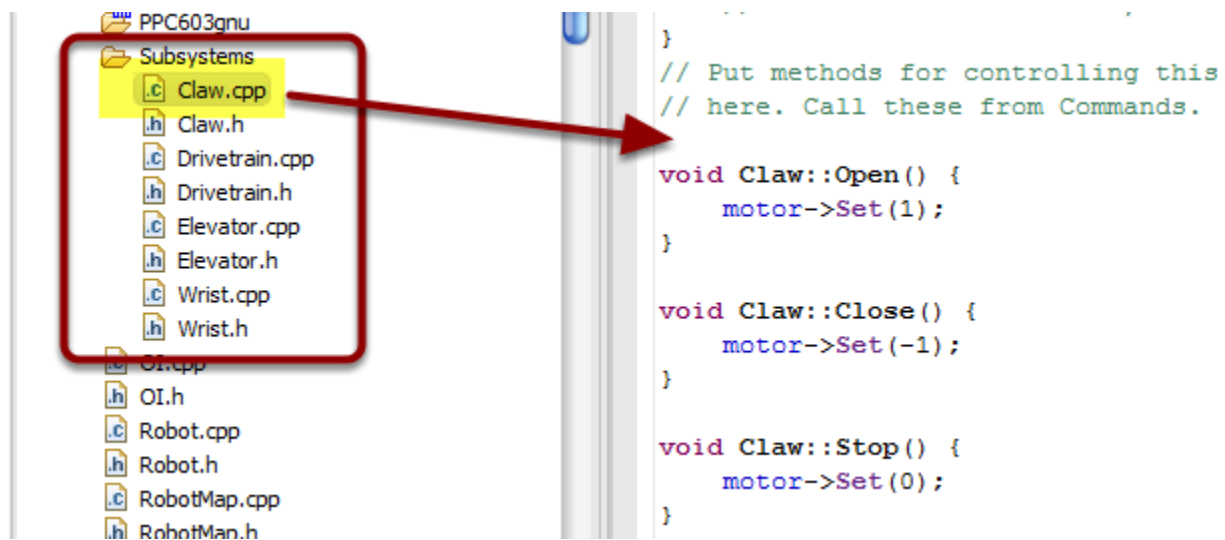


## How to write an easy to test robot program

# How to write an easy to test robot program

The command based programming model is designed to simplify creating very easy to write and especially easy to test robot programs. All the pieces come together when it's time to see how your program works so that pieces developed by multiple programmers can be integrated and tested with the main robot program.

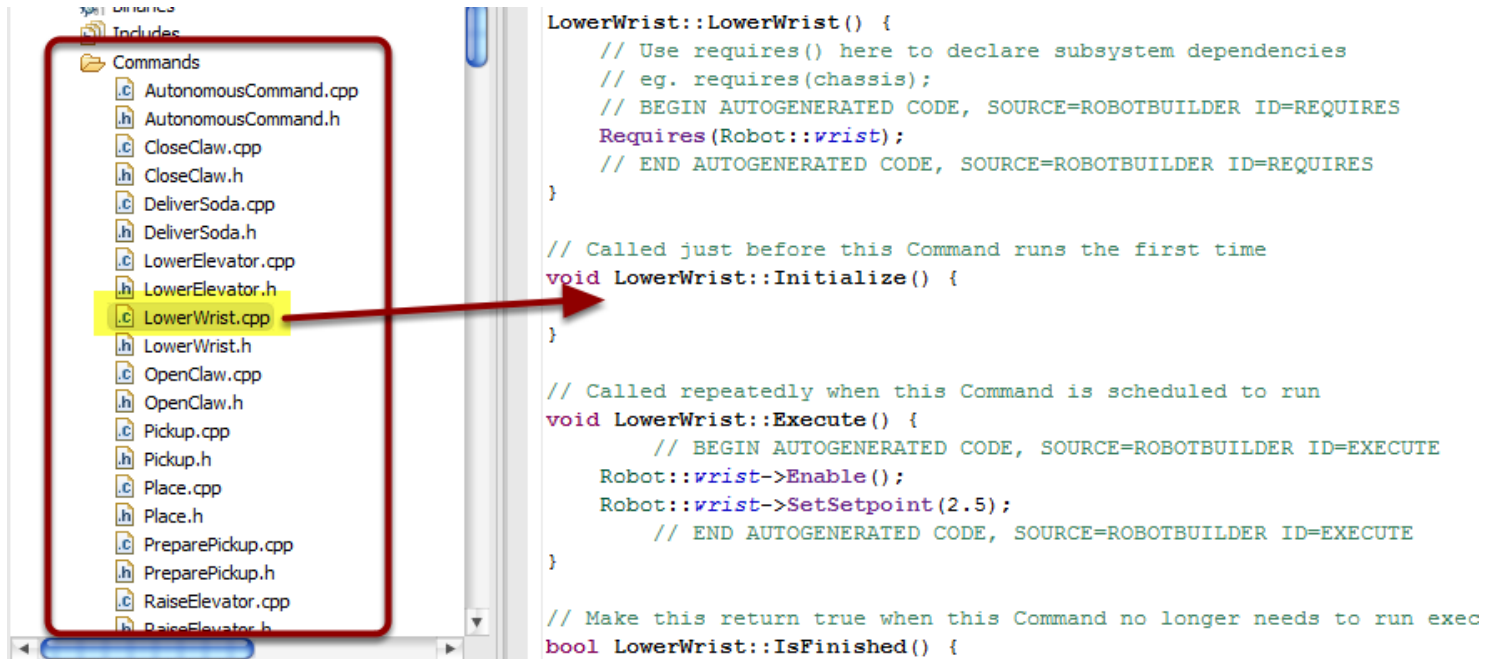
## Break up the robot into subsystems



The first step is to break the robot program into subsystems that each represent a separately controlled mechanism on the robot. Each subsystem has methods that operate the mechanism in some way.

# How to write an easy to test robot program

## Create commands for each robot behavior



Commands define the behaviors of the robot, that is the operation of the subsystems over time. A command starts a subsystem doing something, then waits until it is finished. Upon completion, the next command can be scheduled. It is the defining of commands for each behavior that is the key to making it easy to test the code as you'll see in the next few sections of this lesson.

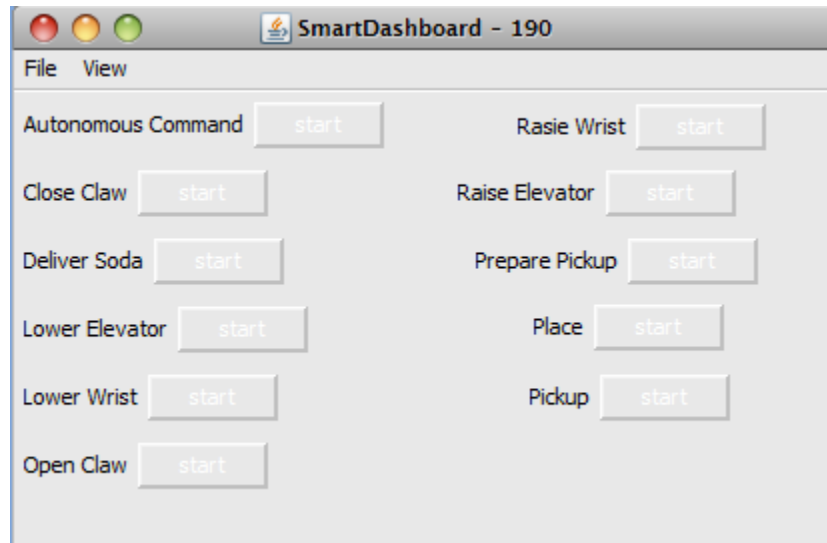
## Add commands to the SmartDashboard

```
// SmartDashboard Buttons  
SmartDashboard::PutData("Autonomous Command", new AutonomousCommand());  
SmartDashboard::PutData("Open Claw", new OpenClaw());  
SmartDashboard::PutData("Close Claw", new CloseClaw());  
SmartDashboard::PutData("Lower Wrist", new LowerWrist());  
SmartDashboard::PutData("Raise Wrist", new RaiseWrist());  
SmartDashboard::PutData("Lower Elevator", new LowerElevator());  
SmartDashboard::PutData("Raise Elevator", new RaiseElevator());  
SmartDashboard::PutData("Prepare Pickup", new PreparePickup());  
SmartDashboard::PutData("Pickup", new Pickup());  
SmartDashboard::PutData("Place", new Place());  
SmartDashboard::PutData("Deliver Soda", new DeliverSoda());
```

Once one more commands are developed, they can be tested by adding them to the SmartDashboard. It's easy to add commands, simply call `SmartDashboard.PutData()` with the command name that should be displayed and the command object itself.

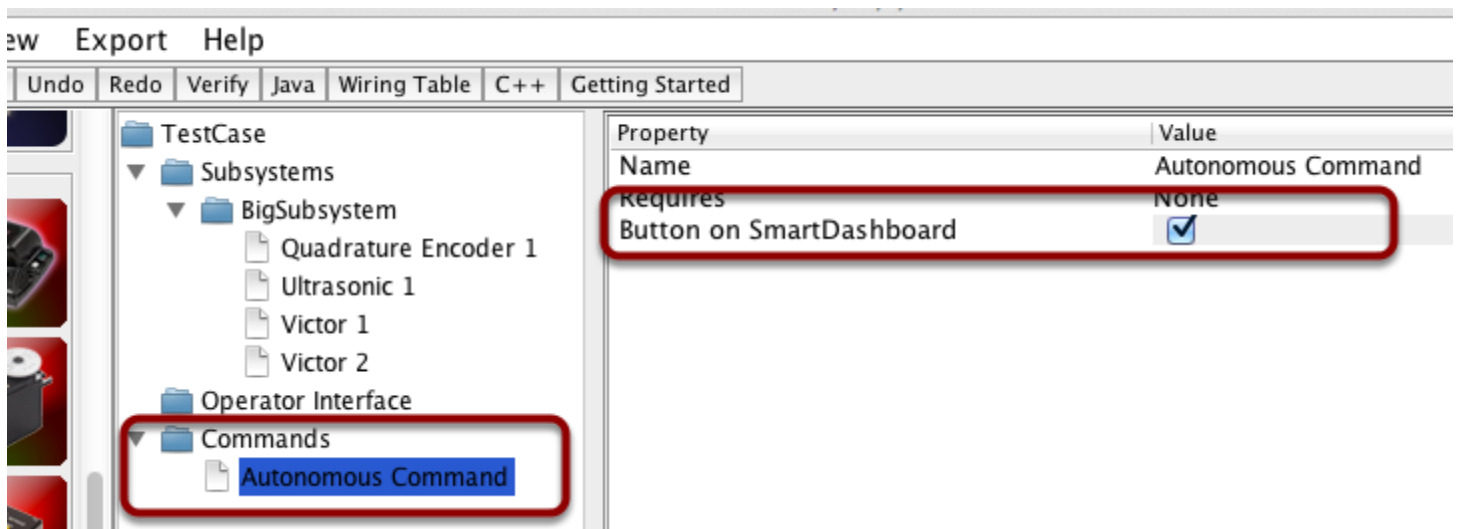
# How to write an easy to test robot program

## Test commands individually using the SmartDashboard



When a command is added to the SmartDashboard a button is created on the screen that will, when pressed, schedule the command to run. By doing this you can easily test each command individually to make sure it works. This is a key principle in programming, that is unit test each part of the larger program separately, then you'll have confidence that putting the units (commands) together into a larger program will also work.

## Adding commands to the SmartDashboard using RobotBuilder

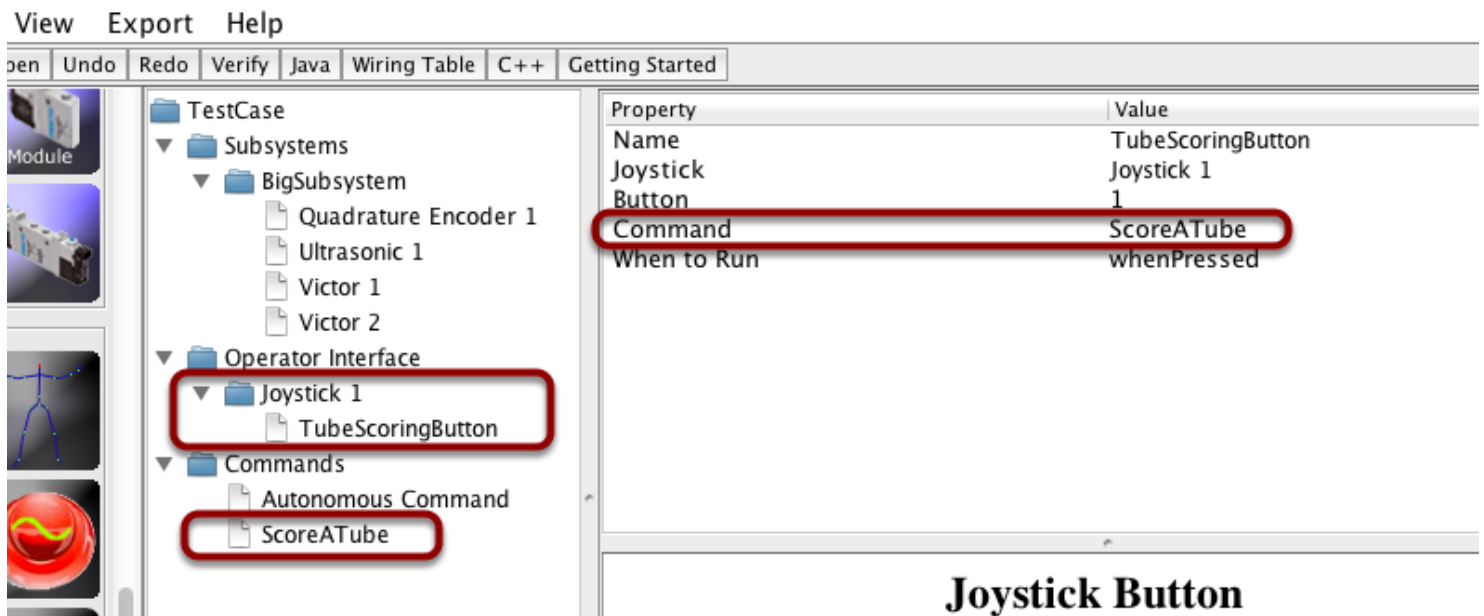


The idea of adding commands to testing pieces of your robot program individually is so useful that a checkbox was added to RobotBuilder to automatically generate SmartDashboard buttons. Just

# How to write an easy to test robot program

check the box for any command and it will automatically appear as a button that, when pressed, will run the command.

## Test commands individually using a joystick button



The screenshot shows the RobotC software interface. On the left, a tree view displays the project structure: **TestCase** (expanded) contains **Subsystems** (expanded) which includes **BigSubsystem** (expanded) with **Quadrature Encoder 1**, **Ultrasonic 1**, **Victor 1**, and **Victor 2**; **Operator Interface** (expanded) which includes **Joystick 1** (expanded) with **TubeScoringButton**; and **Commands** (expanded) with **Autonomous Command** and **ScoreATube**. The **Joystick 1** and **ScoreATube** items are highlighted with red boxes. On the right, a **Property** table for the selected **TubeScoringButton** is shown:

Property	Value
Name	TubeScoringButton
Joystick	Joystick 1
Button	1
Command	ScoreATube
When to Run	whenPressed

The **Command** and **When to Run** rows are highlighted with a red box. Below the table, the text **Joystick Button** is displayed.

In addition to testing commands using the SmartDashboard, you can also "wire" commands to joystick buttons. This is another easy way to verify that commands work as you expect. And, in fact, in your finished robot program, you will likely have a number of commands wired to many of the joystick buttons as a way of controlling the robot.

In this example there is a command called **ScoreATube** that presumably places a scoring object somewhere. Then there is a **JoystickButton** object called **TubeScoringButton** that has as its command the **ScoreATube** command. When the button is pressed, the command will run.

## Combine simple commands into Command Groups

```
DeliverSoda::DeliverSoda() {  
    AddSequential(new PreparePickup());  
    AddSequential(new Pickup());  
    AddSequential(new Place());  
    AddSequential(new RasieWrist());  
    AddSequential(new CloseClaw());  
}
```

# How to write an easy to test robot program

Once individual commands are working, they can be combined into Command Groups. Command Groups are commands that each consist of a list of commands to run when scheduled. This is where much of the power of the system comes into play. Suppose the robot needs to score an inner tube by driving forward for some distance, spinning rollers to release the tube for 1 second, moving a wrist joint, and backing up the robot. If there are commands for each of those operations, they can be individually tested, then a command group just lists all the individual commands, and the group is tied to a button. When the button is pressed, all the individual commands in the group run sequentially. Commands can even be scheduled to run in parallel for operations that can happen simultaneously.

## Add robot status to the SmartDashboard

Often you want to see the status of the robot by displaying values on the driver station. In the past this was handled with adding lots of print statements to the program and values poured out of the program, usually so fast that it was hard to know what was going on. With the SmartDashboard, you can create a display just by sending values, and they will automatically be displayed in the graphical user interface. In fact, you can change the format of the fields to more appropriate displays. For example, for a gyro, its heading can be displayed as a compass.

In addition display "widgets" or graphical elements can be created as standard Java code and added to the dashboard to make custom dashboards easily by programmers. For more detail see the [section on the SmartDashboard](#).