

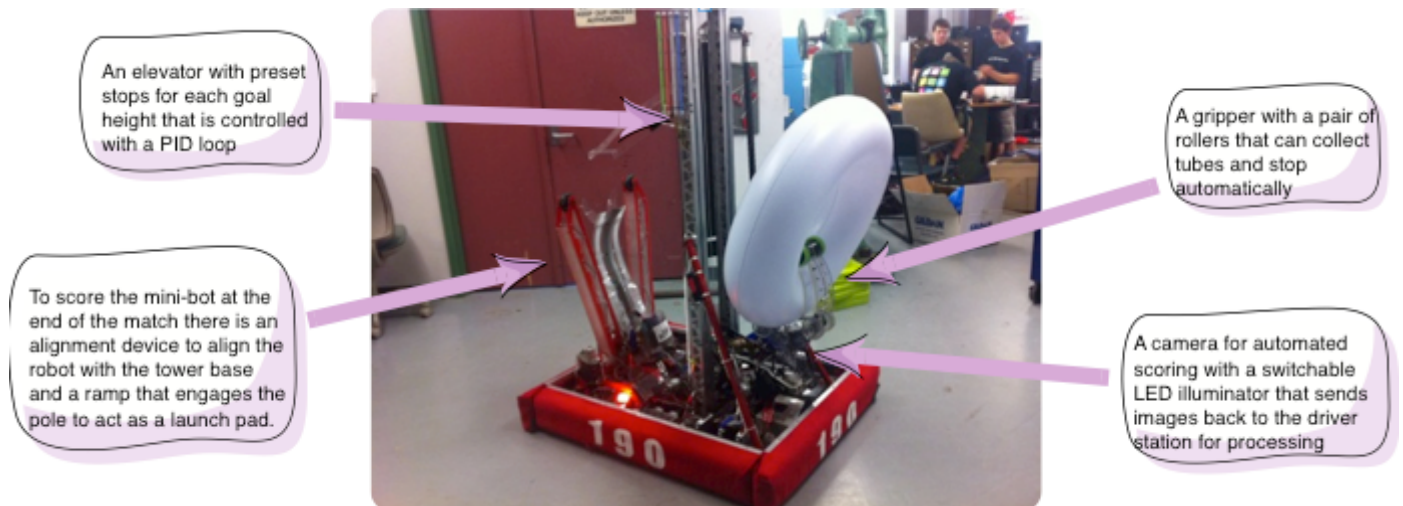
## Overview of RobotBuilder

# Overview of RobotBuilder

Creating a program with RobotBuilder is a very straight forward procedure by following a few steps that are the same for any robot. This lesson describes the steps that you can follow. You can find more details about each of these steps in subsequent sections of the document.

In addition to the text documentation provided here, a series of [videos about RobotBuilder and many other FRC Robotics Engineering topics](#) is also available.

## Divide the robot into subsystems

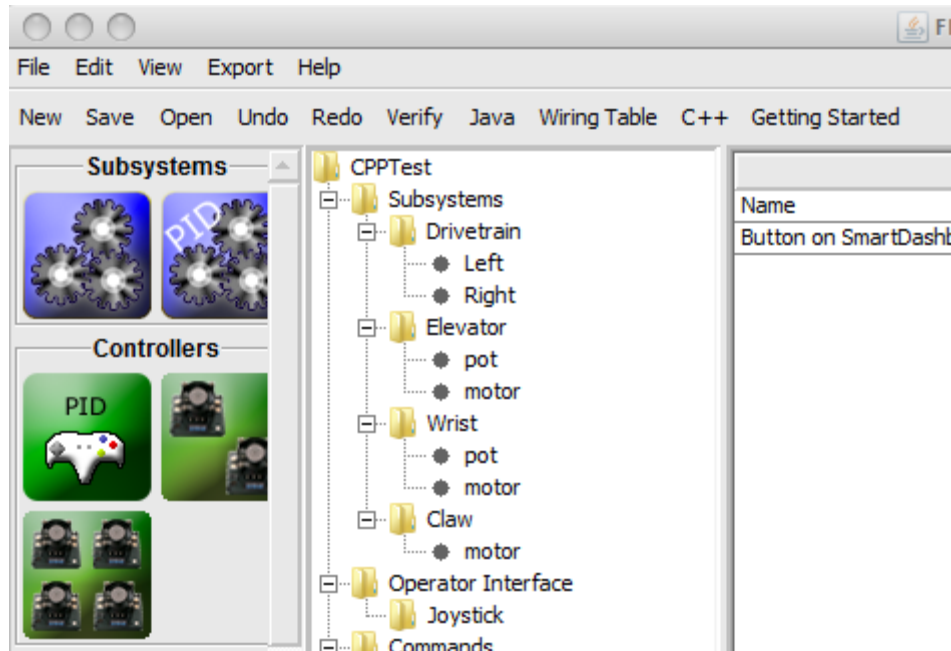


Your robot is naturally made up of a number of smaller systems like the drive trains, arms, shooters, collectors, manipulators, wrist joints, etc. You should look at the design of your robot and break it up into smaller, separately operated subsystems. In this particular example there is an elevator, a minibot alignment device, a gripper, and a camera system. In addition one might include the drive base. Each of these parts of the robot are separately controlled and make good candidates for subsystems.

For more information see: [Creating a Subsystem](#)

# Overview of RobotBuilder

## Add each subsystem to the RobotBuilder project



Each subsystem will be added to the "Subsystems" folder in the RobotBuilder and given a meaningful name. For each of the subsystems there are several attributes that get filled in to specify more information about the subsystems. In addition there are two types of subsystems that you might want to create:

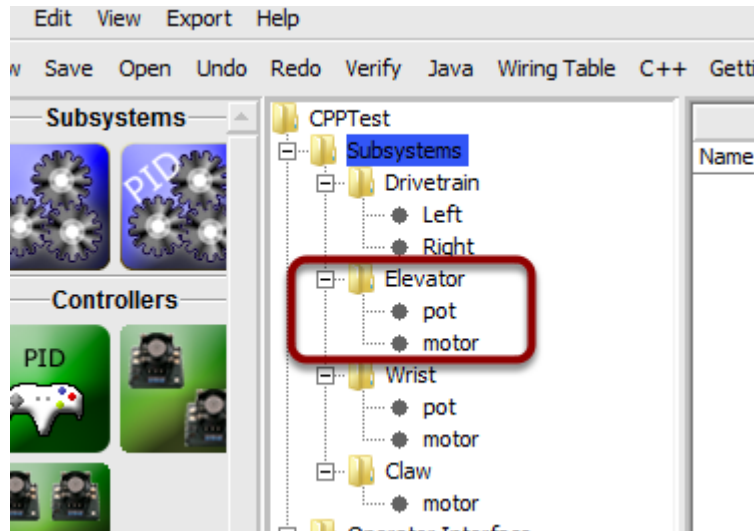
1. **PIDSubsystems** - often it is desirable to control a subsystems operation with a PID controller. This is code in your program that makes the subsystem element, for example arm angle, more quickly to a desired position then stop when reaching it. PIDSubsystems have the PID Controller code built-in and are often more convenient then adding it yourself. PIDSubsystems have a sensor that determines when the device has reached the target position and an actuator (speed controller) that is driven to the setpoint.
2. **Regular subsystem** - these subsystems don't have an integrated PID controller and are used for subsystems without PID control for feedback or for subsystems requiring more complex control than can be handled with the default embedded PID controller.

As you look through more of this documentation the differences between the subsystem types will become more apparent.

For more information see: [Creating a subsystem](#), [Writing Java code for a subsystem](#) and [Writing C++ code for a subsystem](#)

# Overview of RobotBuilder

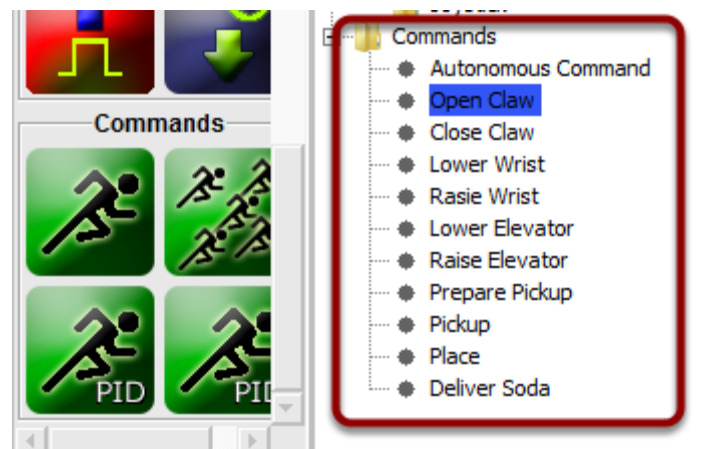
## Add components to each of the subsystems



Each subsystem consists of a number of actuators, sensors and controllers that it uses to perform its operations. These sensors and actuators are added to the subsystem with which they are associated. Each of the sensors and actuators comes from the RobotBuilder palette and is dragged to the appropriate subsystem. For each, there are usually other properties that must be set such as port numbers and other parameters specific to the component.

In this example there is an Elevator subsystem that uses a motor and a potentiometer (motor and pot) that have been dragged to the Elevator subsystem.

## Add commands to describe the goals for each subsystem



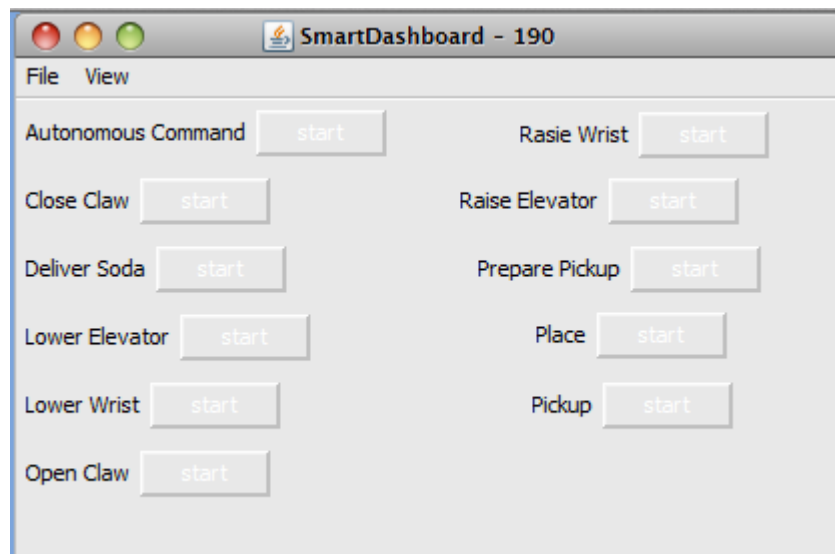
# Overview of RobotBuilder

Commands are distinct goals that the robot will perform. These commands are added by dragging the command under the "Commands" folder. When creating a command, there are 3 primary choices (shown on the palette on the left of the picture):

- Normal commands - these are the most flexible command, you have to write all of the code to perform the desired actions necessary to accomplish the goal.
- Command groups - these commands are a combination of other commands running both in a sequential order and in parallel. Use these to build up more complicated actions after you have a number of basic commands implemented.
- Setpoint commands - setpoint commands move a PID Subsystem to a fixed setpoint, or the desired location.

For more information see: [Creating a command](#), [Writing the code for a command in Java](#) and [Writing the code for a command in C++](#)

## Test each command individually by starting it from the SmartDashboard



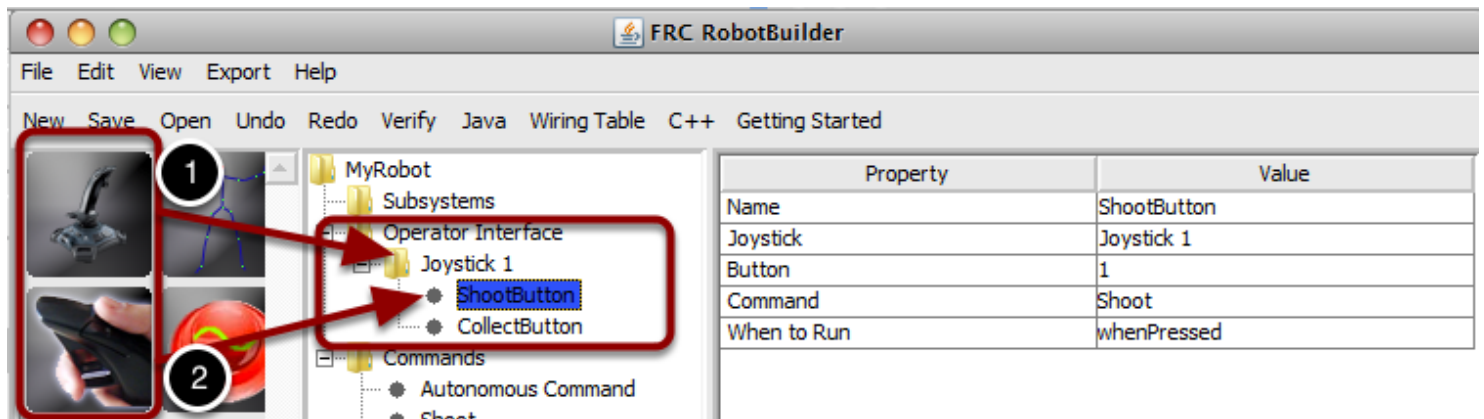
Each command can be run from the SmartDashboard. This is useful for testing commands before you add them to the operator interface or to a command group. As long as you leave the "Button on SmartDashboard" property checked, a button will be created on the SmartDashboard. When you press the start button, the command will run and you can check that it performs the desired action.

# Overview of RobotBuilder

By creating buttons, each command can be tested individually. If all the commands work individually, you can be pretty sure that the robot will work as a whole.

For more information see: [Adding a button to SmartDashboard to run a command](#)

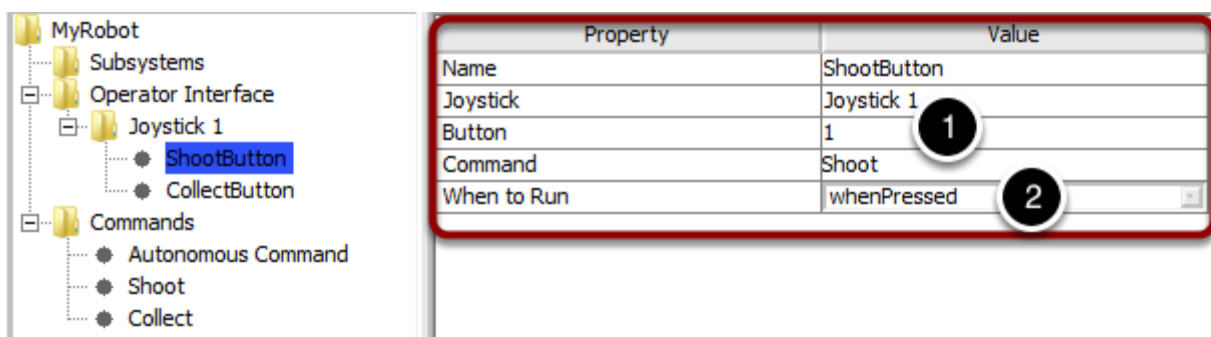
## Add Operator Interface components



The operator interface consists of joysticks, gamepads and other HID input devices. You can add operator interface components (joysticks, joystick buttons) to your program in RobotBuilder. It will automatically generate code that will initialize all of the components and allow them to be connected to commands.

The operator interface components are dragged from the palette to the "Operator Interface" folder in the RobotBuilder program. First (1) add Joysticks to the program then put buttons under the associated joysticks (2) and give them meaningful names, like ShootButton.

## Connect the commands to the Operator Interface



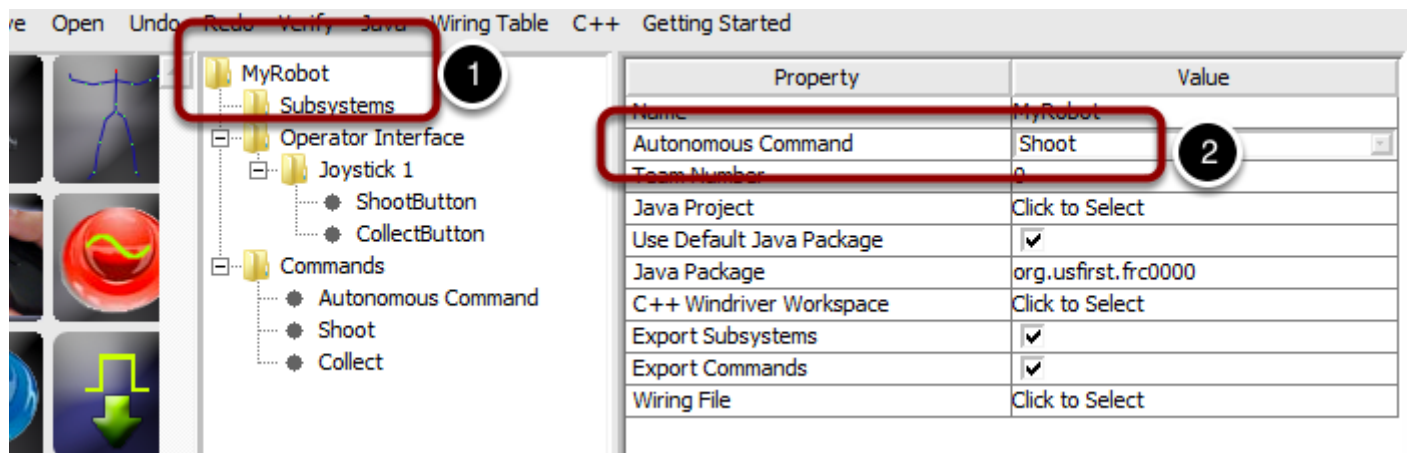
# Overview of RobotBuilder

Commands can be associated with buttons so that when a button is pressed the command is scheduled. This should, for the most part, handle most of the tele-operated part of your robot program.

This is simply done by (1) adding the command to the JoystickButton object in the RobotBuilder program, then (2) setting the condition in which the command is scheduled.

For more information see: [Connecting the operator interface to a command](#)

## Develop one or more Autonomous commands



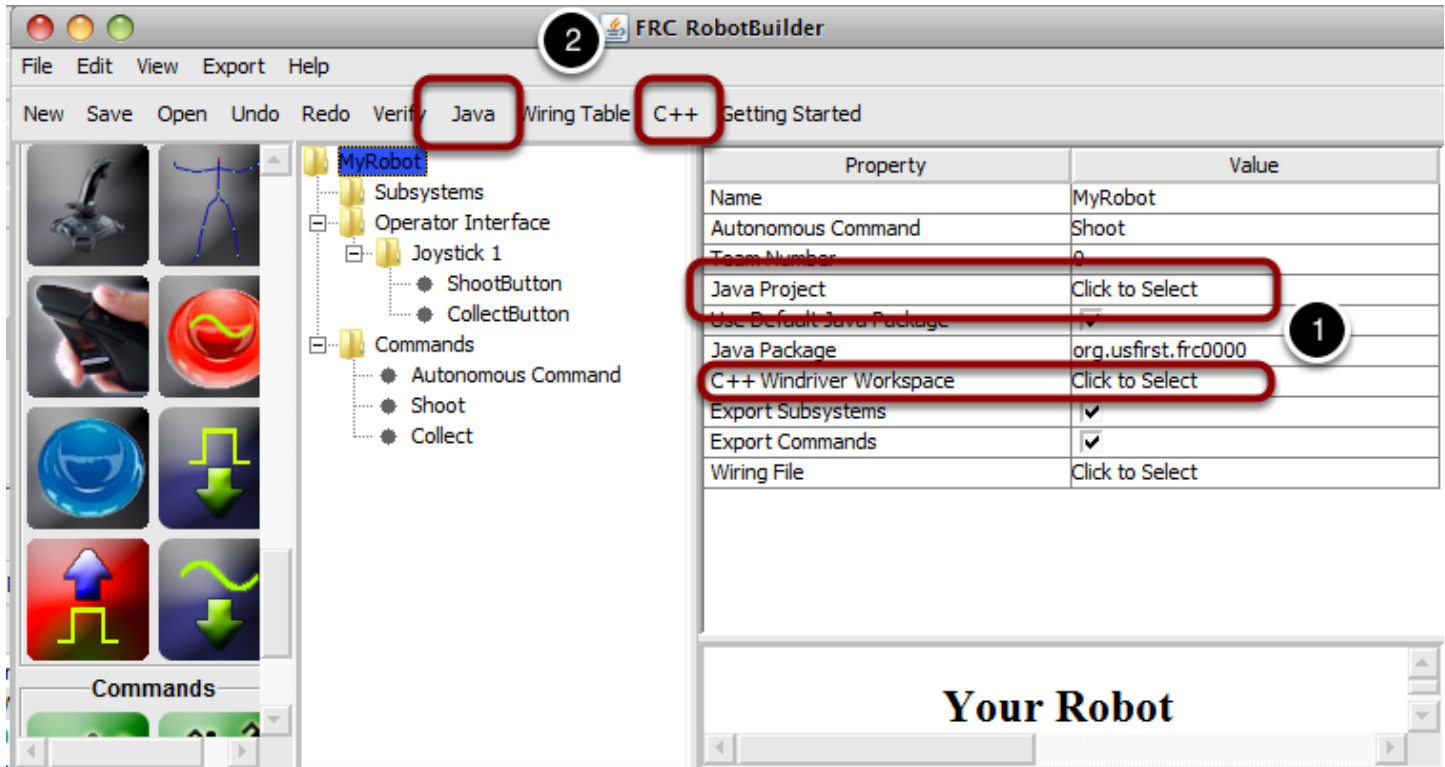
Commands make it simple to develop autonomous programs. You simply specify which command should run when the robot enters the autonomous period and it will automatically be scheduled. If you have tested commands as discussed above, this should simply be a matter of choosing which command should run.

Select the robot at the root of the RobotBuilder project, then edit the Autonomous Command property to choose the command to run. It's that simple!

For more information see: [Setting the default autonomous command](#)

# Overview of RobotBuilder

## Generating code for the program



At any point in the process outlined above you can have RobotBuilder generate a C++ or Java program that will represent the project you have created. This is done by specifying the location of the project in the project properties (1), then clicking the appropriate toolbar button to generate the code.

For more information see: [NEEDS UPDATE](#)