     bool move_in{m_controller->getDigital(in)};
00031     bool move_out{m_controller->getDigital(out)};
00032     if (move_in && move_out)
00033     {
00034         updateElevatorPosition(getElevatorPosition());
00035         manual_input = true;
00036     }
00037     else if (move_in)
00038     {
00039         updateElevatorPosition(IN_POSITION);
00040         manual_input = true;
00041     }
00042     else if (move_out)
00043     {
00044         updateElevatorPosition(OUT_POSITION);
00045         manual_input = true;
00046     }
00047     else if (manual_input)
00048     {
00049         updateElevatorPosition(getElevatorPosition());
00050         manual_input = false;
00051     }
00052 }
```

5.76.4.4 updatePresetSplit()

```
void wisco::user::ElevatorOperator::updatePresetSplit (
            EControllerDigital in,
            EControllerDigital field,
            EControllerDigital match_load,
            EControllerDigital out )  [private]
```

Updates the elevator position based on preset split control.

Parameters

in	The digital control for the in position
field	The digital control for the field position
match_load	The digital control for the match load position
out	The digital control for the out position

Definition at line 54 of file ElevatorOperator.cpp.

```
00055 {
00056     bool move_in{m_controller->getNewDigital(in)};
00057     bool move_field{m_controller->getNewDigital(field)};
00058     bool move_match_load{m_controller->getNewDigital(match_load)};
00059     bool move_out{m_controller->getNewDigital(out)};
00060
00061     if (move_in && !move_field && !move_match_load && !move_out)
00062         updateElevatorPosition(IN_POSITION);
00063     else if (!move_in && move_field && !move_match_load && !move_out)
00064         updateElevatorPosition(FIELD_POSITION);
00065     else if (!move_in && !move_field && move_match_load && !move_out)
00066         updateElevatorPosition(MATCH_LOAD_POSITION);
00067     else if (!move_in && !move_field && !move_match_load && move_out)
00068         updateElevatorPosition(OUT_POSITION);
00069 }
```

5.76.4.5 updatePresetToggle()

```
void wisco::user::ElevatorOperator::updatePresetToggle (
            EControllerDigital toggle )  [private]
```

Updates the elevator position based on a toggle.

Parameters

toggle	The digital control for the toggle

Definition at line 71 of file ElevatorOperator.cpp.

```
00072 {
00073     if (m_controller->getNewDigital(toggle))
00074     {
00075         switch (toggle_state)
00076         {
00077         case EToggleState::IN:
00078             updateElevatorPosition(FIELD_POSITION);
00079             toggle_state = EToggleState::FIELD;
00080             break;
00081         case EToggleState::FIELD:
00082             updateElevatorPosition(MATCH_LOAD_POSITION);
00083             toggle_state = EToggleState::MATCH_LOAD;
00084             break;
00085         case EToggleState::MATCH_LOAD:
00086             updateElevatorPosition(OUT_POSITION);
00087             toggle_state = EToggleState::OUT;
00088             break;
00089         case EToggleState::OUT:
00090             updateElevatorPosition(IN_POSITION);
00091             toggle_state = EToggleState::IN;
00092             break;
00093         }
00094     }
00095 }
```

5.76.4.6 updatePresetLadder()

```
void wisco::user::ElevatorOperator::updatePresetLadder (
            EControllerDigital in,
            EControllerDigital out )  [private]
```

Updates the elevator position based on a ladder toggle system.

Parameters

in	The digital control for the next position inward
out	The digital control for the next position outward

Definition at line 97 of file ElevatorOperator.cpp.

```
00098 {
00099     bool move_in{m_controller->getNewDigital(in)};
00100     bool move_out{m_controller->getNewDigital(out)};
00101     if (move_in && !move_out)
00102     {
00103         switch (toggle_state)
00104         {
00105         case EToggleState::IN:
00106             updateElevatorPosition(FIELD_POSITION);
00107             toggle_state = EToggleState::FIELD;
00108             break;
00109         case EToggleState::FIELD:
00110             updateElevatorPosition(MATCH_LOAD_POSITION);
00111             toggle_state = EToggleState::MATCH_LOAD;
00112             break;
00113         case EToggleState::MATCH_LOAD:
00114             updateElevatorPosition(OUT_POSITION);
00115             toggle_state = EToggleState::OUT;
00116             break;
00117         case EToggleState::OUT:
00118             break;
00119         }
00120     }
00121     else if (!move_in && move_out)
00122     {
00123         switch (toggle_state)
00124         {
00125         case EToggleState::IN:
00126             break;
00127         case EToggleState::FIELD:
00128             updateElevatorPosition(IN_POSITION);
00129             toggle_state = EToggleState::IN;
00130             break;
00131         case EToggleState::MATCH_LOAD:
00132             updateElevatorPosition(FIELD_POSITION);
00133             toggle_state = EToggleState::FIELD;
00134             break;
00135         case EToggleState::OUT:
00136             updateElevatorPosition(MATCH_LOAD_POSITION);
00137             toggle_state = EToggleState::MATCH_LOAD;
00138             break;
00139         }
00140     }
00141 }
```

5.76.4.7 setElevatorPosition()

```
void wisco::user::ElevatorOperator::setElevatorPosition (
            const std::unique_ptr< IProfile > & profile )
```

Set the elevator position.

Parameters

profile	The driver profile

Definition at line 143 of file ElevatorOperator.cpp.

```
00144 {
00145     EControllerDigital in{profile->getDigitalControlMapping(EControl::ELEVATOR_IN)};
00146     EControllerDigital field{profile->getDigitalControlMapping(EControl::ELEVATOR_FIELD)};
00147     EControllerDigital match_load{profile->getDigitalControlMapping(EControl::ELEVATOR_MATCH_LOAD)};
00148     EControllerDigital out{profile->getDigitalControlMapping(EControl::ELEVATOR_OUT)};
00149     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::ELEVATOR_TOGGLE)};
00150
00151     switch (static_cast<EElevatorControlMode>(profile->getControlMode(EControlType::ELEVATOR)))
00152     {
00153     case EElevatorControlMode::MANUAL:
00154         updateManual(in, out);
00155         break;
00156     case EElevatorControlMode::PRESET_SPLIT:
00157         updatePresetSplit(in, field, match_load, out);
00158         break;
00159     case EElevatorControlMode::PRESET_TOGGLE_LADDER:
00160         updatePresetLadder(in, out);
00161         break;
```

```
00162     case EElevatorControlMode::PRESET_TOGGLE_SINGLE:
00163         updatePresetToggle(toggle);
00164         break;
00165     }
00166 }
```

5.76.5 Member Data Documentation

5.76.5.1 ELEVATOR_SUBSYSTEM_NAME

```
constexpr char wisco::user::ElevatorOperator::ELEVATOR_SUBSYSTEM_NAME[] {"ELEVATOR"}  [static], [constexpr], [private]
```

The name of the elevator subsystem.

Definition at line 51 of file ElevatorOperator.hpp.
```
00051 {"ELEVATOR"};
```

5.76.5.2 SET_POSITION_COMMAND

```
constexpr char wisco::user::ElevatorOperator::SET_POSITION_COMMAND[] {"SET POSITION"}  [static], [constexpr], [private]
```

The command to set elevator position.

Definition at line 57 of file ElevatorOperator.hpp.
```
00057 {"SET POSITION"};
```

5.76.5.3 GET_POSITION_STATE

```
constexpr char wisco::user::ElevatorOperator::GET_POSITION_STATE[] {"GET POSITION"}  [static], [constexpr], [private]
```

The state to get elevator position.

Definition at line 63 of file ElevatorOperator.hpp.
```
00063 {"GET POSITION"};
```

5.76.5.4 IN_POSITION

```
constexpr double wisco::user::ElevatorOperator::IN_POSITION {0.0}  [static], [constexpr], [private]
```

The in position for the elevator.

Definition at line 69 of file ElevatorOperator.hpp.
```
00069 {0.0};
```

5.76.5.5 FIELD_POSITION

```
constexpr double wisco::user::ElevatorOperator::FIELD_POSITION {4.0}  [static], [constexpr],
[private]
```

The field position for the elevator.

Definition at line 75 of file ElevatorOperator.hpp.
```
00075 {4.0};
```

5.76.5.6 MATCH_LOAD_POSITION

```
constexpr double wisco::user::ElevatorOperator::MATCH_LOAD_POSITION {7.0}  [static], [constexpr],
[private]
```

The match loading position for the elevator.

Definition at line 81 of file ElevatorOperator.hpp.
```
00081 {7.0};
```

5.76.5.7 OUT_POSITION

```
constexpr double wisco::user::ElevatorOperator::OUT_POSITION {20.0}  [static], [constexpr],
[private]
```

The out position for the elevator.

Definition at line 87 of file ElevatorOperator.hpp.
```
00087 {20.0};
```

5.76.5.8 m_controller

```
std::shared_ptr<user::IController> wisco::user::ElevatorOperator::m_controller {}  [private]
```

The user input controller.

Definition at line 93 of file ElevatorOperator.hpp.
```
00093 {};
```

5.76.5.9 m_robot

```
std::shared_ptr<robot::Robot> wisco::user::ElevatorOperator::m_robot {}  [private]
```

The robot being controlled.

Definition at line 99 of file ElevatorOperator.hpp.
```
00099 {};
```

5.76.5.10 toggle_state

```
EToggleState wisco::user::ElevatorOperator::toggle_state {EToggleState::IN}  [private]
```

The state stored for toggle mode.

Definition at line 105 of file ElevatorOperator.hpp.
```
00105 {EToggleState::IN};
```

5.76.5.11 manual_input

```
bool wisco::user::ElevatorOperator::manual_input {}  [private]
```

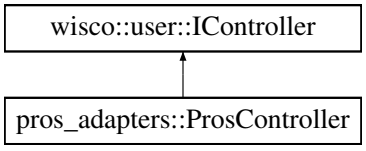
Whether or not there is currently manual input.

Definition at line 111 of file ElevatorOperator.hpp.
```
00111 {};
```

5.77 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.

5.77.1 Detailed Description

Interface for a controller.

Author

Nathan Sandvig

Definition at line 30 of file IController.hpp.

5.77.2 Member Function Documentation

5.77.2.1 initialize()

```
virtual void wisco::user::IController::initialize ( )  [pure virtual]
```

Initializes the controller.

Implemented in pros_adapters::ProsController.

5.77.2.2 run()

```
virtual void wisco::user::IController::run ( )  [pure virtual]
```

Runs the controller.

Implemented in pros_adapters::ProsController.

5.77.2.3 getAnalog()

```
virtual double wisco::user::IController::getAnalog (
            EControllerAnalog analog_channel )  [pure virtual]
```

Get the analog input of a channel from the controller.

Parameters

analog_channel	The channel to read analog input from

Returns

double The value of the analog channel

Implemented in pros_adapters::ProsController.

5.77.2.4 getDigital()

```
virtual bool wisco::user::IController::getDigital (
            EControllerDigital digital_channel )  [pure virtual]
```

Get the digital input of a channel from the controller.

Parameters

digital_channel	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.

5.77.2.5 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

digital_channel	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.

5.77.2.6 rumble()

```
virtual void wisco::user::IController::rumble (
            std::string pattern )  [pure virtual]
```

Rumbles the controller.

Parameters

pattern	The rumble pattern to follow Up to 8 characters, '.' short, '-' long, ' ' pause

Implemented in pros_adapters::ProsController.

5.78 wisco::user::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.

5.78.1 Detailed Description

Runs the operator-controlled intake voltage settings.

Author

Nathan Sandvig

Definition at line 32 of file IntakeOperator.hpp.

5.78.2 Member Enumeration Documentation

5.78.2.1 EToggleState

```
enum class wisco::user::IntakeOperator::EToggleState  [strong], [private]
```

The available states for intake toggles.

Definition at line 39 of file IntakeOperator.hpp.

```
00040     {
00041         OFF,
00042         IN,
00043         OUT
00044     };
```

5.78.3 Constructor & Destructor Documentation

5.78.3.1 IntakeOperator()

```
wisco::user::IntakeOperator::IntakeOperator (
            const std::shared_ptr< user::IController > & controller,
            const std::shared_ptr< robot::Robot > & robot )
```

Construct a new Intake Operator object.

Parameters

controller	The user input controller
robot	The robot to control

Definition at line 74 of file IntakeOperator.cpp.

```
00076     : m_controller{controller}, m_robot{robot}
00077 {
00078
00079 }
```

5.78.4 Member Function Documentation

5.78.4.1 updateIntakeVoltage()

```
void wisco::user::IntakeOperator::updateIntakeVoltage (
            double voltage )  [private]
```

Updates the voltage of the intake subsystem.

Parameters

voltage	The intake voltage

Definition at line 7 of file IntakeOperator.cpp.

```
00008 {
00009     m_robot->sendCommand(INTAKE_SUBSYSTEM_NAME, SET_VOLTAGE_COMMAND, voltage);
00010 }
```

5.78.4.2 updateToggleVoltage()

```
void wisco::user::IntakeOperator::updateToggleVoltage ( )  [private]
```

Updates the intake voltage based on toggle state.

Definition at line 12 of file IntakeOperator.cpp.

```
00013 {
00014     switch(toggle_state)
00015     {
00016     case EToggleState::OFF:
00017         updateIntakeVoltage(VOLTAGE_SETTING);
00018         break;
00019     case EToggleState::IN:
00020         updateIntakeVoltage(-VOLTAGE_SETTING);
00021         break;
00022     case EToggleState::OUT:
00023         updateIntakeVoltage(0);
00024         break;
00025     }
00026 }
```

5.78.4.3 updateSingleToggle()

```
void wisco::user::IntakeOperator::updateSingleToggle (
            EControllerDigital toggle )  [private]
```

Update the voltage for single button toggle.

Definition at line 28 of file IntakeOperator.cpp.

```
00029 {
00030     if (m_controller->getNewDigital(toggle))
00031     {
00032         switch(toggle_state)
00033         {
00034         case EToggleState::OFF:
00035             toggle_state = EToggleState::IN;
00036             break;
00037         case EToggleState::IN:
00038             toggle_state = EToggleState::OUT;
00039             break;
00040         case EToggleState::OUT:
00041             toggle_state = EToggleState::OFF;
00042             break;
00043         }
00044         updateToggleVoltage();
00045     }
00046 }
```

5.78.4.4 updateSplitHold()

```
void wisco::user::IntakeOperator::updateSplitHold (
            EControllerDigital in,
            EControllerDigital out )  [private]
```

Update the voltage for split button hold.

Definition at line 48 of file IntakeOperator.cpp.

```
00049 {
00050     double voltage{(m_controller->getDigital(in) - m_controller->getDigital(out)) * VOLTAGE_SETTING};
00051     updateIntakeVoltage(voltage);
00052 }
```

5.78.4.5 updateSplitToggle()

```
void wisco::user::IntakeOperator::updateSplitToggle (
            EControllerDigital in,
            EControllerDigital out )  [private]
```

Update the voltage for split button toggle.

Definition at line 54 of file IntakeOperator.cpp.

```
00055 {
00056     if (m_controller->getNewDigital(in))
00057     {
00058         if (toggle_state == EToggleState::IN)
00059             toggle_state = EToggleState::OFF;
00060         else
00061             toggle_state = EToggleState::IN;
00062         updateToggleVoltage();
00063     }
00064     else if (m_controller->getNewDigital(out))
00065     {
00066         if (toggle_state == EToggleState::OUT)
00067             toggle_state = EToggleState::OFF;
00068         else
00069             toggle_state = EToggleState::OUT;
00070         updateToggleVoltage();
00071     }
00072 }
```

5.78.4.6 setIntakeVoltage()

```
void wisco::user::IntakeOperator::setIntakeVoltage (
            const std::unique_ptr< IProfile > & profile )
```

Set the intake voltage.

Parameters

profile	The driver profile

Definition at line 81 of file IntakeOperator.cpp.

```
00082 {
00083     EControllerDigital toggle{profile->getDigitalControlMapping(EControl::INTAKE_TOGGLE)};
00084     EControllerDigital in{profile->getDigitalControlMapping(EControl::INTAKE_IN)};
00085     EControllerDigital out{profile->getDigitalControlMapping(EControl::INTAKE_OUT)};
00086     switch(static_cast<EIntakeControlMode>(profile->getControlMode(EControlType::INTAKE)))
00087     {
00088     case EIntakeControlMode::SINGLE_TOGGLE:
00089         updateSingleToggle(toggle);
```

```
00090         break;
00091     case EIntakeControlMode::SPLIT_HOLD:
00092         updateSplitHold(in, out);
00093         break;
00094     case EIntakeControlMode::SPLIT_TOGGLE:
00095         updateSplitToggle(in, out);
00096         break;
00097     }
00098 }
```

5.78.5 Member Data Documentation

5.78.5.1 INTAKE_SUBSYSTEM_NAME

constexpr char wisco::user::IntakeOperator::INTAKE_SUBSYSTEM_NAME[] {"INTAKE"} [static], [constexpr], [private]

The name of the intake subsystem.

Definition at line 50 of file IntakeOperator.hpp.

```
00050 {"INTAKE"};
```

5.78.5.2 SET_VOLTAGE_COMMAND

constexpr char wisco::user::IntakeOperator::SET_VOLTAGE_COMMAND[] {"SET VOLTAGE"} [static], [constexpr], [private]

The command to set intake voltage.

Definition at line 56 of file IntakeOperator.hpp.

```
00056 {"SET VOLTAGE"};
```

5.78.5.3 VOLTAGE_SETTING

constexpr double wisco::user::IntakeOperator::VOLTAGE_SETTING {12.0} [static], [constexpr], [private]

The voltage to run the intake at.

Definition at line 62 of file IntakeOperator.hpp.

```
00062 {12.0};
```

5.78.5.4 m_controller

std::shared_ptr<user::IController> wisco::user::IntakeOperator::m_controller {} [private]

The user input controller.

Definition at line 68 of file IntakeOperator.hpp.

```
00068 {};
```

5.78.5.5 m_robot

```
std::shared_ptr<robot::Robot> wisco::user::IntakeOperator::m_robot {}  [private]
```

The robot being controlled.

Definition at line 74 of file IntakeOperator.hpp.
```
00074 {};
```

5.78.5.6 toggle_state

```
EToggleState wisco::user::IntakeOperator::toggle_state {EToggleState::OFF}  [private]
```

The state stored for toggle mode.

Definition at line 80 of file IntakeOperator.hpp.
```
00080 {EToggleState::OFF};
```

Index