



2015 FTC Kick-Off

FTC programming Fundamentals

Frog Tech University, FRC Team 503
September 12, 2015

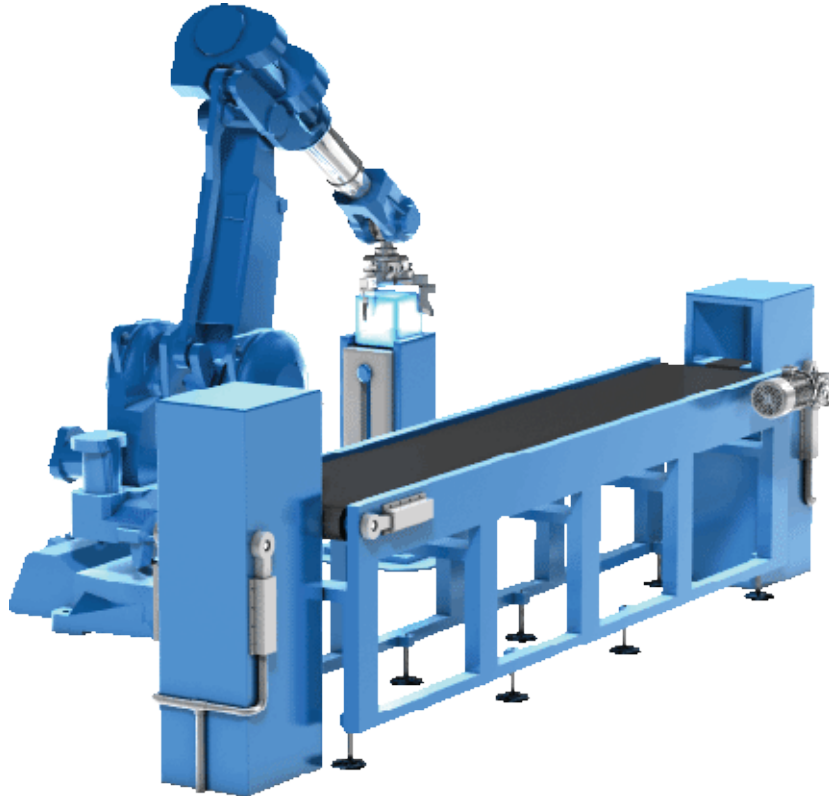
Course 104

Today's Goal.....



The goal of today's session is give you a very basic look at how to develop programming code for the robot

Programming Model Compared to LEGO NXT



Guide

- The programming model for the new FTC control systems is different than the one used for the Lego NXT with tools like RobotC.
- Lego NXT used a linear programming model-which means things executed one after another in the code.
- The new FTC Java-based tools use an Event driven model.
- Sections of code fire off when a certain event occurs.
- For example the loop() method you are about to see fires off repeatedly until stopped.

What is an Op Mode?

Definition

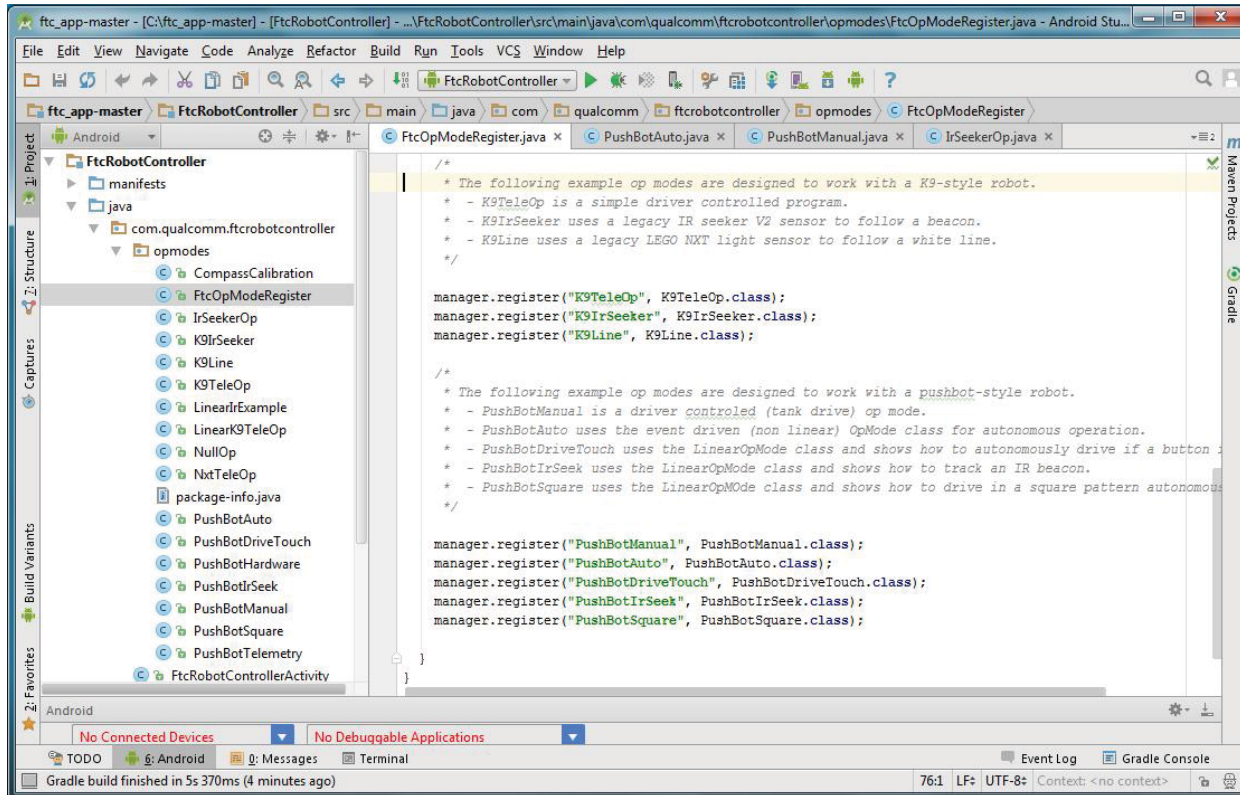


An Operational Mode (Op Mode) is a software module stored on the robot controller that you can execute from the driver station.

These op-mode software modules contain pre-programmed behaviors for your robot.

Registering an OpMode

You must register new OpModes to see them in the driver station list

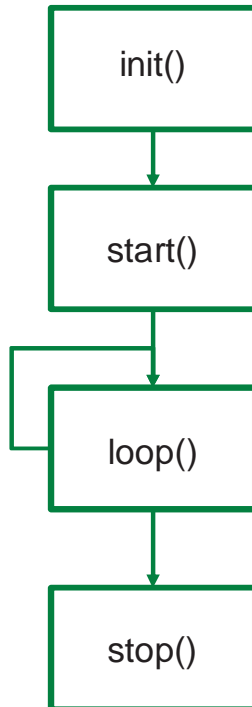


Guide

- A register is a list of op modes that will display on the driver station for selection and execution.
- If you create a new op mode you must add a manager.register command to the FTCOpModeRegister.java.
- The format is:
 - Manager.register("opmode name",Java code classname.class);
- Example:
 - Manager.register("K9TeleOp",K9TeleOp.class);

Anatomy of an OpMode

Overview

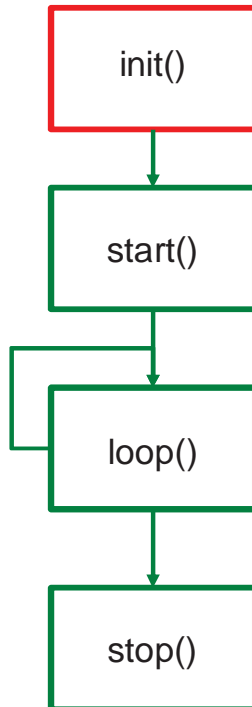


Guide

- Every opMode **should/can** contain 4 methods:
 - Init
 - Start
 - Loop
 - Stop
- Not all methods are required for every Op Mode

Anatomy of an OpMode

init – Initialization

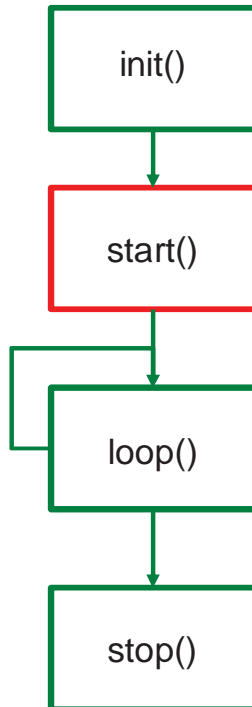


Guide

- This is for initialization tasks
- It is executed only once
- The robot is updated after the method exits-not as each line in the code is executing

Anatomy of an OpMode

start – tasks before loop

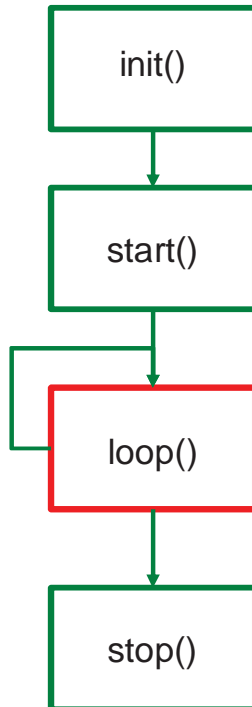


Guide

- This method is triggered when the driver pushes “Start” button on the touch screen
- It is executed only once
- If you have any initialization tasks that you want to execute right before the “loop” method you do so here by adding a public void `start() {}` to your opmode

Anatomy of an OpMode

loop – stuff to do repeatedly

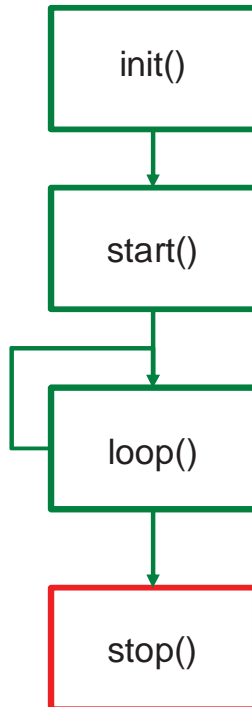


Guide

- When the driver pushed the “Start” button on the driver station, the code in the loop() method will execute repeatedly (approximately every 10-20 milliseconds)
- The robot controller app has a built in event loop that executes the contents of the loop() method repeatedly until a stop command is received from the driver station (or unless an emergency stop condition occurs)
- This is the method where you will put the bulk of your code

Anatomy of an OpMode

stop – tasks to do when the loop is over



Guide

- When the robot controller receives a stop command from the driver station or when emergency stop() condition occurs, the code in the stop() method gets executed
- If you have an cleaning up to do after the opmode loop() method has been run, this is the place to put it
- Just like the start() method this method does nothing by default, you can override it's behavior by adding a public void stop() {} to your opmode
- Lets take a look at the sample code in Android Studio. We will use the K9TeleOp Op mode as it is a very straight forward for novice FTC programmers.

Anatomy of an OpMode

init – K9TeleOp sample code

```
74  @Override
75  public void init() {
76
77      motorLeft = hardwareMap.dcMotor.get("motor_1");
78      motorRight = hardwareMap.dcMotor.get("motor_2");
79      neck = hardwareMap.servo.get("servo_1");
80      jaw = hardwareMap.servo.get("servo_6");
81      wheelController = hardwareMap.dcMotorController.get("wheels");
82
83      motorLeft.setDirection(DcMotor.Direction.REVERSE);
84
85      // set the starting position of the wrist and neck
86      neckPosition = 0.5;
87  }
```

Guide

- Looks for a DCMotor called motor_1 and assigns to a variable called motorLeft
- Looks for a DCMotor called motor_2 and assigns to a variable called motorRight
- Looks for a Server named server_1 and assigns to a variable called neck, etc

Note these are the names you defined in the configuration file on the Android phones. Whatever names you assign in the Robot Controller must match the names here in the program.

Anatomy of an OpMode

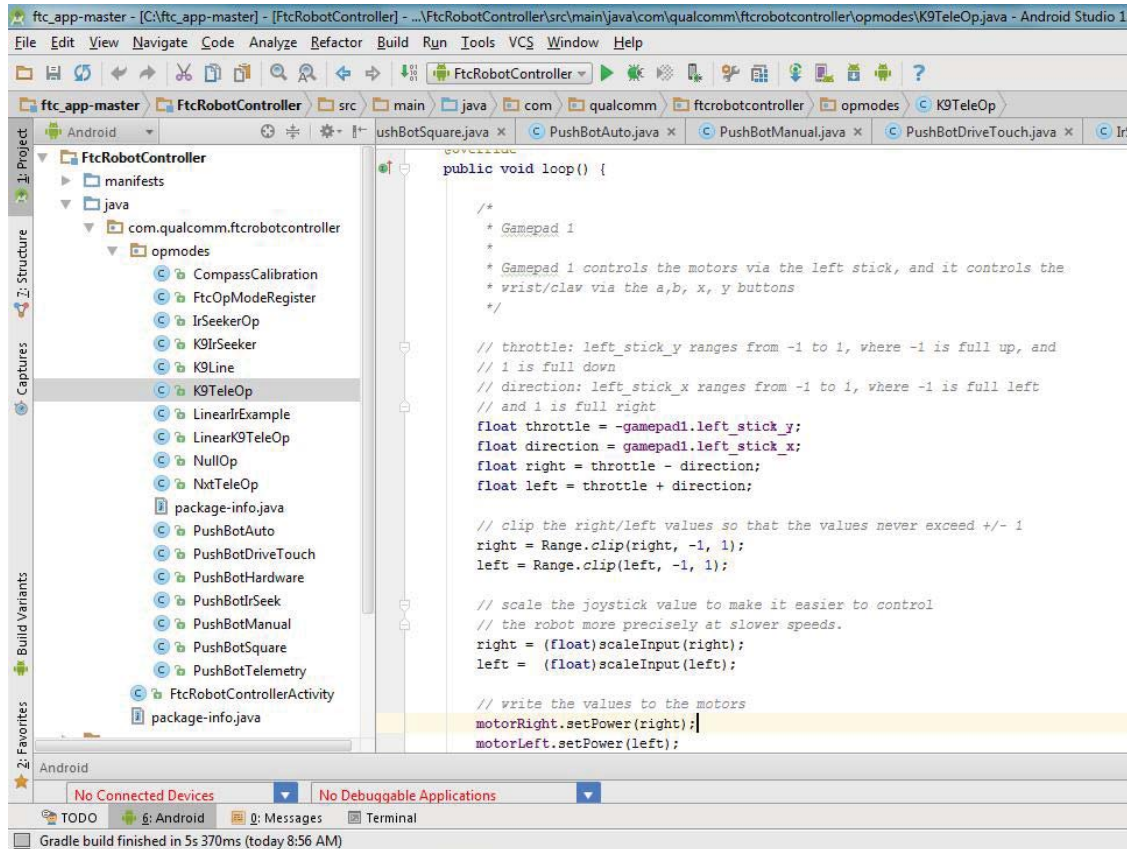
init – K9TeleOp sample code

```
74  @Override
75  public void init() {
76
77      motorLeft = hardwareMap.dcMotor.get("motor_1");
78      motorRight = hardwareMap.dcMotor.get("motor_2");
79      neck = hardwareMap.servo.get("servo_1");
80      jaw = hardwareMap.servo.get("servo_6");
81      wheelController = hardwareMap.dcMotorController.get("wheels");
82
83      motorLeft.setDirection(DcMotor.Direction.REVERSE);
84
85      // set the starting position of the wrist and neck
86      neckPosition = 0.5;
87  }
```

Guide

- Another thing to notice is that the motorLeft variable is using a method called “SetDirection” and then is setting it to reverse.
- So the Motor Right will go forward and motorLeft will go backward. This is to ensure that the robot moves forward when the motors are installed in opposite directions.

Anatomy of an OpMode loop – K9TeleOp sample code



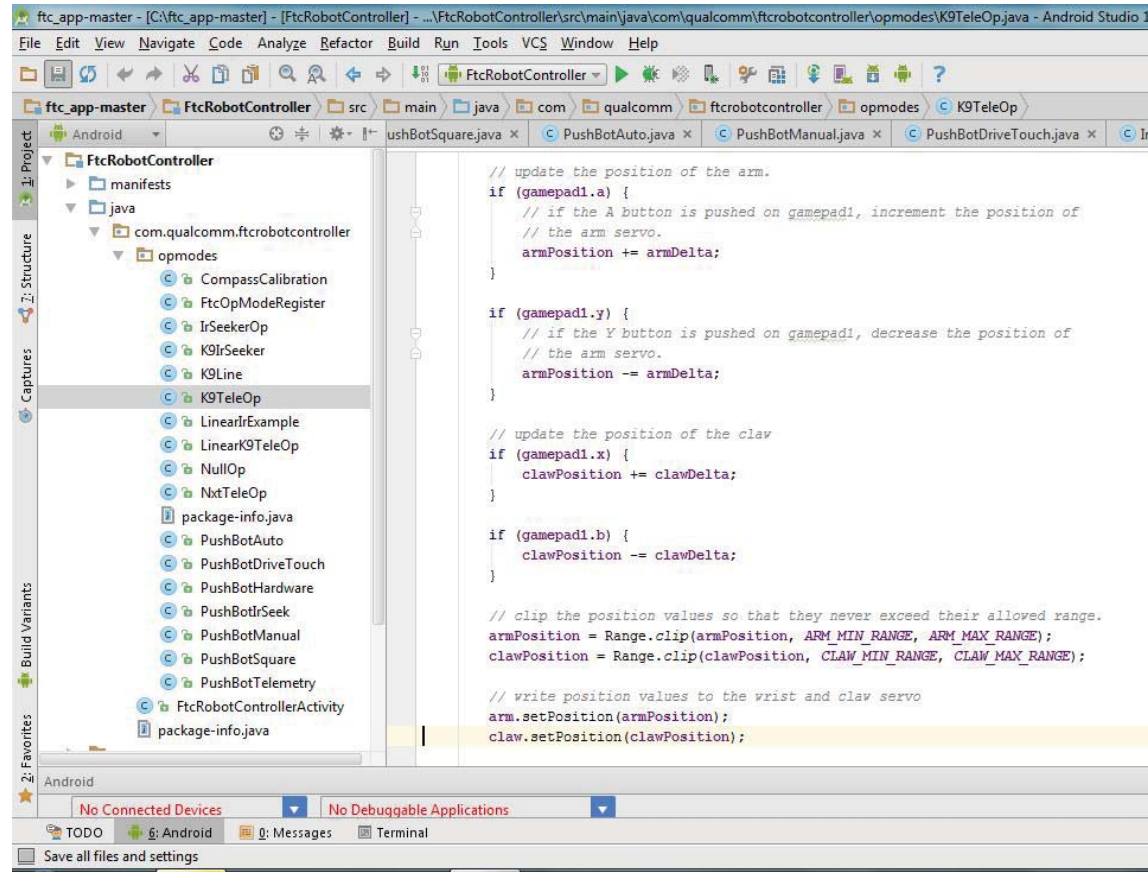
```
public void loop() {  
    /*  
     * Gamepad 1  
     * Gamepad 1 controls the motors via the left stick, and it controls the  
     * wrist/claw via the a,b, x, y buttons  
     */  
  
    // throttle: left_stick_y ranges from -1 to 1, where -1 is full up, and  
    // 1 is full down  
    // direction: left_stick_x ranges from -1 to 1, where -1 is full left  
    // and 1 is full right  
    float throttle = -gamepad1.left_stick_y;  
    float direction = gamepad1.left_stick_x;  
    float right = throttle - direction;  
    float left = throttle + direction;  
  
    // clip the right/left values so that the values never exceed +/- 1  
    right = Range.clip(right, -1, 1);  
    left = Range.clip(left, -1, 1);  
  
    // scale the joystick value to make it easier to control  
    // the robot more precisely at slower speeds.  
    right = (float)scaleInput(right);  
    left = (float)scaleInput(left);  
  
    // write the values to the motors  
    motorRight.setPower(right);  
    motorLeft.setPower(left);  
}
```

Guide

- A new float variable throttle is assigned the value of the gamepad1 (the driver gamepad), left stick, y value
- A new float variable named direction is assigned the value of gamepad1, left stick, x value
- A new float variable named right is set to the throttle minus the direction, etc
- The left and right variables are then calculated to ensure they don't exceed the maximums for the motors
- And then finally MotorRight and MotorLeft are set to the power that we calculated.

Anatomy of an OpMode

loop – K9TeleOp sample code – next page

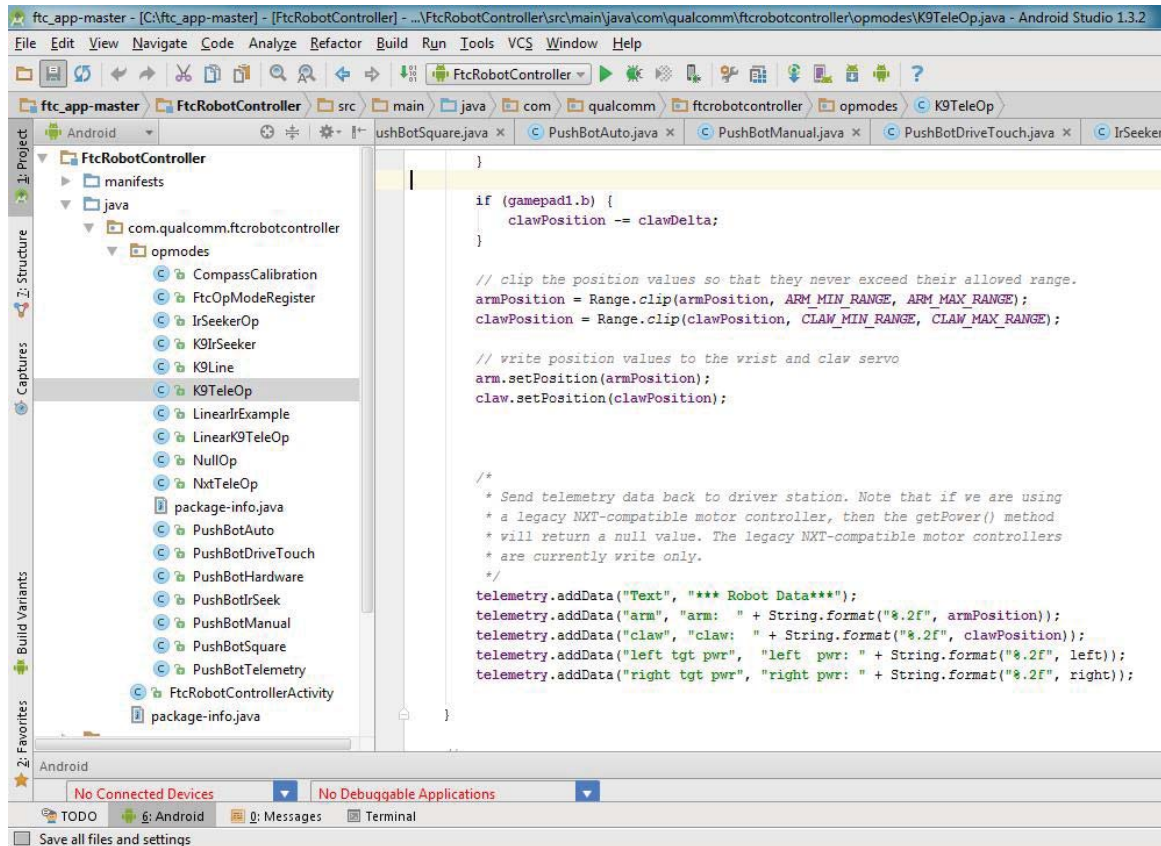


Guide

- Now we check if the 'A' button on the gamepad1 was pressed. If so, armPosition is increased by ArmDelta.
- Then we check the 'Y' button. If pressed armPosition is decreased by ArmDelta.
- The same concept applies in the next lines of code checking the 'X' and 'B' buttons and then adjusting the claw position accordingly.
- We then ensure that the values we calculated are within the specifications for the hardware.
- Then we finally send a command to change the arm and claw positions

Anatomy of an OpMode

loop – K9TeleOp sample code – next page

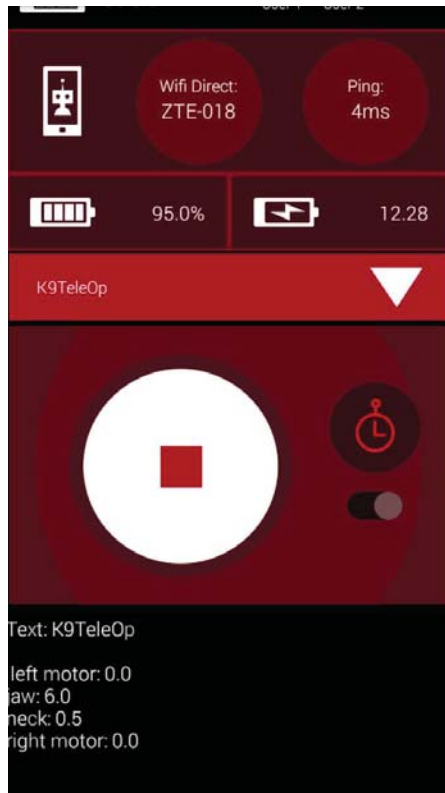


Guide

- Once we move the motors, the arm and the claw, we want to send some data back to the driver station
- The next lines of code use the “telemetry” object with a method of add data.
- Using this method we can send any data back that we want to tell the driver.
- In this case we are sending the:
 - Arm position
 - Claw position
 - Left motor power
 - Right motor power

Anatomy of an OpMode

loop – K9TeleOp sample code – next page

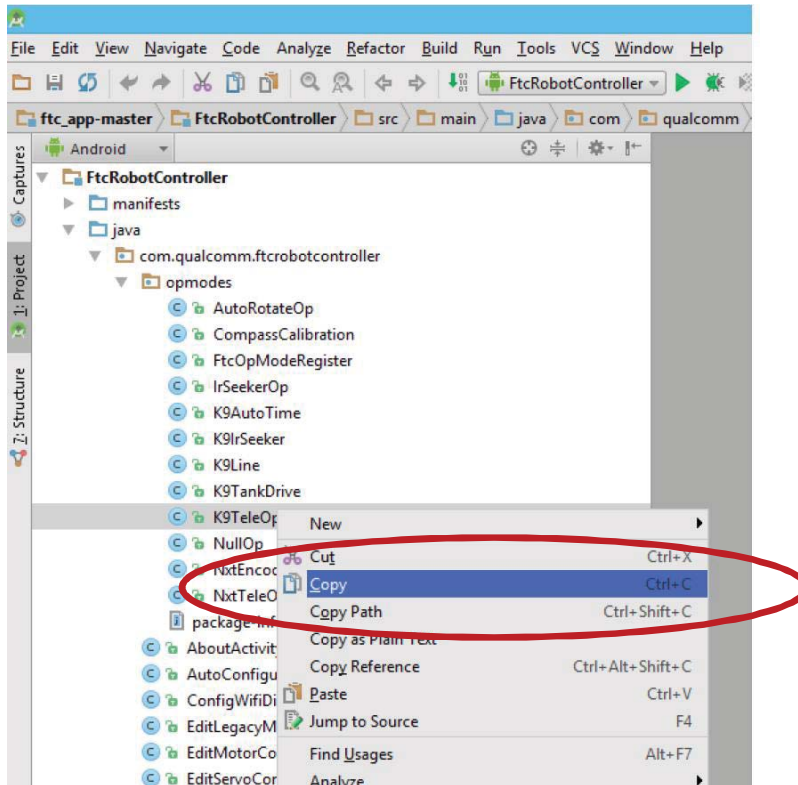


Guide

- The telemetry data will be displayed at the bottom of the driver station similar to that screen shot shown on the left

To make your own Op Mode

1. Copy the existing K9TeleOp

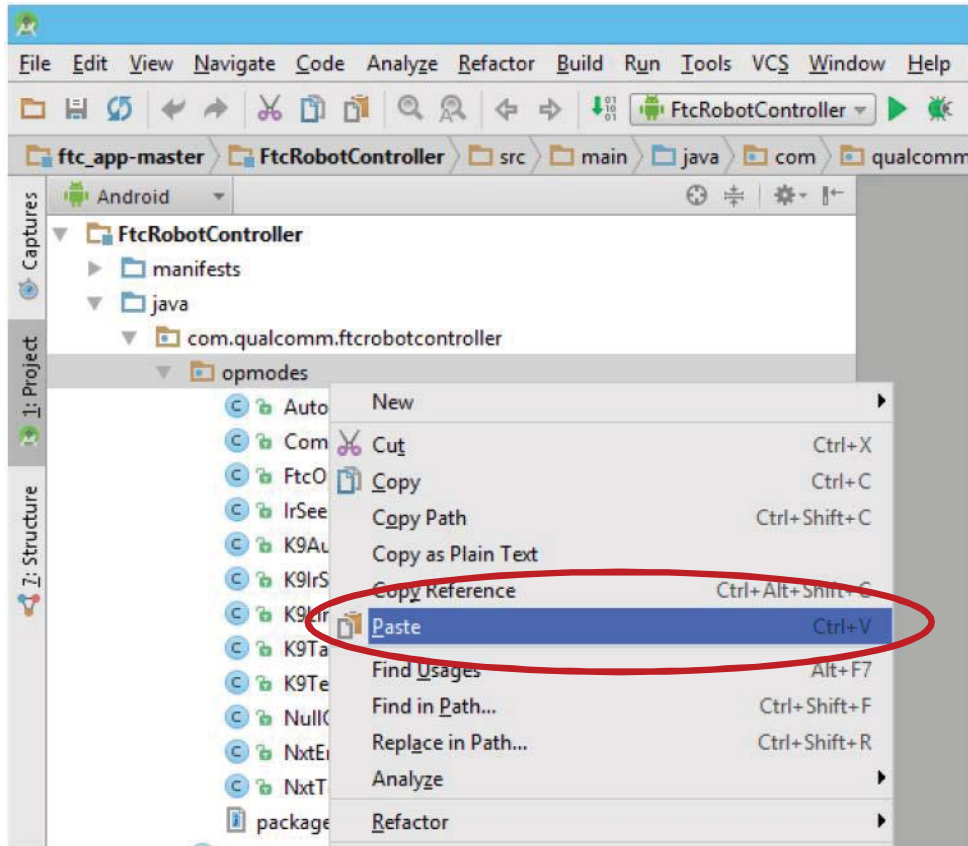


Guide

- Rather than start from scratch, let's make life easy and just copy an existing OpMode and then modify it to fit our purposes.
- In Android Studio, right-click on the K9TeleOp program file and select COPY from the pop-up menu

To make your own Op Mode

2. Paste the copied file into the opmodes folder

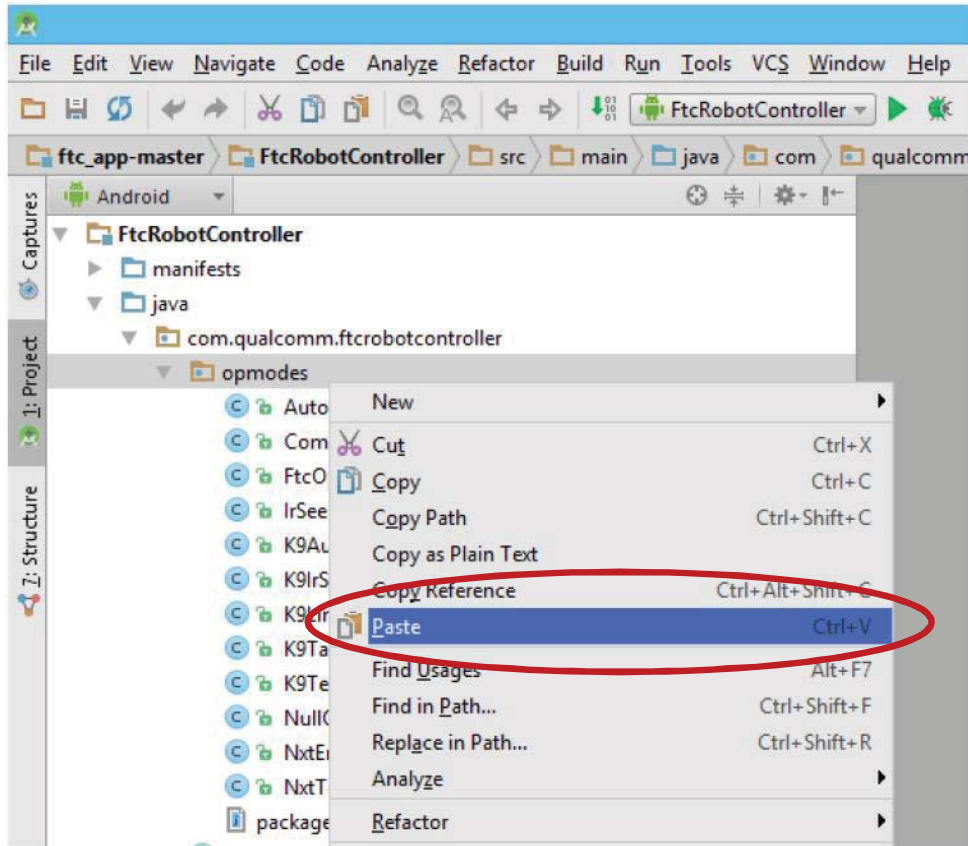


Guide

- Let the file copy
- Right Click on the opmodes folder
- Select "Paste" from the popup menu

To make your own Op Mode

2. Paste the copied file into the opmodes folder

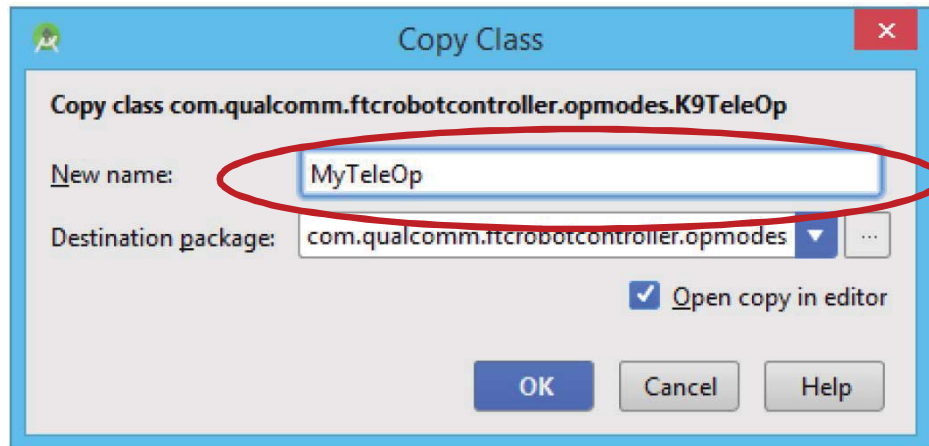


Guide

- Let the file copy
- Right Click on the opmodes folder
- Select "Paste" from the popup menu

To make your own Op Mode

3. Enter name for your new Op Mode



Guide

- Android Studio will prompt you for a New Name for your new Java Class.
- Enter the name of your Op Mode in the New Name text box
 - In this case we entered the name of “MyTeleOp”
- Click “OK”
- Android Studio will create a new class called “MyTeleOp” that will appear in the opmodes folder.
- The contents of the new class will be the same contents of the existing “K9TeleOp” class
- If you were to try and run the app, your new Op Mode called “MyTeleOp” would not display in the list.
- You must register the new Op Mode for it to display in the list.
- We will do this in the next step

To make your own Op Mode

4. Find the Op Mode Register class

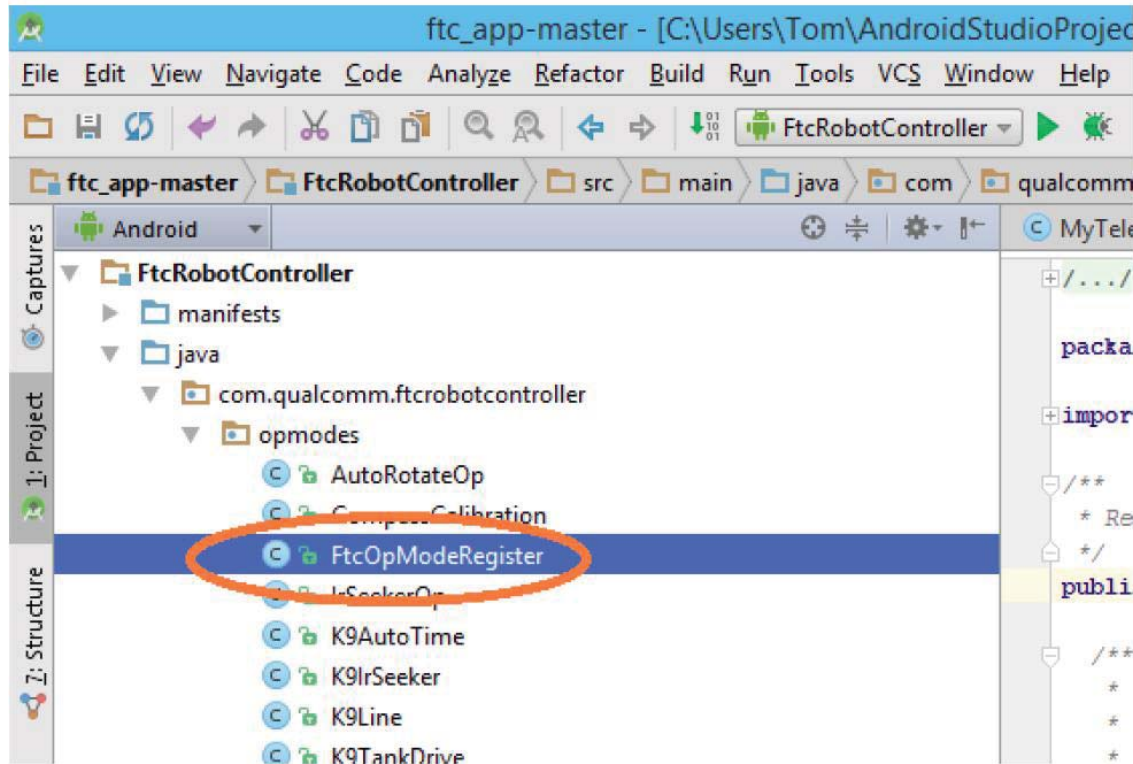


Figure 151 Double click on FtcOpModeRegister to open the file for editing

Guide

- Find the FtcOpModeRegister class in the Android Studio project list
- Double Click on “FtcOpModeRegister”

To make your own Op Mode

5. Find the Op Mode Register class

```
/*  
 * If two or more op modes are registered with the same name, the  
 */  
  
manager.register("NullOp", NullOp.class);  
manager.register("K9TeleOp", K9TeleOp.class);  
manager.register("K9TankDrive", K9TankDrive.class);  
manager.register("K9Line", K9Line.class);  
manager.register("K9IrSeeker", K9IrSeeker.class);  
manager.register("K9AutoTime", K9AutoTime.class);  
/*  
manager.register("IrSeekerOp", IrSeekerOp.class);  
manager.register("CompassCalibration", CompassCalibration.class);  
manager.register("NullOp", NullOp.class);
```

Guide

- Find the manager.register code as shown on the left
- WE need to copy an existing line rather than type it in from scratch
- Select the manager.register("K9AutoTime", K9AutoTime.class); line
- Copy and paste it right below the existing line

To make your own Op Mode

6. Find the Op Mode Register class

```
* If two or more op modes are registered with the same  
*/  
  
manager.register("NullOp", NullOp.class);  
manager.register("K9TeleOp", K9TeleOp.class);  
manager.register("K9TankDrive", K9TankDrive.class);  
manager.register("K9Line", K9Line.class);  
manager.register("K9IrSeeker", K9IrSeeker.class);  
manager.register("K9AutoTime", K9AutoTime.class);  
manager.register("MyTeleOp", MyTeleOp.class);  
/*  
manager.register("IrSeekerOp", IrSeekerOp.class);
```

Guide

- Change the "K9AutoTime" to "MyTeleop"
- Change the "K9AutoTime.class" to "MyTeleOp.class"
- Your changes should look like the page shown on the left
- Android should auto-save the file. But it is good practice to save it anyway
- You now can select the Green Right arrow on the tool bar to run the program.
- Android studio will rebuild the Gradle (executable file)

To make your own Op Mode

7. Once the App is installed on the Driver Station

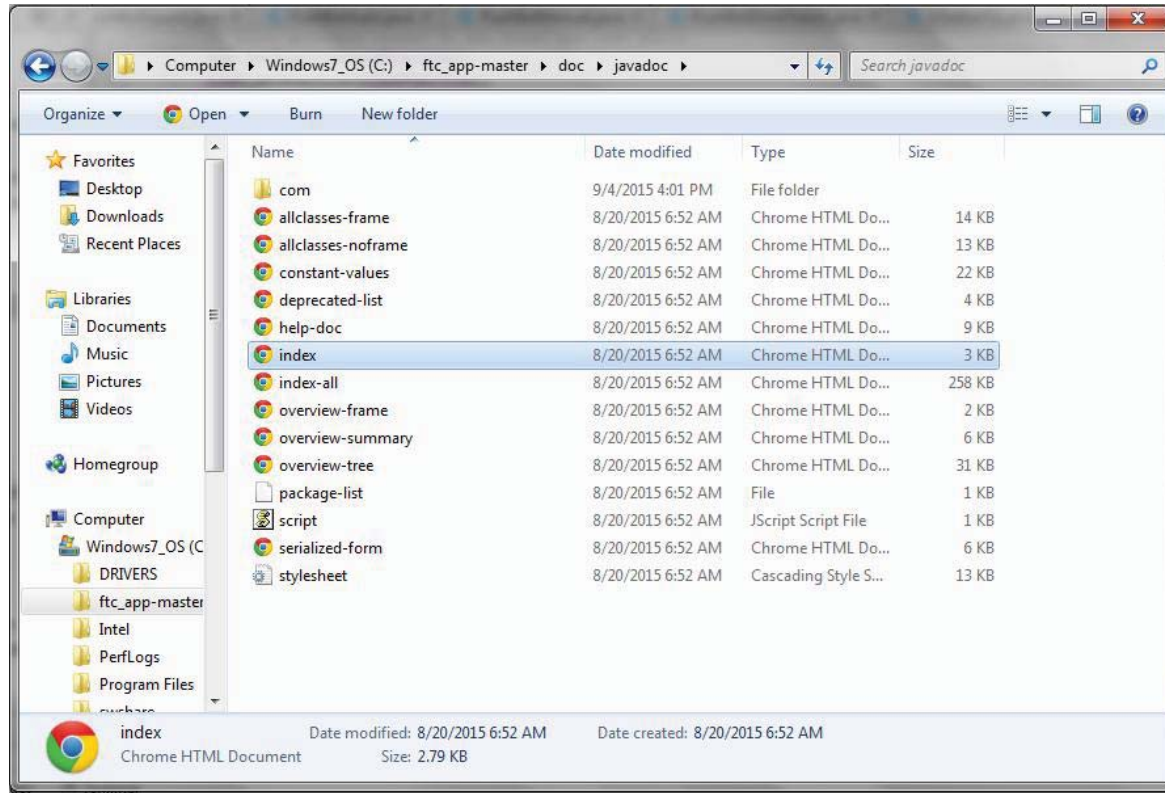


Guide

- Once you connect the USB cable to the driver station to download the new code
- You should now see “MyTeleOp” as a new Op Mode

QualComm SDK Documentation

Finding the documentation on your machine

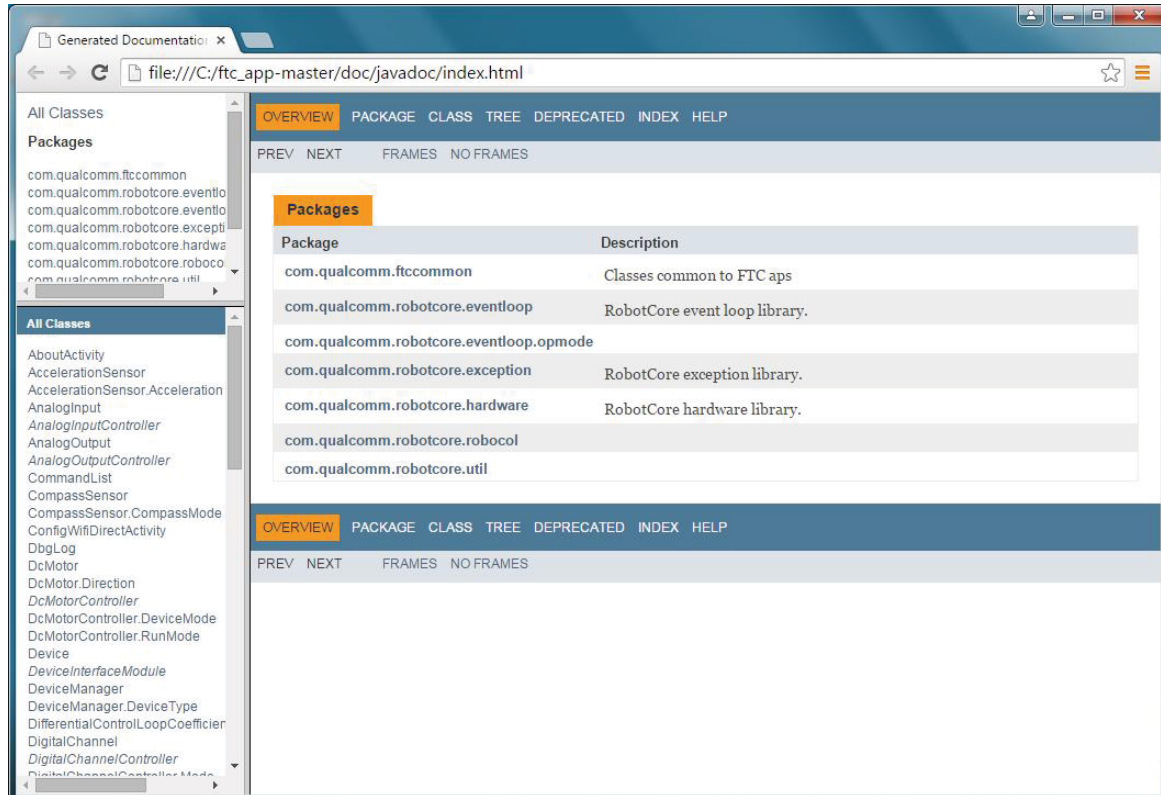


Guide

- Qualcomm has created documentation for their FTC software Development Kit (SDK)
- The documentation was is installed on your machine as part of the ftc_app-master folder.
- You can access the documentation by using Windows Explorer.
- Go to `C:\ftc_app-master\doc\Javadoc` folder
- Click on “index.html”

QualComm SDK Documentation

Finding the documentation on your machine



Guide

- The page on the left is the index page for the QualComm documentation
- You can browse and research various topics in the SDK



Congratulations you have completed the basic programming course!!!!!!



Questions?



What we are doing today will transform tomorrow's culture.