Hello Robot - Programming an Op Mode

In most programming languages the first thing taught is the basic "Hello World" program. This code teaches users basic syntax and logic of a language, while also testing that the system being used to execute the code is functioning properly. This section of the guide will act as the REV Robotics version of the "Hello World" concept. By the end of this section users will have a basic knowledge of **op modes** and code syntax for OnBot Java and Blocks.

> It is important for programming in the REV Control System that you have a **Configuration. Robot Configurations** give names to the different mechanical elements of a robot - like sensors, servos, and motors - that can be referenced in the code. *If you have not set up a configuration yet please visit the Expansion Hub Configuration section or the Control Hub Configuration section to learn how to set one up.*

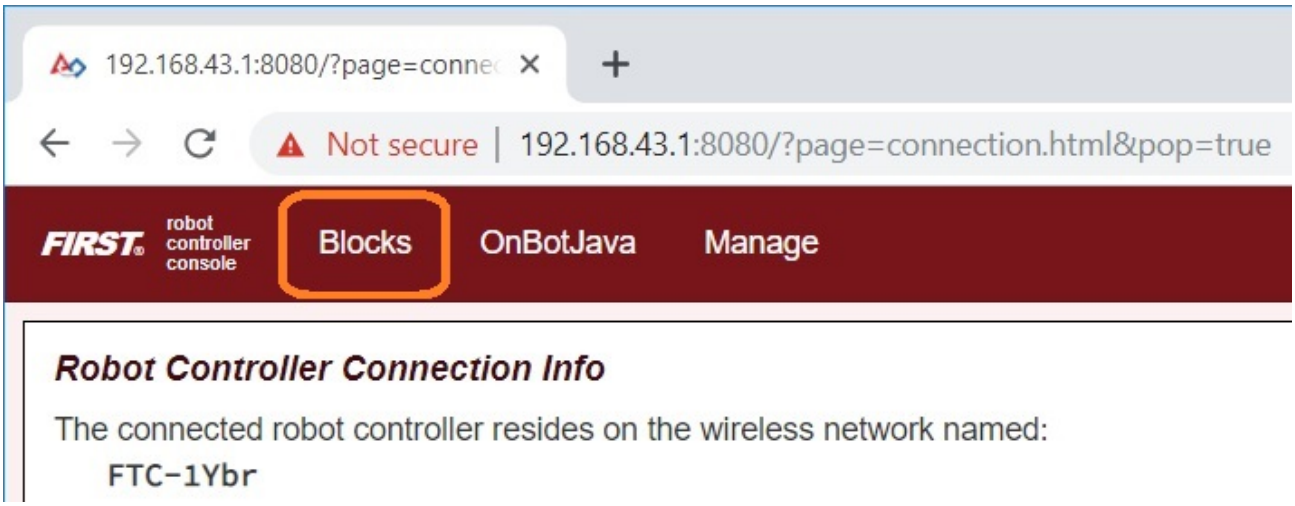
What is an Op Mode?

Op modes (or operational modes) are computer programs that are used to customize or specify the behavior of a robot. Op modes are saved onto and executed by the Robot Controller -either the Control Hub (REV-31-1595) or an Android device paired with an Expansion Hub (REV-31-1153). When op modes are saved to the Robot Controller they can be accessed and started by the Driver Station.

Hello Robot - Basic Op Mode Walkthrough

This section will walk through how to create an op mode in both OnBot Java and Blocks

> *This section assumes that you have already learned how to access the specific programming tool that you have chosen to use. Check out the Programming Language Options to pick a language and learn how to access it.*



Accessing Blocks

Press the Blocks link towards the top of the Console to navigate to the main Blocks Programming screen.

The main Blocks Programming screen is where you create new op modes. It is also the screen where you can see a list of existing Blocks Op Modes on a Robot Controller. Initially, this list will be empty until you create and save your first op mode.

Accessing OnBot Java

Press the OnBot Java link towards the top of the Console to navigate to the main OnBot Java Screen.

The OnBot Java screen is where you create new op modes. It is also the screen where you can see a list of existing op Modes on a Robot Controller. IThe list will be empty until you create and save your first op mode.

Creating an Op Mode

Before diving in and creating your first op mode, you should consider the concept of **naming conventions**. When writing code the goal is to be as clear as possible about what is happening within the code. This is where the concept of naming conventions comes into play. Common naming conventions have been established by the programming world to denote variables, classes, functions, etc. Op modes share some similarities to **classes**. Thus the naming convention for op modes tends to follow that naming convention for classes; where the first letter of every word is capitalized.

For the example below you should also create a configuration file with a motor, servo, touch sensor, color/range sensor, and the IMU. The following list is the naming conventions for the components that will be used in the example.

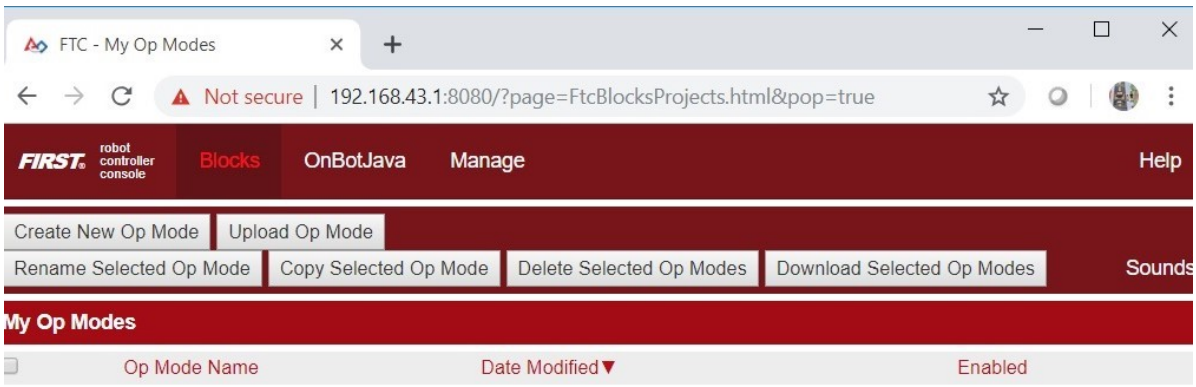
- Motor - motorTest
- IMU - imu
- Servo - servoTest
- Touch Sensor - digitalTouch
- Color/Range Sensor - sensorColorRange

> *To learn how to create a configuration file for your robot check out Configuring Your Hardware on the FTC Wiki*

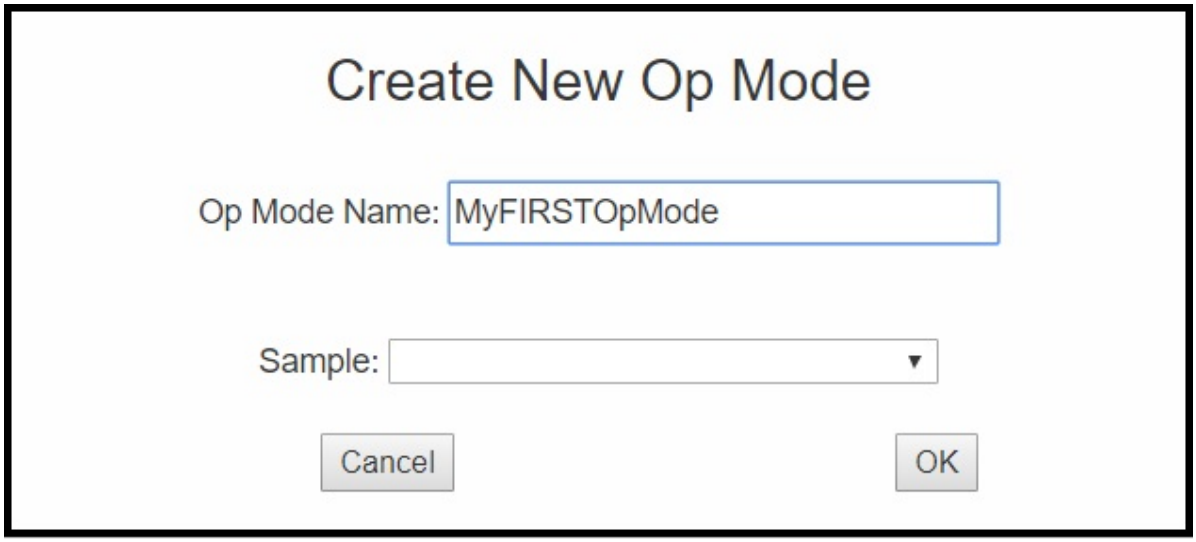
The following tab block will walk through how to make create an op mode. In both examples the op mode will be named **MyFIRSTOpMode**.

Blocks

Press the “Create New Op Mode” button in the upper left corner of the Robot Control Console Blocks Menu, seen in the image below.

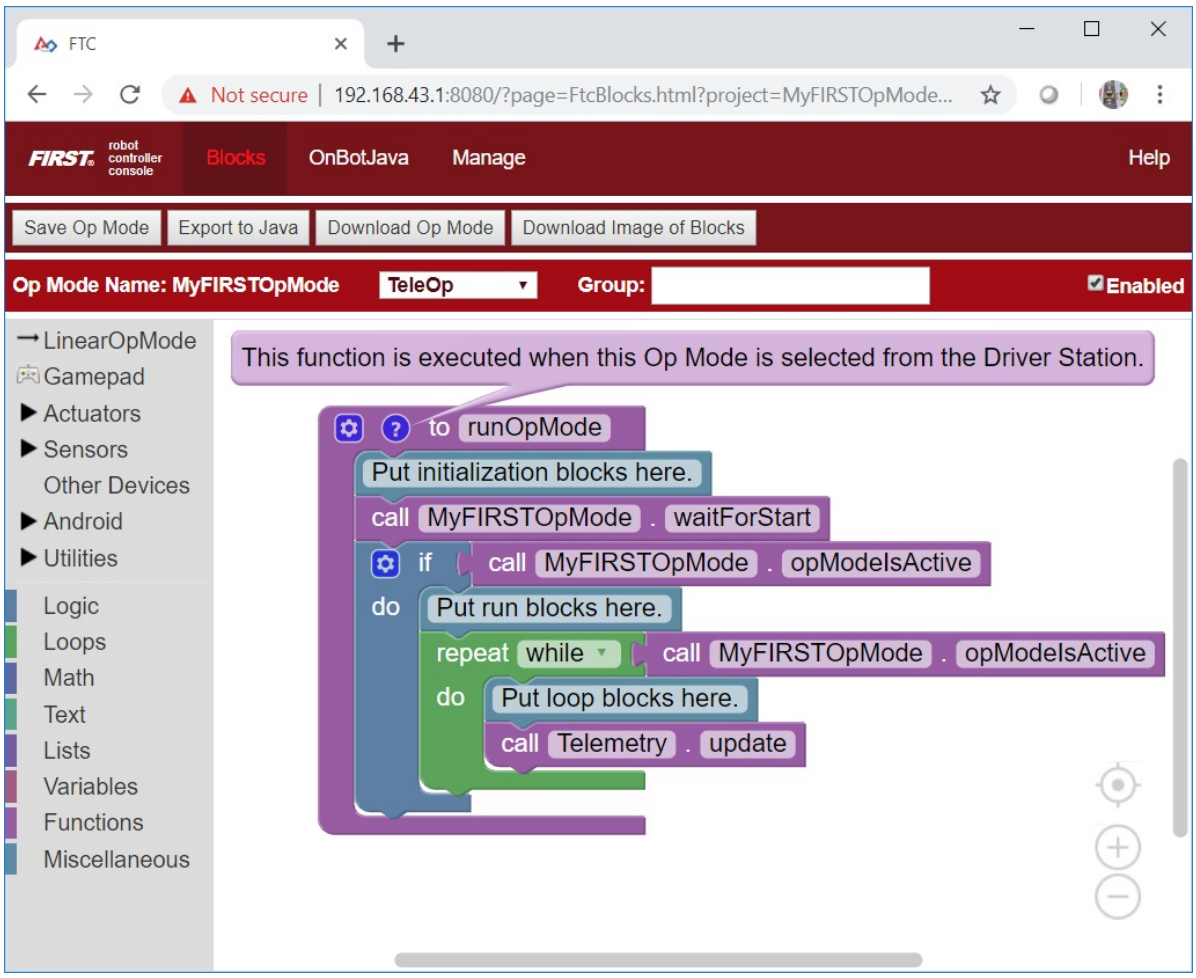


The "Create New Op Mode" window should appear. Enter the name for your op mode and hit "OK"



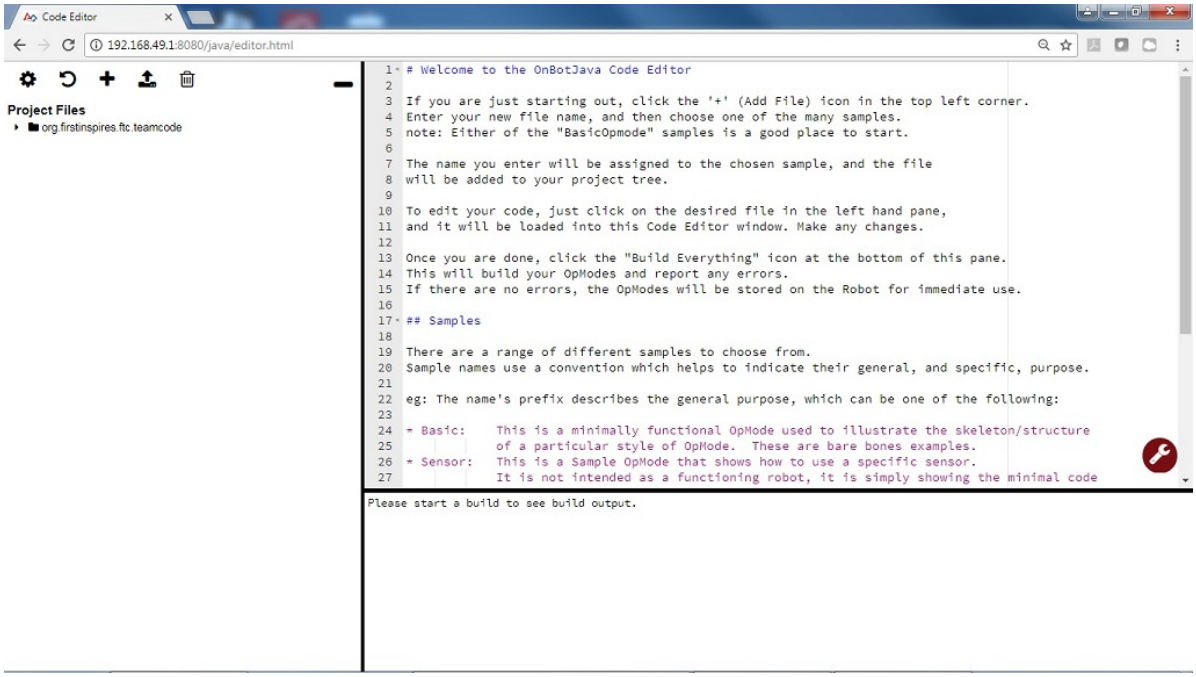
After selecting "OK" a screen similar to the one in the image below will appear. There are several key areas:

1. Clicking Save OP Mode is how the op mode is saved to the Robot Controller
2. Switch between TeleOp and Autonomous modes to determine the function of the op mode
3. Blocks are divided into category types. Click the category to sort through potential block options

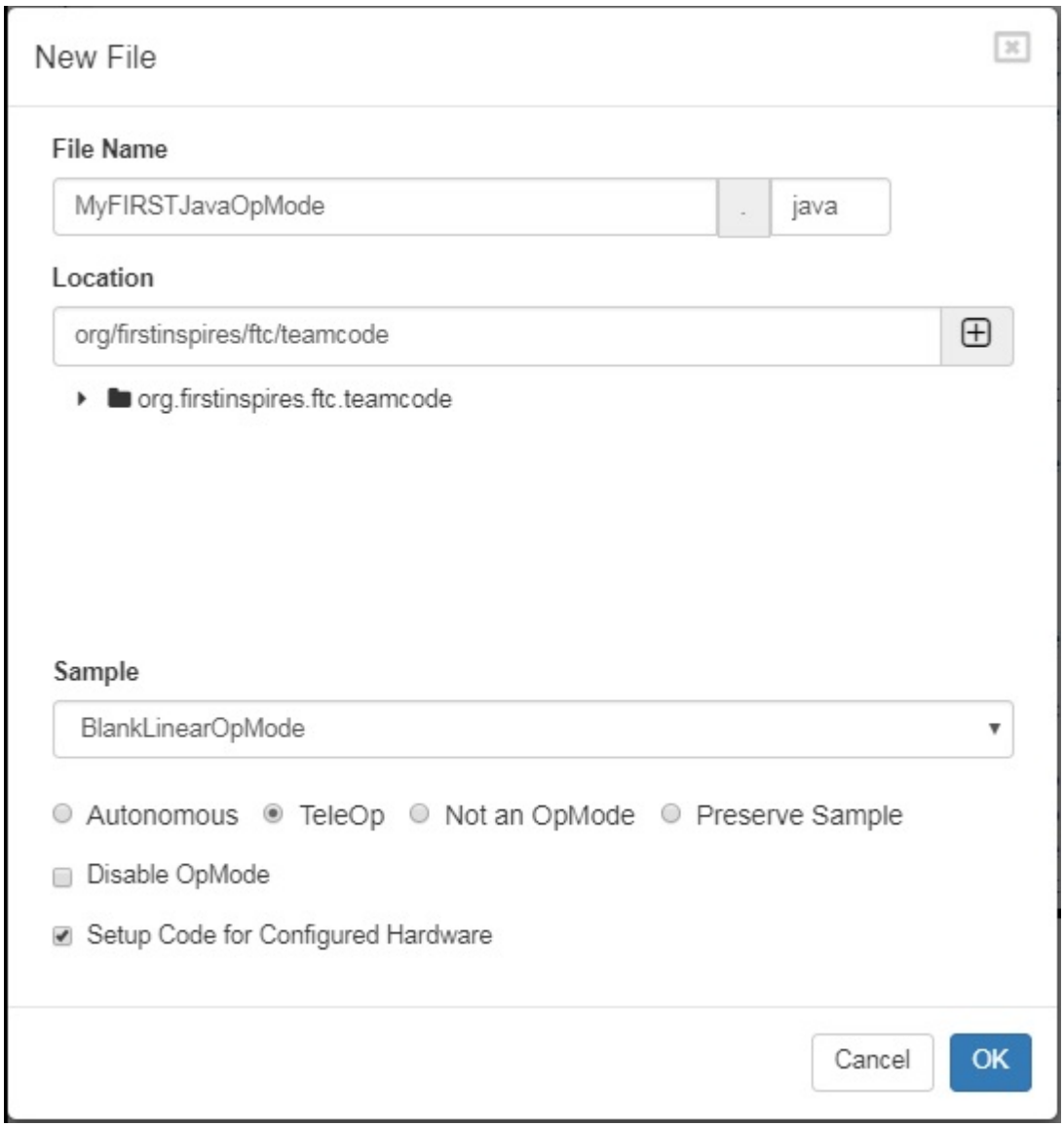

FTC
Not secure | 192.168.43.1:8080/?page=FtcBlocks.html?project=MyFIRSTOpMode...
FIRST robot controller console
Blocks
OnBotJava
Manage
Help
Save Op Mode
Export to Java
Download Op Mode
Download Image of Blocks
Op Mode Name: MyFIRSTOpMode
TeleOp
Group:
Enabled
LinearOpMode
Gamepad
Actuators
Sensors
Other Devices
Android
Utilities
Logic
Loops
Math
Text
Lists
Variables
Functions
Miscellaneous
This function is executed when this Op Mode is selected from the Driver Station.
to runOpMode
Put initialization blocks here.
call MyFIRSTOpMode . waitForStart
if call MyFIRSTOpMode . opModeIsActive
do Put run blocks here.
repeat while call MyFIRSTOpMode . opModeIsActive
do Put loop blocks here.
call Telemetry . update

OnBot Java

The image below shows the OnBot Java user interface. On the left hand side, there is the project browser pane. In the upper right hand corner, there is the source code editing pane. In the lower right hand corner, there is the message pane.



In the project browser pane, press the “+” symbol to create a new file. Pushing this button will launch the New File dialog box. This dialog box has several parameters that you can configure to customize your new file.



Using the Sample dropdown list control, select **BlankLinearOpMode** from the list of available sample op modes (see image above). By selecting **BlankLinearOpMode** the OnBot Java editor will automatically generate a basic LinearOpMode framework for you.

Check the option labeled “TeleOp” to ensure that this new file will be configured as a tele-operated (i.e., driver controlled) op mode.

Also, make sure you check the “Setup Code for Configured Hardware” option. If this option is enabled, the OnBot Java editor will look at the hardware configuration file for your Robot Controller and automatically generate the code that you will need to access the configured devices in your op mode.

Press the “OK” button to create your new op mode.

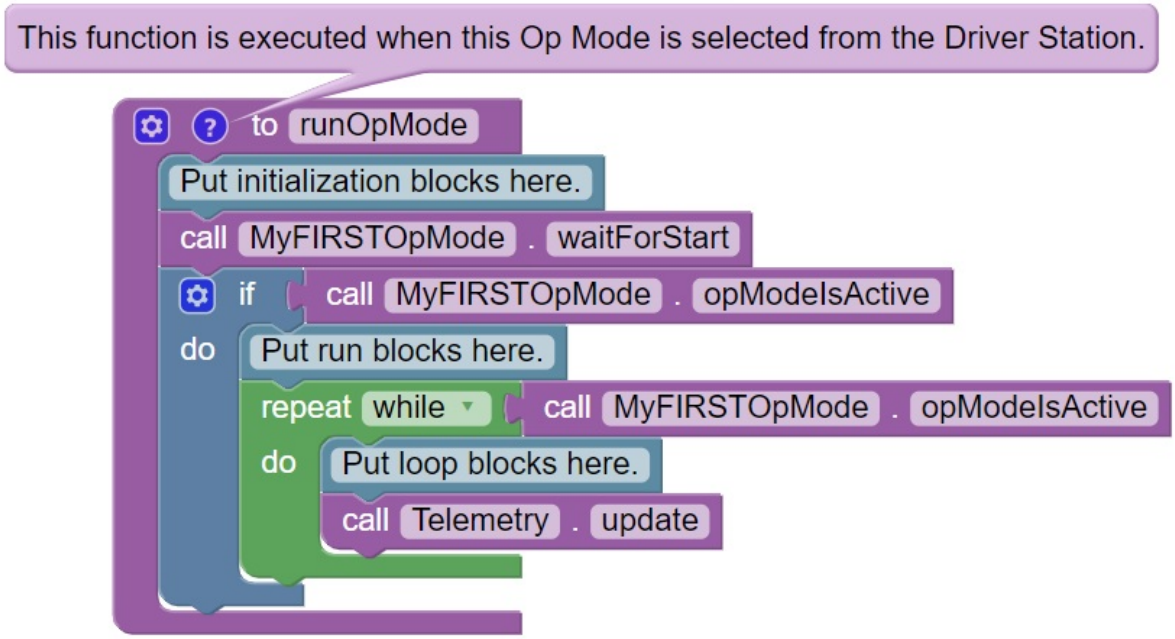
Writing an Op Mode

If you think about an op mode as a list of instructions for the robot, this set of instructions that you created will be executed by the robot whenever a team member selects the op mode called “MyFIRSTJavaOpMode” from the list of available op modes for this Robot Controller. For a linear op mode, the Robot Controller will process this list of tasks sequentially. Users can also use control loops (such as a while loop) to have the Robot Controller repeat (or iterate) certain tasks within a linear op mode.

> The following code walkthrough assumes you have already created an op mode and configured your hardware.

Blocks

When you create a new op mode, there should already be a set of programming blocks that are placed on the design canvas for your op mode. These blocks are automatically included with each new op mode that you create. They create the basic structure for your op mode.



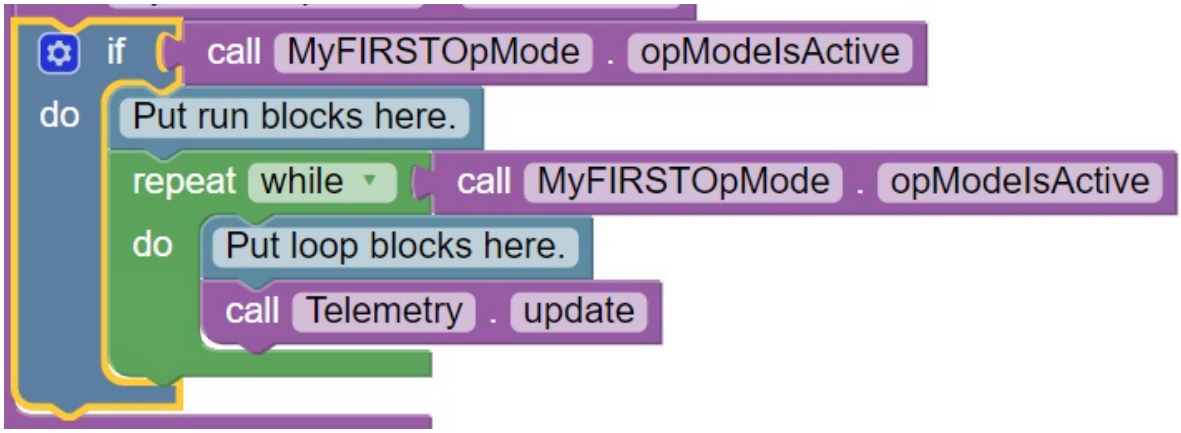
The blue colored block with the words “Put initialization blocks here” is a comment. Comments are placed in an op mode for the benefit of the human user. The robot will ignore any comments in an op mode.

Any programming blocks that are placed after the “Put initialization blocks here” comment (and before the “call MyFIRSTOpMode.waitForStart” block) will be executed when the op mode is first selected by a user at the Driver Station.

When the Robot Controller reaches the block labeled “call MyFIRSTOpMode.waitForStart” it will stop and wait until it receives a Start command from the Driver Station. A Start command will not be sent until the user pushes the Start button on the Driver Station. Any code after the “call

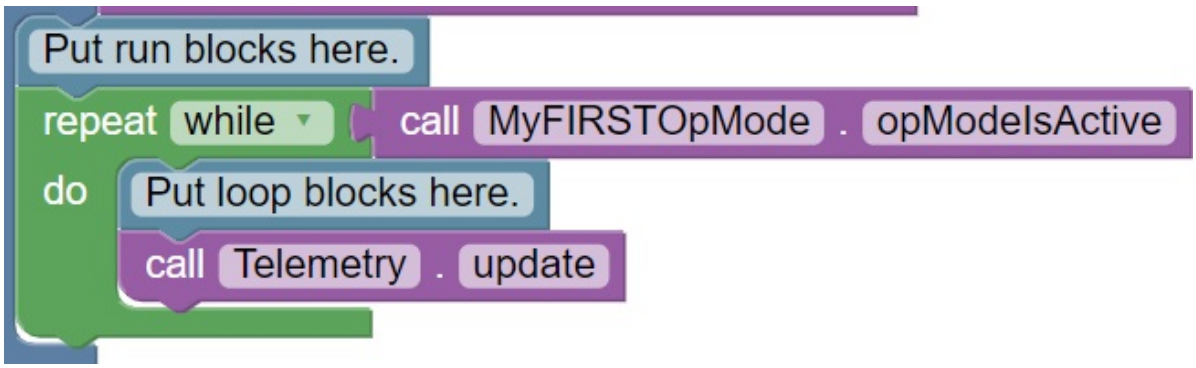
MyFIRSTOpMode.waitForStart” block will get executed after the Start button has been pressed.

After the “call MyFIRSTOpMode.waitForStart”, there is a conditional "if" block ("if call MyFIRSTOpMode.isActive") that only gets executed if the op mode is still active (i.e., a stop command hasn't been received).



Any blocks that are placed after the “Put run blocks here” comment and before the green block labeled “repeat while call MyFirstOpMode.opModeIsActive” will be executed sequentially by the Robot Controller after the Start button has been pressed.

The green block labeled “repeat while call MyFirstOpMode.opModeIsActive” is an iterative or looping control structure.



This green control block will perform the steps listed under the “do” portion of the block as long as the condition “call MyFIRSTOpMode.opModeIsActive” is true. What this means is that the statements included in the “do” portion of the block will repeatedly be executed as long as the op mode “MyFIRSTOpMode” is running. Once the user presses the Stop button, the “call MyFIRSTOpMode.opModeIsActive” clause is no longer true and the “repeat while” loop will stop repeating itself.

OnBot Java

For analytical purposes, the following code will be broken up into pieces and logically explained.

> As you are following through pay attention to the **syntax** of the statements. In most programming languages syntax plays an important part in the code being deciphered and run.

```
@TeleOp

public class MyFIRSTJavaOpMode extends LinearOpMode {
    private Gyroscope imu;
    private DcMotor motorTest;
    private DigitalChannel digitalTouch;
    private DistanceSensor sensorColorRange;
    private Servo servoTest;


    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        motorTest = hardwareMap.get(DcMotor.class, "motorTest");
        digitalTouch = hardwareMap.get(DigitalChannel.class, "digita
        sensorColorRange = hardwareMap.get(DistanceSensor.class, "se
        servoTest = hardwareMap.get(Servo.class, "servoTest");

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

At the start of the op mode there is an annotation that occurs before the class definition. This annotation states that this is a tele-operated (i.e., driver controlled) op mode:

```
@TeleOp
```

> If you wanted to change this op mode to an autonomous op mode, you would replace the “@TeleOp” with an “@Autonomous” annotation instead.

You can see from the sample code that an op mode is defined as a Java class. In this example, the op mode name is called “MyFIRSTJavaOpMode” and it inherits characteristics from the LinearOpMode class.

```
public class MyFIRSTJavaOpMode extends LinearOpMode {
```

You can also see that the OnBot Java editor created five private member variables for this op mode. These variables will hold references to the five configured devices that the OnBot Java editor detected in the configuration file of your Robot Controller.

```
1       private Gyroscope imu;
2       private DcMotor motorTest;
3       private DigitalChannel digitalTouch;
4       private DistanceSensor sensorColorRange;
5       private Servo servoTest;
```

Next, there is an overridden method called runOpMode. Every op mode of type LinearOpMode must implement this method. This method gets called when a user selects and runs the op mode.

```
    @Override
    public void runOpMode() {
```

At the start of the runOpMode method, the op mode uses an object named hardwareMap to get references to the hardware devices that are listed in the Robot Controller's configuration file:

```
        imu = hardwareMap.get(Gyroscope.class, "imu");
        motorTest = hardwareMap.get(DcMotor.class, "motorTest");
        digitalTouch = hardwareMap.get(DigitalChannel.class, "digita
        sensorColorRange = hardwareMap.get(DistanceSensor.class, "se
        servoTest = hardwareMap.get(Servo.class, "servoTest");
```

The hardwareMap object is available to use in the runOpMode method. It is an object of type HardwareMap class.

> When you attempt to retrieve a reference to a specific device in your op mode, the name that you specify as the second argument of the HardwareMap.get method must match the name used to define the device in your configuration file. For example, if you created a configuration file that had a DC motor named "motorTest", then you must use this same name (it is case sensitive) to retrieve this motor from the hardwareMap object.

In the next few statements of the example, the op mode prompts the user to push the start button to continue. It uses another object that is available in the runOpMode method. This object is called telemetry and the op mode uses the addData method to add a message to be sent to the Driver Station. The op mode then calls the update method to send the message to the Driver Station. Then it calls the waitForStart method, to wait until the user pushes the start button on the driver station to begin the op mode run.

```
        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();
```

> All linear op modes should have a waitForStart statement to ensure that the robot will not begin executing the op mode until the driver pushes the start button.

After a start command has been received, the op mode enters a while loop and keeps iterating in this loop until the op mode is no longer active (i.e., until the user pushes the stop button on the Driver Station):

```
        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
```

As the op mode iterates in the while loop, it will continue to send telemetry messages with the index of “Status” and the message of “Running” to be displayed on the Driver Station.

Controlling Actuators

In programming, information is constantly being exchanged. When communicating with the whole of the system (motors, servos, etc) the Control Hub is receiving input and, when coded correctly, using that input to perform an action. While its possible for the robot to run

autonomously off of the information exchange happening within its own system; there are many situations where **user input** is necessary.

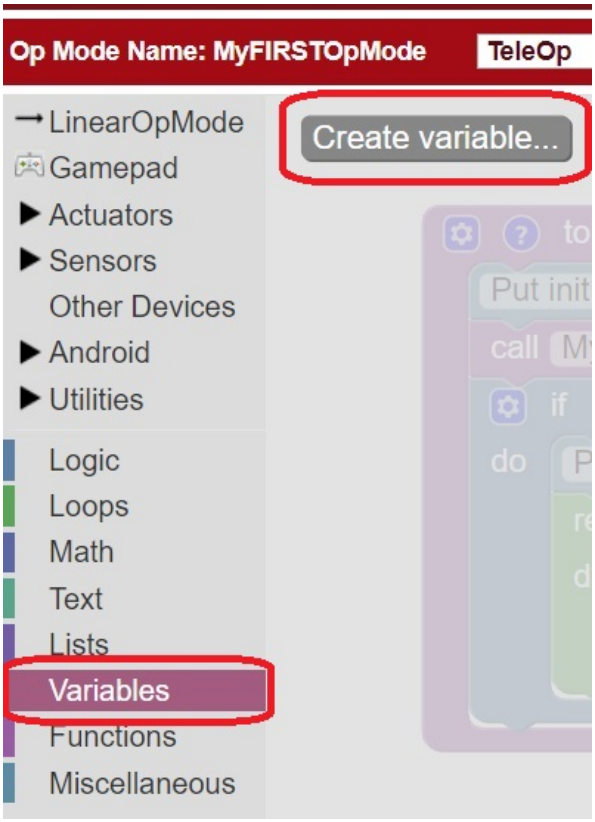
This section will cover how to assign input from a gamepad to control a motor and a servo.

Controlling a Motor

> The following code walkthrough assumes you have already created an op mode and configured your hardware.

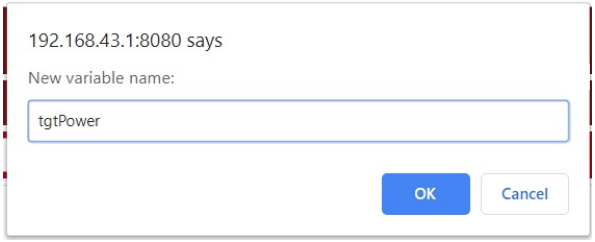
Blocks

On the left-hand side of the screen click on the category called “Variables” to display the list of block commands that are used to create and modify variables within your op mode.

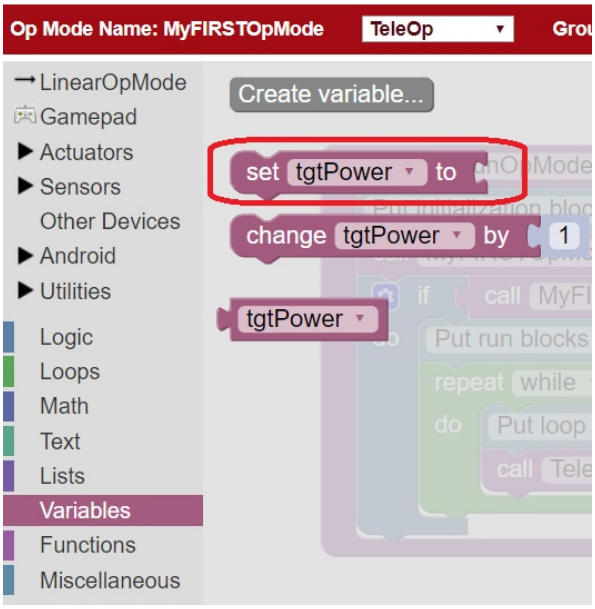


Click on “Create variable…” to create a new variable that will represent the target motor power for our op mode.

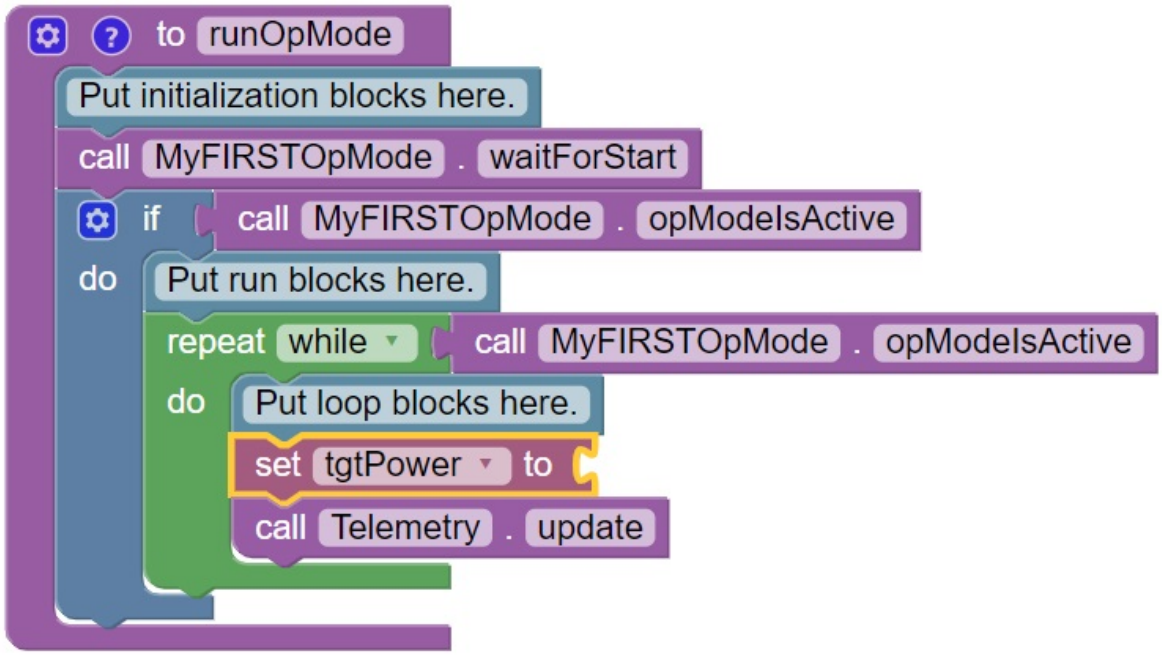
When prompted, type in a name (“tgtPower”) for your new variable.



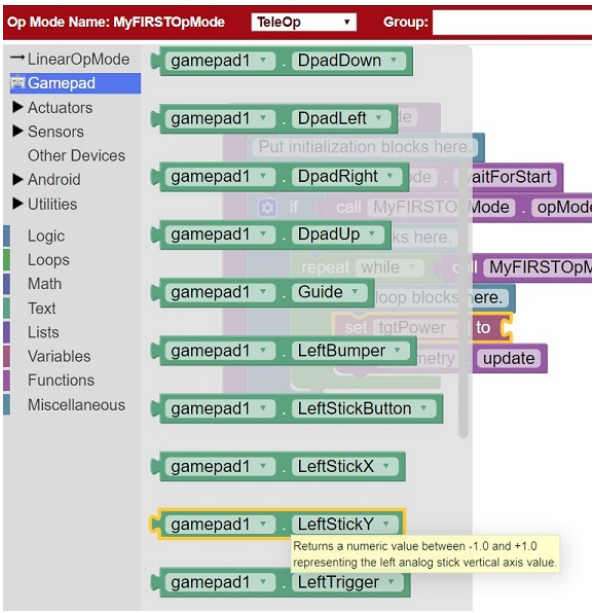
Once you have created your new variable, some additional programming blocks should appear under the “Variables” block category.



Click on the “set tgtPower to” programming block and then use the mouse to drag the block to the spot just after the “Put loop blocks here” comment block.

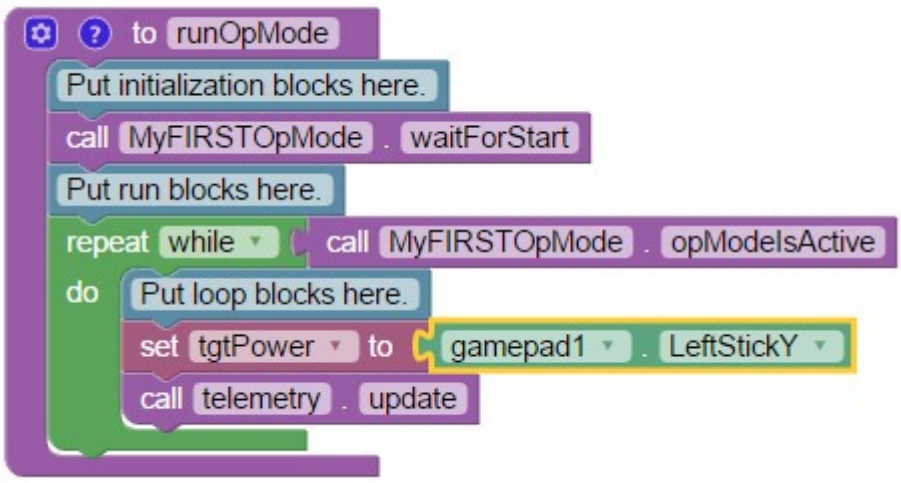


Click on the “Gamepad” category of the programming blocks and select the “gamepad1.LeftStickY” block from the list of available blocks.



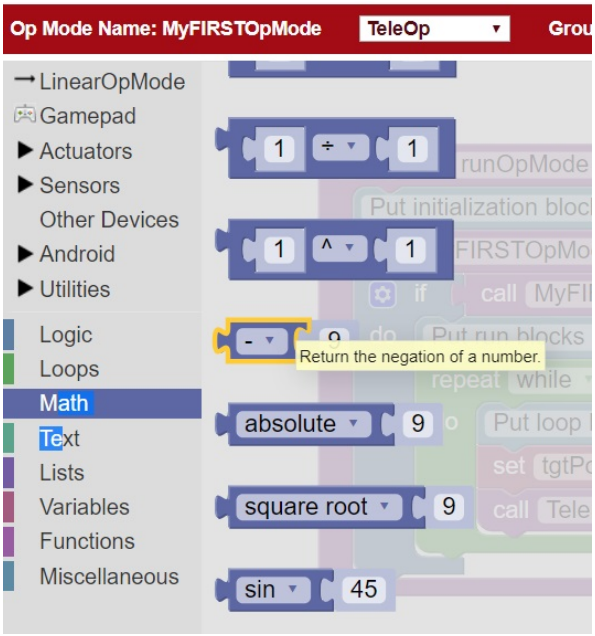
> The control system lets you have up to two gamepads controlling a robot. By selecting “gamepad1” you are telling the op mode to use the control input from the gamepad that is designated as driver #1.

Drag the “gamepad1.LeftStickY” block so it snaps in place onto the right side of the “set tgtPower to” block. This set of blocks will continually loop and read the value of gamepad #1’s left joystick (the y position) and set the variable tgtPower to the Y value of the left joystick.

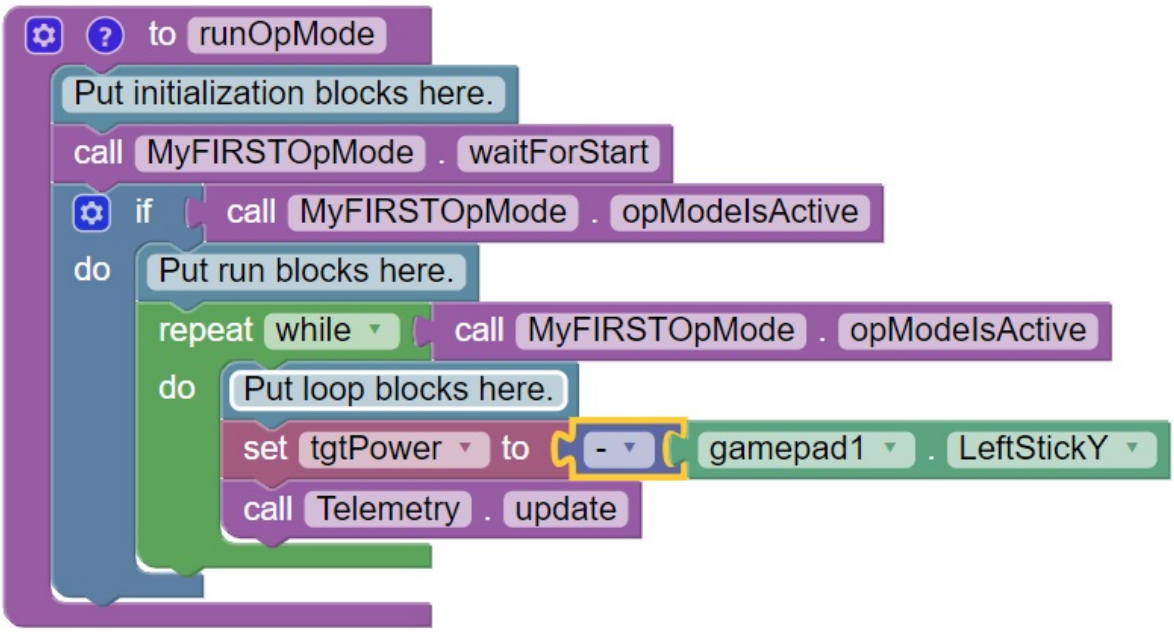


> For the F310 gamepads, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position. In our example, if the left joystick is pushed to the top, the variable tgtPower will have a value of -1.

Click on the “Math” category for the programming blocks and select the negative symbol (“-“).

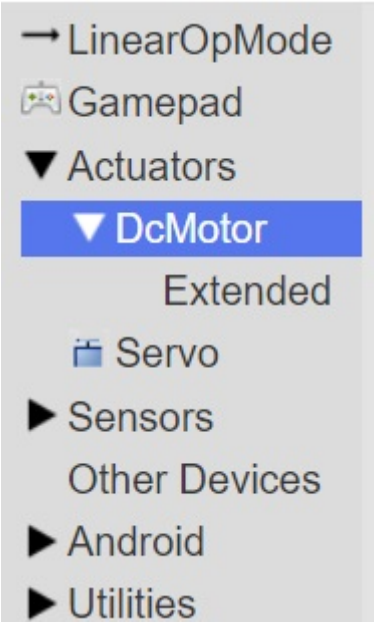


Drag the negative symbol (also known as a “negation operator”) to the left of the “gamepad1.LeftStickY” block. It should click in place after the “set tgtPower to” block and before the “gamepad1.LeftStickY” block.

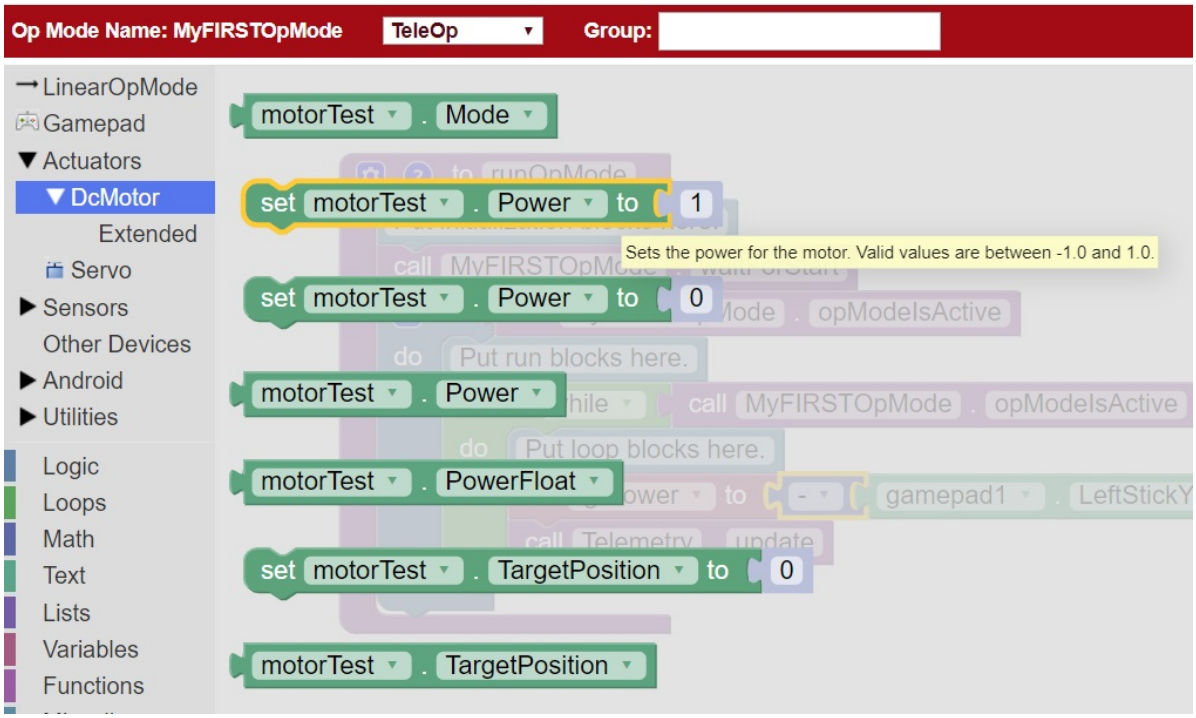


With this change, the variable tgtPower will be set to +1 if the left joystick is in its topmost position and will be set to -1 if the joystick is in its bottommost position.

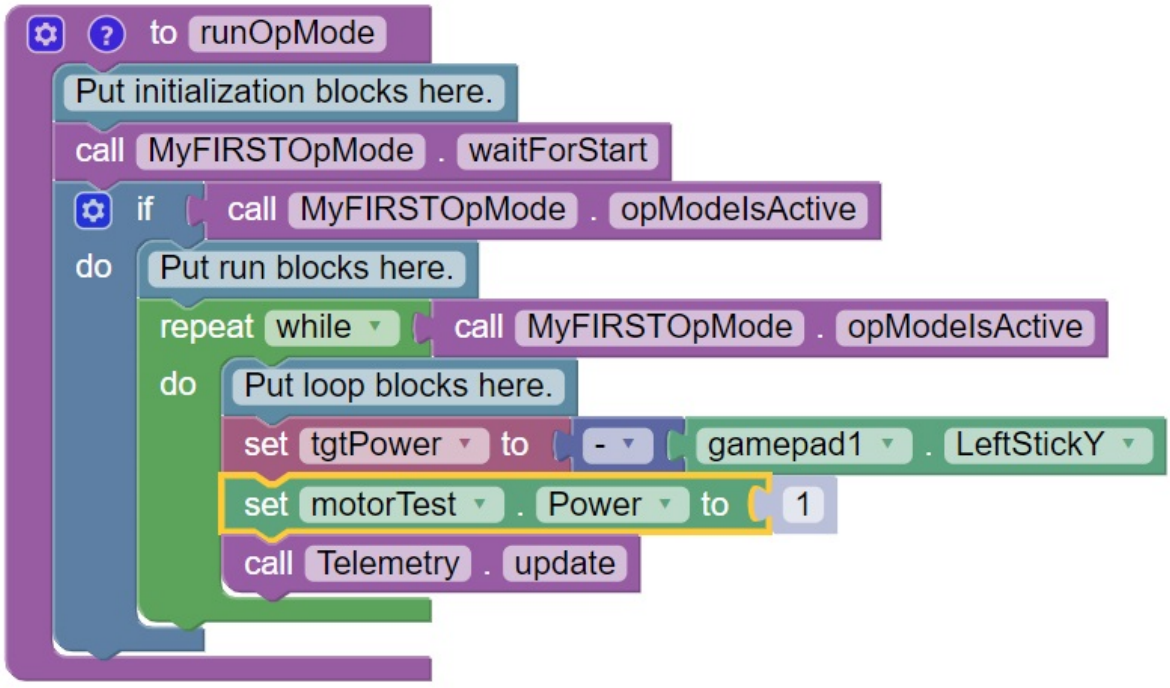
Click on the “Actuators” category of blocks. Then click on the "DcMotor" category of blocks.



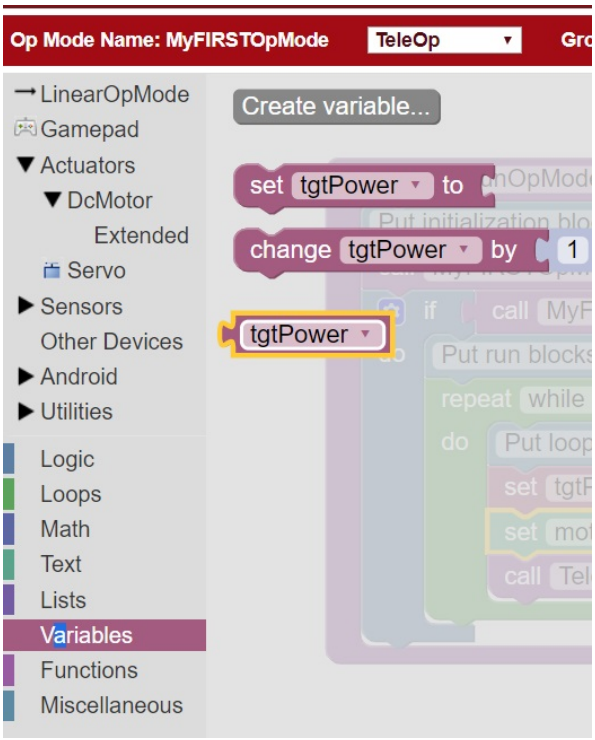
Select the “set motorTest.Power to 1” programming block.



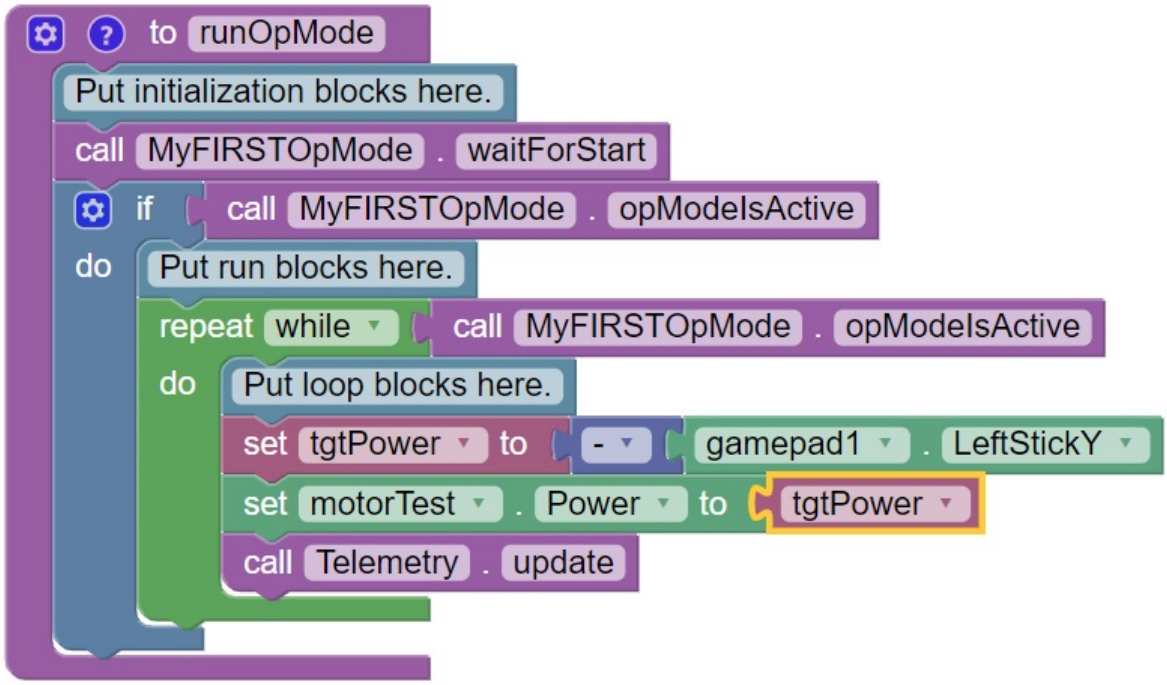
Drag and place the “set motorTest.Power to 1” block so that it snaps in place right below the “set tgtPower to” block.



Click on the “Variables” block category and select the “tgtPower” block.



Drag the “tgtPower” block so it snaps in place just to the right of the “set motor1.Power to” block.



The “tgtPower” block should automatically replace the default value of “1” block.

OnBot Java

Let's modify your op mode to control the DC motor that you connected and configured for your REV Expansion Hub or Control Hub. Modify the code for the program loop so that it looks like the following:

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    telemetry.addData("Target Power", tgtPower);
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

If you look at the code that was added, you will see that we defined a new variable called target power before we enter the while loop.

```
double tgtPower = 0;
```

At the start of the while loop we set the variable tgtPower equal to the negative value of the gamepad1's left joystick:

```
tgtPower = -this.gamepad1.left_stick_y;
```

The object gamepad1 is available for you to access in the runOpMode method. It represents the state of gamepad #1 on your Driver Station. Note that for the F310 gamepads that are used during the competition, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position. In the example code above, you negate the left_stick_y value

so that pushing the left joystick forward will result in a positive power being applied to the motor. Note that in this example, the notion of forwards and backwards for the motor is arbitrary. However, the concept of negating the joystick y value can be very useful in practice.

The next set of statements sets the power of motorTest to the value represented by the variable tgtPower. The values for target power and actual motor power are then added to the set of data that will be sent via the telemetry mechanism to the Driver Station.

```
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    telemetry.addData("Target Power", tgtPower);
    telemetry.addData("Motor Power", motorTest.getPower());
```

After you have modified your op mode to include these new statements, press the build button and verify that the op mode was built successfully.

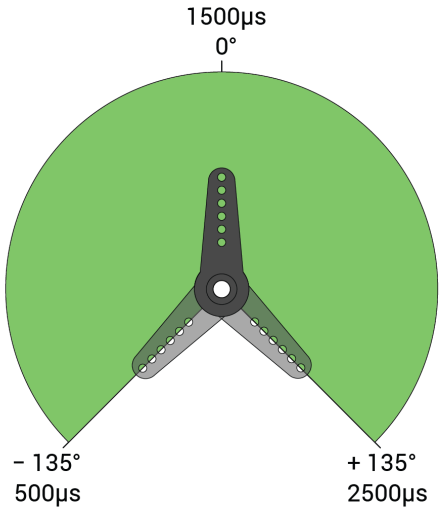
Controlling a Servo

Let’s modify your op mode to add the logic required to control a servo motor. For this example, you will use the buttons on the Logitech F310 gamepad to control the position of a REV Robotics Smart Robot Servo (REV-41-1097).

With a typical servo, you can specify a target position for the servo. The servo will turn its motor shaft to move to the target position, and then maintain that position, even if moderate forces are applied to try and disturb its position.

> This section is considering the Smart Robot Servo in its default mode. If your servo has been changed to function in continuous mode or with angular limits it will not behave the same using the code examples below. You can learn more about the Smart Robot Servo or changing the Servo's mode via the SRS Programmer by clicking the hyperlinks.

For both Blocks and OnBot Java, you can specify a target position that ranges from 0 to 1 for a servo. For a servo with a 270° range, if the input range was from 0 to 1 then a signal input of 0 would cause the servo to turn to point -135°. For a signal input of 1, the servo would turn to +135°. Inputs between the minimum and maximum have corresponding angles evenly distributed between the minimum and maximum servo angle.



In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway or

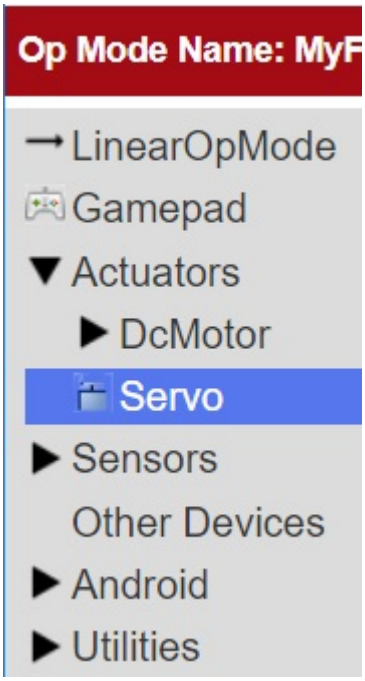
neutral position. If we use the above image of the pulse width range for the servo, this is representative of 0 degrees, but can also be consider 135 degrees of the full 270 degree range.

Pushing the yellow “Y” button will move the servo to the target position where signal input is 0. Pushing the blue “X” button or the red “B” button will move the servo to the target position where signal input is 0.5, which corresponds with the neutral position . Pushing the green “A” button will move the servo to the target position where signal input is 1.

> The following code walkthrough assumes you have already created an op mode and configured your hardware.

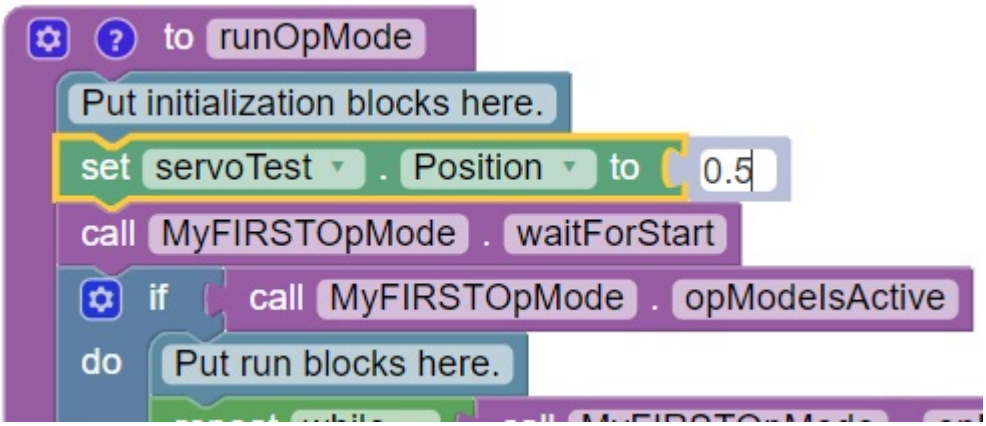
Blocks

On the left-hand side of the screen click on the category called “Actuators” and look for the subcategory called “Servos”.

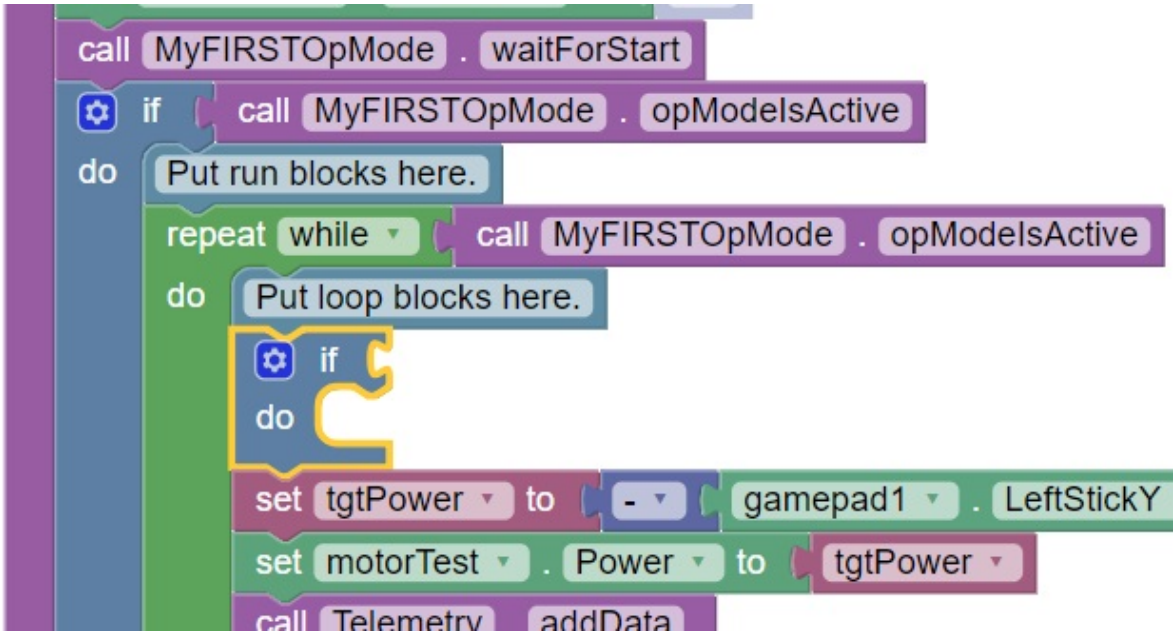


Select the “set servoTest.Position to” block from the list of available Servo blocks.

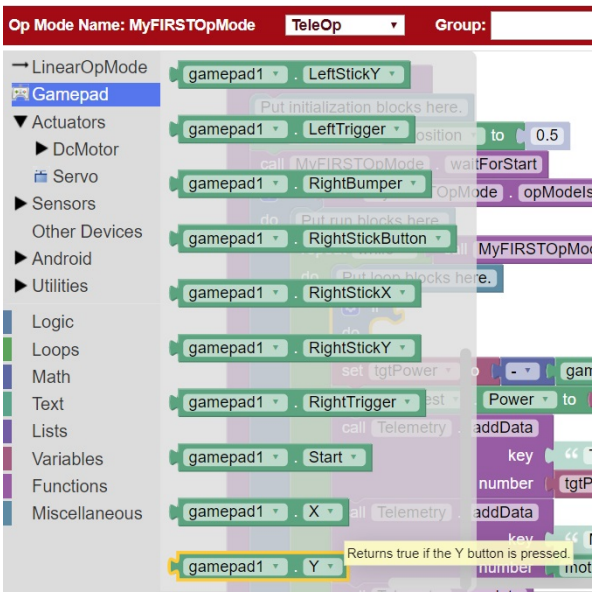
Drag the “set servoTest.Position to” block to the spot just under the comment block that reads “Put initialization blocks here.” The block should click into place. Change the number block to read "0.5" instead of "0"



Click on the “Logic” category of the programming blocks and select the “if do” block from the list of available blocks. Drag the block to the position immediately after the comment block that reads “Put loop blocks here.”



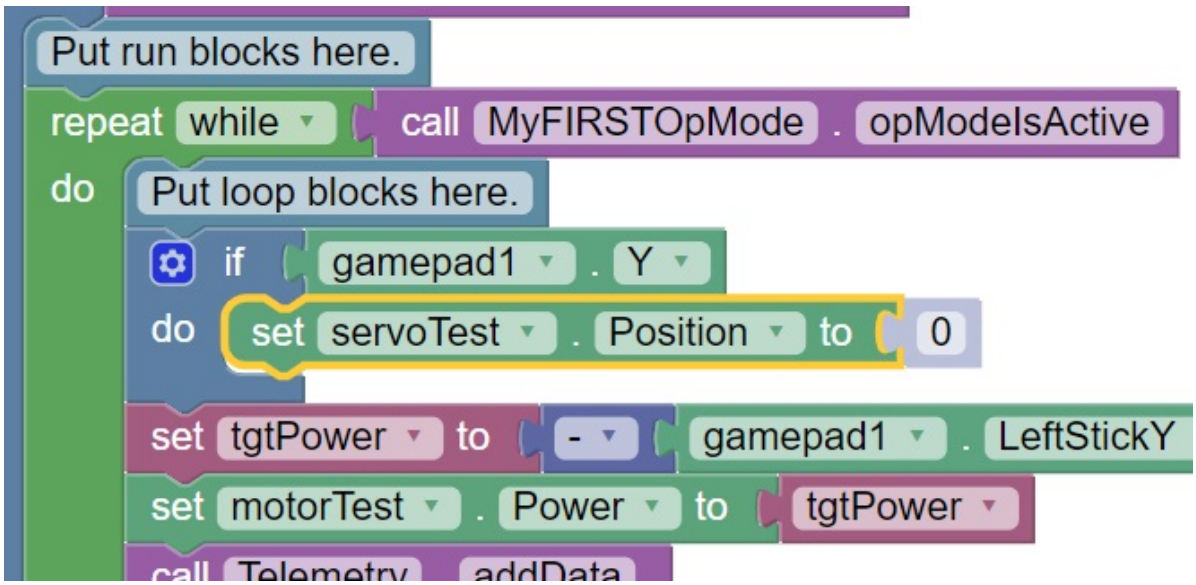
Click on the “Gamepad” category of the programming blocks and select the “gamepad1.Y” block from the list of available blocks.



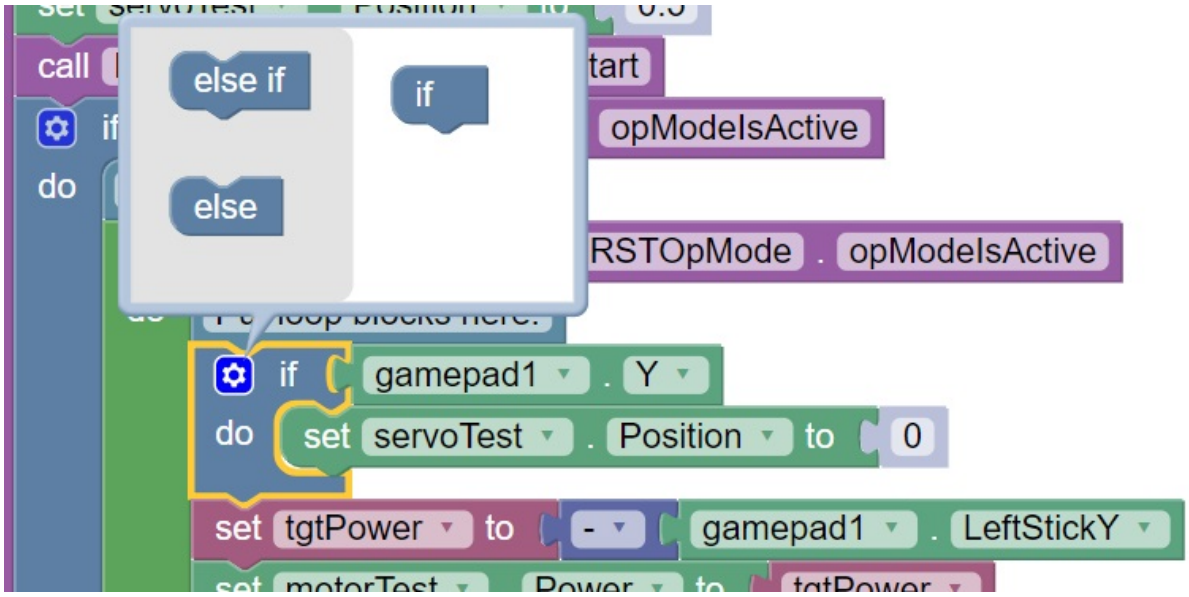
Drag the “gamepad1.Y” block to the right side of the “if do” block. The block should click into place.

> The “if do” block will use the state of the gamepad1.Y value its test condition. If the “Y” button is pressed, the statements within the “do” portion of the block will be executed.

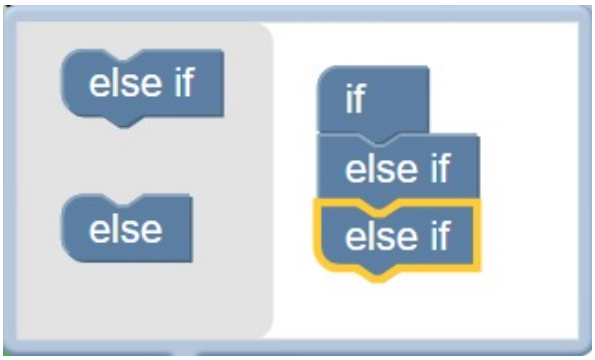
Select the “set servoTest.Position to” block from the list of available Servo blocks. Drag the “set servoTest.Position to” block so that it snaps in place in the do portion of the “if do” block.



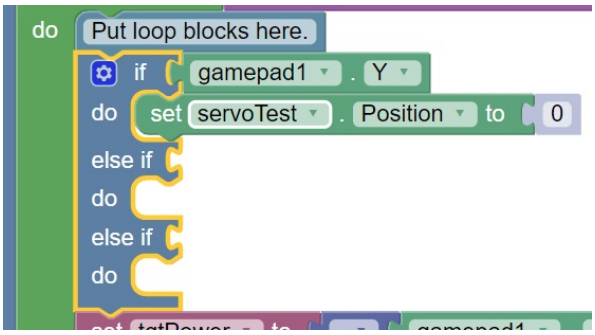
Click on the blue and white Settings icon for the “if do” block. This will display a pop-up menu that lets you modify the “if do” block.



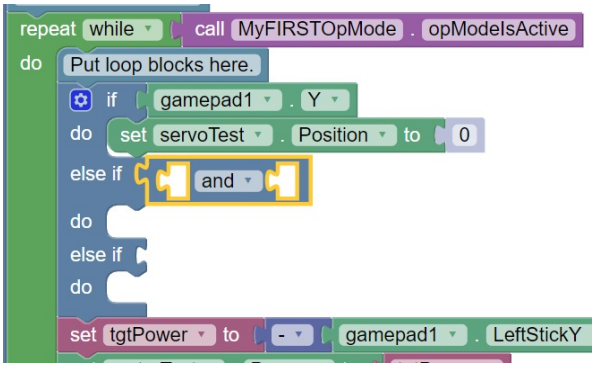
Drag an “else if” block from the left side of the pop-up menu and snap it into place under the “if” block. Drag a second “else if” block from the left side and snap it into place on the right side under the first “else if” block.



Click on the Settings icon to hide the pop-up menu for the “if do” block. The “if do” block should now have two “else if” test conditions added.



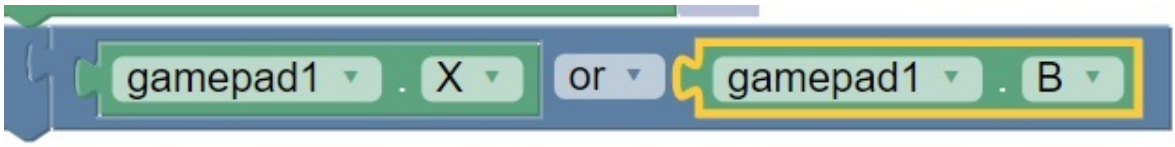
Click on the “Logic” category and select the logical “and” block. Drag the “and” block so it clicks in place as the test condition for the first “else if” block.



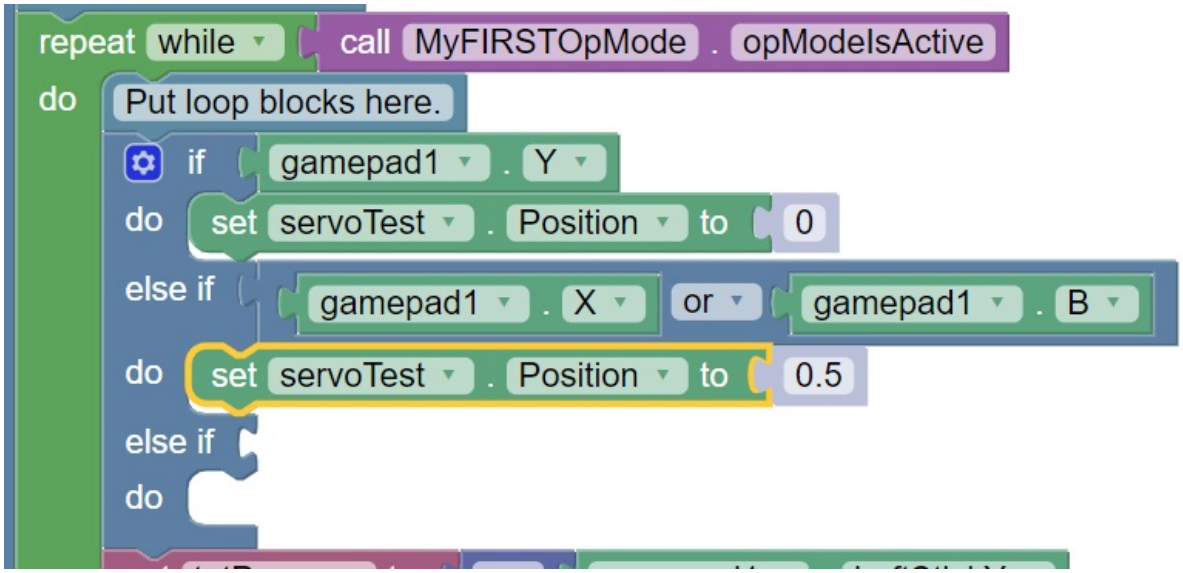
Click on the word “and” and select “or” from the pop-up menu to change the block to a logical “or” block.

Click on the “Gamepad” category and select the “gamepad1.X” block. Drag the block so that it clicks in place as the first test condition of the logical “or” block. The select

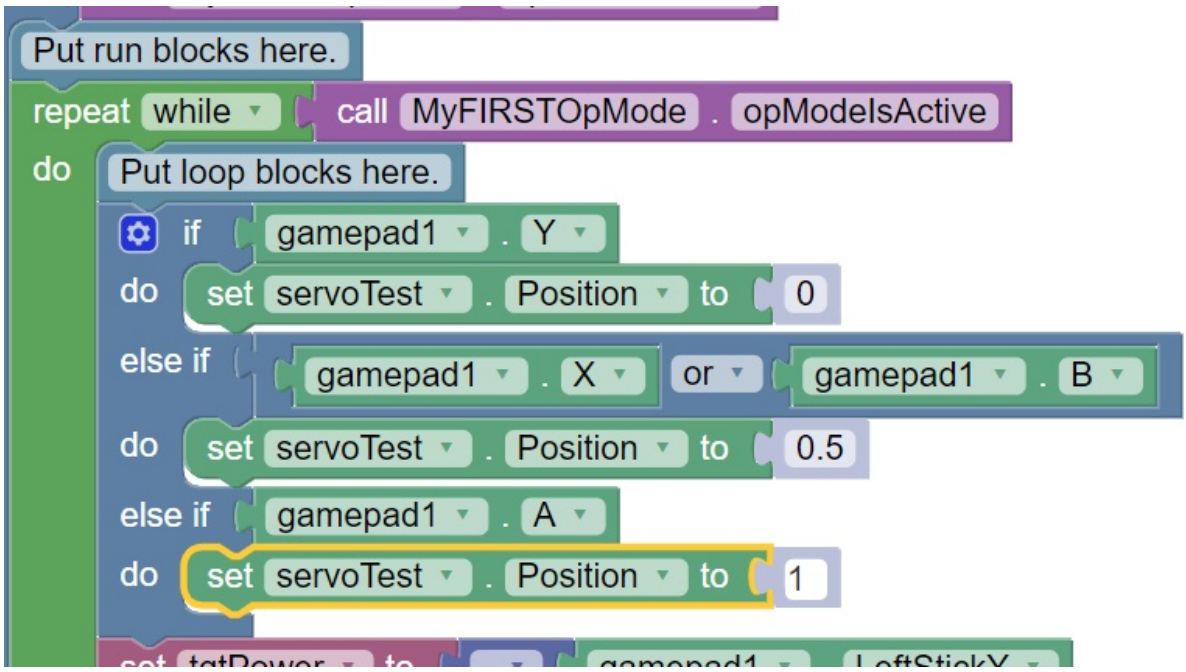
the "gamepad1.B" and drag it into the second test condition of the "or" bloc



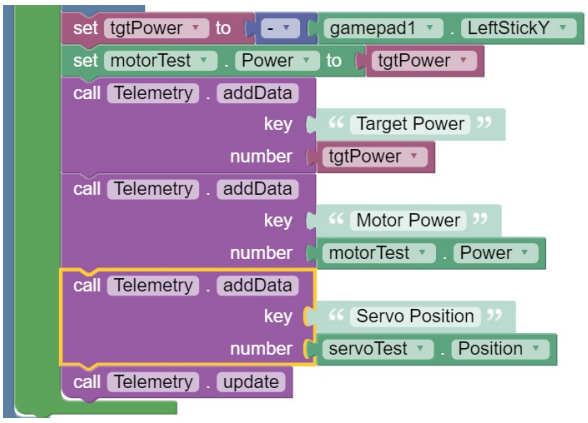
Select a “set servoTest.Position to” block and place it into “do” clause of the first else-if block. Highlight the number “0” and change it to “0.5”. With this change, if the user presses the “X” button or “B” button on gamepad #1, the op mode will move the servo to the neutral position.



Use a “gamepad1.A” block as the test condition for the second “else if” block. Drag a “set servoTest.position to” block to the do clause of the second “else if” block and modify the numeric value so that the servo’s position will be set to a value of 1.



Insert a “call telemetry.addData” block (numeric) before the “call Telemetry.update” block. Rename the key field to “Servo Position” and insert a “servoTest.Position” block for the number field.



This set of blocks will send the current servo position value to the Driver Station while the op mode is running.

OnBot Java

In order to cover the three positions discussed in the scenario above and **If/else statement** needs to be utilized.

If/else statements need to have a condition in order to choose which part of the statement gets triggered. In this case the servo needs to move to position 0 when the "Y" button is pressed.

```
if(gamepad1.y) {
        // move to -135 degrees.
        servoTest.setPosition(0);
```

The gamepad1.y looks for a press of the y button on gamepad #1. The line servoTest.setPosition(0); is setting the position of the servo "servoTest" to 0.

This process can be mostly replicated to apply to the other buttons and other positions in the example using "else if". However, the example asks that both the "X" and "B" be assigned to position 0.5. Which means that the logical or "||" needs to be used to signify that if button "X" or button "B" is pressed.

```
if(gamepad1.y) {
        // move to -135 degrees.
        servoTest.setPosition(0);
    } else if (gamepad1.x || gamepad1.b) {
        // move to 0 degrees.
        servoTest.setPosition(0.5);
    } else if (gamepad1.a) {
        // move to + 135 degrees.
        servoTest.setPosition(1);
    }
```

If you are following along with the full guide for the "MyFIRSTOpMode" the following code works the servo code into the full op mode.

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    // check to see if we need to move the servo.
    if(gamepad1.y) {
        // move to -135 degrees.
        servoTest.setPosition(0);
    } else if (gamepad1.x || gamepad1.b) {
        // move to 0 degrees.
        servoTest.setPosition(0.5);
    } else if (gamepad1.a) {
        // move to + 135 degrees.
        servoTest.setPosition(1);
    }
    telemetry.addData("Servo Position", servoTest.getPosition());
    telemetry.addData("Target Power", tgtPower);
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

This added code will check to see if any of the colored buttons on the F310 gamepad are pressed. The op mode will also send telemetry data on the servo position to the Driver Station.

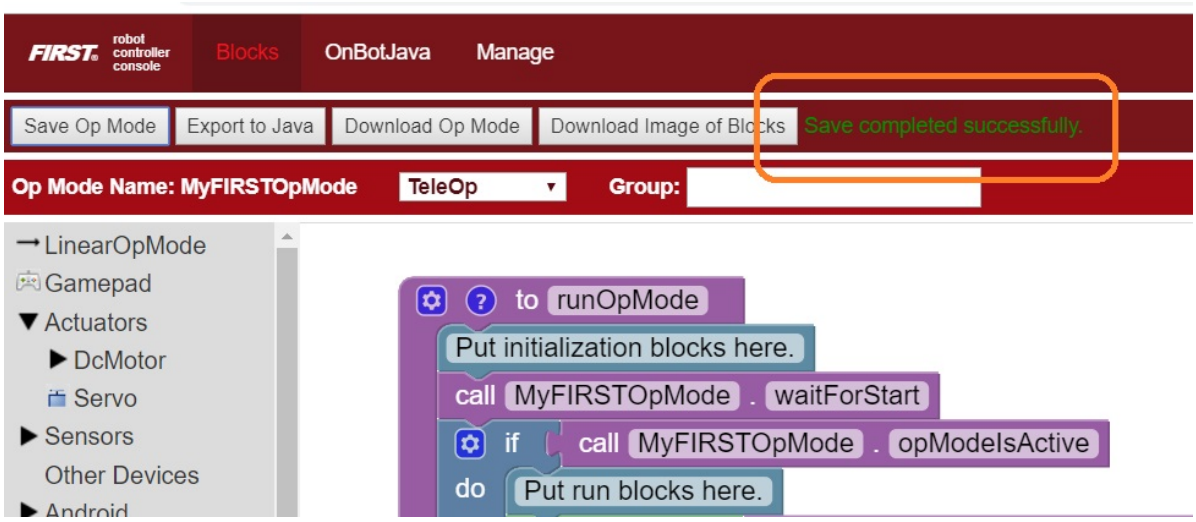
Saving or Building and Op Mode

Blocks

After you have modified your op mode, it is very important to save the op mode to the Robot Controller.

> It will take an estimated 1 minute to complete this task.

Press the “Save Op Mode” button to save the op mode to the Robot Controller. If your save was successful, you should see the words “Save completed successfully” to the right of the buttons.

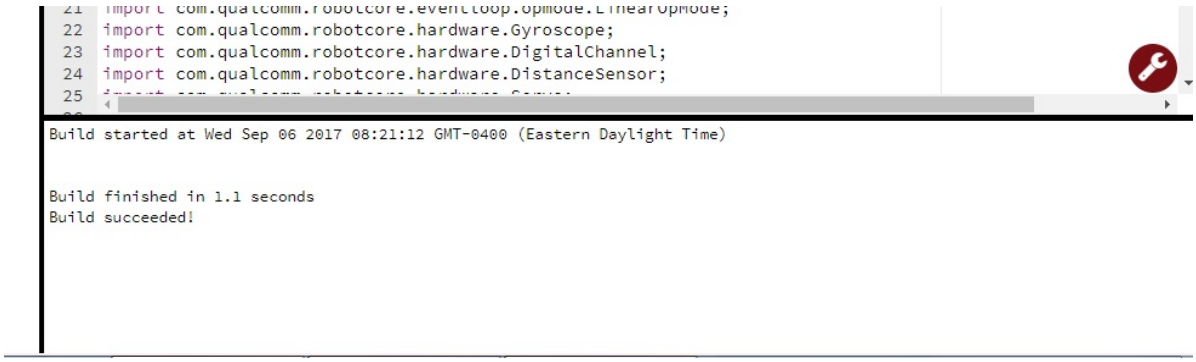


OnBot Java

When you create or edit an op mode the OnBot Java editor will auto-save the .java file to the file system of the Robot Controller. However, before you can execute your changes on the Robot Controller, you must first build the op mode and convert it from a Java text file to a binary that can be loaded dynamically into the FTC Robot Controller app.

If you are satisfied with your op mode and are ready to build, press the Build button (which is the button with the wrench symbol, see image below) to start the build process. Note that the build process will build **all of the .java files** on your Robot Controller.

You should see messages appear in the message pane, which is located in the lower right hand side of the window. If your build was successful, you should see a “Build succeeded!” message in the message pane.



Once you have built the binary files with your updated op modes, they are ready to run on the Robot Controller.

Troubleshooting Common Issues

One of the key aspects of troubleshooting is understanding the most common issues that occur in a system. Because of the structural differences of the languages; the common issues in Blocks and OnBot Java differ.

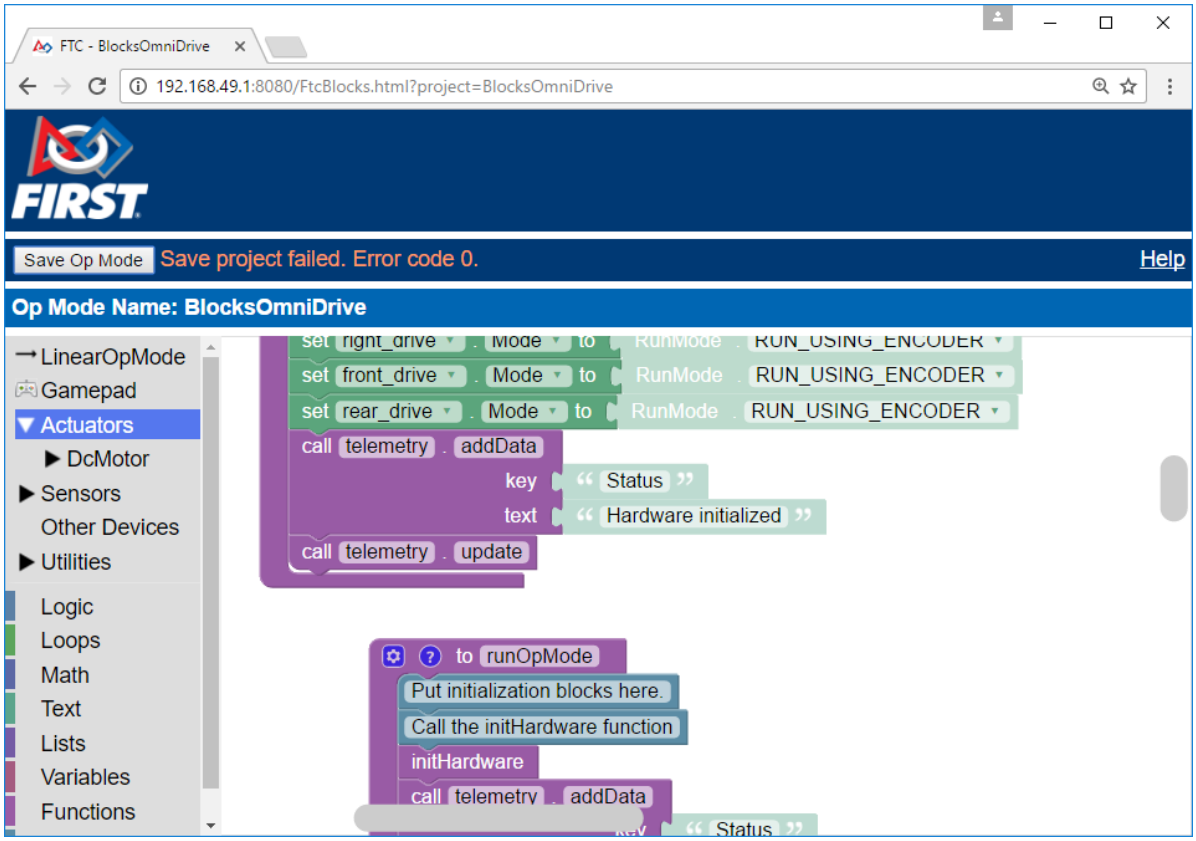
Blocks self contained syntax reduces the amount of potential syntax errors that are typically a part of programming. The two major Blocks errors relate to save issues.

In contrast, most major errors with OnBot Java relate to syntax. This section will cover what potential error codes will appear with even small changes in syntax.

Blocks

"Save Project Failed, Error code 0."

If you attempt to save the op mode that you are currently editing, but you receive an error message indicating that the “Save project failed. Error code 0.” you might have not be connected to the blocks programming mode sever.



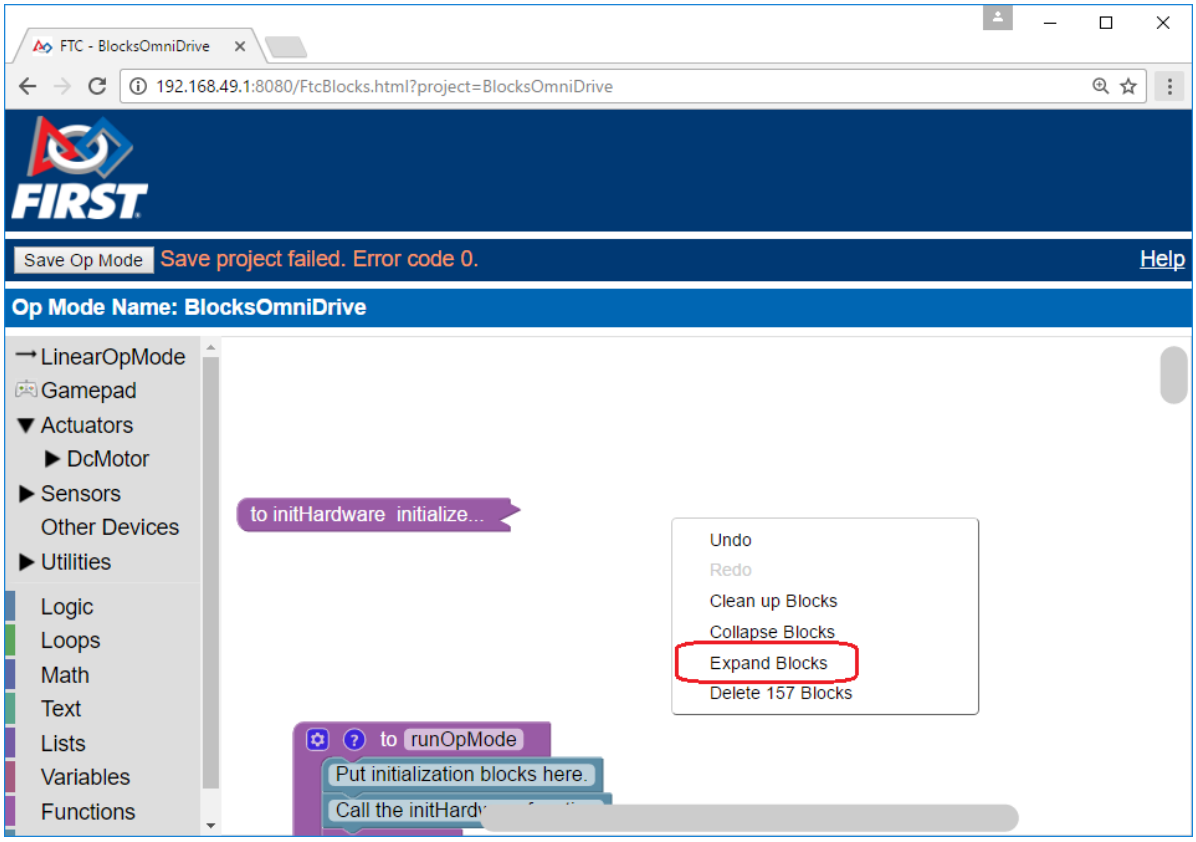
To correct this issue, you will need to reconnect to the blocks programming server on the Control Hub.

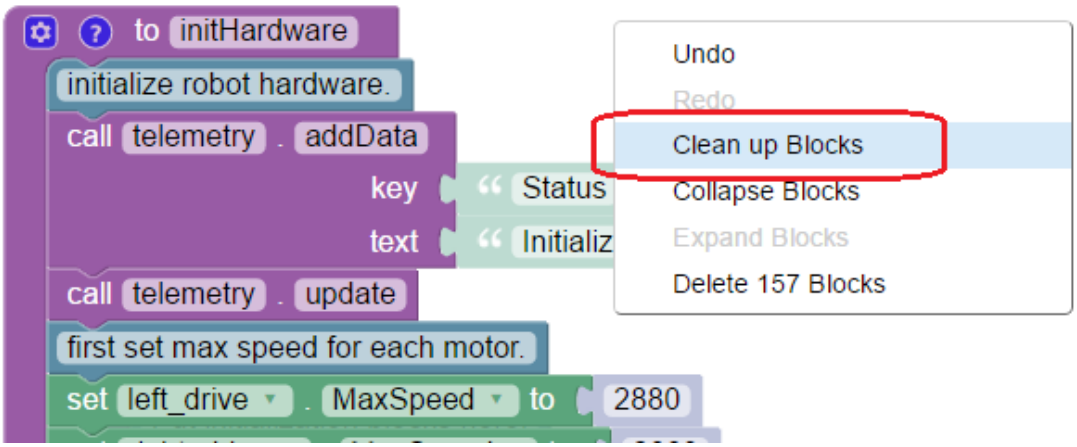
1. Make sure that your laptop is connected to the blocks programming mode Wi-Fi network
2. Press the “Save Op Mode” button again to re-attempt the save operation.

Op Mode Blocks are Missing

If you have opened an existing op mode to edit it in your Javascript-enabled browser, but the programming blocks are missing, check the following:

1. Did you remember to save the op mode the last time you edited and then exited the op mode? If you did not save the op mode after the last editing session, you might have lost some of your changes.
2. Are the blocks collapsed and/or in an area of the design “canvas” (or design pane) that is outside your current browser window? If so, you can use the expand and cleanup functions of the blocks programming tool, seen in the images below" to expand all of the blocks on your screen and to organize them in an easy-to-view (and easy-to-find) manner.




to initHardware
initialize robot hardware.
call telemetry . addData
key
Status
text
Initializ
call telemetry . update
first set max speed for each motor.
set left_drive . MaxSpeed to
2880
Undo
Redo
Clean up Blocks
Collapse Blocks
Expand Blocks
Delete 157 Blocks

OnBot Java

Troubleshooting Build Messages

In the previous section, the build process went smoothly. Let's modify your op mode slightly to cause an error in the build process.

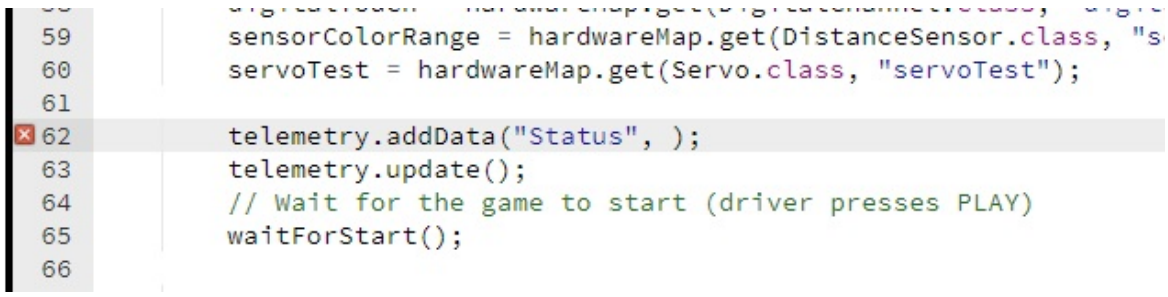
In the editing pane of the OnBot Java window, look for the line that reads "private Servo servoTest;". This should appear somewhere near the beginning of your op mode class definition. Change the word "Servo" to the word "Zervo":

```
private Zervo servoTest;
```

Also, let's modify the telemetry statement that informs the user that the op mode has been initialized, and let's remove one of the two arguments so that the statement looks like this:

```
telemetry.addData("Status",);
```

Note that when you eliminate the second argument, a little "x" should appear next to the line with the modified addData statement. This "x" indicates that there is a syntax error in the statement.



After you have modified your op mode, you can press the build button and see what error messages appear.

```
Build started at Wed Sep 06 2017 09:03:41 GMT-0400 (Eastern Daylight Time)
org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java line 62, column 37: ERROR: illegal start of expression


Build finished in 0.2 seconds
Build FAILED!
```

When you first attempt to build the op mode, you should get an “illegal start of expression error”. This is because the addData method is missing its second argument. The OnBot Java system also directs you to the file that has the error, and the location within the file where the error occurs.

In this example, the problem file is called “org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java” and the error occurs at line 62, column 37. It is important to note that the build process builds all of the .java files on the Robot Controller. If there is an error in a different file (one that you are not currently editing) you will need to look at the file name to determine which file is causing the problem.

Let’s restore this statement back to its original, correct form:

```
telemetry.addData("Status", "Initialized");
```

After you have corrected the addData statement, push the build button again to see what happens. The OnBot Java system should complain that it cannot find the symbol “Zervo” in a source file called “org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java” at line 51, column 13.

```
Build started at Wed Sep 06 2017 09:10:46 GMT-0400 (Eastern Daylight Time)
org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java line 51, column 13: ERROR: cannot find symbol
  symbol:   class Zervo
  location: class org.firstinspires.ftc.teamcode.MyFIRSTJavaOpMode


Build finished in 0.4 seconds
Build FAILED!
```

You should restore the statement back to its original form and then push the build button and verify that the op mode gets built properly.

```
    private Servo servoTest;
```