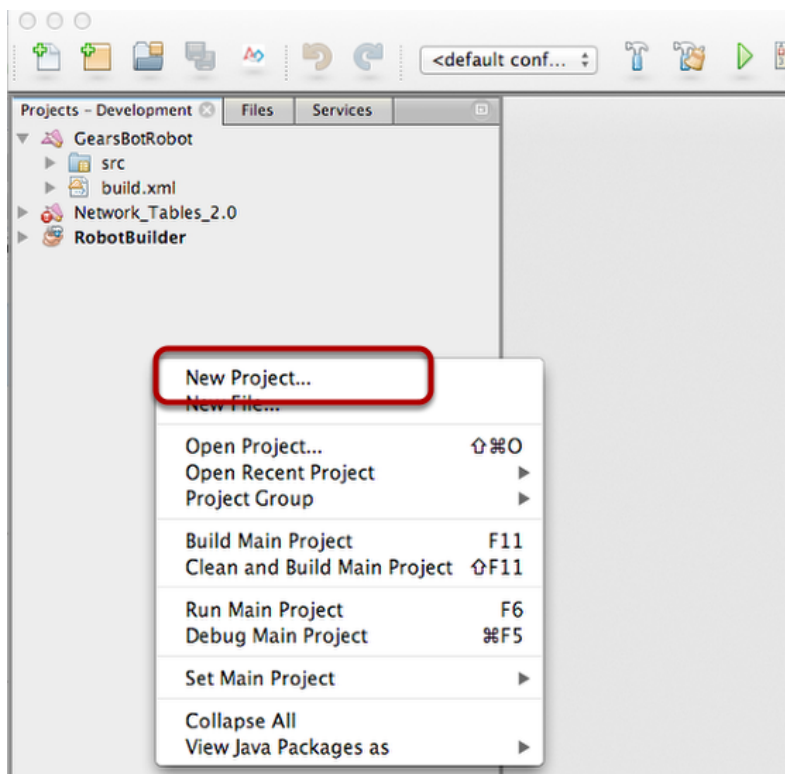


The "Hello world" of FRC robot programming

The "Hello world" of FRC robot programming

Here's how to create the shortest possible robot program that actually does something useful. In this case, it provides tank steering in teleop mode and drives a few feet and stops in autonomous mode. This program uses the SimpleRobotTemplate which lets you write very simple programs very easily. For larger programs we recommend using the CommandBasedRobot template and RobotBuilder.

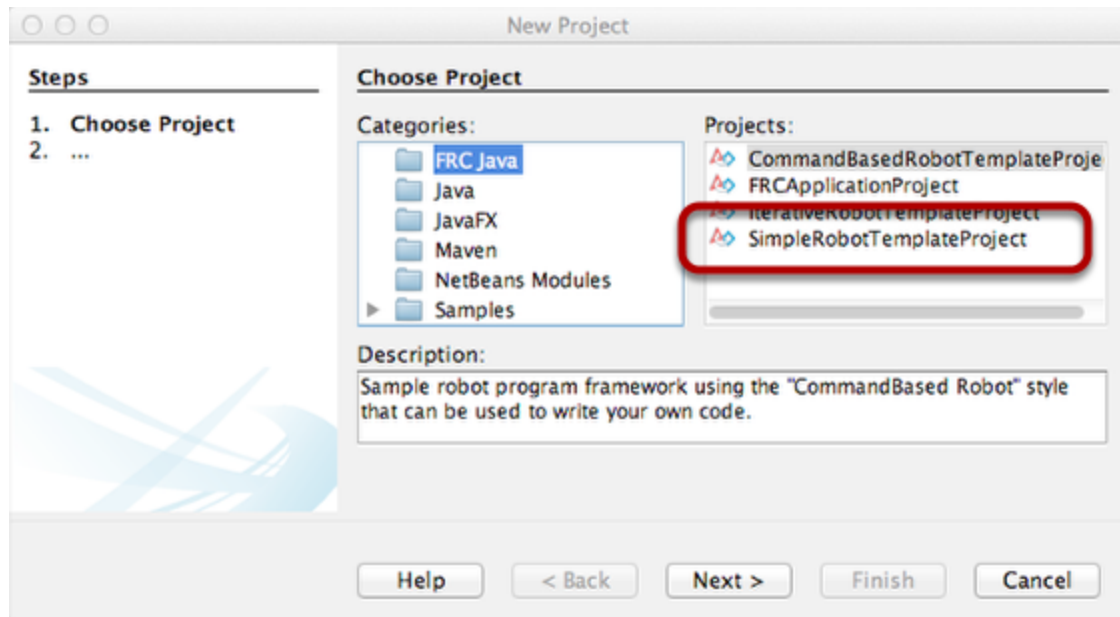
Create the project



Start NetBeans running. In the left pane labeled "Projects" right-click and select "New Project..."

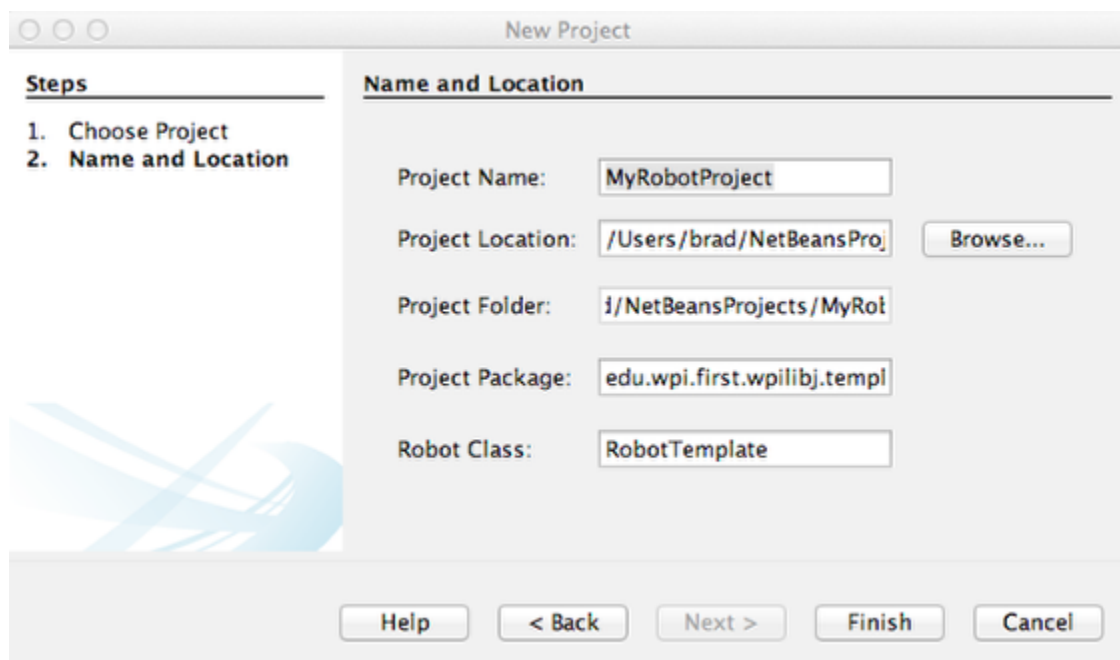
The "Hello world" of FRC robot programming

Select the project template



There a number of project types that you can use to get started with your robot project. The simplest one to use is SimpleRobotTemplateProject. Select this option and click "Next>".

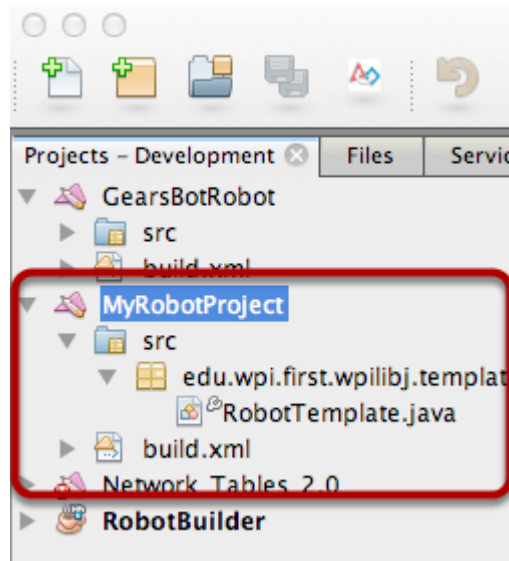
Fill in the project information



The "Hello world" of FRC robot programming

Fill out the "Name and Location" form. The Project Name will be the name you see in NetBeans for your project. The Robot Class will be the name of the class that is created that will be a subclass of SimpleRobot. You can also set the Project Location (directory) the name of the Project Folder and the Java package that is used for your classes. For testing it's OK to just take all the defaults. Then click "Finish".

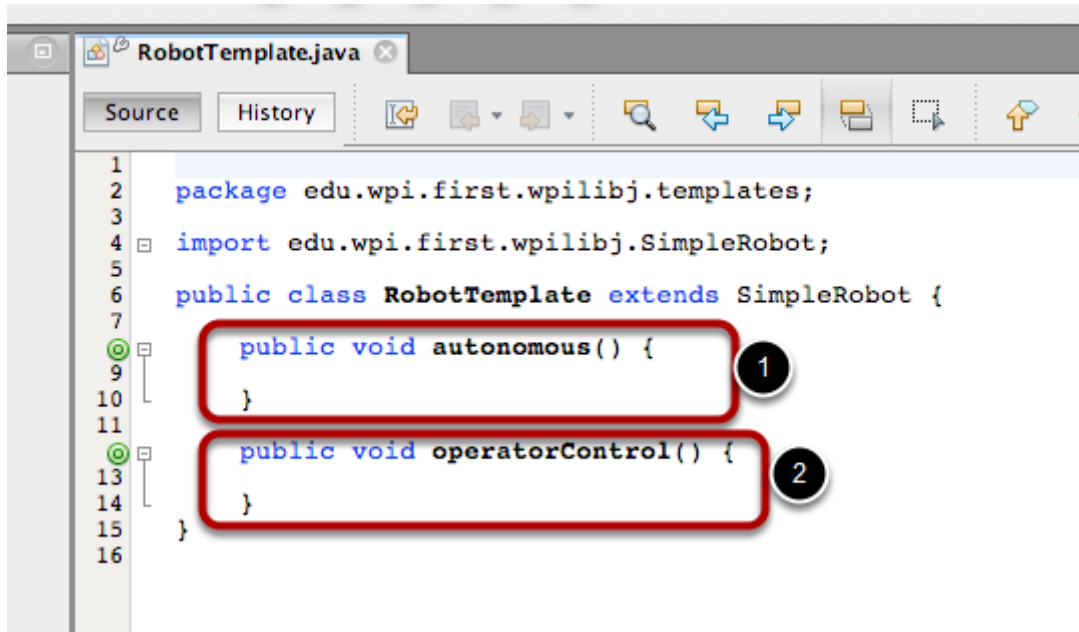
The project is created in the Projects window in NetBeans



Notice that the project is now created (MyRobotProject) with a "src" folder that contains the the class name specified in the previous step in the packa listed in the previous step. You can double-click on the source file "RobotTemplate.java" in this case to see the default code.

The "Hello world" of FRC robot programming

Reviewing the generated source file



Look at the generated source file for your project (the comments are removed from this example to make it better fit on the screen). Notice that there are two methods as part of the main class.

1. The `autonomous()` method - this is where you place all the code that you would like to have the robot run while it is in autonomous mode. The code should run until it is finished, but be sure to exit the method before the time runs out for the autonomous period of the match. If not, your teleop code will be delayed until the autonomous method returns.
2. The `operatorControl()` method - the code in this method runs when the robot is placed into teleop mode. Typically this code just loops reading sensor and controls values and drives actuators until the end of the operator control period.

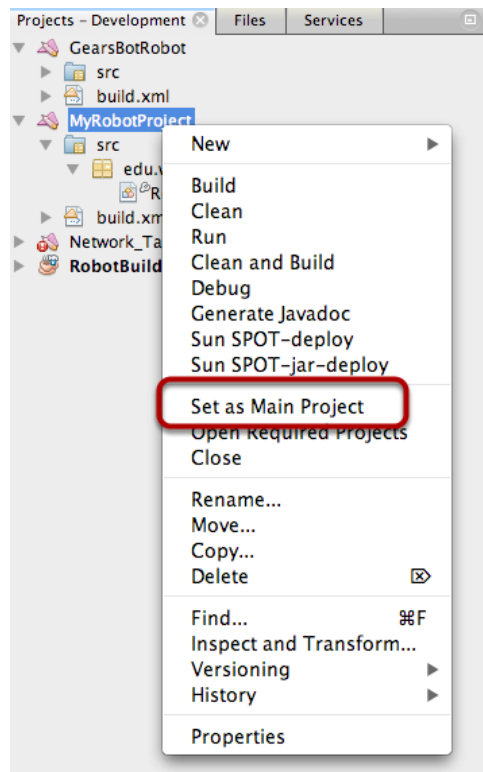
Setting the Main Project



It's a good idea to set the project that you are currently working on as the "Main Project". This is the one that will automatically be deployed to the robot and run when you press the green "Play" button.

The "Hello world" of FRC robot programming

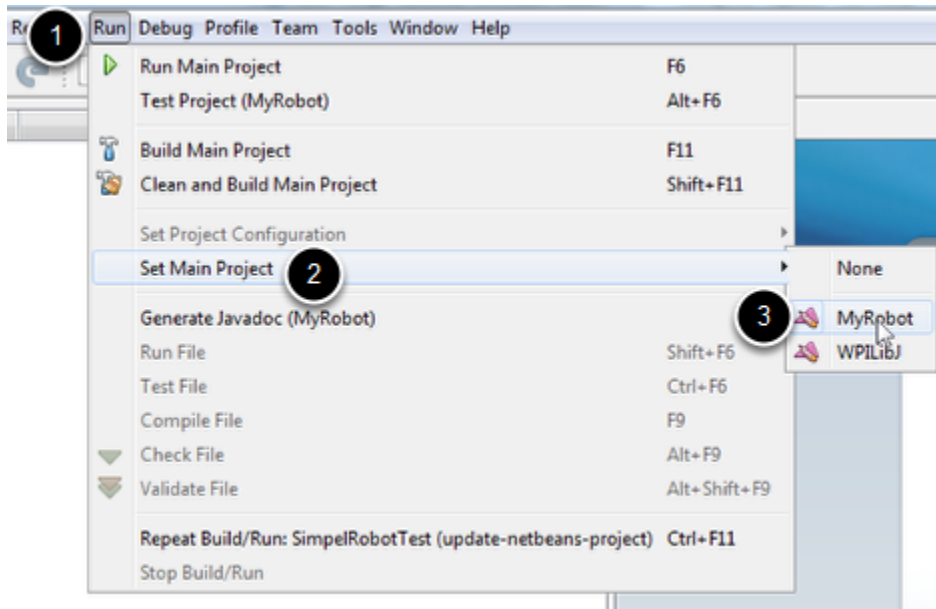
Setting the Main Project (Netbeans 7.1 and earlier)



To set the Main Project in Netbeans 7.1 or earlier, right-click on the project name and select "Set as Main Project". The current main project will be shown in the list in bold type.

The "Hello world" of FRC robot programming

Setting the Main Project (Netbeans 7.2)



To set the Main Project in Netbeans 7.2, click on the Run menu, hover over "Set Main Project" then click on the desired project.

Create the RobotDrive and Joystick objects

```
1 package edu.wpi.first.wpilibj.templates;
2
3
4 import edu.wpi.first.wpilibj.Joystick;
5 import edu.wpi.first.wpilibj.RobotDrive;
6 import edu.wpi.first.wpilibj.SimpleRobot;
7 import edu.wpi.first.wpilibj.Timer;
8
9 public class RobotTemplate extends SimpleRobot {
10
11     RobotDrive chassis = new RobotDrive(1, 2);
12     Joystick leftStick = new Joystick(1);
13     Joystick rightStick = new Joystick(2);
14 }
```

To use components on the robot such as motors, joysticks and sensors you must create objects for each of them. In this case we are using two Joysticks and a RobotDrive object that handles the 2 Jaguar controlled motors in our robot base.

1. To use these objects you need to add import statements to the start of your program that tells Java that you are going to use those objects and the classes are defined as part of WPILib. A shortcut to adding these import declarations is to start typing the name of

The "Hello world" of FRC robot programming

the class where you are using it (step 2) and before it's complete, type ctrl-space. This will complete the name automatically and add the import declaration if necessary.

2. Add the declarations to create the RobotDrive object with the 2 Jaguar speed controllers connected to PWM ports 1 and 2 on the first digital module (If your robot has 4 motor controllers or the motor controllers are connected to different ports, make sure to change this line accordingly. The constructors are ordered left motor(s) then right motor(s).). Also the two joysticks connected to USB channels 1 and 2. You can reorder the joysticks in the driver station when they are plugged in.

Fill in the autonomous part of the program

```
32 public void autonomous() {  
33     chassis.setSafetyEnabled(false);  
34     chassis.drive(-0.5, 0.0);  
35     Timer.delay(2.0);  
36     chassis.drive(0.0, 0.0);  
37 }
```

The sample autonomous program here drives the program drives the robot at half speed (-0.5) and a turn rate of (0.0). A negative speed is used to make the robot drive forward because the joysticks provided in the Kit of Parts (and most other HID joysticks and gamepads) return a negative value when pushed forwards. Then the program delays for 2.0 seconds while the robot continues to drive at half speed. After the delay tell the RobotDrive object to stop (drive 0.0 speed forward).

The first line of the method disables motor safety for the autonomous program. Motor safety is a mechanism built into the RobotDrive object that will turn off the motors if the program doesn't continuously update the motor speed. In this case, the speed is updated once, then there is a 2 second delay before it's updated again. The default setting for motor safety is to require an update every 100 ms. By turning off motor safety, it will prevent the motors from turning off after the first 0.1 seconds.

Fill in the teleop part of the program

```
21  
22 public void operatorControl() {  
23     chassis.setSafetyEnabled(true);  
24     while (isOperatorControl() && isEnabled()) {  
25         chassis.tankDrive(leftStick, rightStick);  
26         Timer.delay(0.01);  
27     }  
28 }
```

The "Hello world" of FRC robot programming

The teleop part of program turns motor safety back on. This will cause the robot to stop driving if the program were to stop running for any reason since the code would stop updating the motor speeds. Then it loops with the RobotDrive object providing tank steering with the two joysticks. The loop continues until the teleop period ends.

Inverting Motors

```
.  
//Inverts rear or only motor on left side  
chassis.setInvertedMotor(RobotDrive.MotorType.kRearLeft, true);  
//If using four motors, invert front motor as well.  
chassis.setInvertedMotor(RobotDrive.MotorType.kFrontLeft, true);
```

Depending on the wiring and construction of your robot, it is possible that you will need to invert the direction of one or motors in your code in order to have all motors spinning the correct direction. If pushing the joystick directly away from you results in anything other than the robot driving forward, one or more motors needs to be inverted. If you have 2 motors in the Robot Drive, invert the side of the robot that moves in the wrong direction. Note that the Robot Drive object refers to the single motor in a 2 motor drive as the rear motor.

If you have 4 motors in your Robot Drive and one side drives the wrong way, invert both motors on that side. If you have 4 motors and one side of the drive appears to not move at all when commanded the motors may be fighting each other, try inverting one of the two motors and observing if that side of the drive now moves when commanded.