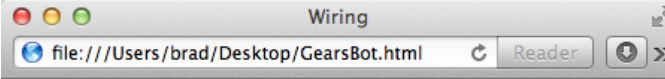


# Constructing the robot program

## Wiring file



**Wiring**

**PWMs on Digital Sidecar 1**

#	Motor
1	Elevator motor Output
2	Wrist motor Output
3	Claw motor Output
5	DriveBase leftFront Output
6	DriveBase rightFront Output
7	DriveBase leftRear Output
8	DriveBase rightRear Output

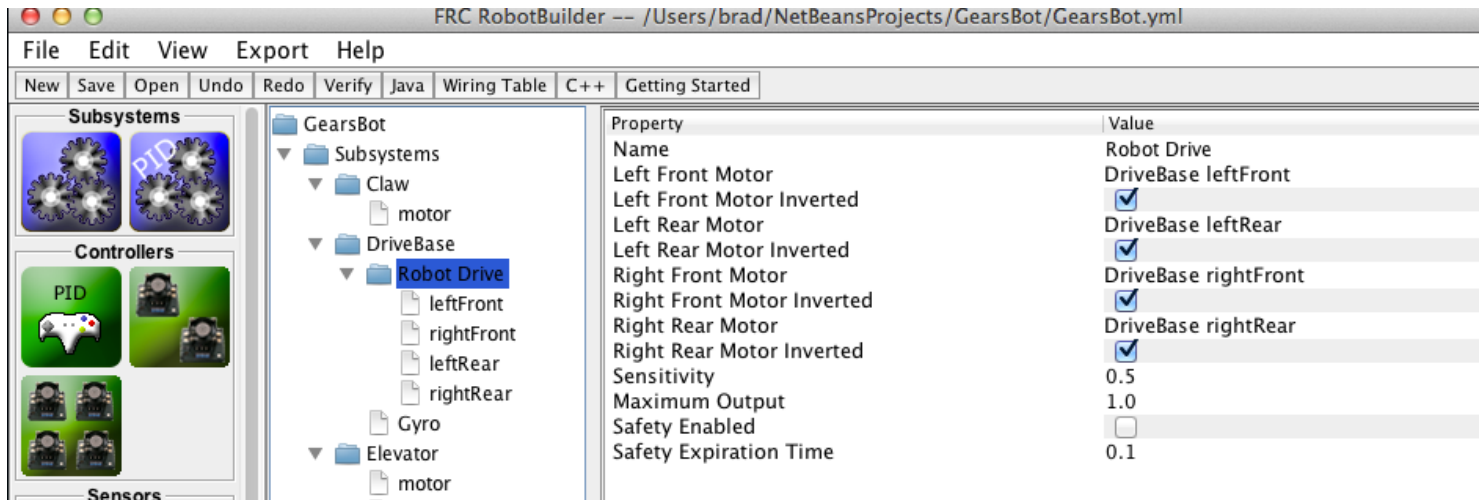
**Analog Module 1**

#	Sensor
1	DriveBase Gyro Input
4	Wrist pot Input
5	Elevator pot Input

The wiring file is automatically generated using the "Wiring Table" toolbar button based on the actuators and sensors added to each subsystem.

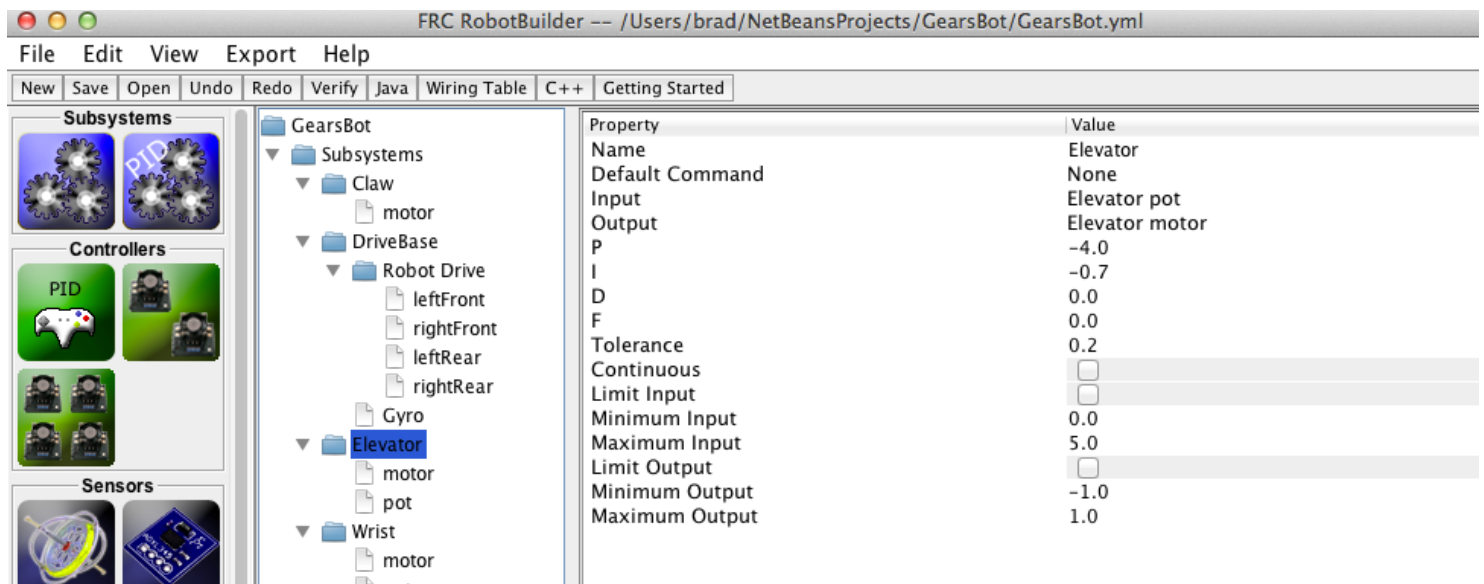
# Constructing the robot program

## RobotDrive object



All the motors are inverted based on the gearing and mounting of the motors. Adding gears reverses the direction of the motor for each gear added.

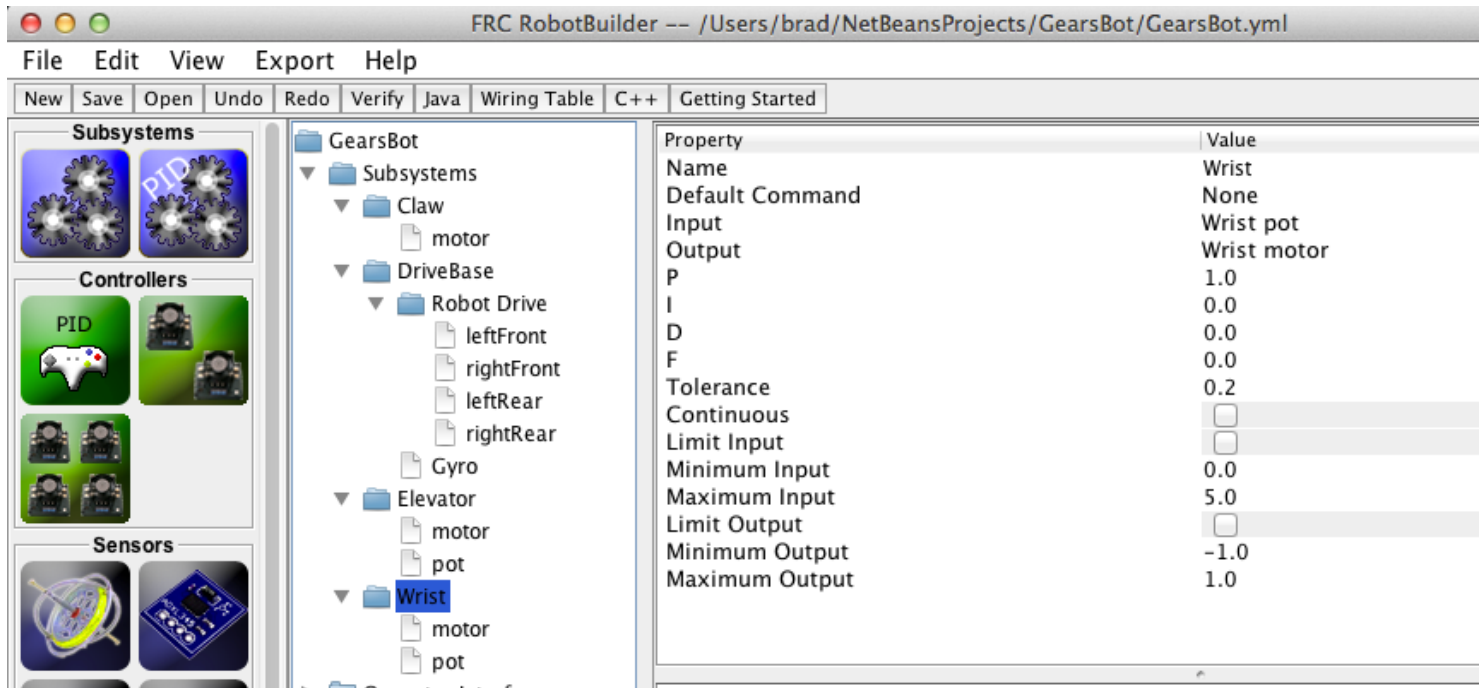
## Elevator



The elevator is controlled by a potentiometer and driven with a motor connected with a Victor speed controller. Since this is a PIDSubsystem, it has P, I, and D constants and a Tolerance value that determines when the elevator has reached it's target. The target is used by commands to know when they are finished for sequential operations.

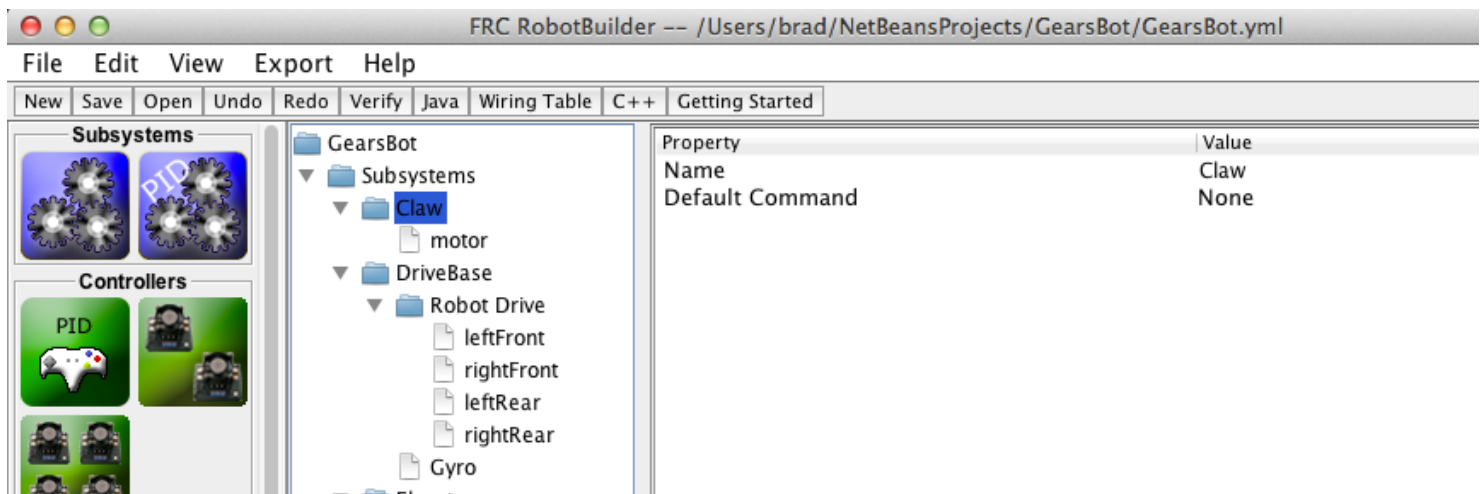
# Constructing the robot program

## Wrist



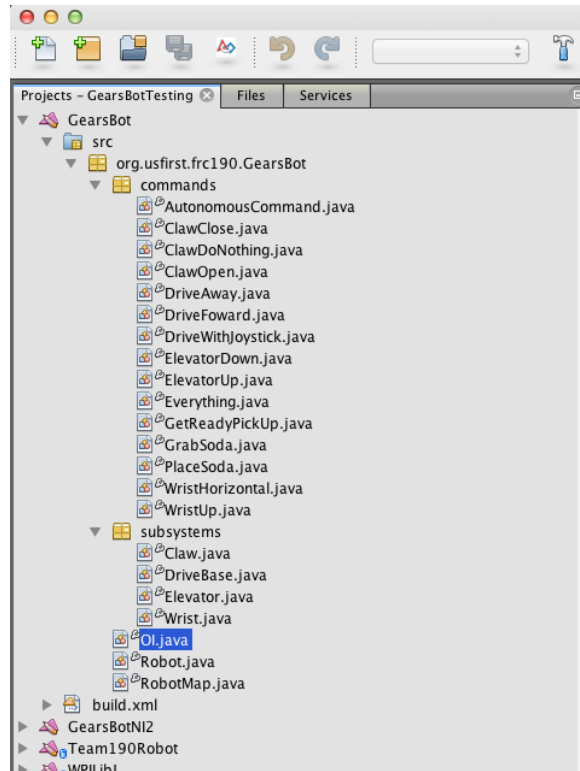
The wrist is a PIDSubsystem with a Victor speed controller motor and a potentiometer to measure the wrist angle. The P, I, and D values are used with the built-in PID controller and the tolerance is used to determine if the wrist has reached the desired angle.

## Gripper



# Constructing the robot program

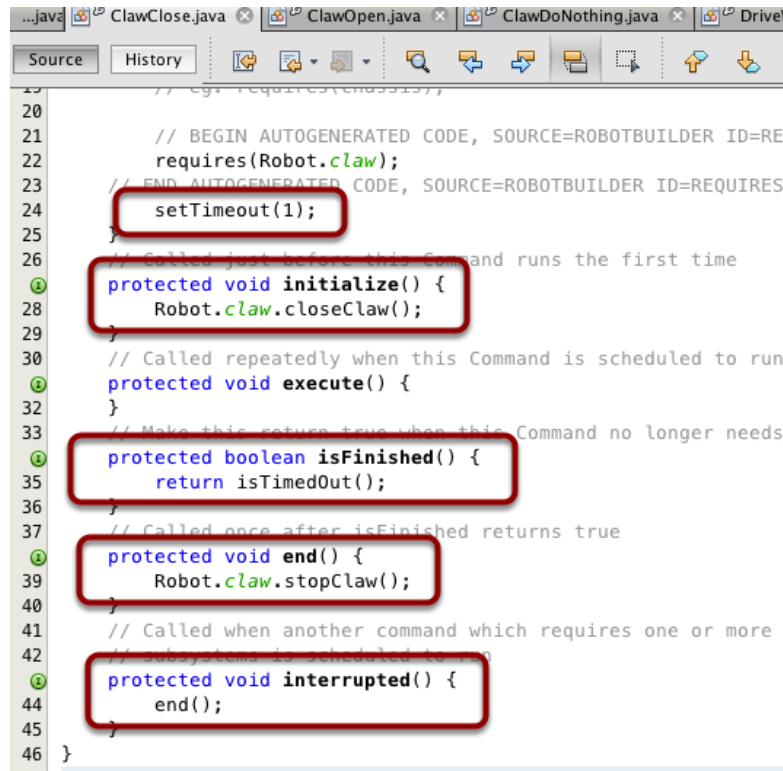
## Subsystems and commands



There are a number of commands that are used to implement this robots operation, but each one only requires writing a few lines of code to make them operate. In most cases the code sets a timeout for timed commands, or sets a PID target for PID controlled commands.

# Constructing the robot program

## Claw commands



```
...java ClawClose.java ClawOpen.java ClawDoNothing.java Drive
Source History
20 // eg. requires(chassis);
21 // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=RE
22 requires(Robot.claw);
23 // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
24 setTimeout(1);
25 }
26 // Called just before this Command runs the first time
27 protected void initialize() {
28     Robot.claw.closeClaw();
29 }
30 // Called repeatedly when this Command is scheduled to run
31 protected void execute() {
32 }
33 // Make this return true when this Command no longer needs
34 // to be executed because it just completed.
35 protected boolean isFinished() {
36     return isTimedOut();
37 }
38 // Called once after isFinished returns true
39 protected void end() {
40     Robot.claw.stopClaw();
41 }
42 // Called when another command which requires one or more
43 // subsystems is scheduled to run
44 protected void interrupted() {
45     end();
46 }
```

There are commands that open and close the claw. It has no sensor and is operated for a fixed time rather than from sensor feedback. It is generally better to have some sensor feedback to control the subsystem rather than relying on time, but this is simply a Vex motor and running it for a longer time than necessary won't hurt anything. For larger FRC motors, the motor could be easily damaged if it runs open-loop like this.

# Constructing the robot program

## DriveForward

```
1 package org.usfirst.frc190.GearsBot.commands;
2 import edu.wpi.first.wpilibj.command.Command;
3 import org.usfirst.frc190.GearsBot.Robot;
4
5 public class DriveForward extends Command {
6     public DriveForward() {
7         // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRE
8         requires(Robot.driveBase);
9         // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRE
10        setTimeout(1.0);
11
12        // Called just before this Command runs the first time
13        protected void initialize() {
14            Robot.driveBase.driveForward(0.5);
15        }
16
17        // Called repeatedly when this Command is scheduled to run
18        protected void execute() {
19        }
20
21        // Make this return true when this Command no longer needs to run
22        protected boolean isFinished() {
23            return isTimedOut();
24        }
25
26        // Called once after isFinished returns true
27        protected void end() {
28            Robot.driveBase.driveForward(0.0);
29        }
30
31        // Called when another command which requires one or more
32        // subsystems is scheduled to run
33        protected void interrupted() {
34            end();
35        }
36    }
37 }
```

This command has the robot driving forward for a fixed period of time, in this case about 0.5 seconds. The command starts the motors running, then uses a timeout to stop them after driving for a while. The end() method stops the robot from driving and the interrupted() method simply calls the end() method.

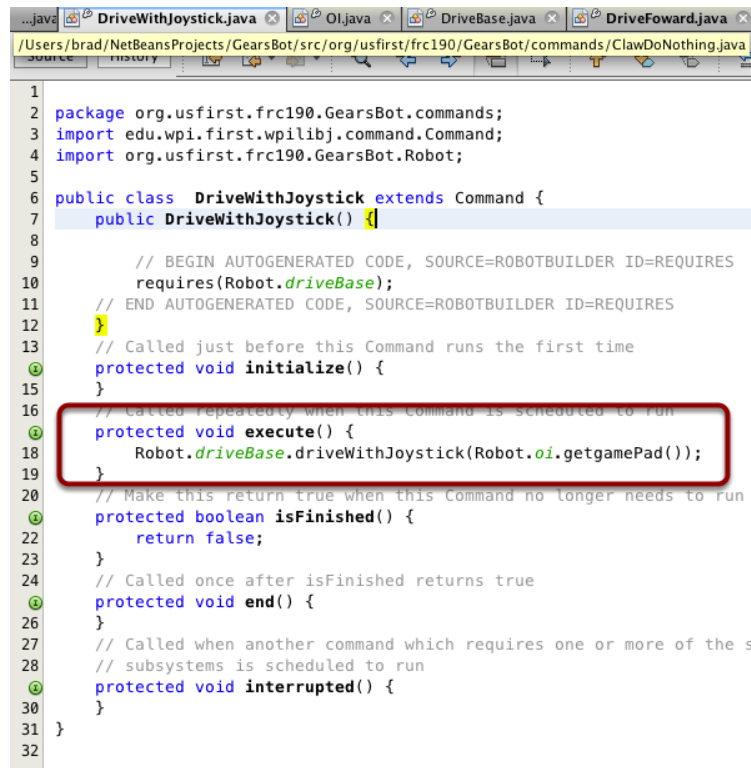
## Operator Interface



# Constructing the robot program

The operator interface in this robot is fairly simple, just a logitech gamepad with a few buttons to operate some commands.

## DriveWithJoystick



```
1 package org.usfirst.frc190.GearsBot.commands;
2 import edu.wpi.first.wpilibj.command.Command;
3 import org.usfirst.frc190.GearsBot.Robot;
4
5 public class DriveWithJoystick extends Command {
6     public DriveWithJoystick() {}
7
8     // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
9     requires(Robot.driveBase);
10    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
11
12    // Called just before this Command runs the first time
13    protected void initialize() {
14    }
15
16    // Called repeatedly when this Command is scheduled to run
17    protected void execute() {
18        Robot.driveBase.driveWithJoystick(Robot.oi.getgamePad());
19    }
20
21    // Make this return true when this Command no longer needs to run
22    protected boolean isFinished() {
23        return false;
24    }
25
26    // Called once after isFinished returns true
27    protected void end() {
28    }
29
30    // Called when another command which requires one or more of the s
31    // subsystems is scheduled to run
32    protected void interrupted() {
33    }
34 }
```

DriveWithJoystick calls the method on the drive base to drive the robot using the joystick that is created as part of the operator interface. This command never finishes because it's the default command for the drive base subsystem so the robot can drive using the joystick when it's not doing anything else.

# Constructing the robot program

## Drive base code for mecanum drive

```
public class DriveBase extends Subsystem {
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
    SpeedController leftFront = RobotMap.driveBaseleftFront;
    SpeedController rightFront = RobotMap.driveBaserightFront;
    SpeedController leftRear = RobotMap.driveBaseleftRear;
    SpeedController rightRear = RobotMap.driveBaserightRear;
    RobotDrive robotDrive = RobotMap.driveBaseRobotDrive;
    Gyro gyro = RobotMap.driveBaseGyro;
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS

    // Put methods for controlling this subsystem
    // here. Call these from Commands.
    public void initDefaultCommand() {
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
        setDefaultCommand(new DriveWithJoystick());
        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND

        // Set the default command for a subsystem here.
        //setDefaultCommand(new MySpecialCommand());
    }

    public void driveForward(double speed) {
        robotDrive.mecanumDrive_Cartesian(0, -speed, 0, 0);
    }

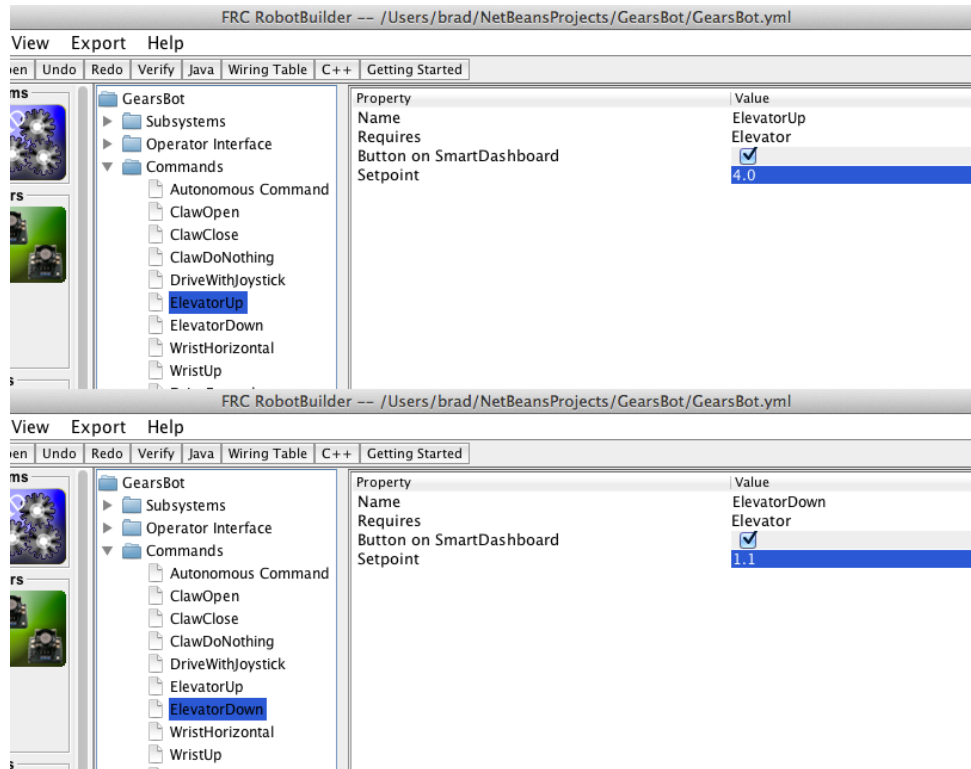
    public void driveWithJoystick(Joystick gamePad) {
        robotDrive.mecanumDrive_Cartesian(gamePad.getX(), gamePad.getY(), gamePad.getTwist(), 0);
    }
}
```

the DriveBase subsystem has a default command (DriveWithJoystick) that allows the user to drive the robot when no other command is using the drive base. It drives using the mecanum drive methods as part of the RobotDrive class. The speed is inverted for forward driving because the joystick has positive (forward) values when the stick is pulled backwards.



# Constructing the robot program

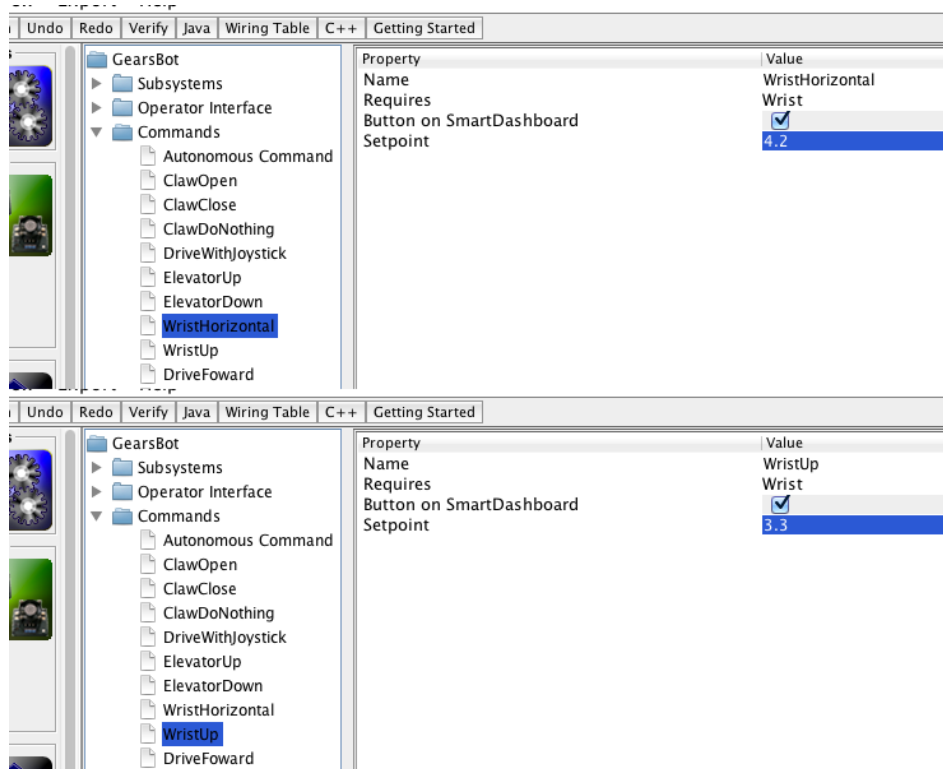
## Elevator Commands



Elevator commands move the elevator from the upper (drop off) position to the lower (pickup) positions. Both use the potentiometer for feedback.

# Constructing the robot program

## Wrist Commands



Wrist PIDSetpoint commands that cause the wrist to go to the horizontal and up (stowed) positions.