

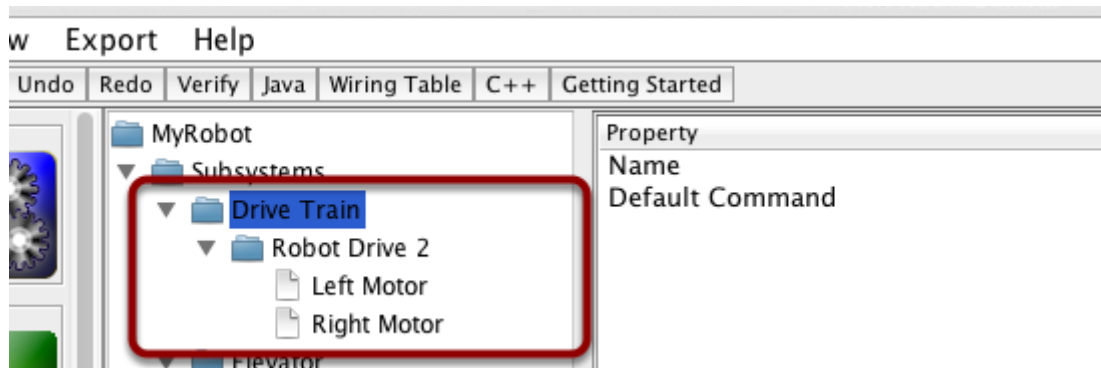
## Driving the robot with tank drive and joysticks

# Driving the robot with tank drive and joysticks

A common use case is to have a joystick that should drive some actuators that are part of a subsystem. The problem is that the joystick is created in the OI class and the motors to be controlled are in the subsystem. The idea is to create a command that, when scheduled, reads input from the joystick and calls a method that is created on the subsystem that drives the motors.

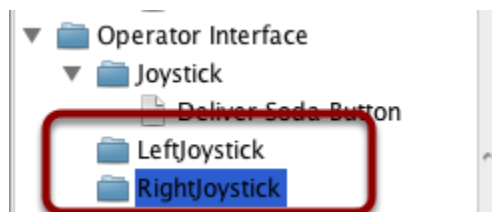
In this example a drive base subsystem is shown that is operated in tank drive using a pair of joysticks.

## Create a Drive Train subsystem



Create a subsystem called Drive Train. Its responsibility will be to handle the driving for the robot base. Inside the Drive Train is a Robot Drive object for a two motor drive robot (in this case). There is a left motor and right motor as part of the Robot Drive 2 class.

## Add the joysticks to the Operator Interface



Add two joysticks to the Operator Interface, one is the left stick and the other is the right stick. The y-axis on the two joysticks are used to drive the robots left and right sides.

Note: be sure to [export your program to C++](#) or [Java](#) before continuing to the next step.

# Driving the robot with tank drive and joysticks

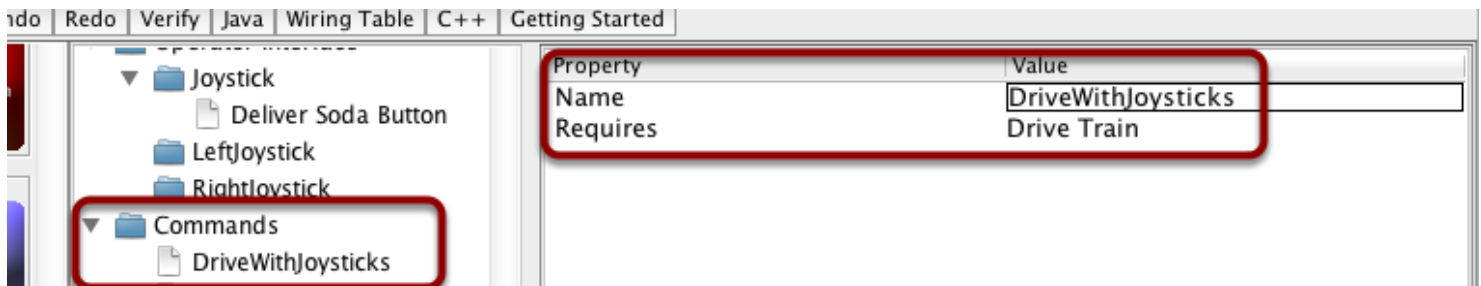
## Create a method to write the motors on the subsystem

```
1 // Generated with RobotBuilder version 3.0.1
2 package org.usfirst.frc190.MyRobot.subsystems;
3 import edu.wpi.first.wpilibj.*;
4 import edu.wpi.first.wpilibj.command.Subsystem;
5 import org.usfirst.frc190.MyRobot.RobotMap;
6
7 public class DriveTrain extends Subsystem {
8     // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
9     RobotDrive robotDrive2 = RobotMap.DRIVE_TRAIN_ROBOT_DRIVE_2;
10    Jaguar rightMotor = RobotMap.DRIVE_TRAIN_RIGHT_MOTOR;
11    Jaguar leftMotor = RobotMap.DRIVE_TRAIN_LEFT_MOTOR;
12    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
13
14    public void initDefaultCommand() {
15        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
16        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DEFAULT_COMMAND
17
18        // Set the default command for a subsystem here.
19        //setDefaultCommand(new MySpecialCommand());
20    }
21
22    public void takeJoystickInputs(Joystick left, Joystick right) {
23        robotDrive2.tankDrive(left, right);
24    }
25
26    public void stop() {
27        robotDrive2.drive(0, 0);
28    }
29 }
```

Create a method that takes the joystick inputs, in this case the the left and right driver joystick. The values are passed to the RobotDrive object that in turn does tank steering using the joystick values. Also create a method called stop() that stops the robot from driving, this might come in handy later.

*Note: the extra RobotBuilder comments have been removed to format the example for the documentation.*

## Create a command that reads the joystick values and calls the subsystem method



# Driving the robot with tank drive and joysticks

Create a command, in this case called DriveWithJoysticks. Its purpose will be to read the joystick values and send them to the Drive Base subsystem. Notice that this command Requires the Drive Train subsystem. This will cause it to stop running whenever anything else tries to use the Drive Train.

Note: be sure to [export your program to C++](#) or [Java](#) before continuing to the next step.

## Add the code for the command to do the actual driving

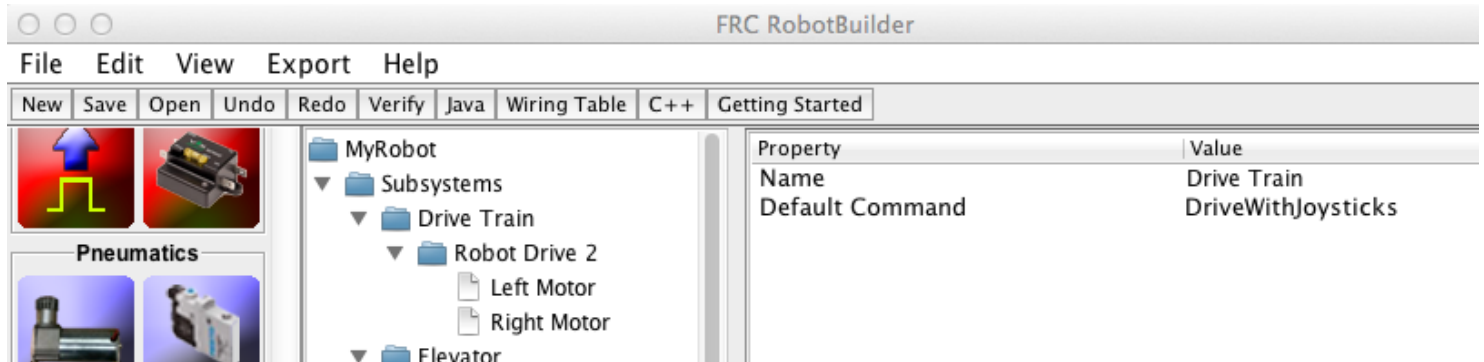
```
3 package org.usfirst.frc190.MyRobot.commands;
4
5 import edu.wpi.first.wpilibj.command.Command;
6 import org.usfirst.frc190.MyRobot.Robot;
7
8 public class DriveWithJoysticks extends Command {
9
10     public DriveWithJoysticks() {
11         // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
12         requires(Robot.driveTrain);
13         // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
14     }
15
16     protected void initialize() {
17     }
18
19     protected void execute() {
20         Robot.driveTrain.takeJoystickInputs(Robot.oi.getLeftJoystick(),
21                                             Robot.oi.getRightJoystick());
22     }
23
24     protected boolean isFinished() {
25         return false;
26     }
27
28     protected void end() {
29         Robot.driveTrain.stop();
30     }
31
32     protected void interrupted() {
33         end();
34     }
35 }
```

Add code to the execute method to do the actual driving. All that is needed is to get the Joystick objects for the left and right drive joysticks and pass them to the Drive Train subsystem. The subsystem just uses them for the tank steering method on its RobotDrive object. And we get tank steering.

We also filled in the end() and interrupted methods so that when this command is interrupted or stopped, the motors will be stopped as a safety precaution.

# Driving the robot with tank drive and joysticks

## Make the command the "default command" for the subsystem



The last step is to make the DriveWithJoysticks command be the "Default Command" for the Drive Train subsystem. This means that whenever no other command is using the Drive Train, the Joysticks will be in control. This is probably the desirable behavior. When the autonomous code is running, it will also require the drive train and interrupt the "DriveWithJoystick" command. When the autonomous code is finished, the DriveWithJoysticks command will restart automatically (because it is the default command), and the operators will be back in control. If you write any code that does teleop automatic driving, those commands should also "require" the DriveTrain so that they too will interrupt the DriveWithJoysticks command and have full control.

Note: be sure to [export your program to C++](#) or [Java](#) before continuing.