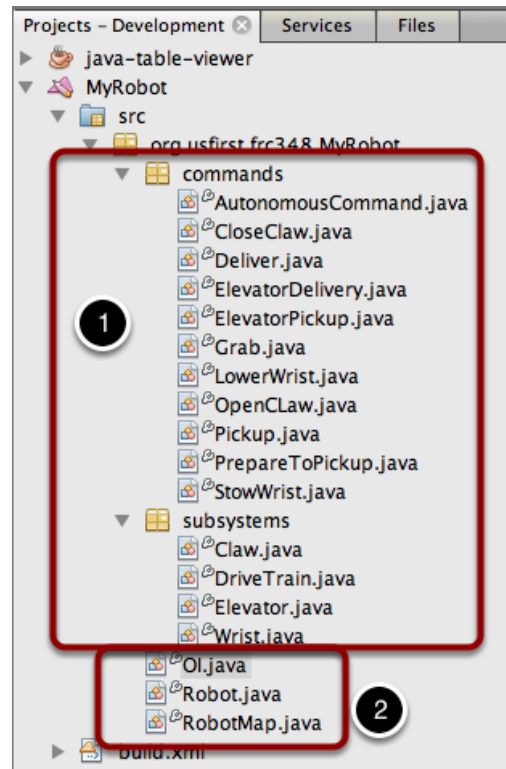


RobotBuilder created code

RobotBuilder created code

The layout of a RobotBuilder generated project



A RobotBuilder generated project consists of a package (in Java) or a folder (in C++) for Commands and another for Subsystems (1). Each command or subsystem object is stored under those containers (2). At the top level of the project you'll find the robot main program (Robot.java), the Operator Interface file (OI.java) and the RobotMap that contains the code to create all the subsystem components that were added to the robot description.

Autogenerated code

```
// BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
requires(Robot.claw);
// END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=REQUIRES
setTimeout(1):
```

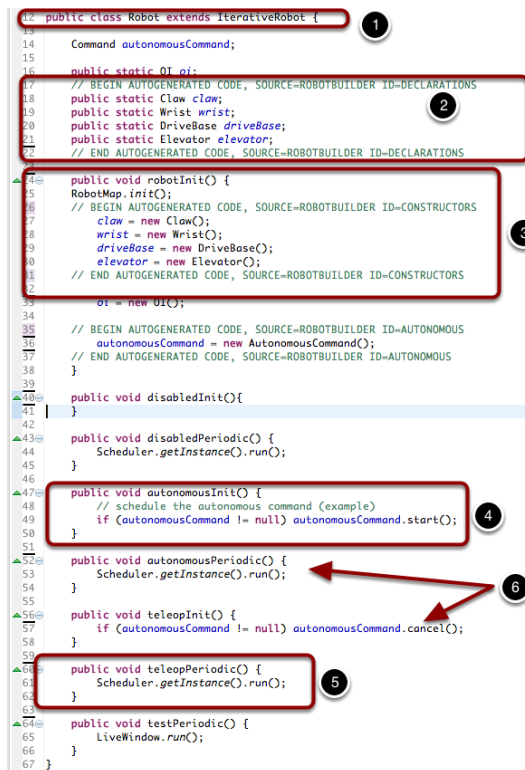
When the robot description is modified and code is re-exported RobotBuilder is designed to not modify any changes you made to the file, thus preserving your code. This makes RobotBuilder a

RobotBuilder created code

full-lifecycle tool. To know what code is OK to be modified by RobotBuilder, it generates sections that will potentially have to be rewritten delimited with some special comments. These comments are shown in the example above. Don't add any code within these comment blocks, it will be rewritten next time the project is exported from RobotBuilder.

If code inside one of these blocks must be modified, the comments can be removed, but this will prevent further updates from happening later. In the above example, if the `//BEGIN` and `//END` comments were removed, then later another required subsystem was added in RobotBuilder, it would not be generated on that next export.

Main robot program



```
12 public class Robot extends IterativeRobot {
13
14     Command autonomousCommand;
15
16
17     public static OI oi;
18     // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
19     public static Claw claw;
20     public static Wrist wrist;
21     public static DriveBase driveBase;
22     public static Elevator elevator;
23     // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS
24
25     public void robotInit() {
26         RobotMap.init();
27         // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
28         claw = new Claw();
29         wrist = new Wrist();
30         driveBase = new DriveBase();
31         elevator = new Elevator();
32         // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS
33
34         OI = new OI();
35
36         // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
37         autonomousCommand = new AutonomousCommand();
38         // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=AUTONOMOUS
39
40     public void disabledInit() {
41     }
42
43     public void disabledPeriodic() {
44         Scheduler.getInstance().run();
45     }
46
47     public void autonomousInit() {
48         // schedule the autonomous command (example)
49         if (autonomousCommand != null) autonomousCommand.start();
50     }
51
52     public void autonomousPeriodic() {
53         Scheduler.getInstance().run();
54     }
55
56     public void teleopInit() {
57         if (autonomousCommand != null) autonomousCommand.cancel();
58     }
59
60     public void teleopPeriodic() {
61         Scheduler.getInstance().run();
62     }
63
64     public void testPeriodic() {
65         LiveWindow.run();
66     }
67 }
```

The image shows a Java code file for a robot program. It is a public class named `Robot` that extends `IterativeRobot`. The code is annotated with six numbered circles (1-6) and red boxes highlighting specific sections. Circle 1 points to the class declaration. Circle 2 points to the static variable declarations for `Claw`, `Wrist`, `DriveBase`, and `Elevator`, which are enclosed in a `// BEGIN AUTOGENERATED CODE...` and `// END AUTOGENERATED CODE...` block. Circle 3 points to the `robotInit()` method, which contains constructor calls for the subsystems, also enclosed in a `// BEGIN AUTOGENERATED CODE...` and `// END AUTOGENERATED CODE...` block. Circle 4 points to the `autonomousInit()` method, which schedules the `autonomousCommand`. Circle 5 points to the `teleopPeriodic()` method, which calls `Scheduler.getInstance().run()`. Circle 6 points to the `autonomousPeriodic()` method, which also calls `Scheduler.getInstance().run()`. A red arrow points from circle 6 to circle 5, indicating that both methods call the same scheduler.

This is the main program generated by RobotBuilder. There are a number of parts to this program:

1. This class extends `IterativeRobot`. `IterativeRobot` will call your `autonomousPeriodic()` and `teleopPeriodic()` methods every 20ms (each time the driver station exchanges Joystick and other data with the robot).
2. Each of the subsystems is declared here These are public static variables so that they can be referenced from throughout your robot program by writing `Robot.<subsystem-name>.method()`, for example `Robot.elevator.setSetpoint(4)`.

RobotBuilder created code

3. The subsystems are instantiated in the robotInit() method that is called after the constructor runs for this class. It is important to create the subsystems after the constructor to avoid recursive loops. Also instance of the OI() class (for your operator interface) and the autonomous command are created here.
4. In the autonomousInit() method which is called every 20ms, make one scheduling pass. That is call the isFinished() and execute() methods of every command that is currently scheduled.
5. In the teleopPeriodic method which is called every 20ms, make one scheduling pass.
6. If there is an autonomous command provided in RobotBuilder robot properties, it is scheduled at the start of autonomous in the autonomousInit() method and canceled at the end of the autonomous period in teleopInit().

RobotMap - generation of all the actuator and sensor objects

```
public class RobotMap {  
    public static Jaguar DRIVE_TRAIN_LEFT_MOTOR;  
    public static Jaguar DRIVE_TRAIN_RIGHT_MOTOR;  
    public static RobotDrive DRIVE_TRAIN_ROBOT_DRIVE_2;  
    public static AnalogChannel DRIVE_TRAIN_ULTRASONIC;  
    public static Victor ELEVATOR_MOTOR;  
    public static AnalogChannel ELEVATOR_POTENTIOMETER;  
    public static AnalogChannel WRIST_POTENTIOMETER;  
    public static Victor WRIST_MOTOR;  
    public static Victor CLAW_MOTOR;  
  
    public static void init() {  
        DRIVE_TRAIN_LEFT_MOTOR = new Jaguar(1, 2);  
        LiveWindow.addActuator("Drive Train ", "Left 1 or", DRIVE_TRAIN_LEFT_MOTOR);  
  
        DRIVE_TRAIN_RIGHT_MOTOR = new Jaguar(1, 1);  
        LiveWindow.addActuator("Drive Train ", "Right Motor", DRIVE_TRAIN_RIGHT_MOTOR);  
  
        DRIVE_TRAIN_ROBOT_DRIVE_2 = new RobotDrive(DRIVE_TRAIN_LEFT_MOTOR, DRIVE_TRAIN_RIGHT_MOTOR);  
  
        DRIVE_TRAIN_ROBOT_DRIVE_2.setSafetyEnabled(false);  
        DRIVE_TRAIN_ROBOT_DRIVE_2.setExpiration(0.1);  
        DRIVE_TRAIN_ROBOT_DRIVE_2.setSensitivity(0.5);  
        DRIVE_TRAIN_ROBOT_DRIVE_2.setMaxOutput(1.0);  
  
        DRIVE_TRAIN_ULTRASONIC = new AnalogChannel(1, 3);  
  
        ELEVATOR_MOTOR = new Victor(1, 6);  
        LiveWindow.addActuator("Elevator ", "Motor", ELEVATOR_MOTOR);  
  
        ELEVATOR_POTENTIOMETER = new AnalogChannel(1, 4);  
        LiveWindow.addSensor("Elevator ", "Potentiometer", ELEVATOR_POTENTIOMETER);  
  
        WRIST_POTENTIOMETER = new AnalogChannel(1, 2);  
        LiveWindow.addSensor("Wrist ", "Potentiometer", WRIST_POTENTIOMETER);  
  
        WRIST_MOTOR = new Victor(1, 5);  
        LiveWindow.addActuator("Wrist ", "Motor", WRIST_MOTOR);  
  
        CLAW_MOTOR = new Victor(1, 7);  
        LiveWindow.addActuator("Claw ", "Motor", CLAW_MOTOR);  
    }  
}
```

The RobotMap is a mapping from the ports sensors and actuators are wired into to a variable name. This provides flexibility changing wiring, makes checking the wiring easier and significantly reduces the number of magic numbers floating around. All the definitions of sensors and motors from the robot description are generated here.

Notice that each sensor and actuator is added to the LiveWindow class (1) so that they can be automatically displayed when the SmartDashboard is set to LiveWindow mode. Also any

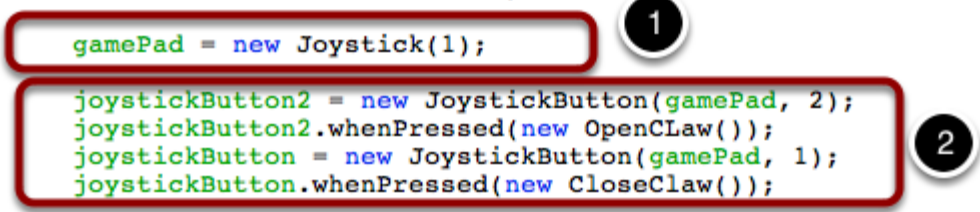
RobotBuilder created code

properties for the particular sensor or actuator is set here to reflect the settings made in the robot description. (2)

Each of the references for the objects are declared and instantiated here (3, 1), but they are copied into every subsystem to make it easy and clean to write code that uses them.

OI class - the Operator Interface

```
public class OI {  
    // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS  
    public JoystickButton joystickButton;  
    public JoystickButton joystickButton2;  
    public Joystick gamePad;  
    // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=DECLARATIONS  
  
    public OI() {  
        // BEGIN AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS  
        gamePad = new Joystick(1);  
        joystickButton2 = new JoystickButton(gamePad, 2);  
        joystickButton2.whenPressed(new OpenClaw());  
        joystickButton = new JoystickButton(gamePad, 1);  
        joystickButton.whenPressed(new CloseClaw());  
        // END AUTOGENERATED CODE, SOURCE=ROBOTBUILDER ID=CONSTRUCTORS  
    }  
}
```



The code for all the operator interface components is generated here (1). In addition the code to link the OI buttons to commands that should run is also generated here (2).