

REV Control System

Introduction

The REV Control System is an affordable robotics control platform providing the interfaces required for building robots. These devices are most commonly used within the FIRST Tech Challenge (FTC), FIRST Global Challenge (FGC), and in the classroom for educational purposes.

How to use this documentation?

This documentation is intended as the place to answer any questions related to the REV Robotics Control System, used in the FIRST Tech Challenge and FIRST Global Challenge.

- **Looking to get an idea of how to use the system before your Control Hub arrives?** Reading through each section will help, but we specifically recommend the guides on [getting started with the Control Hub](#) and the [programming language options](#) section.
- **Have a specific question?** Feel free to head straight to it using the navigation bar to the left. Each section is grouped with other topics that are similar.
- **Having trouble finding what you are looking for?** Try the search bar in the upper right or read the section descriptions below to find the best fit.

Getting started building robots can be an intimidating process. The following documentation is here to make getting started a bit easier. There are a number of examples to get started with the Control System and we are committed to adding content to make it more accessible for people to use REV.

If there is a question that is not answered by this space, send our support team an email; support@revrobotics.com. We are happy to help point you in the right direction.

What is in each section?

Control System Overview

This section contains information regarding all of the major mechanical specifications of the REV [Control Hub](#) and [Expansion Hub](#). These sections include [port pinout](#) information, [protection features](#), and the [types of cables](#) used with the devices.

Getting Started

Take the Control Hub or Expansion Hub from out of the box through generating the first configuration file. This includes the process for changing your Control Hub's [Name](#) and [Password](#) as well as [pairing to your](#)

Driver Station. Also includes information on ways to add additional motors to the Control System through adding a **SPARKmini Motor Controller** or an **Expansion Hub**.

Managing the Control System

This section covers how the information needed to keep your Control System up to date and how to **troubleshoot your Control System** if issues arise.

Programming

From just getting started by writing your **first Op Mode** to working with **closed loop control**, this section covers the information needed to start programming.

Sensors

Sensors are often vital for robots to gather information about the world around them. Use this section to find how to use REV sensors and information on the different sensor types.

Control System Overview

Control Hub Basics

The REV Robotics Control Hub ([REV-31-1595](#)) is an affordable all in one educational robotics controller that provides the interfaces required for building robots, as well as other mechatronics, with multiple programming language options. The Control Hub was designed and built as an easy to use, dependable, and durable device for use in classroom and the competition. It features an Android operating system, and a mature software package designed for both basic and advanced use cases. When the Control Hub software is updated with performance enhancements and features, the controller can receive a "field upgrade," through an update process that is fast and simple.

The Control Hub is an approved device for use in FIRST® Global and FIRST Tech Challenge.



- Physical Dimensions
 - 143mm X 103mm X 29.5 mm
 - Mounting holes on a 16mm spacing
- Input Voltage
 - 12V nominal (8-15V DC)
- Processors
 - RK3328 Quad-core ARM® Cortex-A53
 - Texas Instruments ARM® Cortex-M4
- 3.3V Ports
 - 8x digital I/O: 1A source max
 - 4x I2C 100kHz/400kHz busses: 500mA source max
 - 4x 12-bit analog inputs: 500mA source max
 - 4x quadrature encoder inputs: 500mA source max
- 5V Ports
 - +5V Power: 2A source max

- Servos: 2A source maximum per pair (0-1, 2-3, 4-5)
- USB 2.0: 1.5A source max
- USB 3.0: 1.5A source max

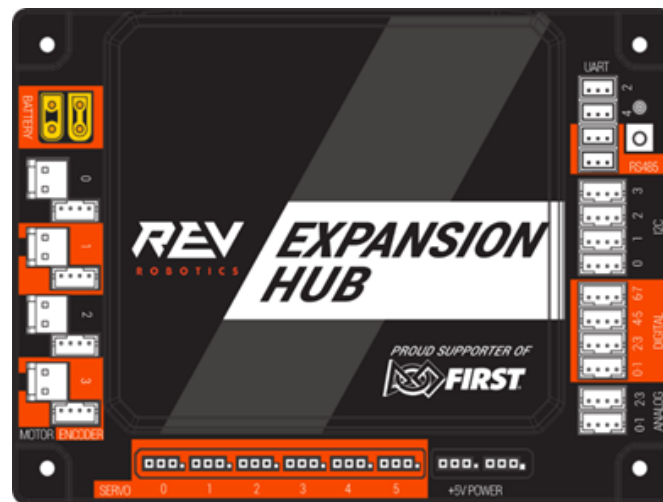
PORT LABEL	QTY	CONNECTOR	DESCRIPTION
Battery	2	XT30	Connect one 12V NiMh battery, add an Expansion Hub with second port
Motor	4	JST VH, 2-pin	Motor power output
Encoder	4	JST PH, 4-pin	Quadrature encoder input
Servo	6	0.1" Header	Extended range 5V servo output (500-2500ms)
+5V Power	2	0.1" Header	Auxiliary device 5V/2A
Analog	4	JST PH, 4-pin	Analog input 0-5.0V measurement range with two channels per connector. 3.3V provided on the connector power pin.
Digital	8	JST PH, 4-pin	Digital Input/Output with two channels per connector
I2C	4	JST PH, 4-pin	Four separate I2C busses, 100kHz/400kHz bus speed
RS485	2	JST PH, 3-pin	Use this serial communication port to add an Expansion Hub
UART	2	JST PH, 3-pin	Debugging only
USB C	1	USB C	Connect directly to the Control Hub via PC, USB 2.0
USB 2.0	1	USB A	Connect USB cameras and other USB peripherals to the Control Hub
USB 3.0	1	USB A	Connect USB cameras and other USB peripherals to the Control Hub
HDMI	1	HDMI A	Supports 4k @ 60Hz

Expansion Hub Basics

The REV Robotics Expansion Hub (REV-31-1153) is a low-cost education device that can communicate with any computer (Commonly the REV Robotics Control Hub or an Android Phone) to provide the interfaces required for building robots and other mechatronics. The Expansion Hub was purposely built to stand up to the rigors of the classroom and competition field. It features a mature firmware designed for basic and advanced use cases with the ability to be field upgraded in the future.

The IO ports of the Expansion Hub are identical in specification to the Control Hub. Within this documentation, many sections may refer to the Control Hub, but the connections are the same for the Expansion Hub.

The REV Robotics Expansion Hub is an approved device for use in the FIRST Tech Challenge and FIRST Global.



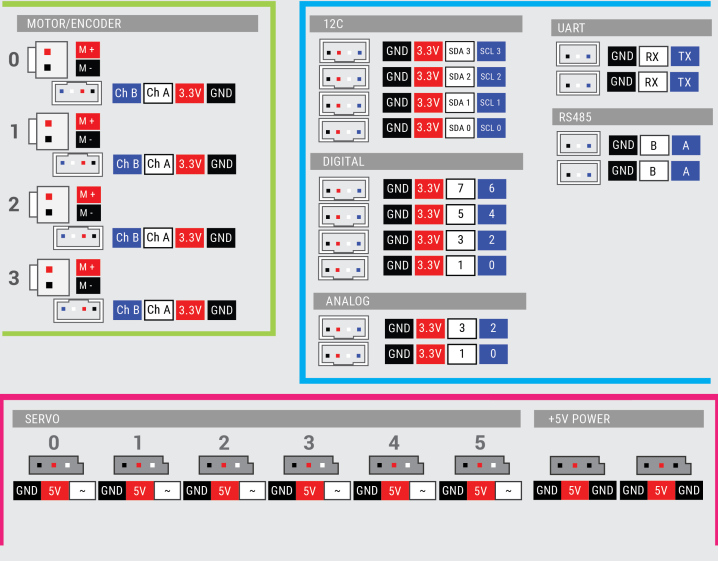
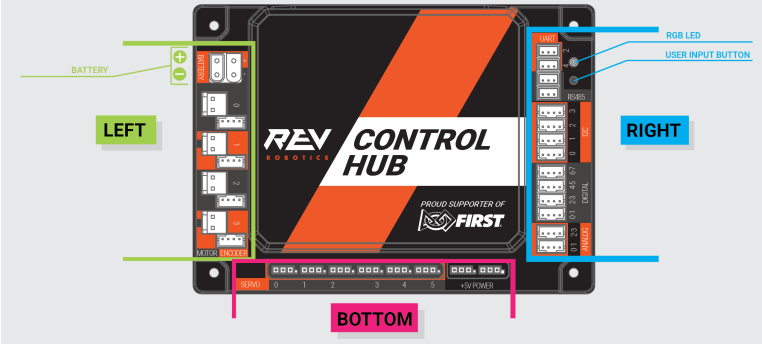
- Physical Dimensions
 - 143mm X 103mm X 29.5 mm
 - Mounting holes on a 16mm spacing
- Input Voltage
 - 12V nominal (8-15V DC)
- Processors
 - Texas Instruments ARM® Cortex-M4
- 3.3V Ports
 - 8x Digital I/O: 1A source max
 - 4x I2C 100kHz/400kHz Buses: 500mA source max
 - 4x 12-bit Analog Inputs: 500mA source max
 - 4x Quadrature Encoder Inputs: 500mA source max

- 5V Ports
 - +5V Power: 2A source max
 - Servos: 2A source maximum per pair (0-1, 2-3, 4-5)

PORT LABEL	QTY	CONNECTOR	DESCRIPTION
Battery	2	XT30	Connect one 12V NiMh battery, add an Expansion Hub with second port
Motor	4	JST VH, 2-pin	Motor power output
Encoder	4	JST PH, 4-pin	Quadrature encoder input
Servo	6	0.1" Header	Extended range 5V servo output (500-2500ms)
5V Aux Power	2	0.1" Header	Auxiliary device 5V/2A
Analog	4	JST PH, 4-pin	Analog input 0-3.3V with two channels per connector
Digital	8	JST PH, 4-pin	Digital Input/Output with two channels per connector
I2C	4	JST PH, 4-pin	Four separate I2C busses, 100kHz/400kHz bus speed
RS485	2	JST PH, 3-pin	Serial communication port to add a Hub (Control or Expansion)
UART	2	JST PH, 3-pin	Debugging only
MINI USB	1	USB Mini-B	Connect directly to the Robot Controller Android device or PC

Port Pinouts

CONTROL HUB PIN OUT



Protection Features

The Control ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) were designed with a number of protection features built into the device. These include the following:

- Reverse battery input protection
- Electrostatic discharge (ESD) protection on all connections
- Over-current protection
 - on all power buses
 - Digital I/O bus
 - I2C bus
 - Analog bus
 - USB
 - Servo bus per pair (0-1, 2-3, 4-5)
 - Encoder bus
- Over-current monitoring for individual Motor Channels
- Keyed and locking connectors
- Fail-safe mode at communication loss

Cables and Connectors

The REV Robotics Control Hub ([REV-31-1595](#)) connector selection provides a robust high-density solution for the user. All connectors are keyed and locking except for the Servo, 5V auxiliary power, HDMI , and USB ports.

→ **XT-30 - Power Cable**

</control-system-overview/cables-and-connectors/xt-30-power-cable>

→ **JST VH - Motor Power**

</control-system-overview/cables-and-connectors/jst-vh-motor-power>

→ **JST PH - Sensors and RS485**

</control-system-overview/cables-and-connectors/jst-ph-sensors-and-rs485>

XT-30 - Power Cable

The XT30 connector is used for connecting a battery and powering a Control/Expansion Hub. Each Control/Expansion Hub has both a Male and Female XT30 connector, as determined from the metal contacts, not the plastic housing. While either connector can provide power to the hub, it is the convention to use the male connector for "power in" to the hub, and to use the female connector for "power out" to a connected secondary device, like an Expansion Hub or XT30 Power Distribution Block, from the single battery source.

Passing power from one device to another in a chain is often called "daisy-chaining."

Most teams will want to use pre-made cables which can be conveniently sourced from the [REV Robotics website](#). However, teams can also make their own cables. These connectors are solder-cup style, do not require any crimping tools, and are available from various online vendors. Because these connectors are an open design, they are manufactured by a variety of sources and quality may vary. AMASS branded connectors are recommended, and are what is used on REV products, but there are many other quality vendors available.

Table 1: Premade XT-30 Cables and Accessories

Cable Type	Length	REV Robotics Part Number
XT-30 Male - XT-30 Female	30 cm	REV-31-1392
XT-30 Male - XT-30 Female	50 cm	REV-31-1394
XT-30 Female - Tamiya	8 cm	REV-31-1382
XT-30 Female - Anderson Power Pole Style	8 cm	REV-31-1385
Power Switch Cable (XT30 Male – XT30 Female)	12 cm	REV-31-1387
XT30 Connector Pack – 5 Pairs	-	REV-31-1399

JST VH - Motor Power

Motor Power connections on the Control Hub ([REV-31-1595](#)) use the JST VH style connector. This connector is keyed and locking with a small latch, seen below, which must be depressed to release the cable.

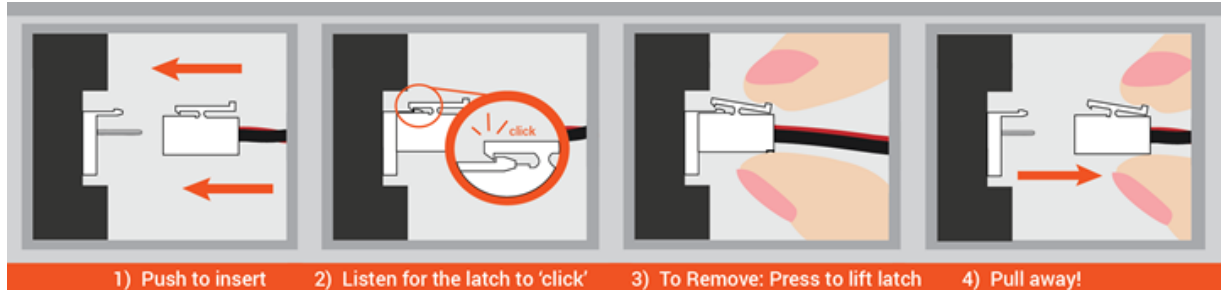


Figure 1: How to Use a JST VH Cable

REV Robotics recommends, in most cases, that teams use pre-made cables because crimp quality is better when made using industrial tooling. These cables can be purchased directly from the [REV Robotics website](#) or through other online vendors.

Premade JST VH Cables and Accessories

Cable/Accessory	Pins	Length	REV Robotics Part Number
JST VH 2-Pin Motor Cable	2 pins	30 cm	REV-31-1412
JST VH 2-Pin Motor Cable	2 pins	50 cm	REV-31-1413
JST VH 2-Pin Motor Cable	2 pins	100 cm	REV-31-1526
Anderson to JST VH Cable	2 pins	12 cm	REV-31-1381
JST VH 2-pin Joiner Board	2 pins	-	REV-31-1429

For teams that want to try crimping their own cables, or to find more information about the connectors, Table 3 lists the appropriate part numbers.

Connector Specifications

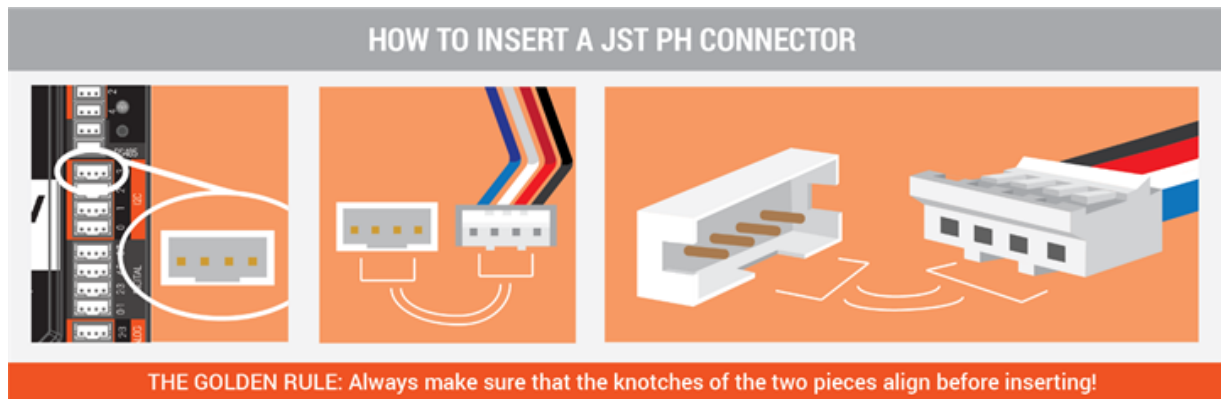
- 10A Continuous Current (16AWG)
- 3.96mm Pitch
- Accepts 22-16AWG Wire

JST VH Connector Part Number Reference

	Manufacturer Part Number	DigiKey Part Number
Contact, JST VH, 18-22AWG	SVH-21T-P1.1	455-1133-1-ND
Contact, JST VH, 16-20AWG	SVH-41T-P1.1	455-1319-1-ND
Housing, JST VH, 2-pin	VHR-2N	455-1183-ND
Header, JST VH, 2-pin, Top Entry	B2P-VH	455-1639-ND
Header, JST VH, 2-pin, Side Entry	B2PS-VH	455-1648-ND

JST PH - Sensors and RS485

The JST PH style connector is used for motor encoder, analog, digital, I2C, RS485, and UART connections on the Control Hub and Expansion Hub. These are all 4-pin connections except for the RS485 and UART which are 3 pin. The connectors are keyed (they only insert in one orientation) and are friction locking. Below the keying feature aligned with the cable is shown.



REV Robotics recommends in most cases that teams use pre-made cables because the quality of the crimp is better when made using industrial tooling. These cables can be bought directly from the REV Robotics Website or through other online vendors.

Premade 4-pin JST PH Cables

Cable/Accessory	Pins	Length	REV Robotics Part Number
JST PH 4-Pin Sensor Cable	4	30 cm	REV-31-1407
JST PH 4-Pin Sensor Cable	4	50 cm	REV-31-1408
JST PH 4-Pin Sensor Cable	4	100 cm	REV-31-1409
JST PH 4-pin Joiner Board	4		REV-31-1388

Premade 3-pin JST PH Cables

Cable	Pins	Length	REV Robotics Part Number
JST PH 3-pin Communication Cable	3	30 cm	REV-31-1417
JST PH 3-pin Communication Cable	3	50 cm	REV-31-1418
JST PH 3-pin Communication Cable	3	100 cm	REV-31-1565

For teams that want to try crimping their own cables, or to find more information about the connectors, the table below lists the appropriate part numbers.

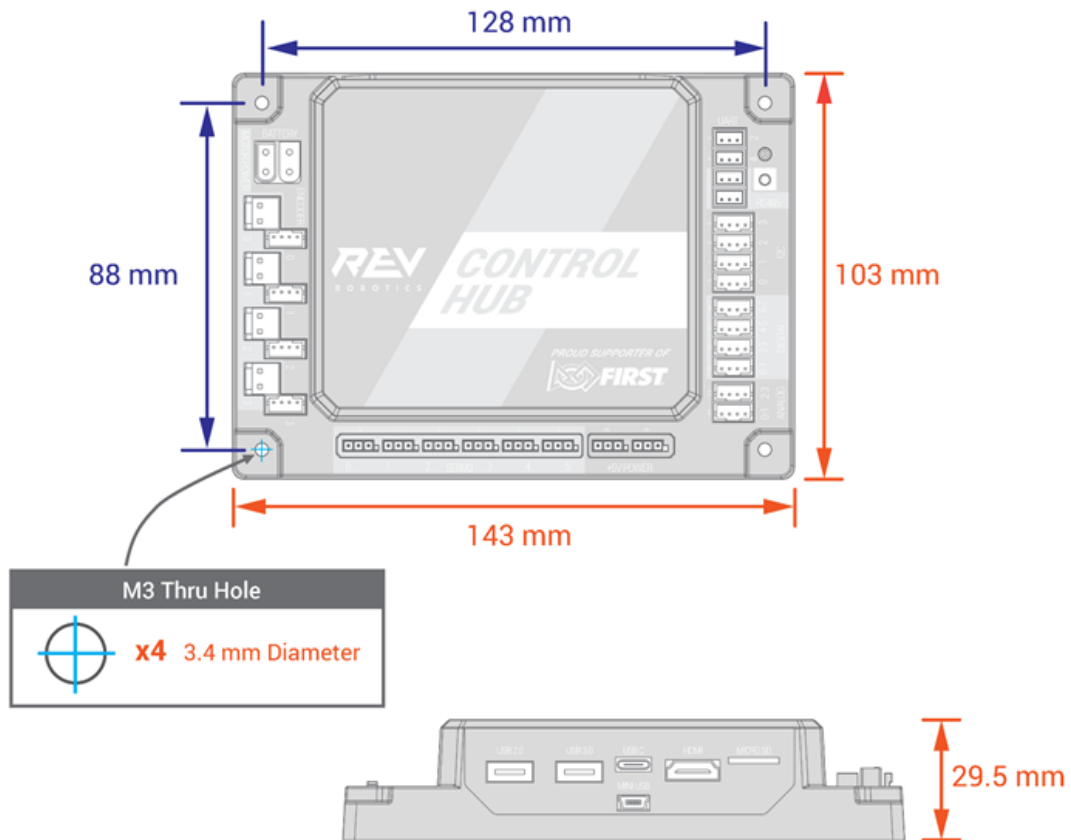
Connector Specifications

- 2A continuous current (24AWG)
- 2.0mm pitch
- Accepts 32-24AWG wire

JST PH Connector Part Number Reference

Connector Parts	Manufacturer Part Number	DigiKey Part Number
Contact, JST PH, 30-24AWG	SPH-002T-P0.5S	455-1127-1-ND
Contact, JST PH, 28-24AWG	SPH-002T-P0.5L	455-2148-1-ND
Housing, JST PH, 4-pin	PHR-4	455-1164-ND
Header, JST PH, 4-pin, Top Entry	B4B-PH-K-S	455-1706-ND
Header, JST PH, 4-pin, Side Entry	S4B-PH-K-S	455-1721-ND
Housing, JST PH, 3-pin	PHR-3	455-1126-ND
Header, JST PH, 3-pin, Top Entry	B3B-PH-K-S	455-1705-ND
Header, JST PH, 3-pin, Side Entry	S3B-PH-K-S	455-1720-ND

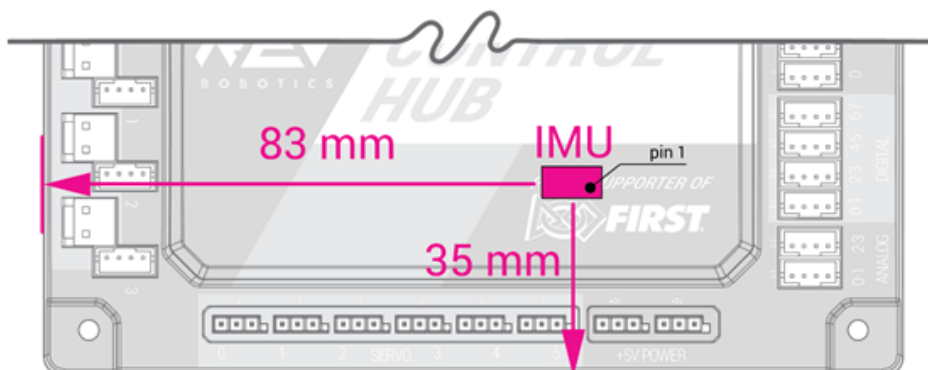
Dimensions and Important Component Locations



IMU Location

When using the Control Hub (REV-31-1595) or Expansion Hub (REV-31-1153) please note the location of the IMU in the graphic below. The Hub's orientation may impact the values received from the embedded IMU.

BELOW: IMU Details Shown in Enlarged View

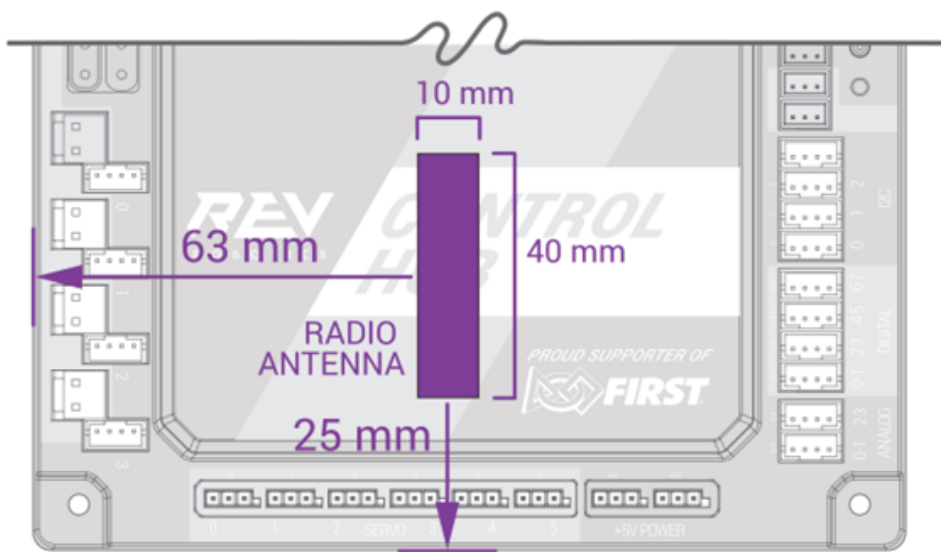


WiFi Radio Location

The Control Hub has an embedded WiFi radio for wireless communication. The antenna is located towards the top of the Control Hub itself. The graphic below shows the location of antenna.

- ⚠ DO NOT put a battery or other WiFi blocking object on top of the Control Hub. This can lead to higher ping times for communication between the Control Hub and the Driver Station.

BELOW: Radio Antenna Shown in Enlarged View



Getting Started

Control Hub

After receiving the Control Hub it is advised to unbox the device, power the Control Hub on, and start the configuration process. Below are the required materials to run through the initial bring up of the Control Hub and links to the different steps of the process.

Required Materials


- Control Hub ([REV-31-1595](#))
- 12v Slim Battery ([REV-31-1302](#))
- [Properly Configured Driver Station \(DS\)](#)
- USB Game Pad ([REV-39-1647](#))
- USB A Female to Micro USB ([REV-31-1426](#))

Optional Additional Materials needed to [Connect an Expansion Hub](#):

- Expansion Hub ([REV-31-1153](#))
- XT30 Extension Cable ([REV-31-1392](#), included with Expansion Hub)
- JST PH 3-pin Communication Cable ([REV-31-1417](#), included with Expansion Hub)

Connect to the Robot Control Console


In order to manage the Control Hub ([REV-31-1595](#)) or programming using the onboard programming languages a computer or other WiFi enabled device will need to connect to the Control Hub's Robot Control Console. The Robot Control Console is a local network created by the Control Hub to program and manage the device.

 This example assumes the user uses Windows 10 as their operating system. If you are not using a Windows 10, the procedure to connect to the network will differ. Refer to your device's documentation for details on how to connect to a WiFi network.


By default, the Control Hub has a name that begins with the phrase "FTC-" or "FIRST-" followed by four characters that are assigned randomly. The default password for the network is "password". If either of these is forgotten, follow the procedure to [reset the WiFi Network](#).

WiFi Connection Procedure

- With the Control Hub powered, select the WiFi Network icon in the lower right corner of the desktop.
- Look for the WiFi that matches the naming protocol of the device.
 - To ensure you are able to locate the correct device, it is recommended that you first connect in a location without other active Control Hubs or significant WiFi connections.
- Once you have found the target network in the list, click on it to select it then press connect.
- Provide the network password (in this example "password") and press "Next" to continue.

 Passwords are case sensitive. Make sure that your spelling and capitalization matches the original spelling and capitalization for the password.

- Once a wireless connection is established, the status is displayed in the wireless settings for the Windows device.

 When connected to the Control Hub, the Windows Device will not have access to the Internet. It only has direct access to the Control Hub.

- Open a web browser (Chrome, Firefox, Internet Explorer) and navigate to "192.168.43.1:8080" through the address bar.

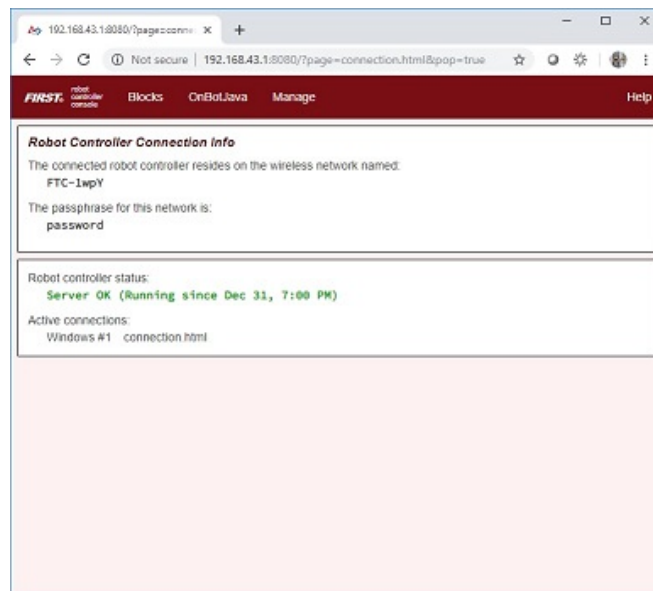
From the Robot Control Console users can change the **Control Hub name**, **update the password**, upgrade the **operating system** and **firmware**, as well as **program** the device. It is strongly recommended that you go through all steps above before you begin programming.

Changing Control Hub Name

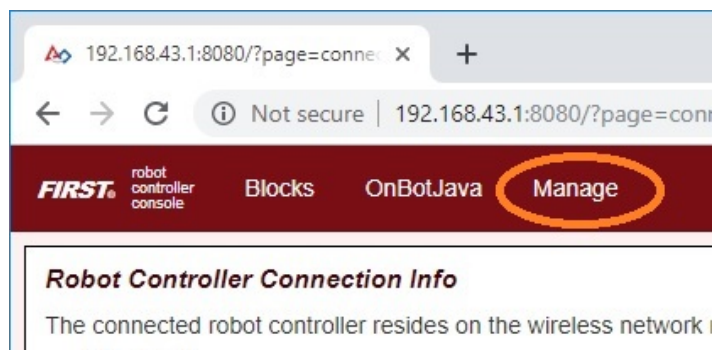
You can change the name of a Control Hub (REV-31-1595) using a device connect to the Control Hub's WiFi network and using a web browser to navigate to the Robot Controller Console.

Steps to Change the Control Hub Name

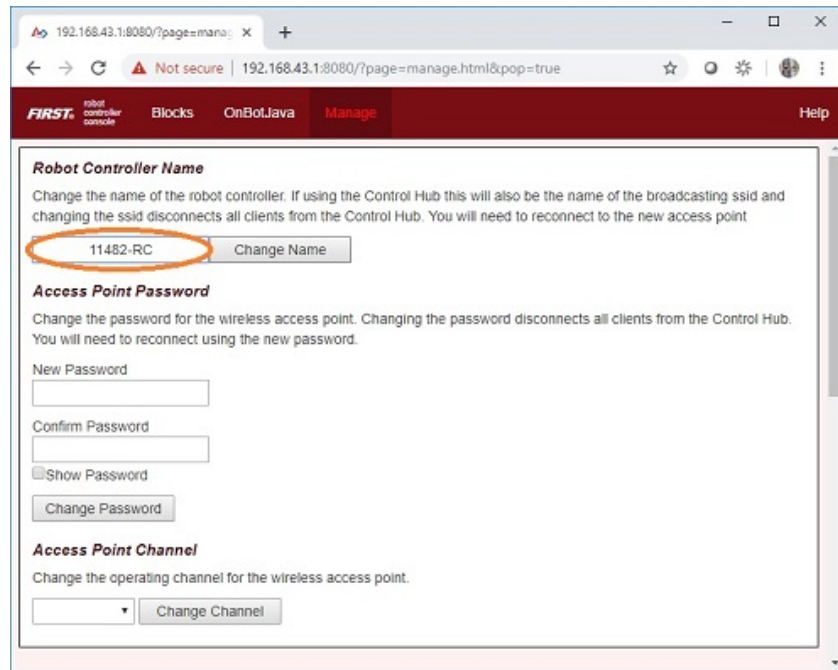
- Verify that connection to the Robot Control Console of the Control Hub. The Robot Controller Connection Info page will be visible when navigating to address "192.168.43.1:8080"



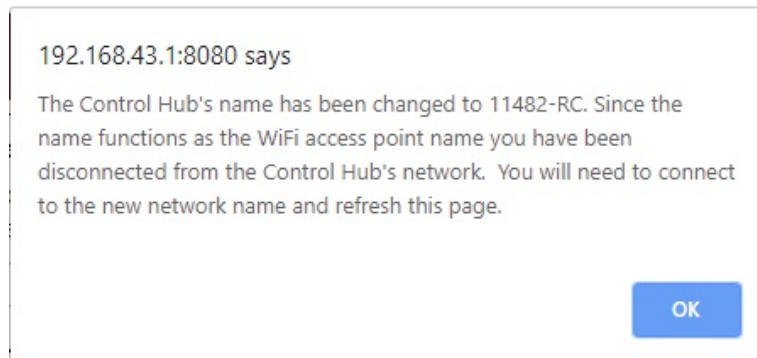
- Click on the Manage link towards the top of the Robot Controller Connection Info page to navigate to the Manage page.



- Change the name in the "Robot Controller Name" field and click on the Change Name button to change the Control Hub's name.



- After pressing the Change Name button, a dialog box will appear indicating that the name has been changed. You will need to reconnect to the wireless network and refresh the current page.

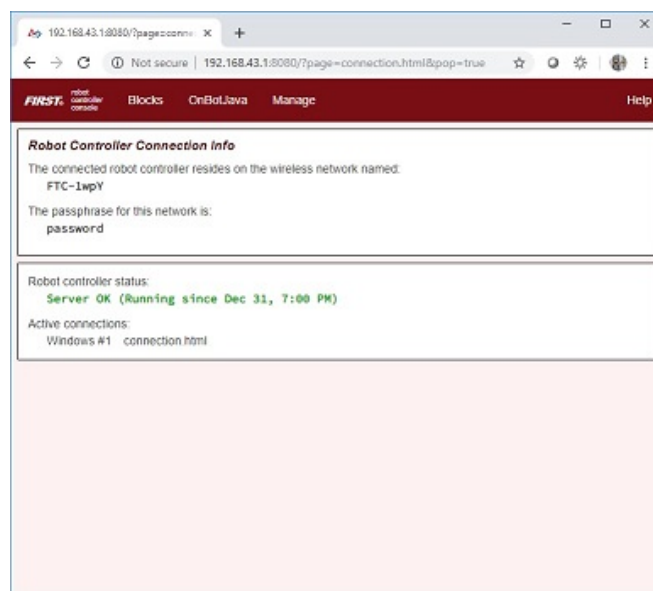


Changing Control Hub Password

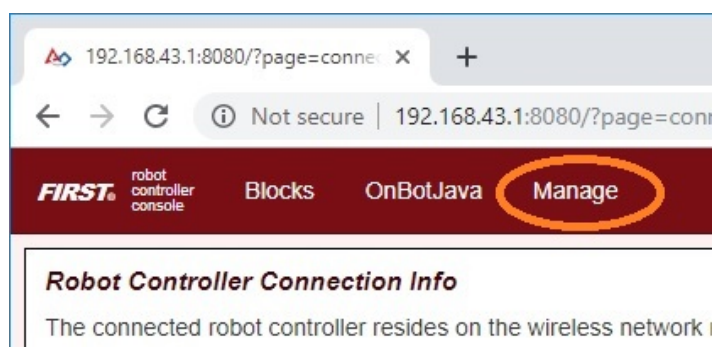
You can change the password of a Control Hub ([REV-31-1595](#)) using a device connect to the Control Hub's WiFi network and using a web browser to navigate to the Robot Controller Console. By default, the Control Hub has its password set to "password" at the factory. It is a good idea to change the password from its default value before you begin using your Control Hub.

Steps to Change the Control Hub Password

- Verify that you are connected to the Robot Control Console of the correct Control Hub. The Robot Controller Connection Info page will be visible when navigating to address "192.168.43.1:8080"



- Click on the Manage link towards the top of the Robot Controller Connection Info page to navigate to the Manage page.



- On the Manage page of the Control Hub Robot Controller Console, specify your new password and then confirm this new password in the Access Point Password section of the page. Press the Change Password to change the password.

Access Point Password

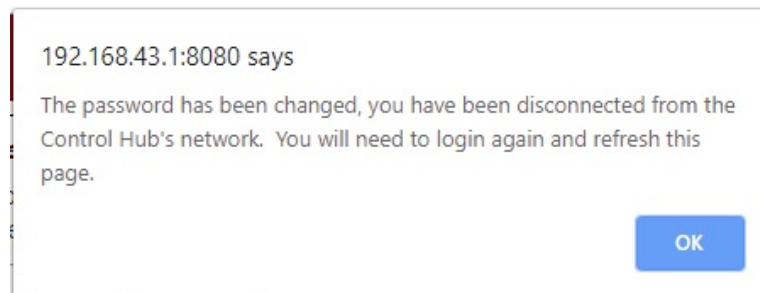
Change the password for the wireless access point. Changing the password disconnects all clients from the Control Hub. You will need to reconnect using the new password.

New Password

Confirm Password

Show Password

- After you press the Change Password button, a dialog box will appear, indicating that the password has been changed. You will need to reconnect to the wireless network using the new password and refresh the current page.

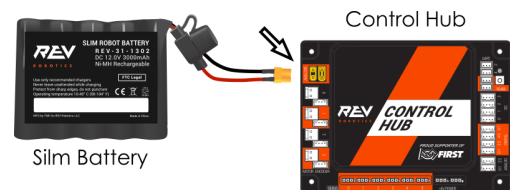


Driver Station Pairing to Control Hub

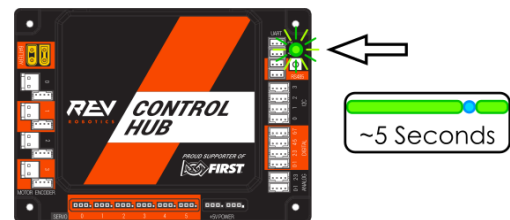
When you first receive your Control Hub (REV-31-1595) , you will have to pair (link) your Driver Station (Android Device) to your Control Hub. This procedure only needs to be performed once for each set of hardware. If you replace your Driver Station or Control Hub, this procedure will need to be repeated. The steps below assume the FTC Driver Station Application or FIRST Global Driver Station Application are installed on your Android device.

Pairing the Driver Station with the Control Hub

Power on the Control Hub by plugging the 12V Slim Battery into the XT30 connector labeled "BATTERY" on the Control Hub. You may also choose to include a switch between the Battery and Control Hub, if you prefer.



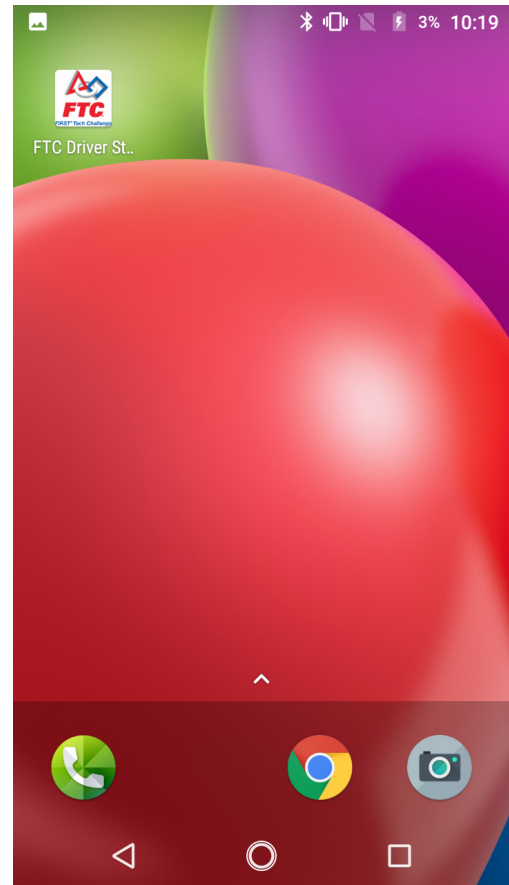
The Control Hub is ready to pair with the Driver Station when the LED turns green. Note: the light blinks blue every ~5 seconds to indicate that the Control Hub is healthy.



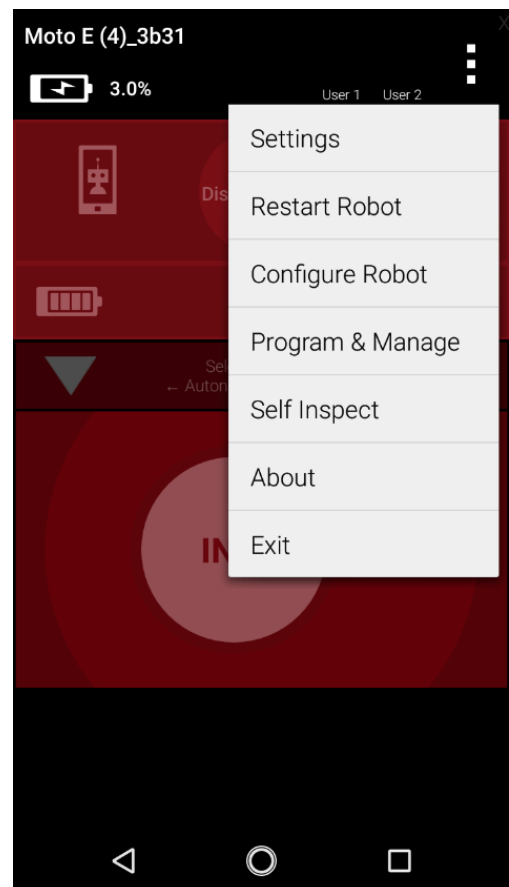
Power on your Android Device by holding down the power button.



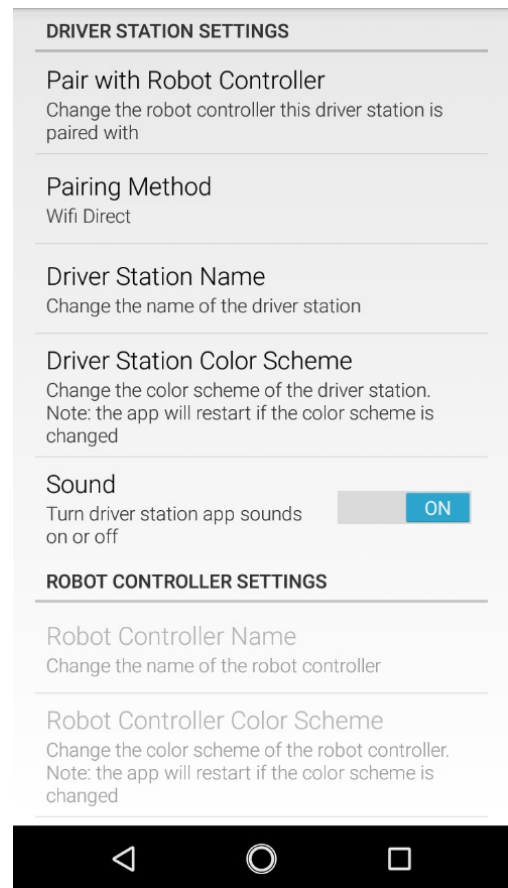
Open the Driver Station application from the HOME Screen.



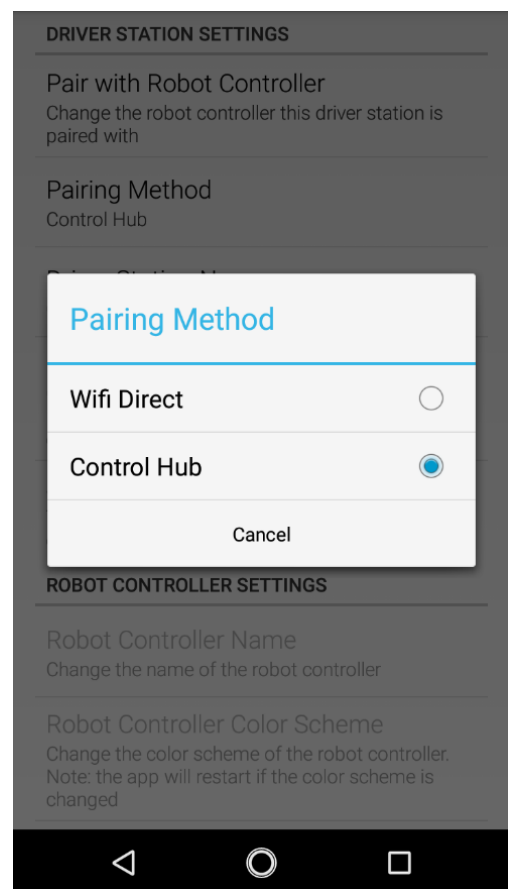
On the Driver Station page, open the menu from the top right corner, then select "Settings".



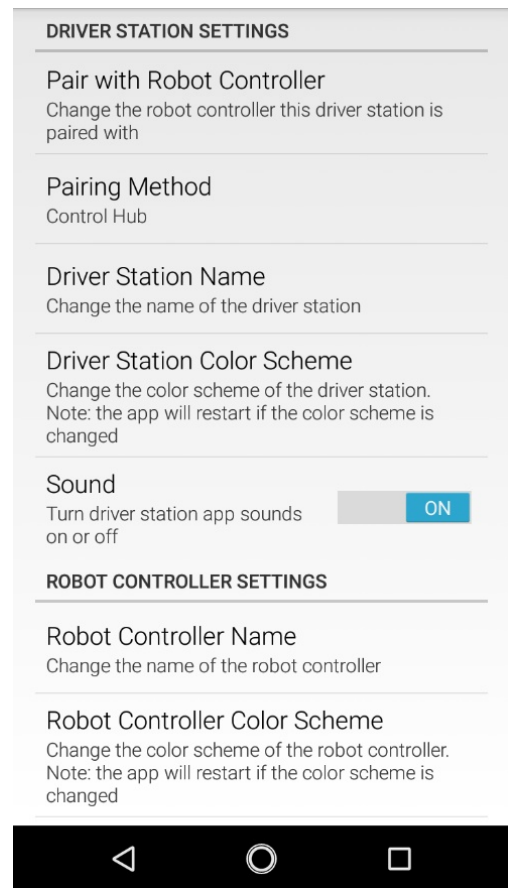
Select, "Pairing Method"



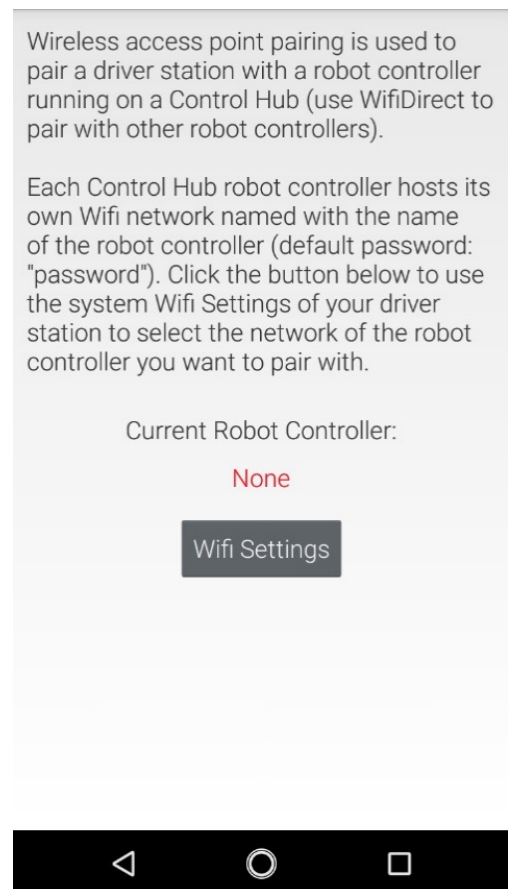
Select, "Control Hub"



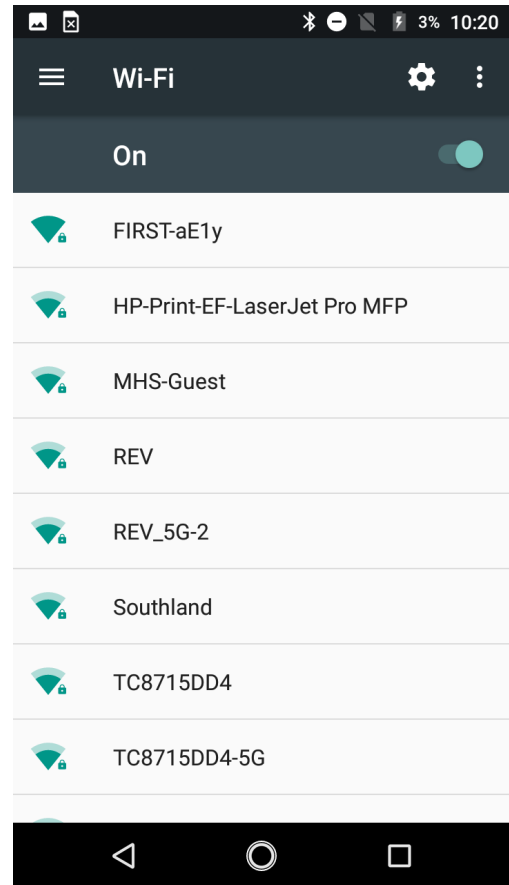
Select, "Pair with Robot Controller".



Select "Wifi Settings"

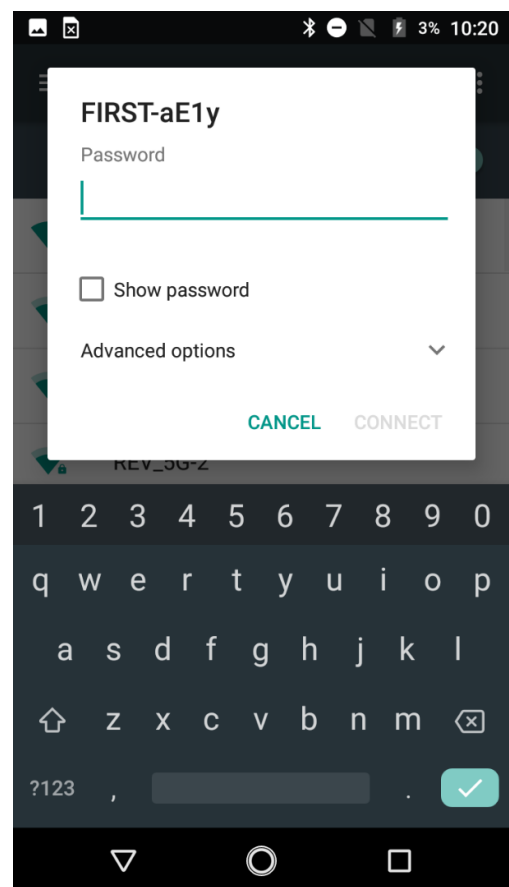


Select the name of the Wifi network generated by your Control Hub. The default SSID name starts with either "FIRST-" or "FTC-".

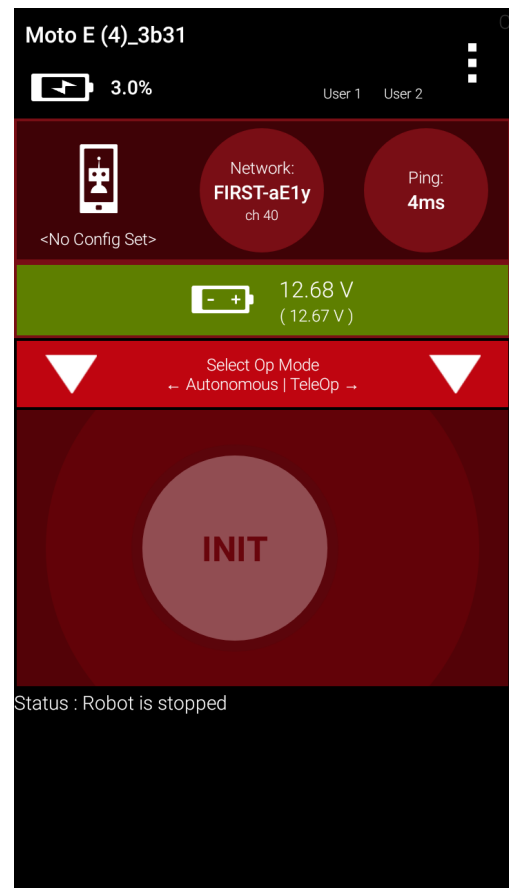


Enter the password to the Wifi network in the password field. This defaults to "password". Press "CONNECT".

After pressing connect, press the back arrow at the bottom of the display until you return to the main driver station screen.



After a couple of seconds, the Driver Station page will indicate the network name, a ping time, and battery voltage.




Your Driver Station is now paired with your Control Hub!

Changing WiFi Channel

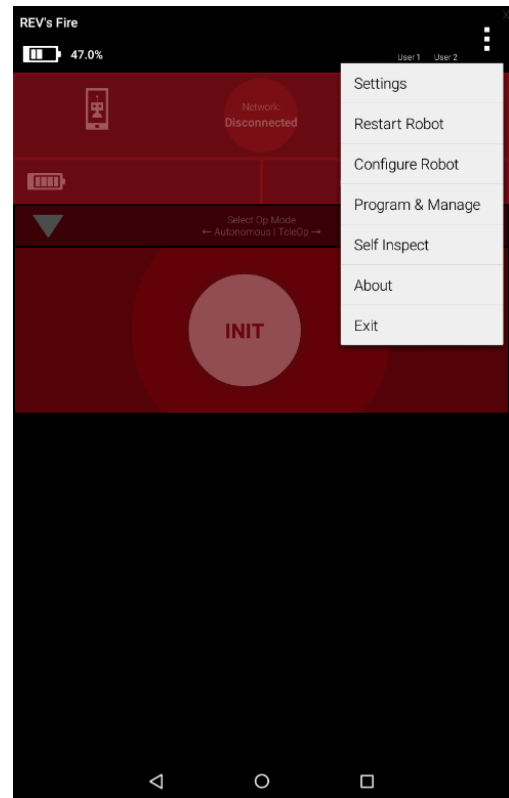
The Control Hub ([REV-31-1595](#)) can utilize either the 2.4 GHz or 5 GHz WiFi band. By default the Control Hub is set to a channel on the 2.4 GHz band. REV Robotics advises that during competition teams utilize a 5 GHz channel for robot communication. Consult the table below for Driver Station devices that can operate on the 5 GHz band.

Phone	WiFi Band
Moto G (2nd generation)	2.4 GHz (Single Band)
Moto G (3rd generation)	2.4 GHz (Single Band)
Moto G (4th generation)	2.4 GHz (Single Band)
Moto G5	2.4 GHz & 5 GHz (Dual Band)
Moto G5 Plus	2.4 GHz & 5 GHz (Dual Band)
Moto E4	2.4 GHz & 5 GHz (Dual Band)
Moto E5	2.4 GHz & 5 GHz (Dual Band)
Moto E5 Play	2.4 GHz & 5 GHz (Dual Band)

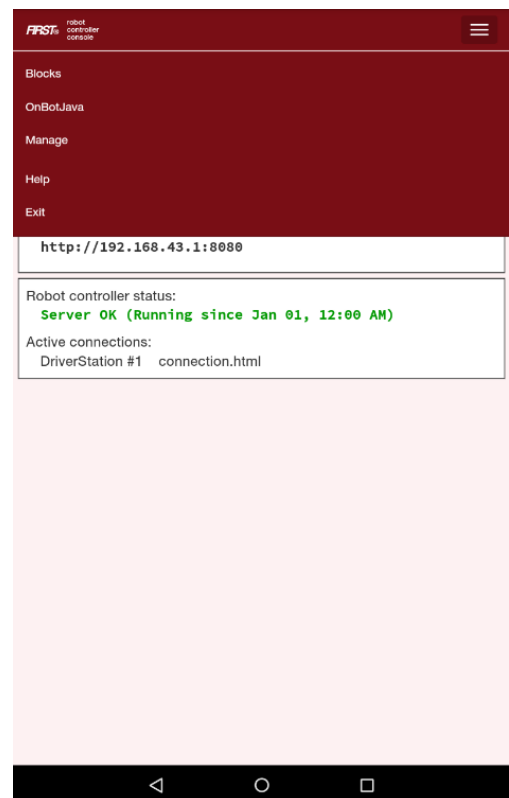
Steps to changing the WiFi Channel on your Robot

Step	Image
	

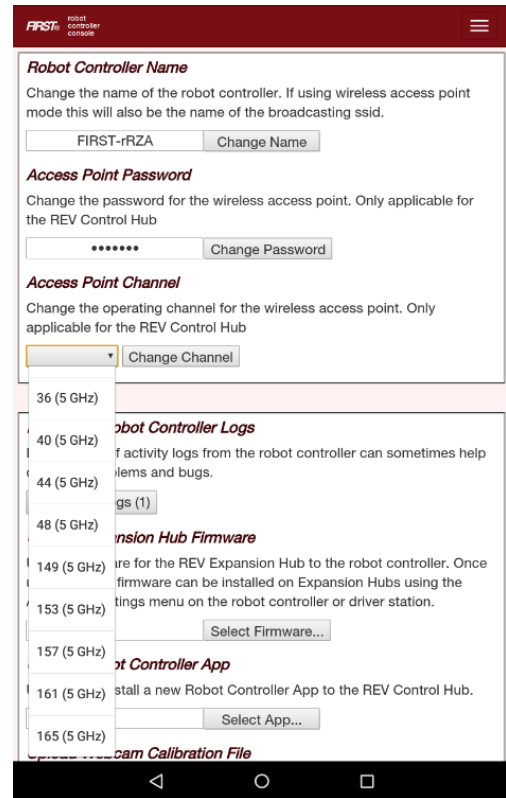
Press the triple dots in the upper right. Then select “Program & Manage” from the Driver Station Menu.



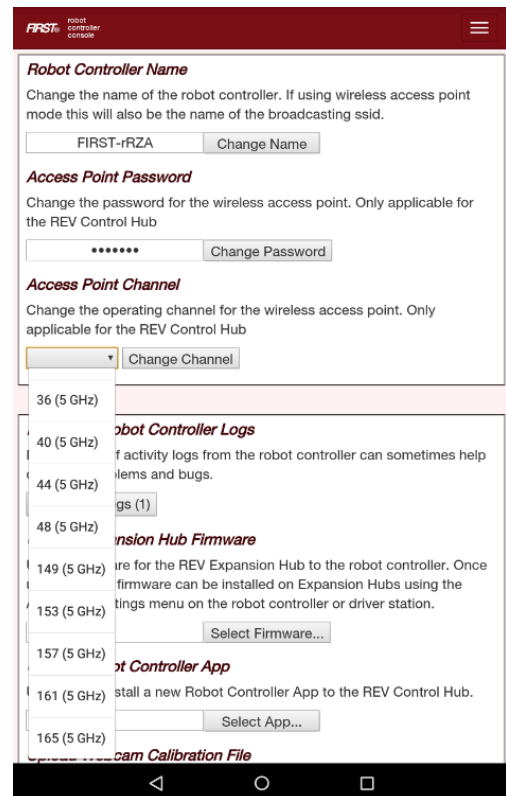
Select the menu button in the top right. Then select “Manage”.



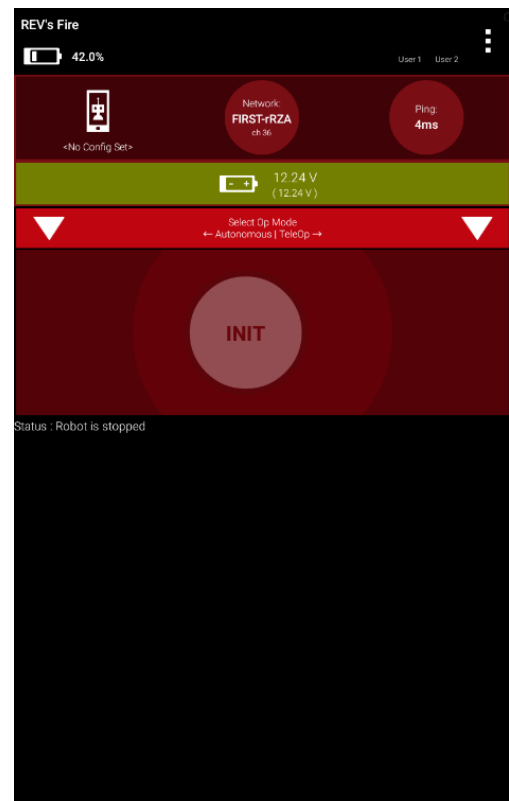
Select the drop down menu under “Access Point Channel”.



Select a 5 GHz channel noted in the () next to the channel number. Then select the “Change Channel” button next to the drop down.

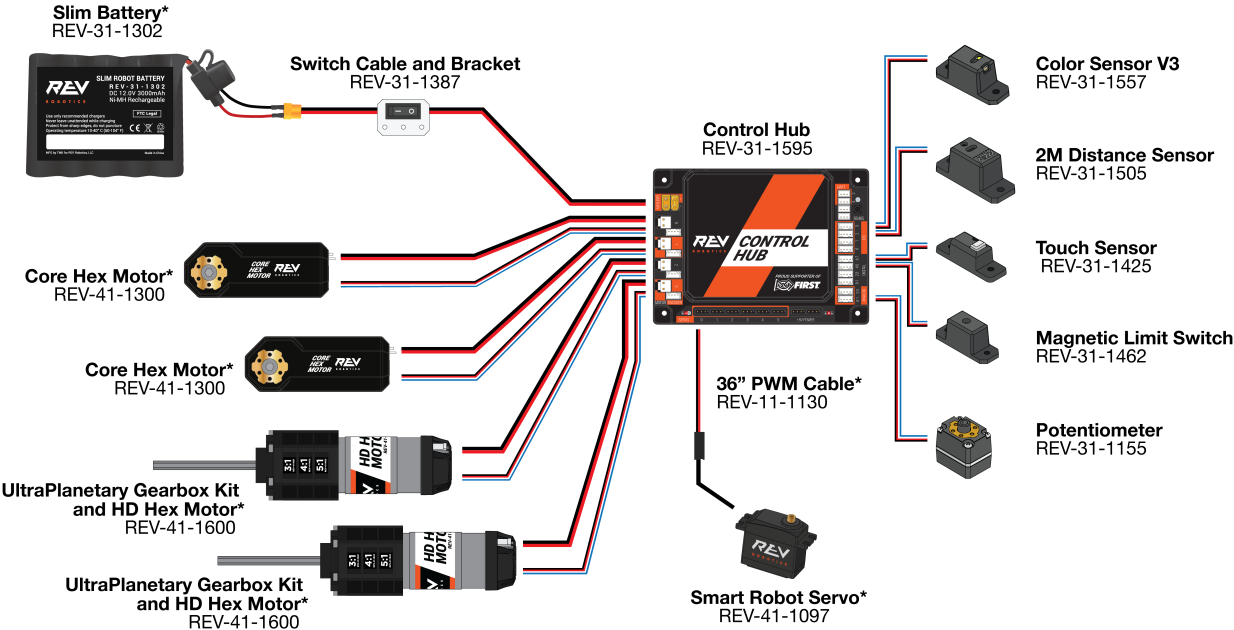


At the main screen, confirm the channel is changed under "Network".



Wiring Diagram

Before configuring your Control Hub, devices must be connected to the Control Hub. Below is a sample wiring diagram to show a sample of actuators and sensors usable with the Control Hub.



Configuration

Every device connected to the Control Hub (REV-31-1595) will need to be added to the Robot Configuration file before you can use the device in your program. The Robot Configuration will allow you to give your sensors and actuators meaningful names that you can reference while programming.

For this example, we will configure a simple two motor robot drivetrain.

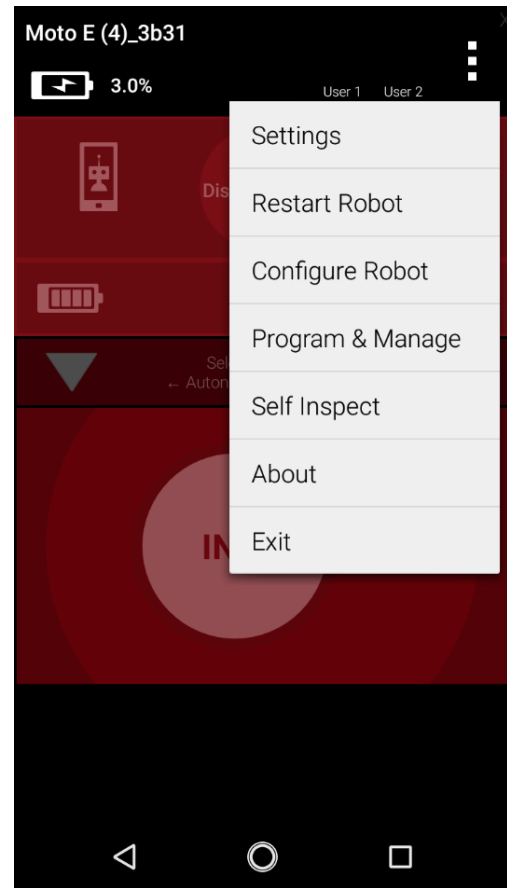


Step

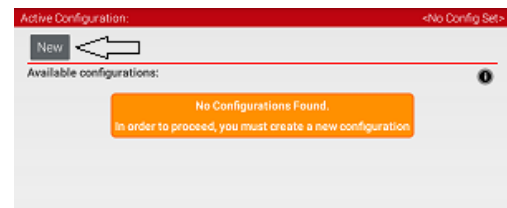
Image



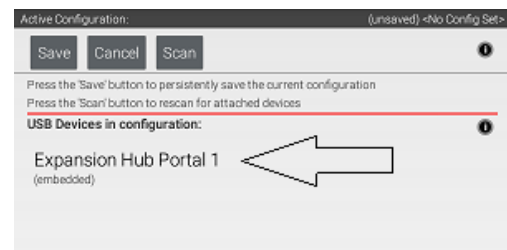
Select the menu on either the Driver Station or Robot Controller. Then select "Configure Robot".



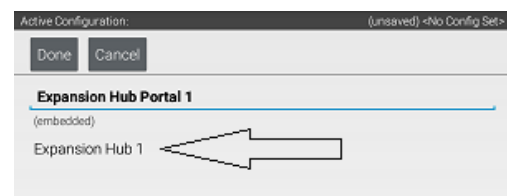
Select "New" in the top left hand corner.



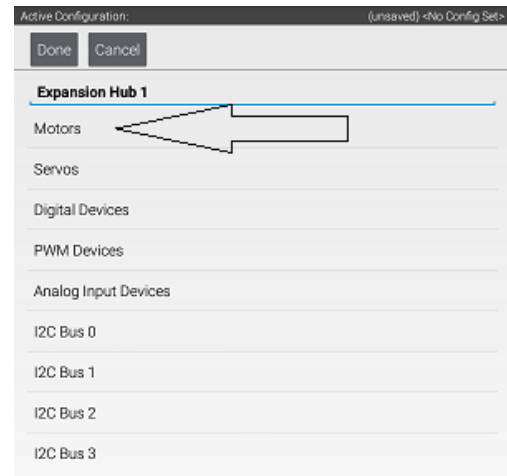
Select "Expansion Hub Portal 1" (embedded).



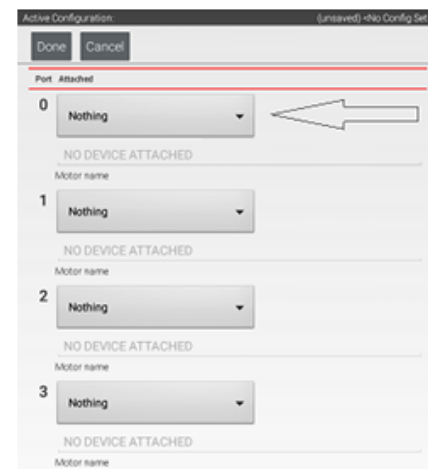
Select "Expansion Hub 1".



Select "Motors".

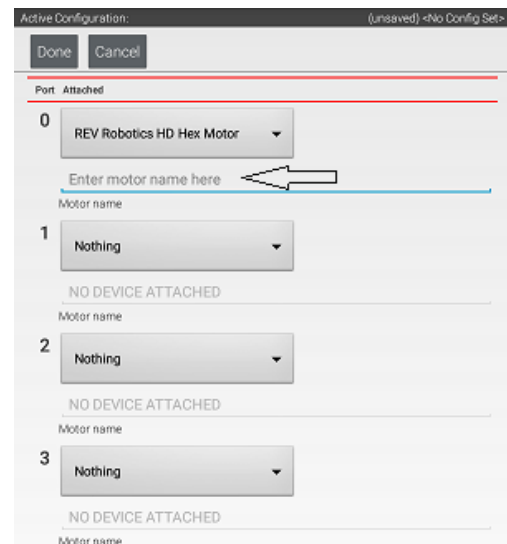


Select the Drop Down menu for "Port 0" then select the motor type attached to the port. In the case of the Minibot in Figure 4, select the "Rev Robotics Core Hex Motor".

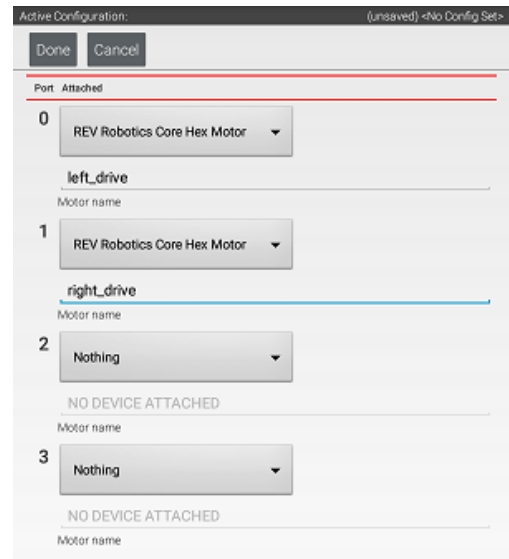


Press "Enter motor name here" and name the motor "left_drive".

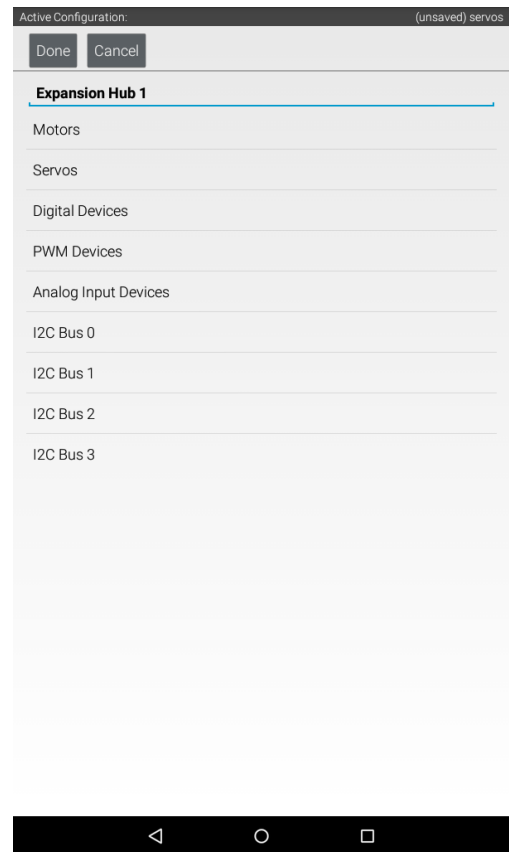
This is the name that you will use when you are programming your robot to control this motor. Always use descriptive names so that you can remember what a device does when you are programming.



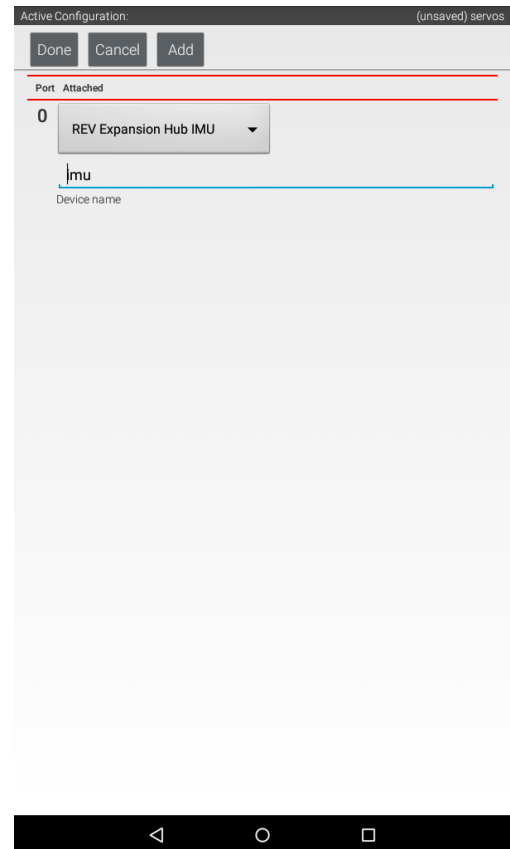
Repeat the process for "Port 1" and name the motor "right_drive".



Press "Done" once to go back to the list of device ports and then select I2C Bus 0.



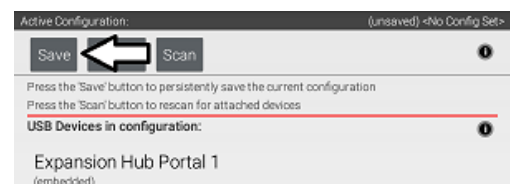
Add the built-in REV Expansion Hub IMU. Name it "imu"



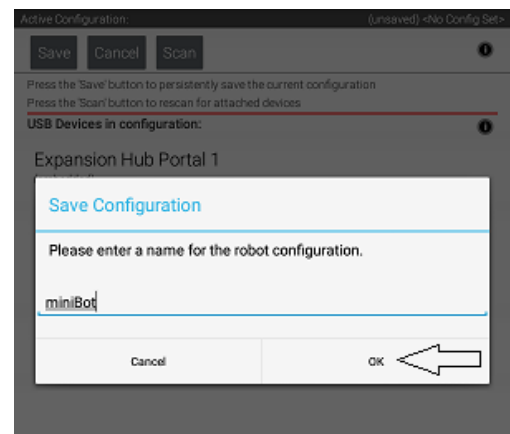
Press the "Done" button (at the top left corner of the page) 3 times.



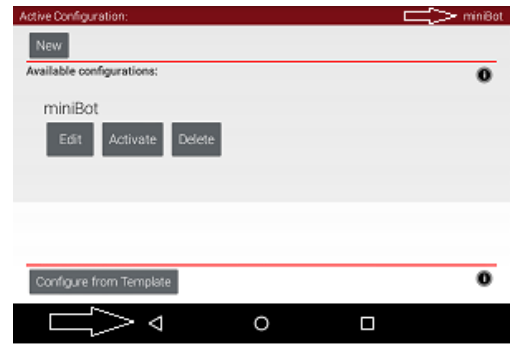
Press "Save".



Enter "miniBot" as your configuration name, then select "OK".



You now have an active configuration called "miniBot".
Press the Android back button to return to the Driver Station
page.



Expansion Hub

After receiving the Expansion Hub it is advised to unbox the device, power the Expansion Hub on, and start the configuration process. Below are the required materials to run through the initial bring up of the Expansion Hub and links to the different steps of the process.

Required Materials

- Expansion Hub ([REV-31-1153](#))
- 12v Slim Battery ([REV-31-1302](#))
- [Properly Configured Driver Station \(DS\)](#)
- [Properly Configured Robot Controller \(RC\)](#)
- USB Game Pad ([REV-39-1647](#))
- USB A Female to Micro USB ([REV-31-1426](#))

Optional Additional Materials needed to [Connect an Expansion Hub](#):

- Expansion Hub ([REV-31-1153](#))
- XT30 Extension Cable ([REV-31-1392](#))
- JST PH 3-pin Communication Cable ([REV-31-1417](#))

Driver Station and Robot Controller Pairing

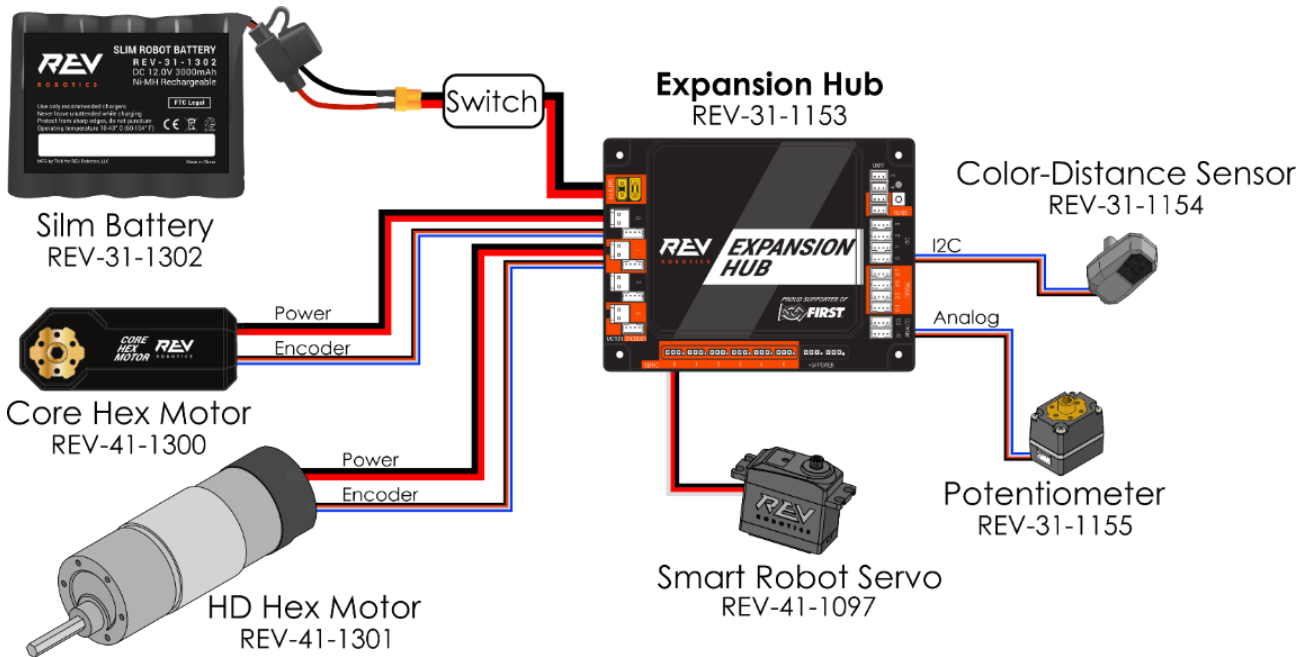
NOTE: *You should update your Driver Station(DS) and Robot Controller(RC) phones to the latest app version in order to use the new Expansion Hub controller. The minimum compatible version is 3.1 released on May 10th, 2017*

Please ensure that the Driver Station and Robot Controller phones are properly configured and paired. Refer to the latest pairing and troubleshooting instructions provided by in the [FTC Control System Wiki](#).

Wiring Diagram

System Wiring Diagram

Before configuring your Expansion Hub, devices must be connected to the Expansion Hub. Below is a sample wiring diagram to show a sample of actuators and sensors usable with the Expansion Hub.

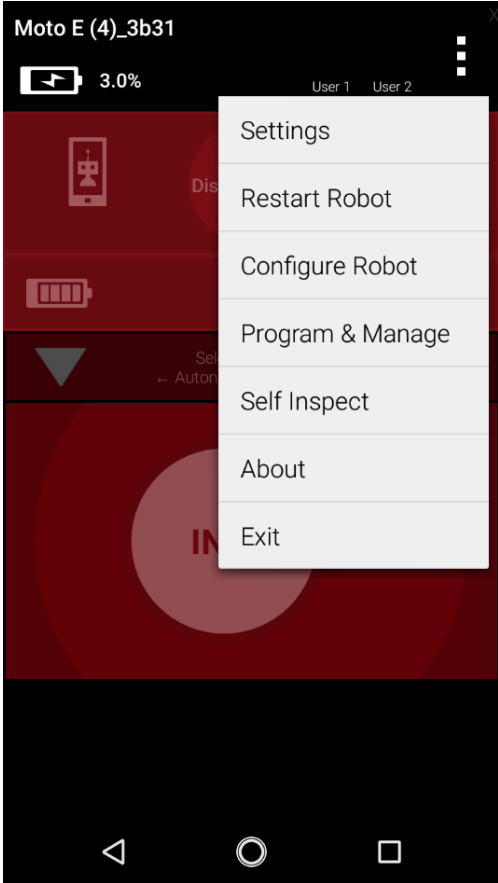
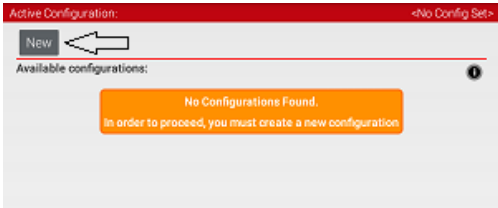
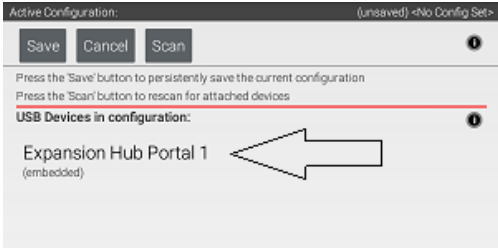


System Wiring Diagram

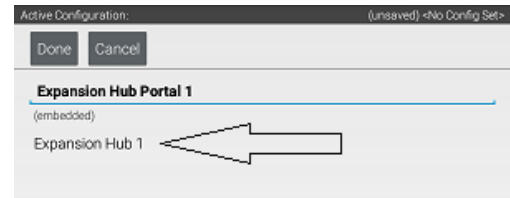
Configuration

Every device connected to the Expansion Hub (REV-31-1153) will need to be added to the Robot Configuration file before you can use the device in your program. The Robot Configuration will allow you to give your sensors and actuators meaningful names that you can reference while programming.

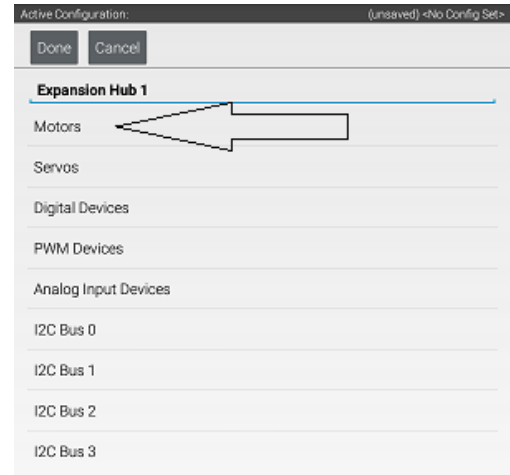
For this example, we will configure a simple two motor robot drivetrain.

Step	Image
Select the menu on either the Driver Station or Robot Controller. Then select "Configure Robot".	 A screenshot of a mobile application interface on a Moto E (4)_3b31. The top status bar shows a battery icon at 3.0% and user names 'User 1' and 'User 2'. A menu is open, listing options: Settings, Restart Robot, Configure Robot, Program & Manage, Self Inspect, About, and Exit. The 'Configure Robot' option is highlighted.
Select "New" in the top left hand corner.	 A screenshot of a configuration screen titled 'Active Configuration: <No Config Set>'. In the top left corner, there is a button labeled 'New' with a white arrow pointing to it. Below the button, it says 'Available configurations:' followed by a message box that reads 'No Configurations Found. In order to proceed, you must create a new configuration'.
Select "Expansion Hub Portal 1" (embedded).	 A screenshot of the configuration screen titled 'Active Configuration: (unsaved) <No Config Set>'. At the top, there are buttons for 'Save', 'Cancel', and 'Scan'. Below these, there are instructions: 'Press the 'Save' button to persistently save the current configuration' and 'Press the 'Scan' button to rescan for attached devices'. Under the heading 'USB Devices in configuration:', the text 'Expansion Hub Portal 1 (embedded)' is displayed with a white arrow pointing to it.

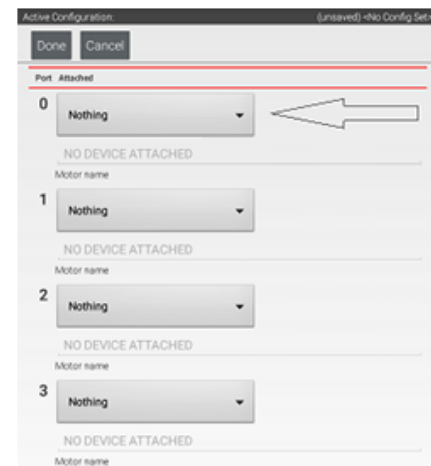
Select "Expansion Hub 1".



Select "Motors".

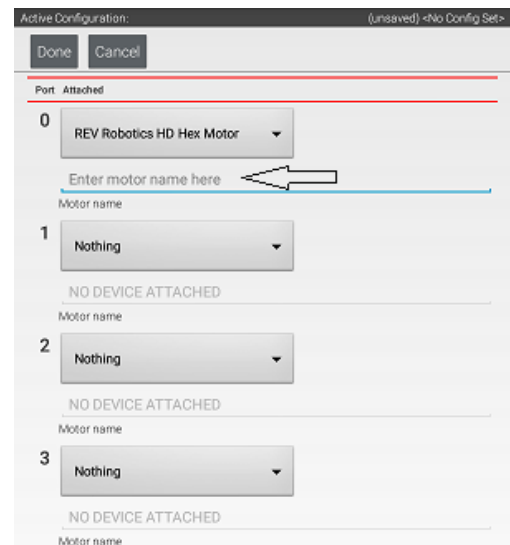


Select the Drop Down menu for "Port 0" then select the motor type attached to the port. In the case of the Minibot in Figure 4, select the "Rev Robotics Core Hex Motor".

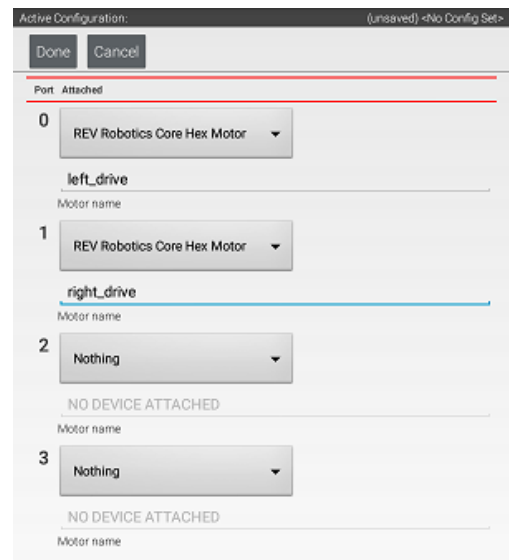


Press "Enter motor name here" and name the motor "left_drive".

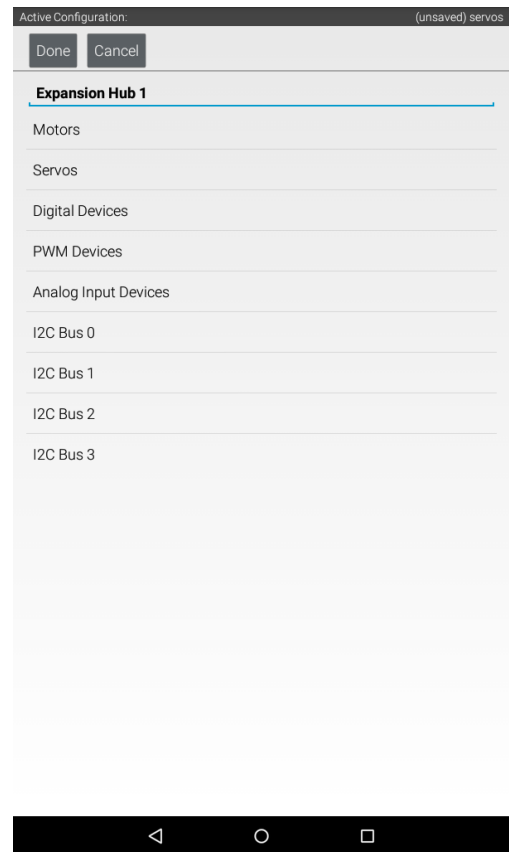
This is the name that you will use when you are programming your robot to control this motor. Always use descriptive names so that you can remember what a device does when you are programming.



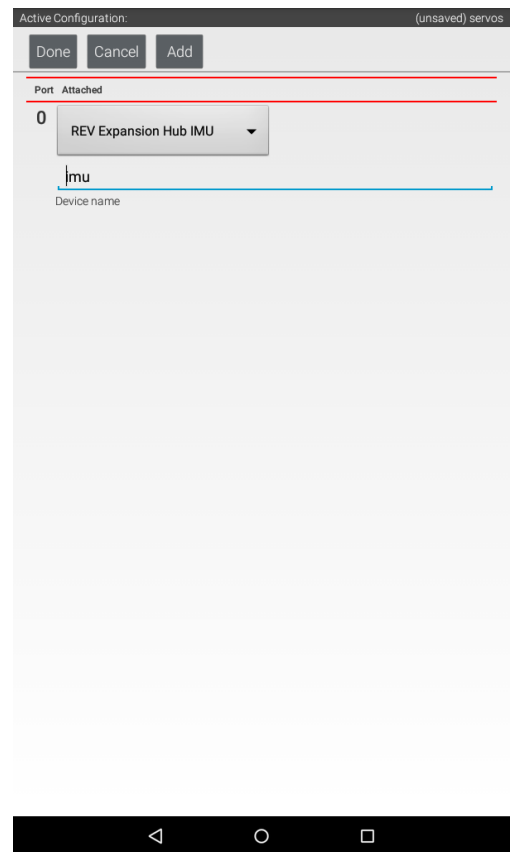
Repeat the process for “Port 1” and name the motor “right_drive”.



Press “Done” once to go back to the list of device ports and then select I2C Bus 0.



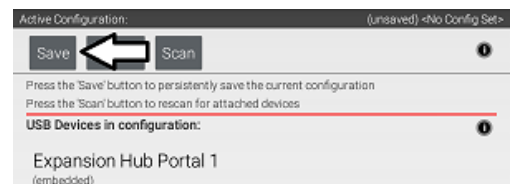
Add the built-in REV Expansion Hub IMU. Name it "imu"



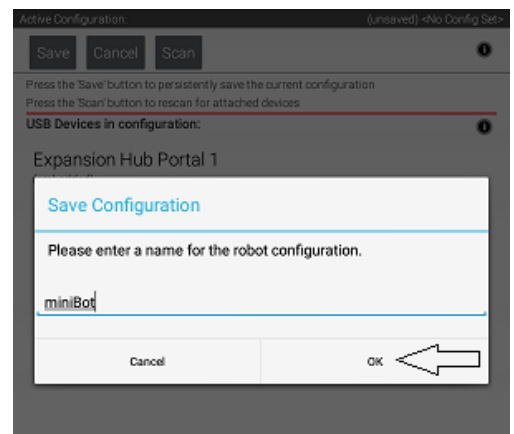
Press the "Done" button (at the top left corner of the page) 3 times.



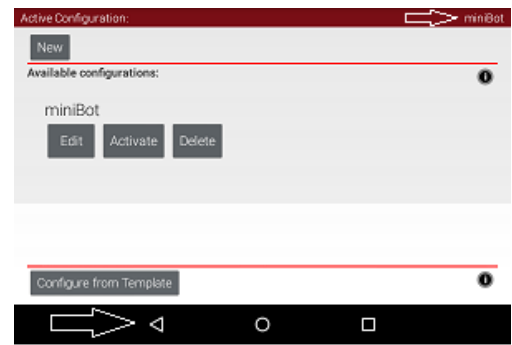
Press "Save".



Enter "miniBot" as your configuration name, then select "OK".



You now have an active configuration called "miniBot".
Press the Android back button to return to the Driver Station page.



Adding More Motors

The Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) can each drive up to four DC brushed motors. As mechanisms are added to the robot the number of motor ports may not be sufficient. There are two ways to add more motors to the Control System, either the SPARKmini Motor Controller ([REV-31-1230](#)) or adding an Expansion Hub. The Following two rules give a general idea of when to choose one method over another:

The Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) can each drive up to four DC brushed motors. As mechanisms are added to the robot the number of motor ports may not be sufficient. There are two ways to add more motors to the Control System, either the SPARKmini Motor Controller ([REV-31-1230](#)) or adding an Expansion Hub. The Following two rules give a general idea of when to choose one method over another:

- If one or two motors are needed, consider using the SPARKmini Motor Controller.
- If three to four additional motors are needed, consider adding an Expansion Hub.

For additional information on [how to use a SPARKmini](#) or [how to add an Expansion Hub](#), visit the linked pages!

SPARKmini Motor Controller

The SPARKmini Motor Controller ([REV-31-1230](#)) is an inexpensive in-line brushed DC motor controller designed to give *FIRST*® Tech Challenge teams more bang for their buck. It offers the same performance characteristics as the REV Control Hub ([REV-31-1595](#)) or Expansion Hub ([REV-31-1153](#)) motor ports in a small 60mm x 22mm footprint. Now FTC teams can add a SPARKmini Motor Controller to utilize more than four DC motors from a single Hub in a space-efficient package.

POWER AND MOTOR CONNECTIONS

The SPARKmini has three integrated wires with connectors dedicated to power, control, and the motor; one [XT30 connector](#) for power, one 3-wire servo-PWM connector for control, and one [JST-VH connector](#) for the motor. The figure below shows each of these connections.



Connect the power wire to a free XT30 port on the REV Control Hub , REV Expansion Hub ([REV-31-1153](#)), or through an XT30 Power Distribution Block ([REV-31-1293](#)) that is connected to a free Control/Expansion Hub XT30 port. Connect the control wire to an open servo port on the hub and the motor wire to a JST-VH port on a motor, like the REV HD Hex Motor ([REV-41-1301](#)) or the REV Core Hex Motor ([REV-41-1300](#)).

⚠ DO NOT reverse polarity on the power input connections. The SPARKmini does not contain reverse polarity protection. This can permanently damage the SPARKmini and will void the warranty.

⚠ DO NOT swap the motor and power connections. This can result in uncontrolled motor operation and can permanently damage the SPARKmini, voiding the warranty.

SERVO-PWM INPUT

A motor's speed is controlled by varying the voltage that is applied to it. The SPARKmini's output voltage can be controlled by sending it an extended-range servo-PWM pulse. The extended 500µs to 2500µs servo-pulse corresponds to full-reverse and full-forward rotation with 1500µs as the neutral position (no rotation). The

pulses are proportionally related to the motor output duty cycle, therefore variable speed can be achieved with pulses in between the extremes. The following table describes the pulse ranges in more detail.

Table - Control Signal Pulse Ranges

Pulse Width (p in μs)				
Full Reverse	Prop. Reverse	Neutral	Prop. Forward	Full Forward
$p \leq 500$	$500 < p < 1490$	$1490 \leq p \leq 1510$	$1510 < p < 2500$	$2500 \leq p$

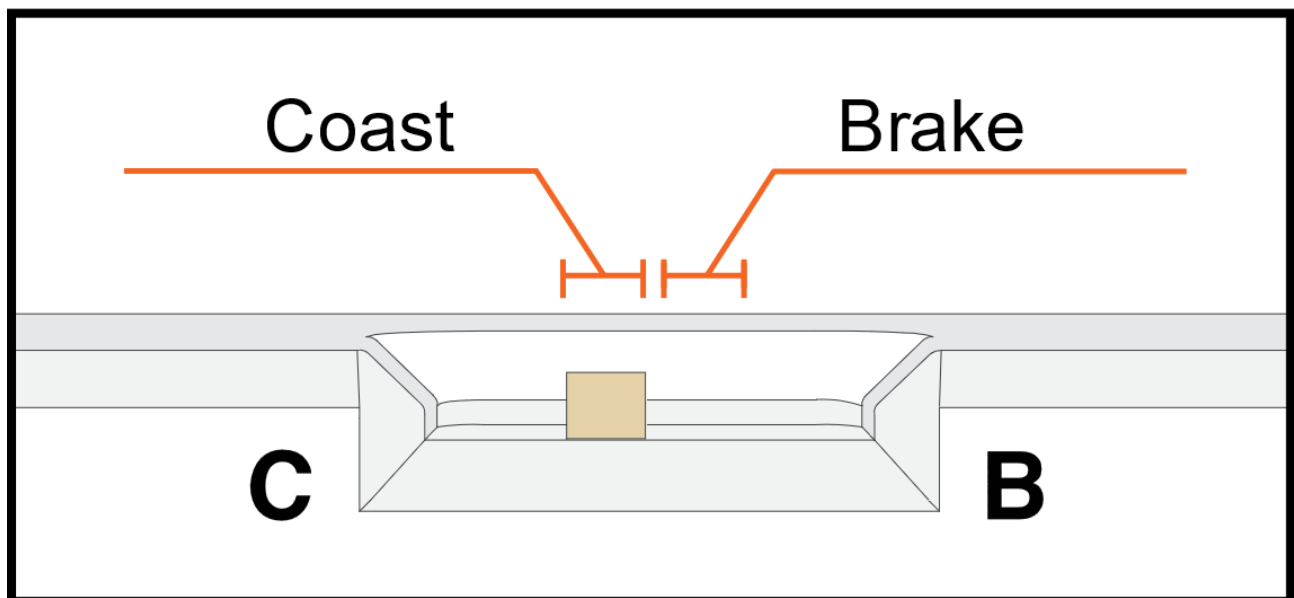
ZERO-POWER BEHAVIOR

When the SPARKmini is receiving a neutral command it will not provide any power to the attached motor. There are two options for how the SPARKmini handles this zero-power state:

Brake - Motor terminals are shorted to each other to dissipate electrical energy, effectively braking the motor.

Coast - Motor terminals are disconnected, allowing the motor to spin down at its own rate.

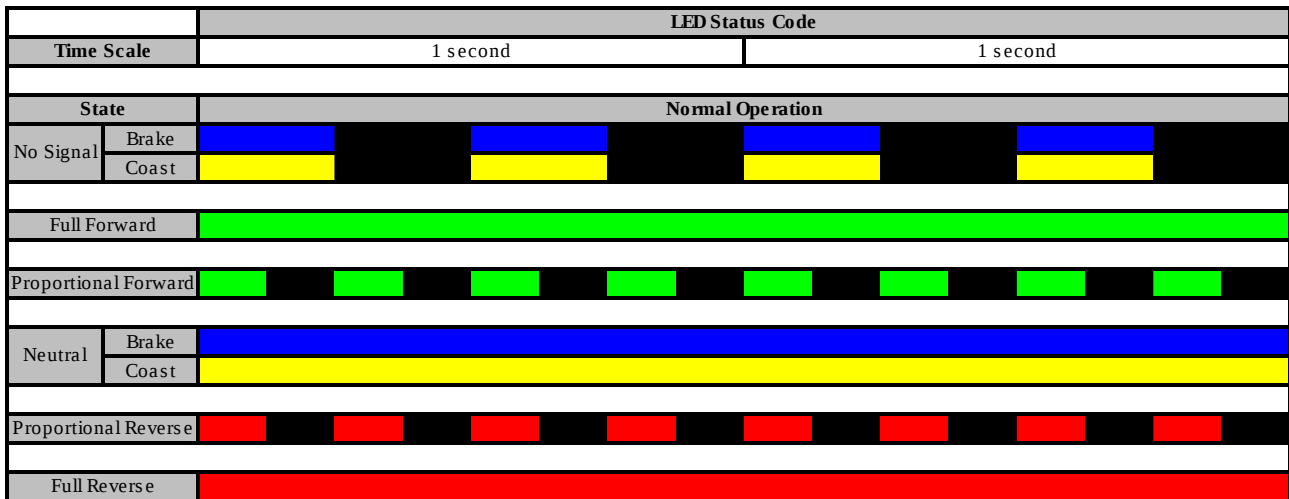
The zero-power behavior can be selected via a switch located towards the center of the SPARKmini housing, shown in Figure 2. Each mode can be selected by sliding the switch to either the Brake (B) or Coast (C) positions.



Coast/Brake Switch

The SPARKmini will indicate whether it is in Brake or Coast mode via the Status LED, located in the center of the housing, whenever it is outputting zero-power. Solid or flashing blue indicates Brake Mode while solid or flashing yellow indicates Coast Mode. See the LED Status Codes section for more details.

LED STATUS CODES



SPECIFICATIONS

Parameter	Min	Typ	Max	Unit
Supply voltage range (VIN)	6.0	12	20	V
Supply voltage absolute maximum	-	-	25	V
Continuous output current	-	-	15	A
Peak output current	-	-	20	A
Output voltage range	- VIN	-	+ VIN	V
Output frequency	-	10	-	kHz
Input pulse width range	500	-	2500	µs
Input frequency	16	50	200	Hz
Input timeout	-	65.5	-	ms
Input deadband	-	±10	-	µs
Input low-level voltage	-0.3	-	0.8	V
Input high-level voltage	2.0	5.0	5.3	V
Weight	-	0.87	-	oz
Dimensions (excluding wires)	-	60 x 22 x 12	-	mm

Adding an Expansion Hub

If you want to use more than 4 motors or 6 servos, you can add an Expansion Hub to your robot. An Expansion Hub ([REV-31-1153](#)) can be added to a Control Hub ([REV-31-1595](#)) or another Expansion Hub. The Expansion Hub has all of the same ports as the Control Hub but without the wireless capability.

Control Hub vs Expansion Hub in FIRST

FIRST Tech Challenge

FIRST Tech Challenge teams may use one (1) Control Hub and may add one (1) Expansion Hub starting in the 2020-2021 season. Read the official FTC Game Manuals for complete game rules.

FIRST Global

FIRST Global teams must use one (1) Control Hub and may add one (1) Expansion Hub to their robot. Read the official FIRST Global manual for complete game rules.

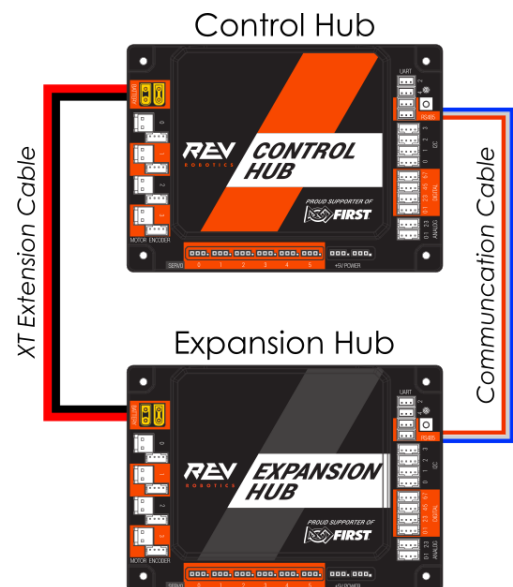
Adding an Expansion Hub to your Robot

Step

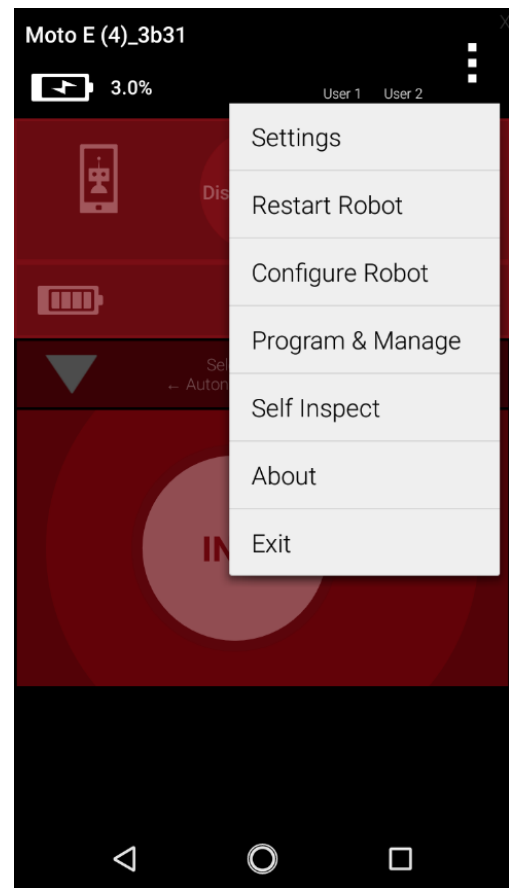
Image

Use the XT Extension Cable to connect power between the Control Hub and the Expansion Hub.

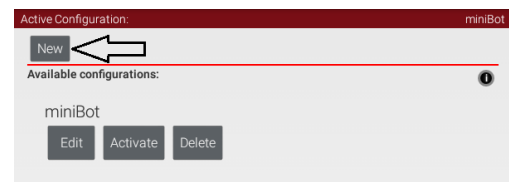
Use a 3-pin JST PH cable to connect the RS485 port on the Control Hub to the Expansion Hub.



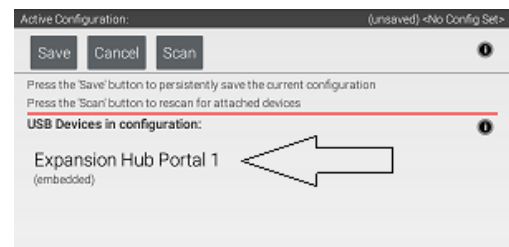
From the Driver Station choose “Configure Robot”



Select “New” in the top left hand corner.



Select “Expansion Hub Portal 1”



Now you have two Expansion Hubs to choose from. “Expansion Hub 1” is the embedded device of the Control Hub or the Android Device the Expansion Hub is connected to via USB.

“Expansion Hub 2” is the connected Expansion Hub that is communicating over RS485.


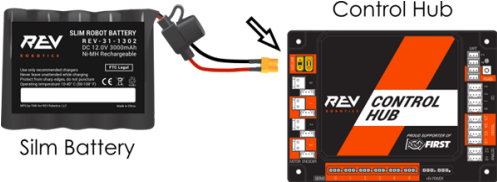


Configure and program as necessary. Please see the [Configuration](#) section of for an overview of configuration.

Managing the Control System

Resetting the WiFi Network

The Control Hub can be reset to the default WiFi settings. This will reset the password and SSID name while keeping Op Modes and the Robot Controller application installed on the Control Hub.

Step	Image
Press and hold the button on the front of the Control Hub.	
While pressing the button, power on the Control Hub.	
Release button when the Control Hub LED flashes PINK. When the Control Hub flashes BLUE then GREEN, it has completed the reset and is ready to connect.	

Updating Firmware

Updating the Expansion Hub Firmware

The Control Hub has its own built-in REV Robotics Expansion Hub board. The purpose of the Expansion Hub board is to facilitate communication between the Control Hub's Android controller and the motors, servos, and sensors of the robot. Periodically, REV Robotics will release new versions of the firmware which contains fixes and improvements for the Expansion Hub. The firmware releases are in the form of a binary (".bin") file.

There are two boards within the Control Hub: an Expansion Hub and an Android controller. The Expansion Hub board built into the Control Hub, facilitates a line of communication between the built in Robot Controller and the motors, servos, and sensors. In order to improve the quality of the Hubs, REV Robotics will release firmware updates for the Expansion Hub. When a firmware release occurs, both Control Hub and Expansion Hub users will need to update their Expansion Hub firmware to the newest version.

You can use the *Manage* interface to upload the firmware file to the Control Hub. You can then use a Driver Station that is connected to the Control Hub to initiate the firmware update. You can download the latest firmware below.

[Download the Latest REV Hub Firmware - Version 1.08.02](#)

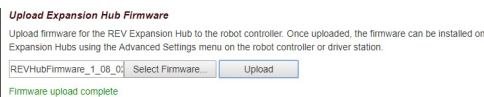
Updating the Expansion Hub Firmware

1. On the *Manage* page of the Control Hub user interface, press the *Select Firmware* button to to select the firmware file that you would like to upload.



An *Upload* button should appear after you successfully selected a file.

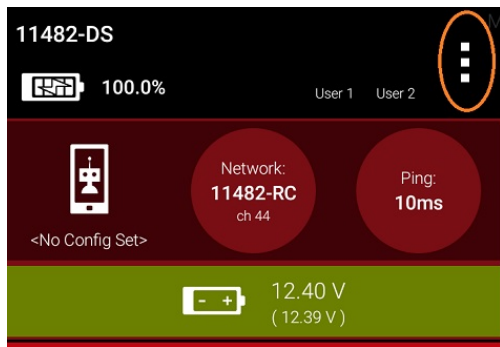
2. Press the *Upload* button to upload the firmware file from your computer to the Control Hub.



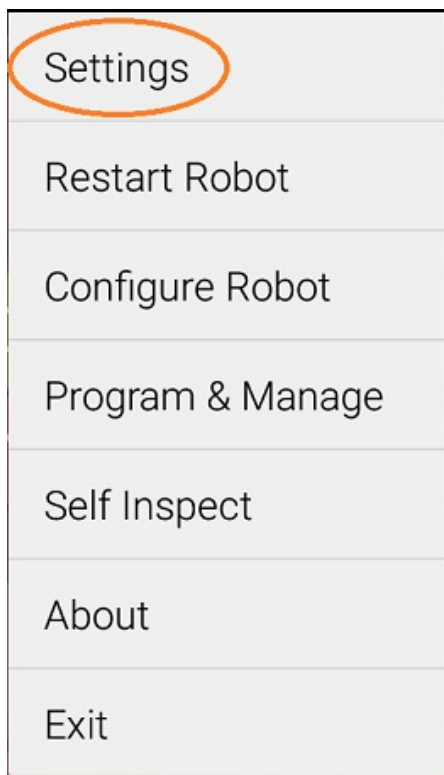
The words "Firmware upload complete" should appear once the file has been uploaded successfully.



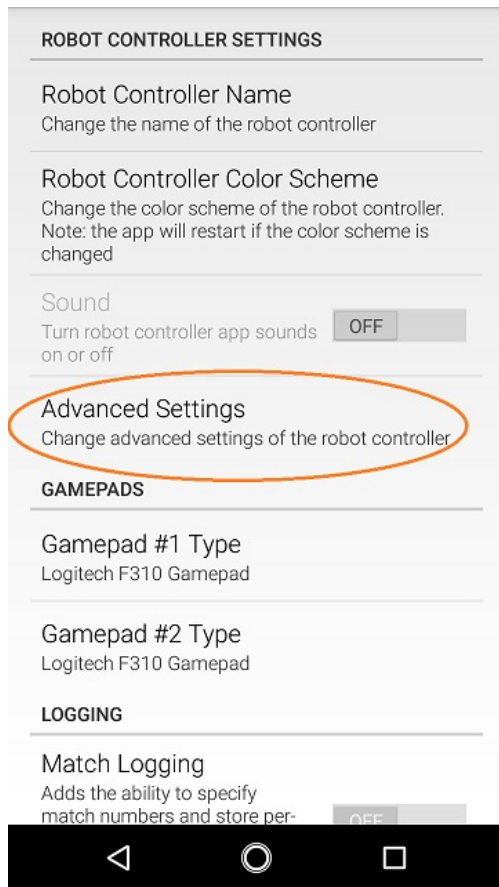
3. On the Driver Station, touch the three dots in the upper right hand corner to display a pop-up menu.



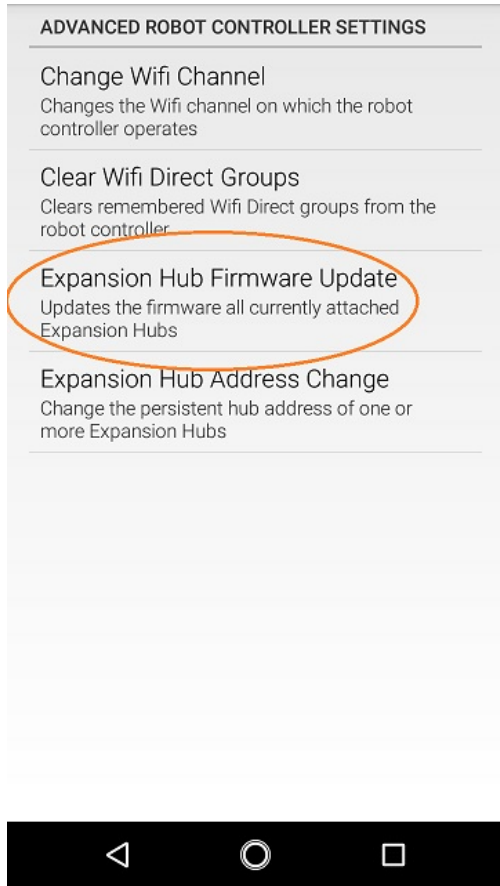
4. Select *Settings* from the pop-up menu to display the Settings activity.



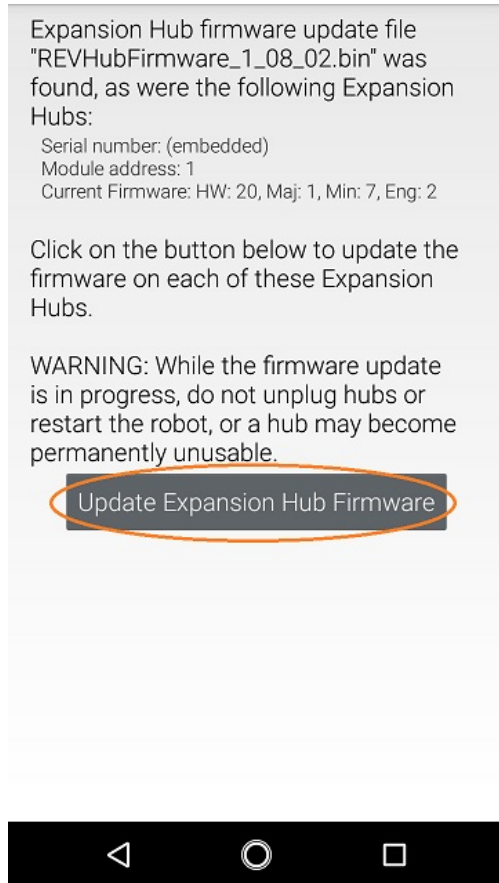
5. On the Driver Station, scroll down and select the *Advanced Settings* item (under the *ROBOT CONTROLLER SETTINGS* category).



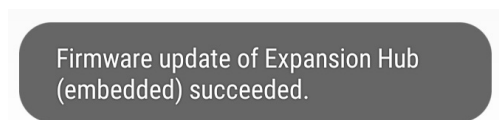
6. Select the *Expansion Hub Firmware Update* item on the *ADVANCED ROBOT CONTROLLER SETTINGS* activity.



7. If a firmware file that is different from the version currently installed on the Expansion Hub was successfully uploaded, the Driver Station should display some information about the current firmware version and the new firmware version. Press the *Update Expansion Hub Firmware* button to start the update process.



8. A progress bar will display while the firmware is being updated. Do not power off the Control Hub/Expansion Hub during this process. The Driver Station will display a message when the update process is complete.



Firmware Changelog

Version 1.8.2 (Latest Version)

- Improved USB recovery in case of fault event (e.g. ESD fault)
- Improved DC motor output linearity
- Improved closed-loop control modes
- Improved I2C speeds
- Minor bug fixes

[Download REV Hub Firmware Version 1.8.2](#)

Version 1.7.2

- Fixes a bug where encoder counts would occasionally reset.

[Download REV Hub Firmware Version 1.7.2](#)

Version 1.7.0

- Fixes a bug where some I2C sensors can lock up the bus causing other additional performance issues.
- Added new status LED blink code:
 - Blinking orange indicates the Hub is only powered over USB. In other words, turn on your main power switch!
- Other minor performance tweaks.

[Download REV Hub Firmware Version 1.7.0](#)

Version 1.6.0

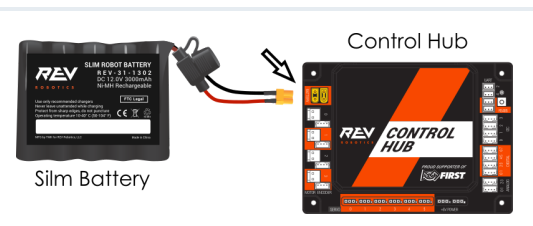

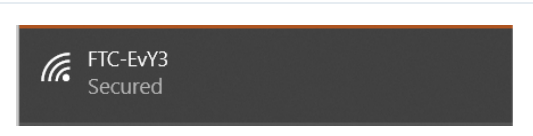

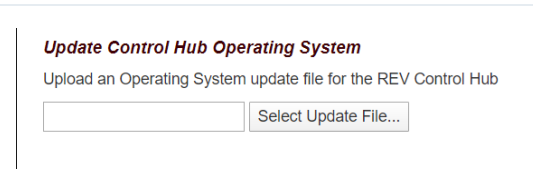
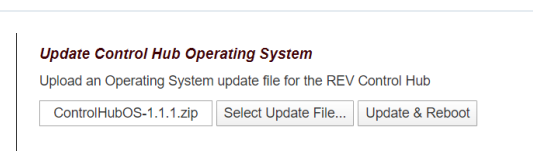
- Original Release

[Download REV Hub Firmware Version 1.6.0](#)

Updating Operating System

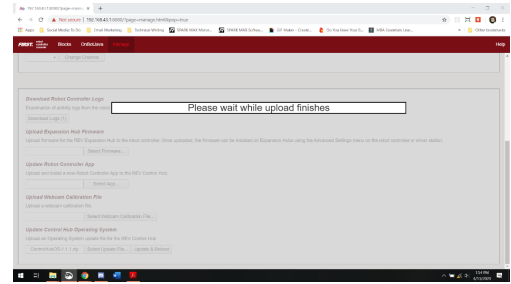
The Control Hub's Operating System is field upgradable. New updates are released to incorporate fixes, improvements, and new features as they are developed. Using a web browser and a PC, follow the steps below for updating the Operating System on the Control Hub. You can download the latest Operating System below.

Download the Latest REV Control Hub Operating System - Version 1.1.1

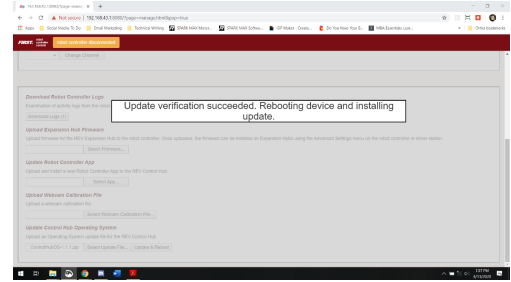
Step	Image
<p>Power on the Control Hub, by plugging the 12V Slim Battery (REV-31-1302) into the XT30 connector labeled "BATTERY" on the Control Hub.</p>	
<p>The Control Hub is ready to connect with a PC when the LED turns green. Note: the light blinks blue every ~5 seconds to indicate that the Control Hub is healthy.</p>	
<p>Connect to the Control Hub's WiFi Network. If it is not renamed, the name will begin with either "FIRST-" or "FTC-".</p>	
<p>Open a browser and navigate to the FIRST Robot Controller Console (type 192.168.43.1:8080 in the navigation bar). Select the Manage Tab.</p>	
<p>Scroll down to "Update Control Hub Operating System" and press the "Select Update File" button.</p>	
<p>Choose the latest version downloaded in Step 1 and press the "Update & Reboot" button.</p>	



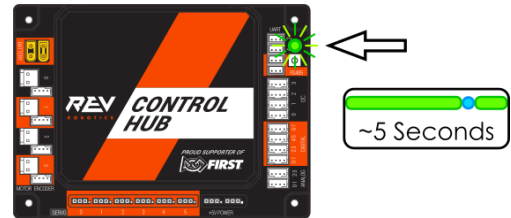
Keep the Control Hub powered while the upload finishes.



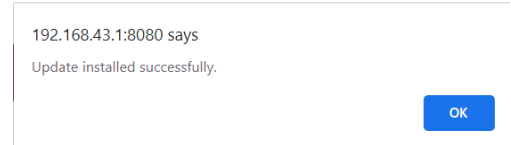
Keep the Control Hub powered while the update is installed.
The Control Hub will reboot to complete the update.



When the OS update has completed, the Control Hub LED will switch from blue, back to its normal blink pattern.



Reconnect your computer to the Control Hub network and verify that the update was a success.



Operating System Changelog

Version 1.1.1 - Latest Version

- Fixed bug where Wifi access point would sometimes fail to start after an Operating System update
- Stopped the FtcAccessPointService UI auto-starting on boot
- Allowed WiFi beacon rate to be changed by the FTC Robot Controller app

Version 1.1.0

- Improved reliability of making changes to WiFi access point settings
- Updated to latest Realtek WiFi driver
- Increased WiFi beacon rate to 6mbps, which reduces congestion when many Control Hubs are being used in an area
- Enabled 802.11w, which prevents WiFi deauthentication attacks when the Control Hub is used with a client device that also supports 802.11w
- Added WifiLog.txt file for debugging and disconnection analysis
- Improved reliability of FtcAccessPointService UI (accessed through an HDMI monitor)
- Added 5 GHz channels to FtcAccessPointService UI
- Ensured app data is not lost when installing a Robot Controller with a different signature via the Manage webpage
- Fixed issue where WiFi SSID would sometimes be AndroidAP

Source Files for Control Hub OS:

- [Linux Kernel Source](#)
- [U-Boot Source](#)

Updating Robot Controller Application

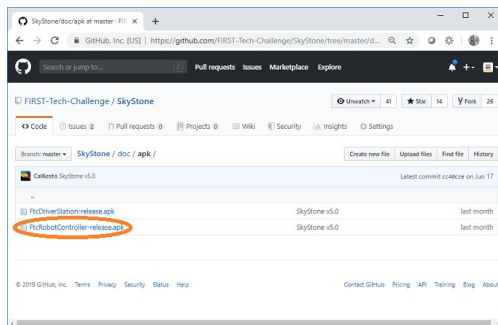
The Robot Controller Application is periodically updated to incorporate fixes, improvements, and new features as they are developed. Follow the steps below for updating the Robot Controller Application.

! If you update your Robot Controller, then you should also update your Driver Station software to the same version number.

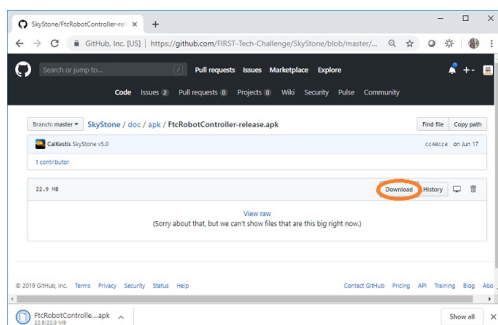
Updating the Robot Controller App

1. Go to the current season's GitHub repository and look in the "doc/apk" subdirectory to download the appropriate APK file. For the Skystone season, the APK files can be found [here](#).

2. Click on the *FtcRobotController-release.apk* link in the repository to access the Robot Controller file.



3. Click on the *Download* button to download the Robot Controller app as an APK file to your computer.



4. On the *Manage* page, click on the *Select App* button to select the Robot Controller app that you would like to upload to the Control Hub.

Update Robot Controller App

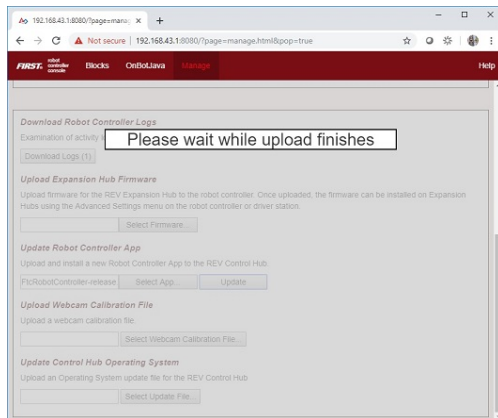
Upload and install a new Robot Controller App to the REV Control Hub.

FtcRobotController-release

An *Update* button should appear if an APK file was

successfully selected.

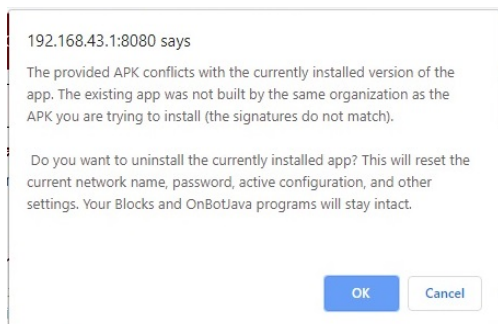
5. Click on the *Update* button to begin the update process.



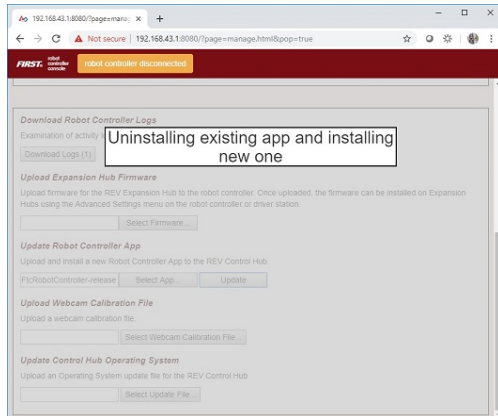
6. During the update process, if the Control Hub detects that the digital signature of the APK that is being installed is different from the digital signature of the APK that is already installed, the Hub might prompt you to ask if it is OK to uninstall the current app and replace it with the new one.

This difference in digital signatures can occur, for example, if the previous version of the app was built and installed using Android Studio, but the newer app was downloaded from the GitHub repository.

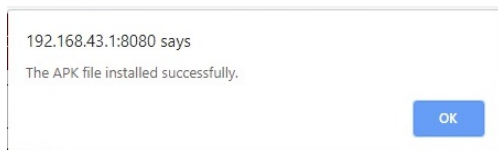
Press *OK* to uninstall the old app and continue with the update process.



7. If the update process had to uninstall the previous version of the Robot Controller app, the network name and password for the Control Hub will be reset back to their factory values. If this happens, then you will need to reconnect your computer to the Control Hub using the factory default values.




8. When the update process is complete and you have successfully reconnected your computer to the Control Hub's network, you should see an "installed successfully" message on the *Manage* web page.



Troubleshooting the Control System

One of the key aspects of troubleshooting is understanding the most common issues that occur in a system. Once those problems, and their indicators, are defined -a flow has to be created. For example, a check engine light in a car indicates any number of issues. When a cars check engine light comes on, a mechanic pulls the codes from the car to narrow down the issue to a specific part of the engine. Even if the code leads to a specific part of the engine, like the transmission, it is not always indicative of the exact problem. However, there is a process flow. Each step narrows down the problem to a potential solution. Troubleshooting the REV Control system is no different!


 The status LED is the REV Control System equivalent to the check engine light mentioned in the example. Visit the [LED Blink Code](#) section to understand what each code is and what it indicates.

Many issues can be solved by systematic troubleshooting without needing to contact REV Support. Take a look at the troubleshooting tips below for help in determining the cause of the issue you are seeing. Should you need to contact us, describing the steps you've taken in detail will help us get you up and running quickly. The section is divided by general best practices, Control Hub ([REV-31-1595](#)) troubleshooting and Expansion Hub ([REV-31-1153](#)) troubleshooting.

General Best Practices

Before diving into common troubleshooting paths its important to understand the general guidelines, or best practices, for Control System Health.

- **Charge the Battery** - While a charged battery and phone are crucial to a healthy control system in general; it is also helpful to ensure batteries and phones are charged before a match.
- **Update** - The applications, firmware, and operating system have periodic updates to improve the control system. Keeping the control system up to date ensures the best performance!
- **Isolate the Issue** - This is key to effective troubleshooting. Many issues can show the same symptom, so eliminating failure points one at a time is critical to finding the root cause.

 DO NOT plug a battery charger into either the Control Hub or Expansion Hub. It will damage the Hub and cause eventual device failure

Control Hub Troubleshooting

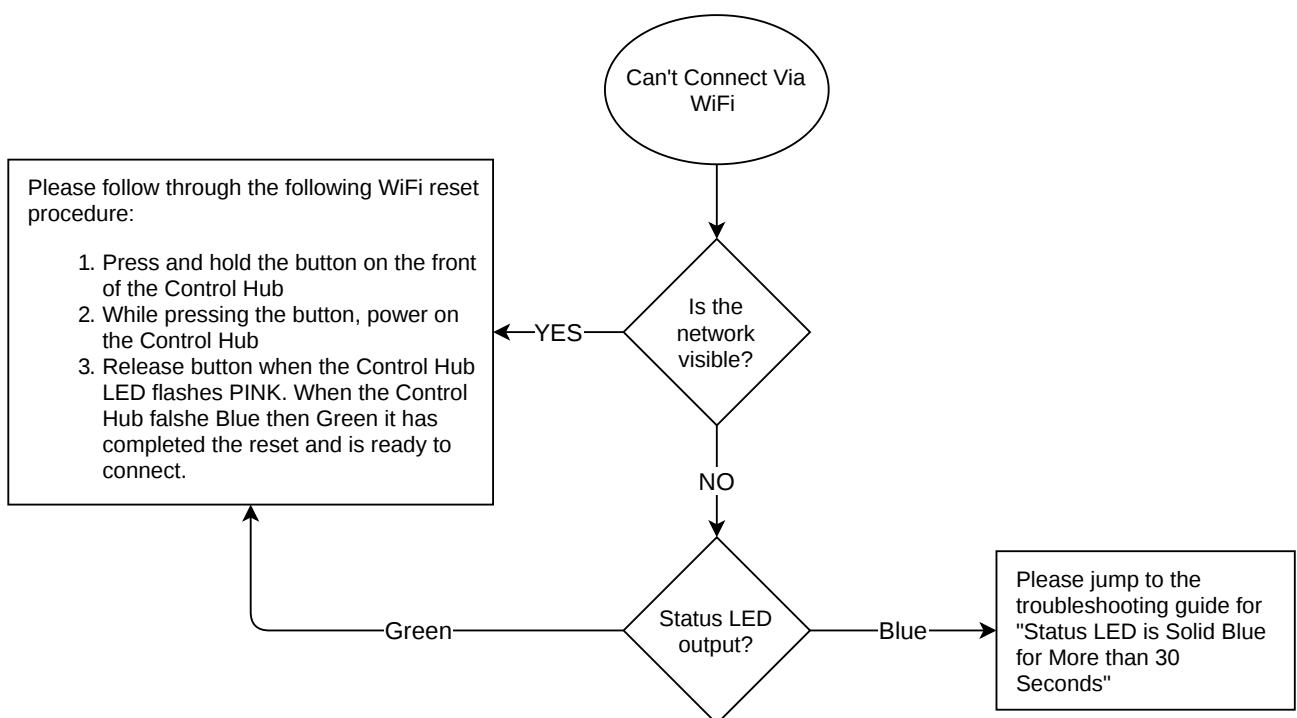
The following questions consider common indicators of issues seen in the Control Hub. Think about what the potential indicators your Hub is currently exhibiting and consider the following questions:

Is the Driver Station phone unable to connect to to the Control Hub WiFi?	Yes
Is the Driver Station connected to the WiFi but not showing a ping or any other signs of communication?	Yes
Has the Status LED been solid blue for longer than 30 seconds (after start up)?	Yes

Sometimes, its possible that you may have to try multiple routes when troubleshooting. Perhaps, at first, it seems like the only indicator is that the Driver Station wont connect to the Hub WiFi; but it becomes apparent that the Status LED has been solid blue the entire time since startup. In this guide, the process flow charts have been crafted to tie in to each other where indicators may overlap.

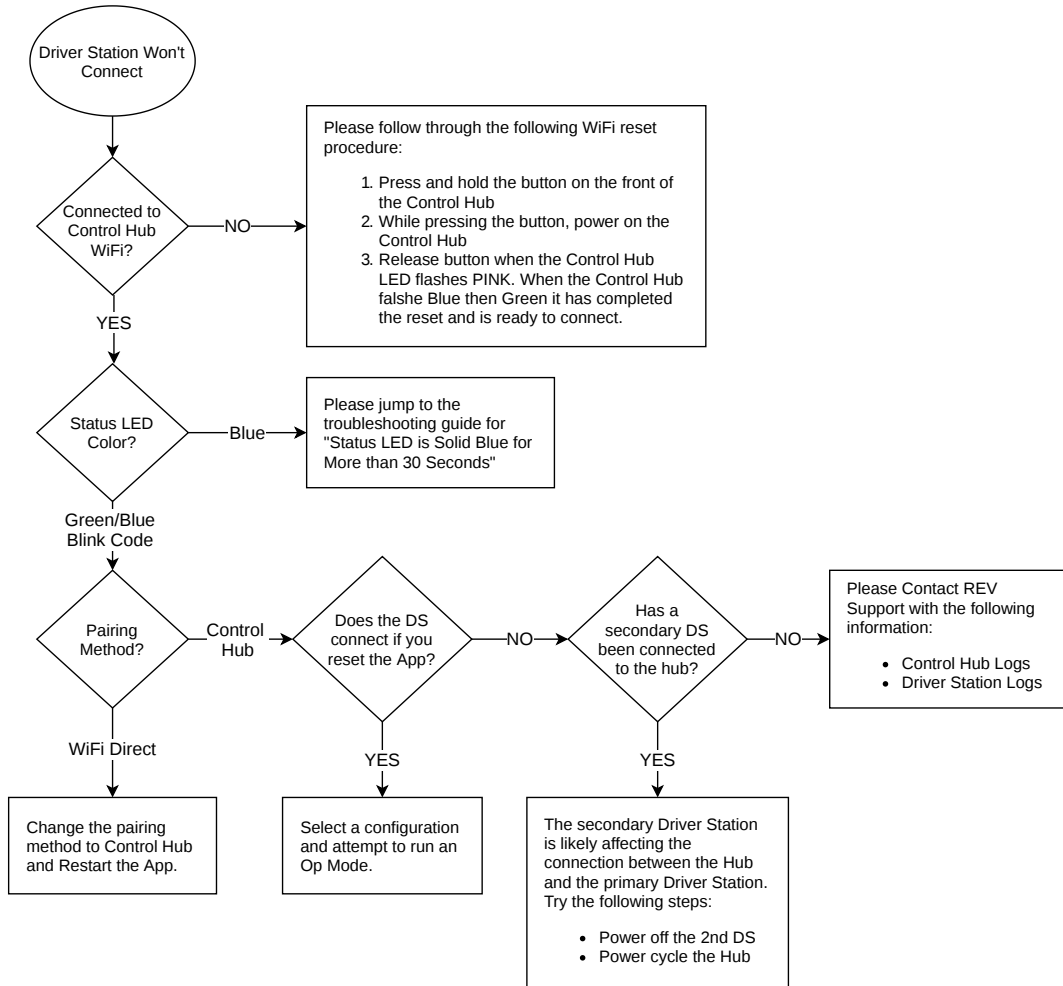
i If a path in this guide does not resolve the issue please contact REV Robotics Support at support@revrobotics.com

Cant Connect to the Hub Via WiFi



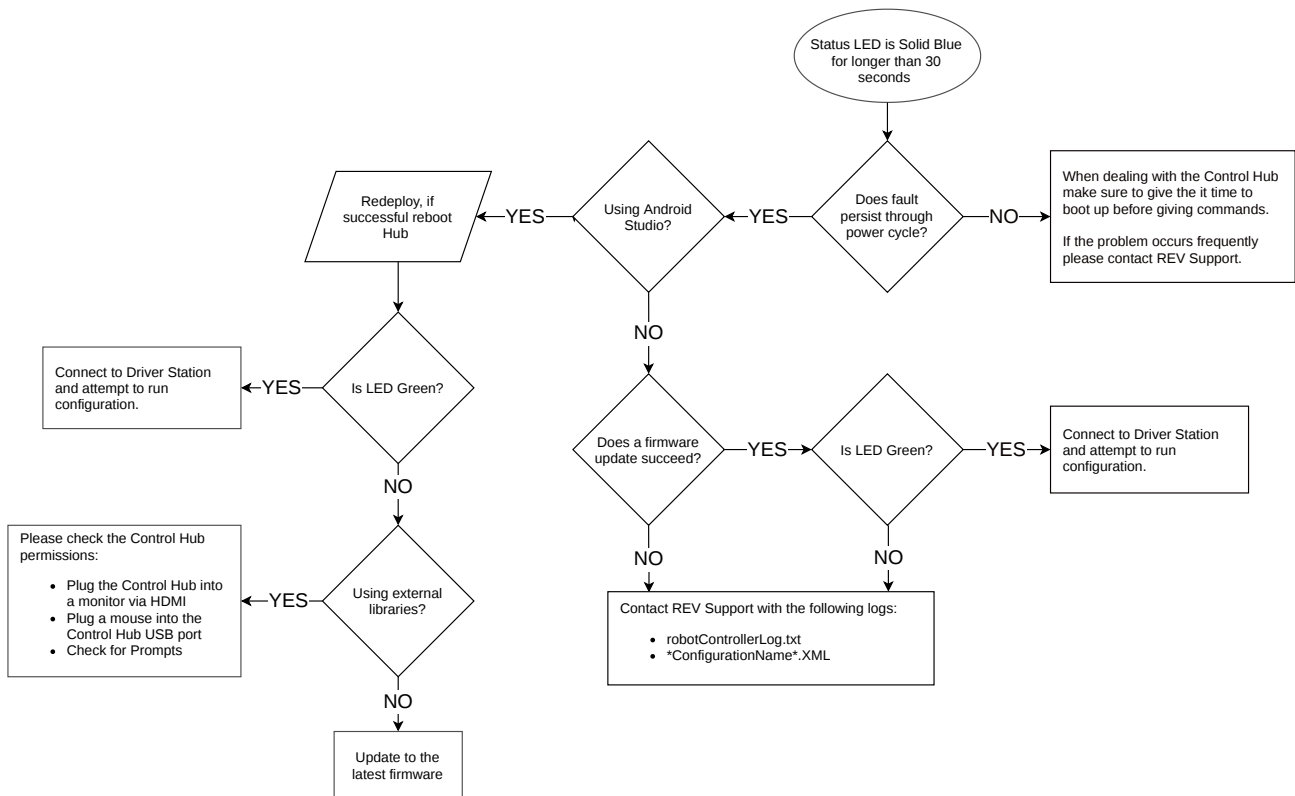
! The WiFi reset will down grade the WiFi connection to 2.4GHz. If you have an android device with 5GHz you may want to switch the WiFi Band in order to run on 5GHz. *Check out the [Changing WiFi Channel](#) Section to learn more about making this switch.*

Driver Station Won't Connect



i Need help getting the Log Files to send to REV Support? See [Downloading Log File](#) for more information.

Status LED is Solid Blue for Longer than 30 Seconds



i Need help getting the Log Files to send to REV Support? See [Downloading Log File](#) for more information.

Expansion Hub Troubleshooting

The following sections, "[Common Indicators and their Solution Steps](#)," provides common indicators of issues seen in the Expansion Hub. Think about what the potential indicators your Hub is currently exhibiting and consider the following questions:

- Did you perform a firmware update before the Hub began to have issues?
- What is the behavior of the Status LED on the Expansion Hub?
- Is the Driver Station showing an error message 'Cant find the Expansion Hub Portal'?
- Did the Robot Controller app open when you plugged in the RC phone and gave power to the Hub?
- Are you experiencing issues with communication between a primary and secondary Hub?

i If a path in this guide does not resolve the issue please contact REV Robotics Support at support@revrobotics.com

Common Indicators and their Solution Steps

- The firmware update failed and the Hub is unresponsive
 - Try a [Firmware Update](#)
- The LED on the Expansion Hub is not lighting up
 - Try a [Firmware Update](#)
 - [The LED is still not lighting up](#)
- The Hub is not being recognized or communicating with the phones
 - Try doing the [Hub Startup Procedure](#)
- There are issues seeing a secondary Hub
 - Try doing the [Hub Startup Procedure](#)
 - [There are still issues seeing the secondary Hub](#)

Firmware Update

1. Install the REV Hub Interface Software (www.revrobotics.com/software)
You may need FTDI Drivers depending on the version of Windows you are running:
<http://www.ftdichip.com/Drivers/VCP.htm>
2. Plug your Expansion Hub into your computer
3. Boot the REV Hub Interface Software
4. Select Firmware tab then click the Choose .bin file button
 1. Select the version of the firmware you want to install (hint it comes with the latest and greatest version)
5. Let the firmware install
6. After install is finished, select Connect on the REV Interface Software.
 1. Hub should read connected and have a Green Light showing connection

USB Serial Converter Check


1. Plug your Expansion Hub into a Windows PC
2. Open the Device Manager in Settings
3. Click the arrow next to Universal Serial Bus Controllers
4. Find USB Serial Converter under the menu
5. If this is not present there maybe a larger issue with your hub. Email support@revrobotics.com with details of the steps you have taken so far, and any order numbers for the Expansion Hub (if you have them)



If you are using a Mac you can use System Information in Lion or later (or System Profiler in Snow Leopard and earlier versions of Mac OS) in Spotlight (press ⌘ and Space). The program is in /Applications/Utilities and is the tool to see the connected USB devices and other hardware details.

Hub Startup Procedures

1. Unplug the USB from your RC phone
2. Power off the main robot switch (turn off 12V power from the Expansion Hub(s))
3. Wait a few seconds
4. Turn on the Main Robot Switch (supply 12V power to the Expansion Hub(s))
5. On your RC phone, press the square button and the swipe to close the FTC RC app
6. Plug your RC phone into the USB– the FTC app should automatically open
 1. If the app doesn't automatically open you do not have a good connection from the Expansion Hub to the Phone. Check your cables first, followed by the micro and mini USB connections.
 2. Consider using some form of strain relief (like the [REV USB Retention Mount](#) or one of the many 3d printable options available on places like Thingiverse) to keep the USB-mini port from being damaged.

 *If the issues persists after applying the Retention Mount try running through the [Firmware Update](#) procedure.*

Persistent inability to see a secondary Hub

1. See the Expansion Hub Guide (www.revrobotics.com/resources)
2. Rerun through the configuration and addressing steps in the guide
 1. Make sure the leader hub is addressed as 2 and the follower hub is addressed as 3.
3. Connect the two Hubs to each other has described in the Expansion Hub Guide
4. Install the REV Hub Interface Software on a PC if not already done
5. Plug the Leader Hub into the Computer with the REV Hub Interface Software installed.
6. Press Connect
7. The DC Motor page should split in two with 8 total motor controls available if both are functioning properly.
 1. Also, you can look at the LEDs on the Hubs. They should be solid Green with occasional blue blinking.

Downloading the Log File


Occasionally, troubleshooting problems in the REV Control System will require that a log file is looked over for potential indicators. Often the first log that is considered is the Robot Controller log, as they are relatively easy to decipher and can be pulled from the Control Hub. However, other indicators and pathways in the [troubleshooting process](#) may require you to pull the XML files off the Hub in addition the the Robot Controller logs.

There are two ways to access the Robot Controller logs. The first way is to search through the Control Hub ([REV-31-1595](#)) files by plugging the hub into the computer. The second way is to connect to the Robot Control Console and download the logs to the computer from there. The table below highlights the scenarios you would use one process over the other.


File Search	Robot Controller Console
The Hub is NOT able to establish a connection with a laptop to open the Robot Control Console	The Hub has a connection but the logs need to be accessed to determine the Firmware version
The XML Files need to be accessed in addition to the Robot Controller Logs	

File Search

1. Provide 12v Power to the Control Hub.
2. Plug the USB-C Cable into the top board of the Hub and into a PC
3. Navigate to This PC\Control Hub v1.0\Internal shared storage should have a file named robotControllerLog.txt (this is for on windows. Mac should have a similar file explorer to pull the txt file)

 The robotControllerLog.txt can either be open via Notepad++ and looked over or sent to REV Support via an email to be further troubleshooted.

4. While in the This PC\Control Hub v1.0\Internal shared storage location navigate to a folder called "FIRST." The folder should have XML files with a naming convention that mirrors the names of the robot configuration.

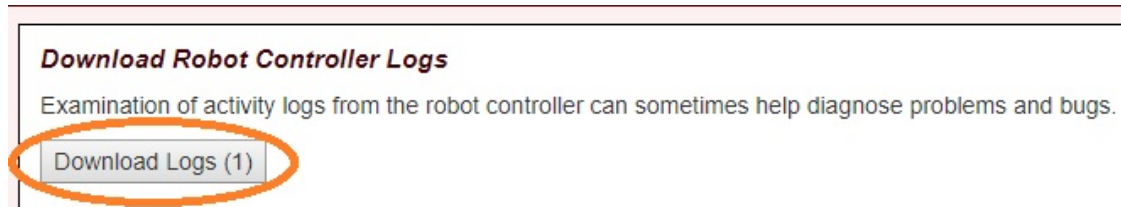
 Problems in the XML files will only be able to be interpreted by REV personnel.

Robot Controller Console

1. Open the Robot Controller Console

2. Select the **Manage** page

3. Press the Download Logs button



4. Check for the robotControllerLog.txt in the Downloads Directory of the Computer







5. Open the Logs via a text editor, like Notepad++, to view the contents of the log or send the logs to REV Support

```
1 |----- beginning of main
2 | 01-01 00:02:04.230 1611 1611 V AppUtil : initializing:
  | getExternalStorageDirectory()=/storage/emulated/0
3 | 01-01 00:02:04.265 1611 1611 I AppUtil : found usbFileSystemRoot: /dev/bus/usb
4 | 01-01 00:02:04.268 1611 1654 V RobotCore: saving logcat to
  | /storage/emulated/0/robotControllerLog.txt
5 | 01-01 00:02:04.268 1611 1654 V RobotCore: logging command line: exec logcat -f
  | /storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
  | UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
  | dalvikvm:W *:V
6 | 01-01 00:02:04.325 1611 1611 V AppUtil : rootActivity=PermissionValidatorWrapper
7 | 01-01 00:02:04.336 1611 1611 I PermissionValidatorActivity: onCreate
8 | 01-01 00:02:04.467 1611 1654 I RobotCore: Done running ps
9 | 01-01 00:02:04.473 1658 1658 I sh : type=1400 audit(0.0:60): avc: denied { read }
  | for name="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
  | tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
10 | 01-01 00:02:04.473 1658 1658 I sh : type=1400 audit(0.0:61): avc: denied { open }
  | for path="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
  | tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
11 | 01-01 00:02:04.511 1611 1654 I RobotCore: Done running exec logcat -f
  | /storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
  | UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
  | dalvikvm:W *:V
12 | 01-01 00:02:04.634 1611 1611 D skia : JPEG Decode 71
13 | 01-01 00:02:04.640 1611 1611 I PermissionValidatorActivity: onStart
```

LED Blink Codes

The RGB LED located on the Control Hub (REV-31-1595) and Expansion Hub (REV-31-1153) near the RS485 ports provides user feedback regarding the status of the Hub. Below is a Table of the Blink Codes

Firmware Version 1.07.00 or Higher LED Codes


LED Status	LED Description	When	Hub Status
	Solid Blue	At Boot	Control Hub has power; Battery is >7V and is waiting to initialize communications.
	Solid Blue	Anytime	Hub is waiting for communication with the Driver Station Host. Control Hub has power; Battery is >7V.
	Solid Green with one or more blue blinks every ~5 Seconds	Anytime	Hub has power and active communication with the Android Platform. The number of blue blinks is the same as the Hub's address. The factory default address is 2 ().
	Blinking Blue	Anytime	Keep alive has timed out. Fault will clear when communication resumes.
	Blinking Orange	Anytime	Battery Voltage is lower than 7V. Either the 12V battery needs to be charged, or the Expansion Hub is running on USB power only. This fault will clear when battery voltage is raised above 7V. This will not be overwritten by the keep alive timeout pattern.

REV Hub Interface Software

The REV Hub Interface is a beta software allowing for a direct connection from a REV Expansion Hub and its peripherals to a Windows PC.


This interface provides a method for teams to prototype with motors, servos, and sensors in a way that is faster and easier than setting up an entire robot control system. It is also a valuable troubleshooting tool that can help isolate the cause of an issue and determine if it is electrical or software related. The REV Hub Firmware can also be updated and recovered through this interface in addition to the Robot Controller Application.

Download the Latest Hub Interface Software - Version 1.2.0

 The REV Hub Interface Software only works with the REV Expansion Hub and not the REV Control Hub

System Requirements

- Operating System: Windows 7 or newer*
- Processor: 64-bit
- RAM: Yes


 The newest versions of Windows should automatically install the required USB drivers. Alternatively, you can download the latest drivers from the [FTDI VCP website](#).

Installation Instructions

1. Download the Hub Interface software installer above.
2. Run the installer.
3. Run the REV Hub Interface Software from the Windows Start Menu or the desktop shortcut

Connecting and Controlling an Expansion Hub

1. Connect your Expansion Hub to the computer with a USB A to USB Mini-B cable.
2. Run the REV Hub Interface Software.
3. The software will scan and connect to the Expansion Hub. The various peripheral tabs will populate with controls once connected.

 Some peripherals, such as DC Motors and Servo Motors, require a battery to be connected to the Expansion Hub in order to operate through the REV Hub Interface.

Alternative Installation Method

You may also download the following zip file if you would rather unzip the application in a directory of your choice. This method shouldn't require administrator privileges.

[REV Hub Interface Software Zip File](#)

LATEST HUB INTERFACE SOFTWARE CHANGE LOG - VERSION 1.2.0

- Display encoder values on 'DC motors' tab.
- Added support for REV Color Sensor V3.
- Display proximity values along with RGBC for REV color sensors.
- Display REV Hub Interface version on the 'Firmware' tab.
- Changed behavior of 'INIT' and 'POLL' buttons on 'I2C'. User can no longer poll a device until it has been successfully initialized.
- Added ability to set LED pattern.
- Bug fix where 'POLL' had to be pressed twice to read values from the IMU.
- Bug fix where status LED would continue to flash blue the second time REV Hub Interface is connected.
- Allow user to press enter key to update motor/servo values.
- Fixed gyro labels on IMU tab and corrected units for linear acceleration.

Programming

Programming Language Options

When programming within the REV Control System, there are three programming tools to choose from: Blocks, OnBot Java, and Android Studio. The following section highlights basic information, use cases, and access examples for the three compilers.

Blocks

The Blocks Programming Tool is a visual, programming tool that lets programmers use a web browser to create, edit and save their op modes. This tool offers preset snippets of code that can be presented visually, using a drag-and-drop interface.

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp(name = "MyTankDrive (Blocks to Java)", group = "MyTankDrive")
public class MyTankDrive extends LinearOpMode {


    private DcMotor right_drive;
    private DcMotor left_drive;

    /**
     * This function is executed when this Op Mode is started
     */
    @Override
    public void runOpMode() {
        right_drive = hardwareMap.dcMotor.get("right_drive");
        left_drive = hardwareMap.dcMotor.get("left_drive");


        // Reverse one of the drive motors.
        // You will have to determine which motor to reverse.
        // In this example, the right motor was reversed.
```

The Blocks Programming Tool lacks the complexity of the Java based programming tools available, which makes it a great place to start for rookie or novice programmers. Rookie programmers can learn programming logic in an intuitive and easy-to-learn environment. Because the Blocks Programming tool is a web-based interface, where programs are saved directly to the robot, it is easy to access on most devices to make code changes.

Accessing Blocks

 This section assumes that you have already gone through the [Configuring Android Device](#) process and that you have JavaScript enabled web browser.

1. Go to WiFi Settings, on a Windows 10 Computer, by clicking on the Wi-Fi symbol.
2. Once the list of available Wi-Fi networks in the vicinity is displayed select the network that matches the name of your WiFi access point.
3. Enter the password you set during the [configuration phase](#).
4. Once connected, open a JavaScript enabled browser (FIRST recommends Google Chrome).
5. Go to IP Address **http://192.168.43.1:8080**
6. At the top of the Robot Controller Console Page, there should be 3 menu options: Blocks, OnBot Java, and Manage. Choose Blocks.
7. Check out the [First Op Mode](#) section to begin coding!

 Passwords are case sensitive. If you do not remember your password, check the Program and Manage menu option on *your driver station*.

OnBot Java

A text-based programming tool that lets programmers use a web browser to create, edit and save their Java op modes.


```
1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
5 import com.qualcomm.robotcore.hardware.DcMotor;
6 import com.qualcomm.robotcore.hardware.DcMotorSimple;
7
8 @TeleOp(name = "MyTankDrive (Blocks to Java)", group = "")
9 public class MyTankDrive extends LinearOpMode {
10
11     private DcMotor right_drive;
12     private DcMotor left_drive;
13
14     /**
15      * This function is executed when this Op Mode is selected from the
16      * menu.
17      */
18     @Override
19     public void runOpMode() {
```

Build started at Mon Jun 17 2019 15:45:25 GMT-0400 (Eastern Daylight Time)

Build finished in 2.9 seconds
Build succeeded!

OnBot Java is great for programmers with basic to advanced Java skills who would like to write text-based op modes. OnBot Java shares a web-based interface with the Blocks Programming tool. The web-based model is easy to access on most devices to make code change and reduces the need to have one set device for code changes.

Accessing OnBot Java

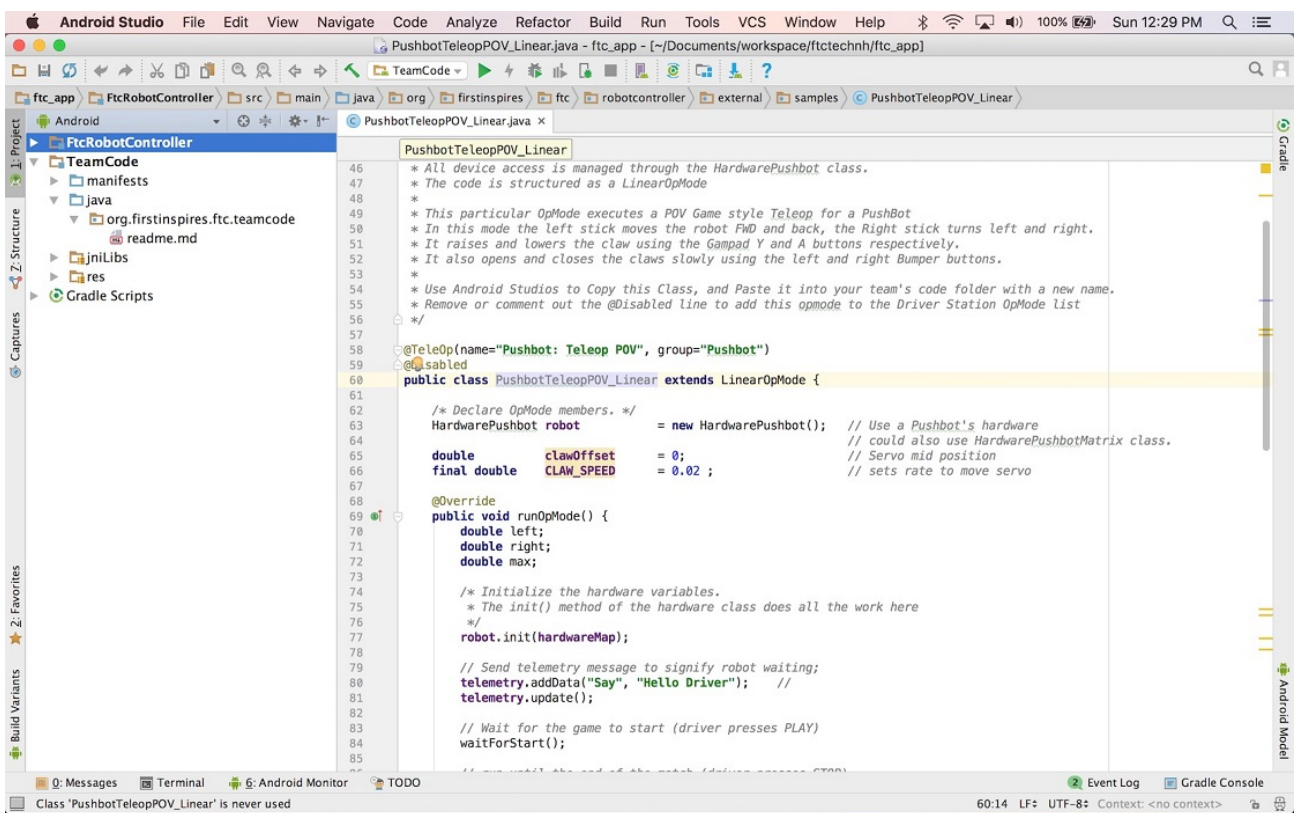
 This section assumes that you have already gone through the [Configuring Android Device](#) process and that you have JavaScript enabled web browser.

1. Go to WiFi Settings, on a Windows 10 Computer, by clicking on the Wi-Fi symbol.
2. Once the list of available Wi-Fi networks in the vicinity is displayed select the network that matches the name of your WiFi access point.
3. Enter the password you set during the [configuration phase](#).
4. Once connected open a JavaScript enabled browser (FIRST recommends Google Chrome).
5. Go to IP Address **http://192.168.43.1:8080**
6. At the top of the Robot Controller Console Page There should be 3 menu options Blocks, OnBot Java, and Manage. Choose OnBot Java
7. Check out the [First Op Mode](#) section to begin coding!

i Passwords are case sensitive. If you do not remember your password check the Program and Manage menu option on *your driver station*.

Android Studio - Java

An advanced integrated development environment for creating Android apps. This tool is the same tool that professional Android app developers use. Android Studio is only recommended for advanced users who have extensive Java programming experience.



```
46  /* All device access is managed through the HardwarePushbot class.
47  * The code is structured as a LinearOpMode
48  *
49  * This particular OpMode executes a POV Game style Teleop for a PushBot
50  * In this mode the left stick moves the robot FWD and back, the Right stick turns left and right.
51  * It raises and lowers the claw using the Gampad Y and A buttons respectively.
52  * It also opens and closes the claws slowly using the left and right Bumper buttons.
53  *
54  * Use Android Studios to Copy this Class, and Paste it into your team's code folder with a new name.
55  * Remove or comment out the @Disabled Line to add this opmode to the Driver Station OpMode List
56  */
57
58  @TeleOp(name="Pushbot: Teleop POV", group="Pushbot")
59  @Disabled
60  public class PushbotTeleopPOV_Linear extends LinearOpMode {
61
62      /* Declare OpMode members. */
63      HardwarePushbot robot = new HardwarePushbot(); // Use a Pushbot's hardware
64      // could also use HardwarePushbotMatrix class.
65      double clawOffset = 0; // Servo mid position
66      final double CLAW_SPEED = 0.02; // sets rate to move servo
67
68      @Override
69      public void runOpMode() {
70          double left;
71          double right;
72          double max;
73
74          /* Initialize the hardware variables.
75           * The init() method of the hardware class does all the work here
76           */
77          robot.init(hardwareMap);
78
79          // Send telemetry message to signify robot waiting;
80          telemetry.addData("Say", "Hello Driver"); //
81          telemetry.update();
82
83          // Wait for the game to start (driver presses PLAY)
84          waitForStart();
85
86          // Loop until the end of the match (driver presses STOP)
87      }
```

Android Studio allows programmer with an advanced understanding of Java a more powerful development environment to work in. It offers enhanced editing and debugging features not available with OnBot Java or Blocks. However, Android Studio is not a web-based software and will need a dedicated laptop to run on.

Accessing Android Studio

To learn about how to properly download and work with Android Studio please visit the [FTC Wiki](#).

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.*

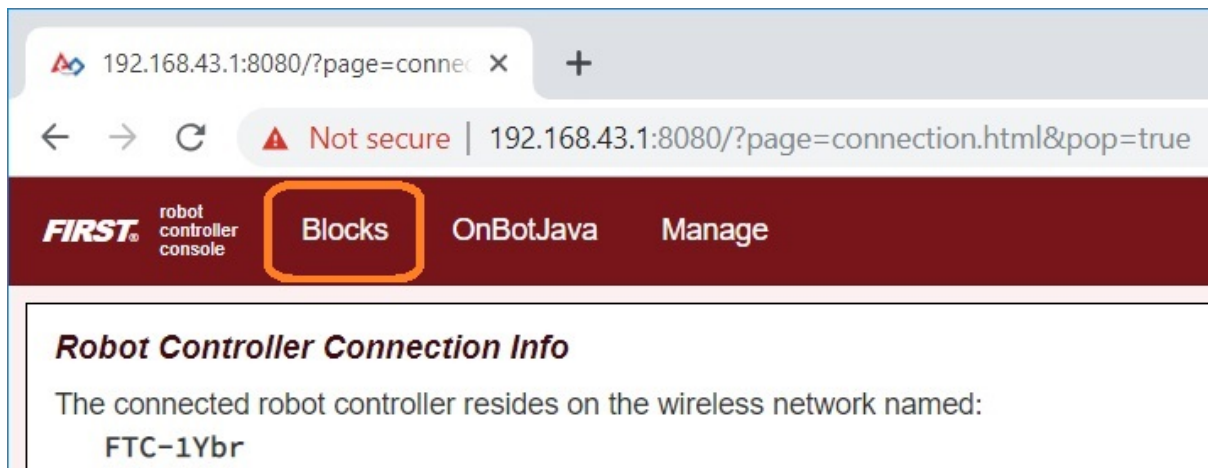
What is an Op Mode?

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. The 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

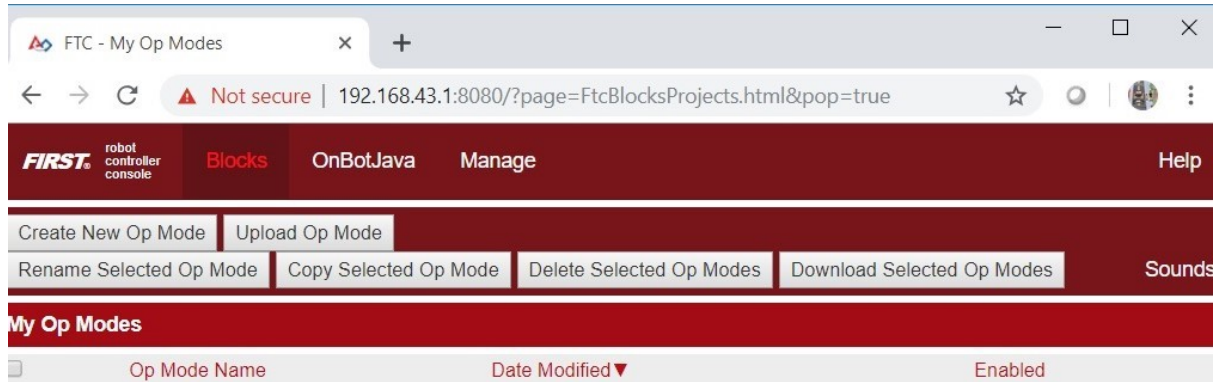
- Touch Sensor - digitalTouch
- Color/Range Sensor - sensorColorRange

 To learn how to create a configuration file for your robot check out [Configuring Your Hardware on the FTC Wiki](#)

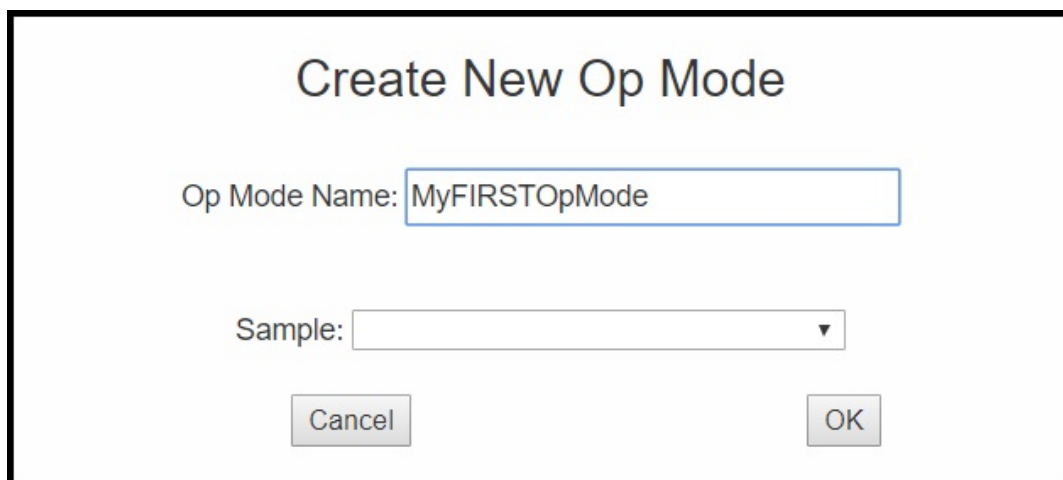
The following tab block will walk through how to make create an op mode. In both examples the op mode will be named **MyFIRSTOpMode**.

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"

A screenshot of a "Create New Op Mode" dialog box. The title is "Create New Op Mode". It contains a text input field labeled "Op Mode Name:" with the text "MyFIRSTOpMode" entered. Below it is a dropdown menu labeled "Sample:". At the bottom of the dialog are two buttons: "Cancel" on the left and "OK" on the right.

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

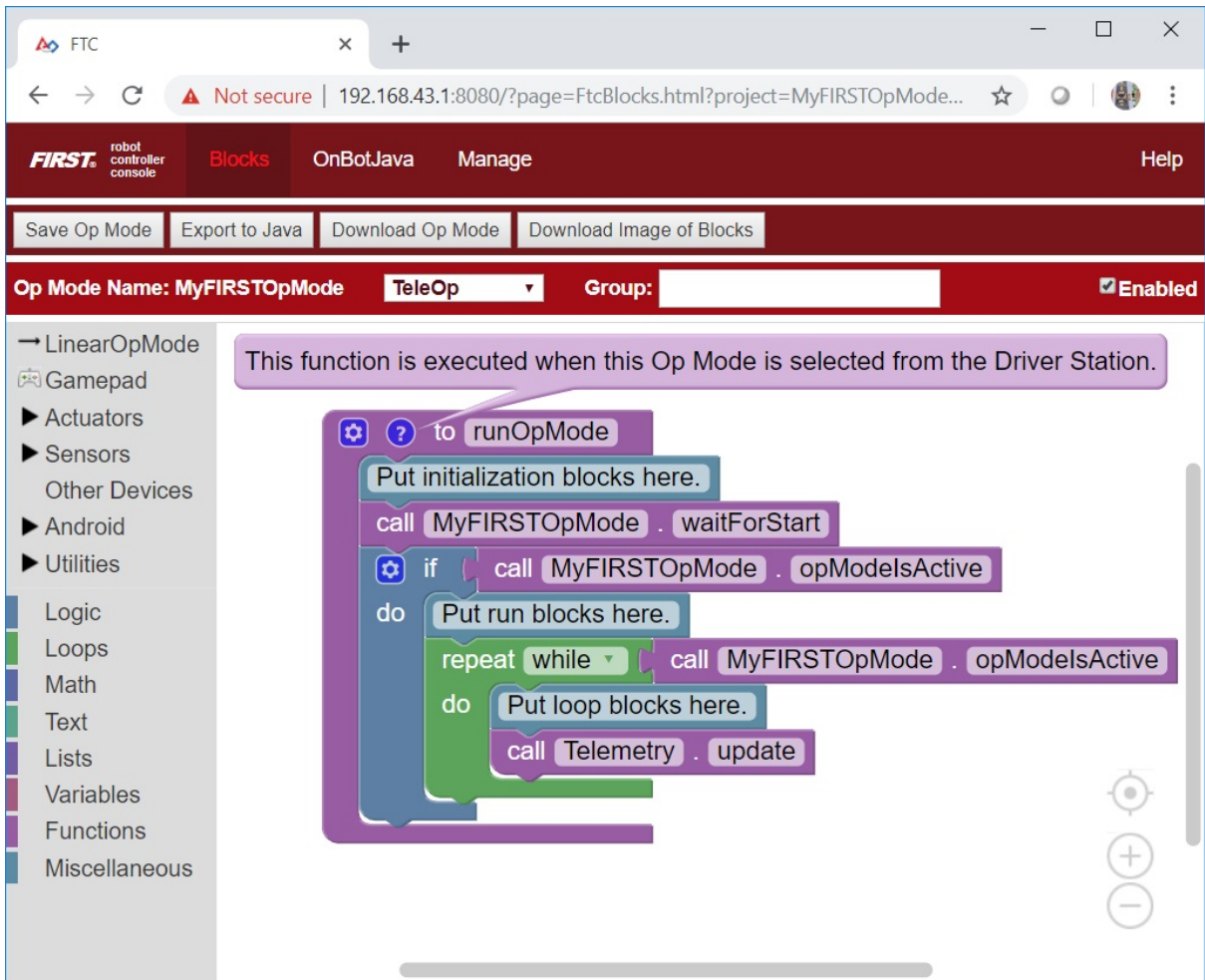
Save Op Mode Export to Java Download Op Mode Download Image of Blocks

Op Mode Name: MyFIRSTOpMode TeleOp Group: Enabled

LinearOpMode
Gamepad
Actuators
Sensors
Other Devices
Android
Utilities
Logic
Loops
Math
Text
Lists
Variables
Functions
Miscellaneous

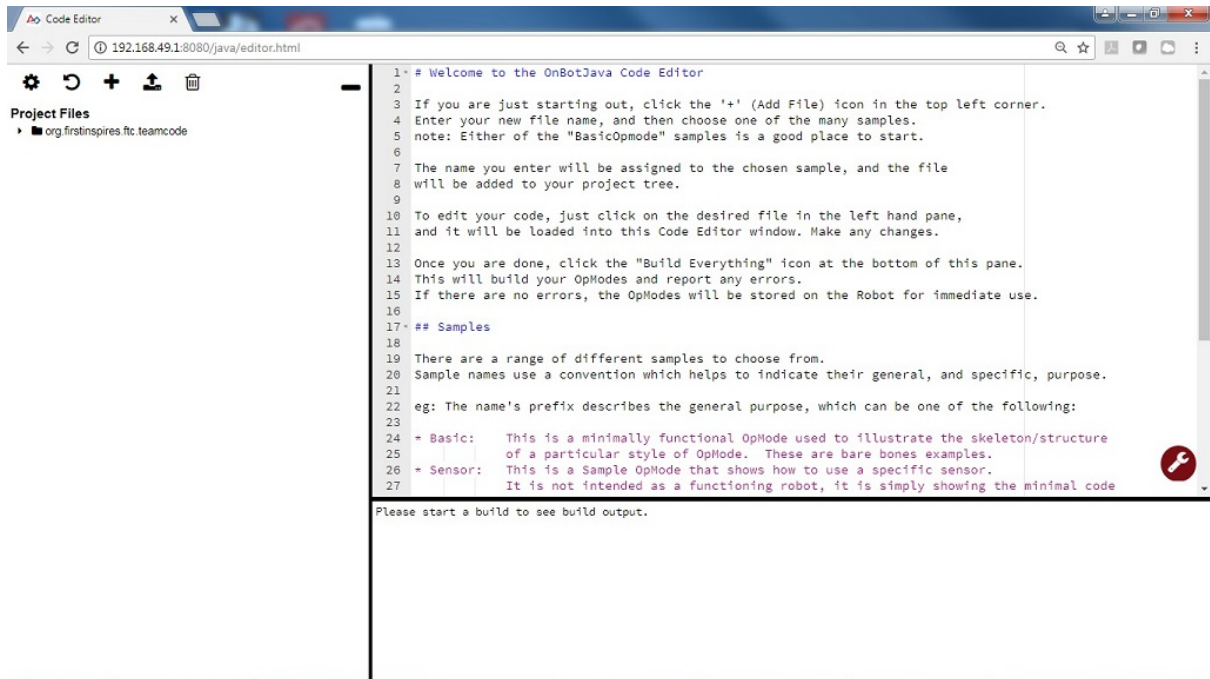
This function is executed when this Op Mode is selected from the Driver Station.

```
to runOpMode
  Put initialization blocks here.
  call MyFIRSTOpMode . waitForStart
  if call MyFIRSTOpMode . opModelsActive
  do
    Put run blocks here.
    repeat while call MyFIRSTOpMode . opModelsActive
    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.

The image shows a 'New File' dialog box. The 'File Name' field contains 'MyFIRSTJavaOpMode' and the file type is set to 'java'. The 'Location' field contains 'org/firstinspires/ftc/teamcode'. The 'Sample' dropdown is set to 'BlankLinearOpMode'. The 'TeleOp' radio button is selected, and the 'Setup Code for Configured Hardware' checkbox is checked.

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”

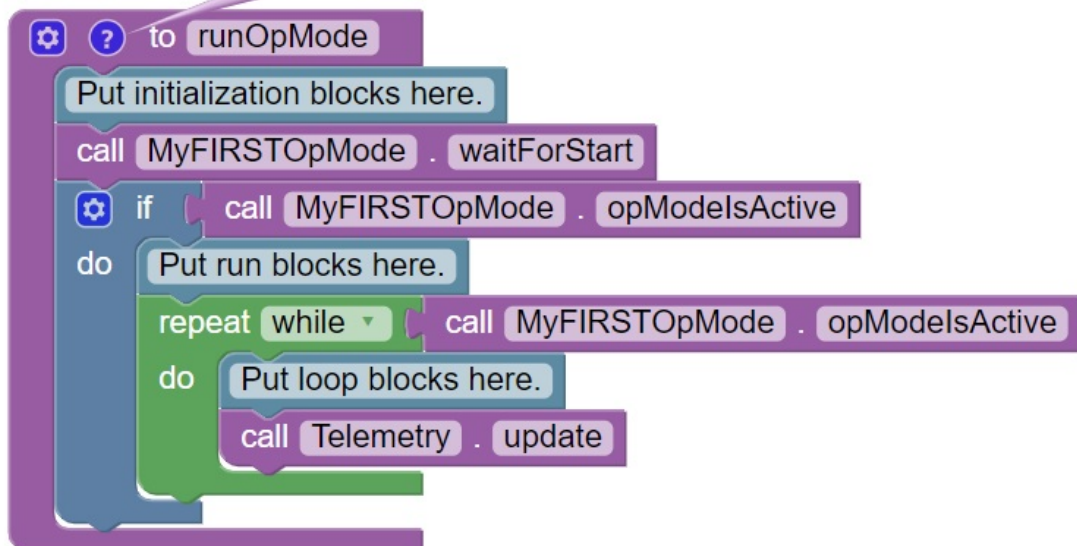
from the list of available op modes for this Robot Controller. For a linear op mode, the Robot Controller will process this list of tasks sequentially. Users can also use control loops (such as a while loop) to have the Robot Controller repeat (or iterate) certain tasks within a linear op mode.

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

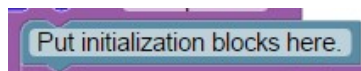
Blocks

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.

This function is executed when this Op Mode is selected from the Driver Station.

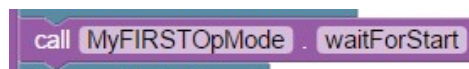


The blue colored block with the words “Put initialization blocks here” is a comment. Comments are placed in an op mode for the benefit of the human user. The robot will ignore any comments in an op mode.



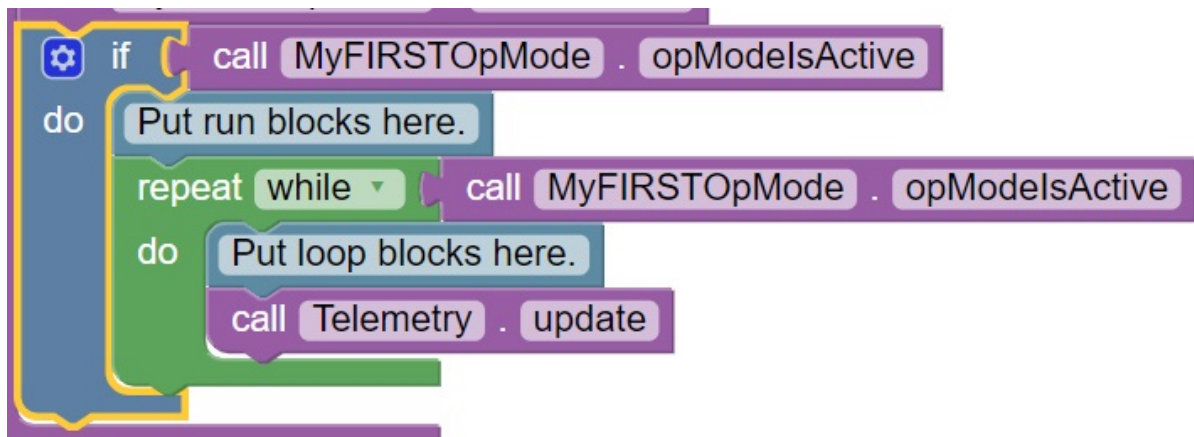
Any programming blocks that are placed after the “Put initialization blocks here” comment (and before the “call MyFIRSTOpMode.waitForStart” block) will be executed when the op mode is first selected by a user at the Driver Station.

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.



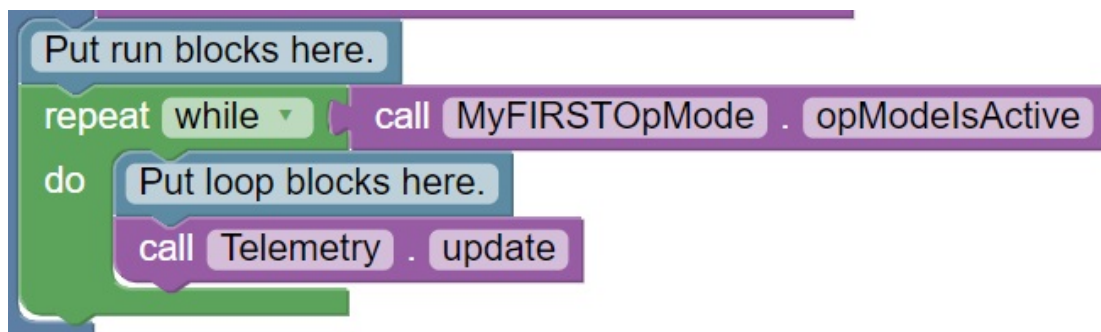
After the “call MyFIRSTOpMode.waitForStart”, there is a conditional “if” block (“if call MyFIRSTOpMode.isActive”) that only gets executed if the op mode is still active (i.e., a stop

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.opModelsActive" will be executed sequentially by the Robot Controller after the Start button has been pressed.

The green block labeled "repeat while call MyFirstOpMode.opModelsActive" 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.opModelsActive" 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.opModelsActive" clause is no longer true and the "repeat while" loop will stop repeating itself.


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.

```
1  @TeleOp
2
3  public class MyFIRSTJavaOpMode extends LinearOpMode {
4      private Gyroscope imu;
5      private DcMotor motorTest;
6      private DigitalChannel digitalTouch;
7      private DistanceSensor sensorColorRange;
8      private Servo servoTest;
9
10
11     @Override
12     public void runOpMode() {
13         imu = hardwareMap.get(Gyroscope.class, "imu");
14         motorTest = hardwareMap.get(DcMotor.class, "motorTest");
15         digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
16         sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
17         servoTest = hardwareMap.get(Servo.class, "servoTest");
18
19         telemetry.addData("Status", "Initialized");
20         telemetry.update();
21         // Wait for the game to start (driver presses PLAY)
22         waitForStart();
23
24         // run until the end of the match (driver presses STOP)
25         while (opModeIsActive()) {
26             telemetry.addData("Status", "Running");
27             telemetry.update();
28         }
29     }
30 }
31 }
```

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.

```
1 @Override  
2 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:

```
1 imu = hardwareMap.get(Gyroscope.class, "imu");  
2 motorTest = hardwareMap.get(DcMotor.class, "motorTest");  
3 digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");  
4 sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");  
5 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.

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

ℹ 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):

```
1     // run until the end of the match (driver presses STOP)
2     while (opModeIsActive()) {
3         telemetry.addData("Status", "Running");
4         telemetry.update();
5
6     }
```

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.

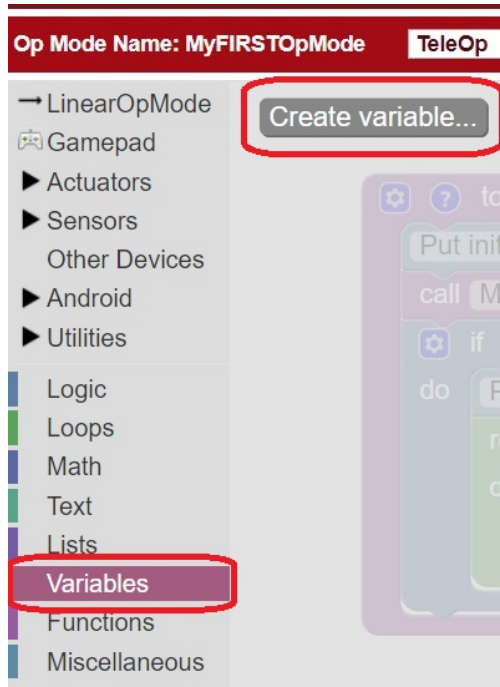
This section will cover how to assign input from a gamepad to control a motor and a servo.

Controlling a Motor

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

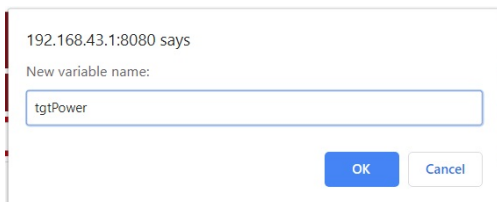
Blocks

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.

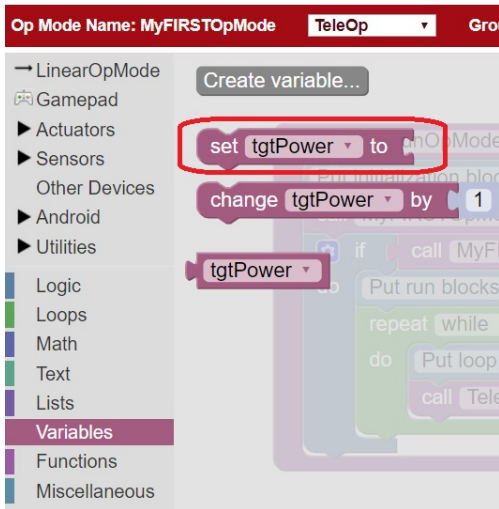


Click on “Create variable...” to create a new variable that will represent the target motor power for our op mode.

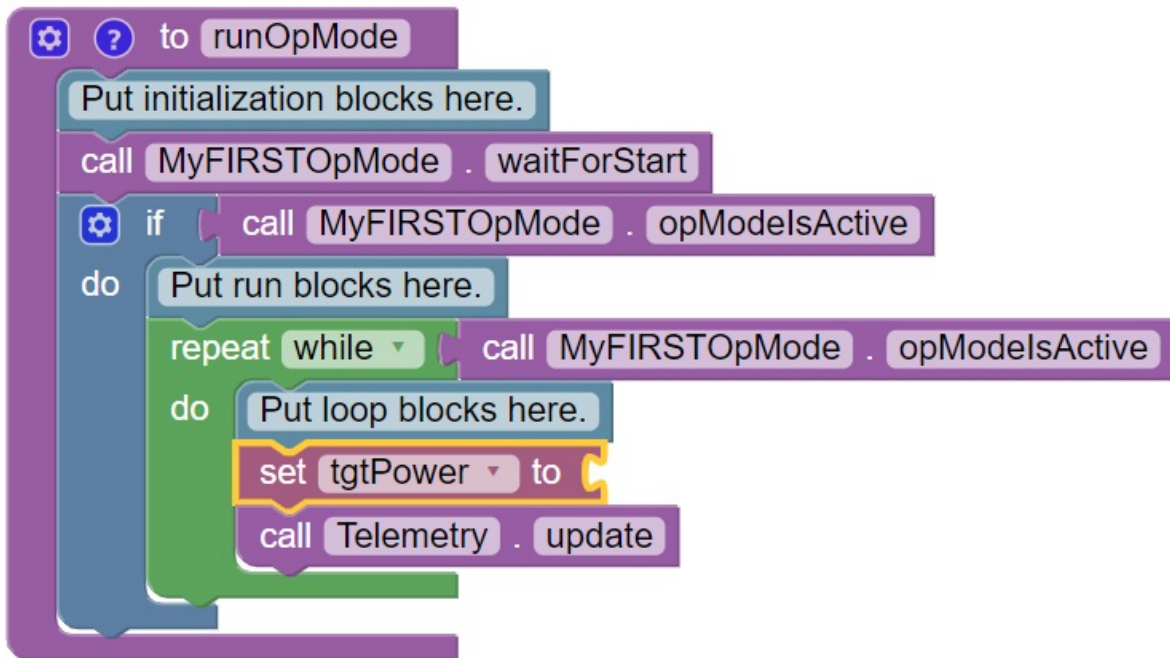
When prompted, type in a name (“tgtPower”) for your new variable.



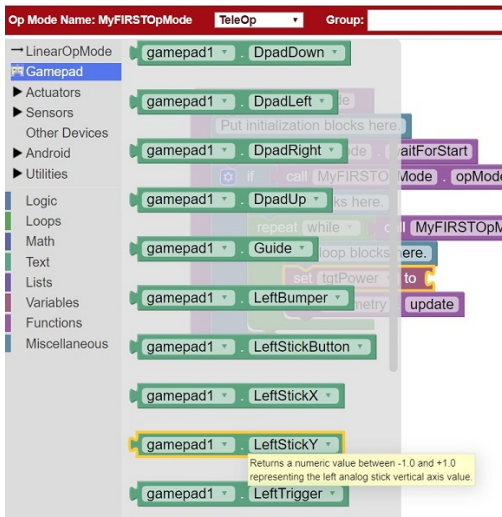
Once you have created your new variable, some additional programming blocks should appear under the “Variables” block category.



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.

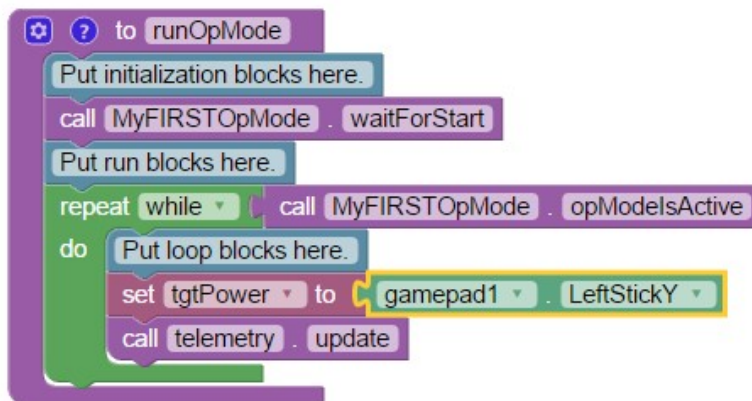


Click on the “Gamepad” category of the programming blocks and select the “gamepad1.LeftStickY” block from the list of available blocks.



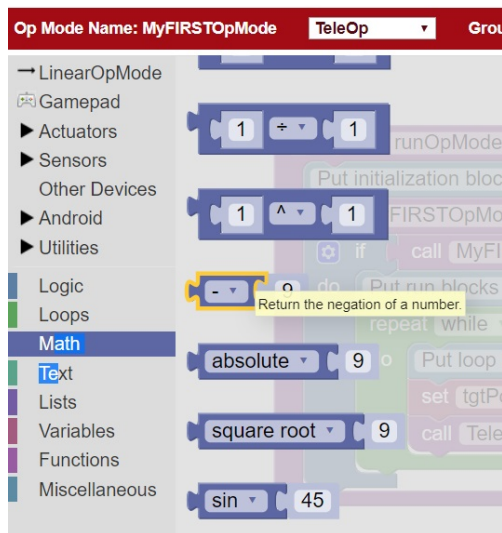
i The control system lets you have up to two gamepads controlling a robot. By selecting “gamepad1” you are telling the op mode to use the control input from the gamepad that is designated as driver #1.

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.

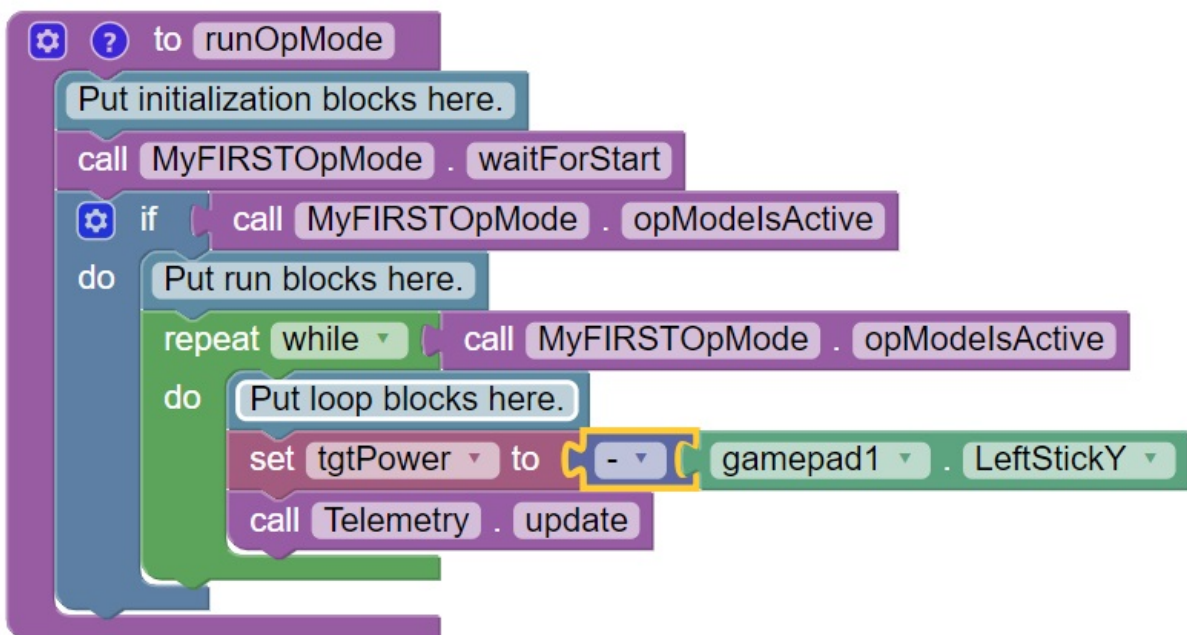


i For the F310 gamepads, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position. In our example, if the left joystick is pushed to the top, the variable tgtPower will have a value of -1.

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

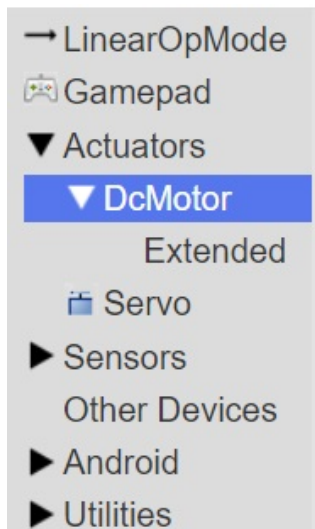


Drag the negative symbol (also known as a “negation operator”) to the left of the “gamepad1.LeftStickY” block. It should click in place after the “set tgtPower to” block and before the “gamepad1.LeftStickY” block.

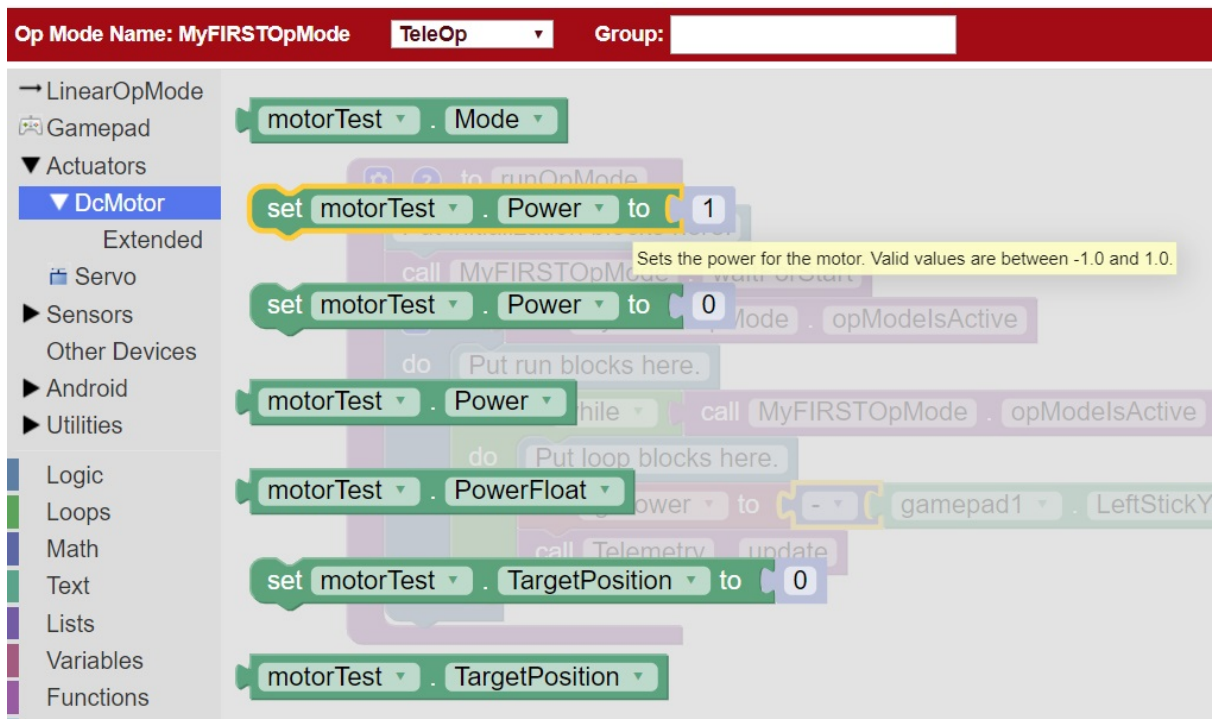


With this change, the variable tgtPower will be set to +1 if the left joystick is in its topmost position and will be set to -1 if the joystick is in its bottommost position.

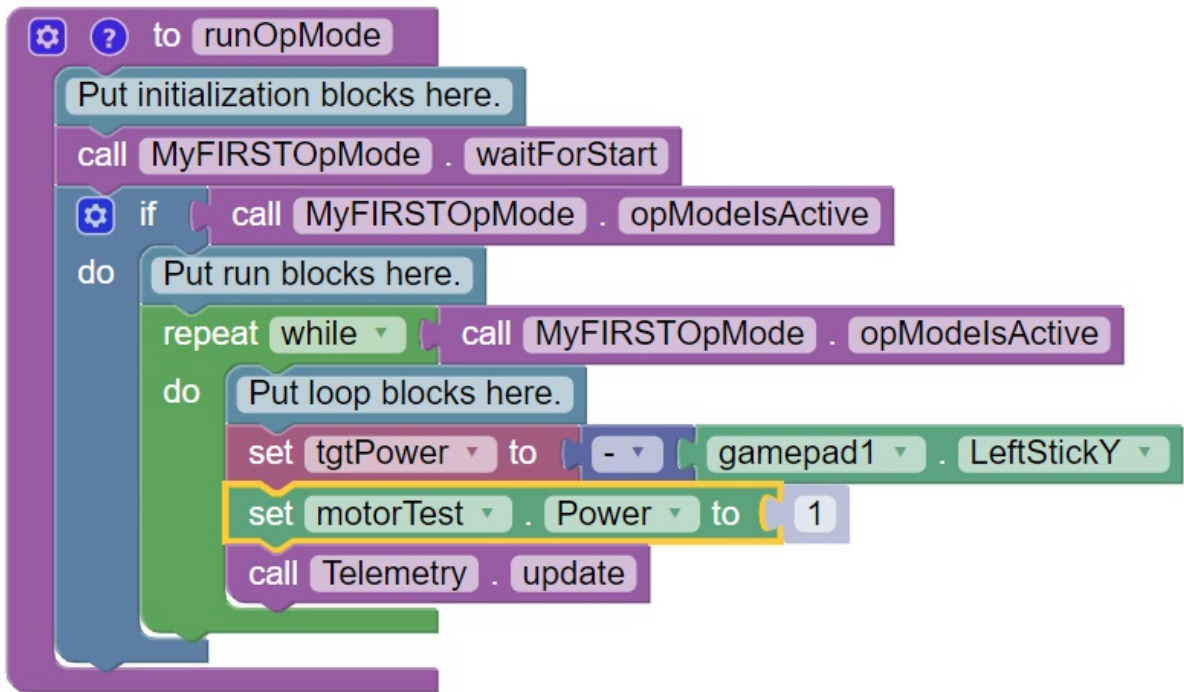
Click on the “Actuators” category of blocks. Then click on the “DcMotor” category of blocks.



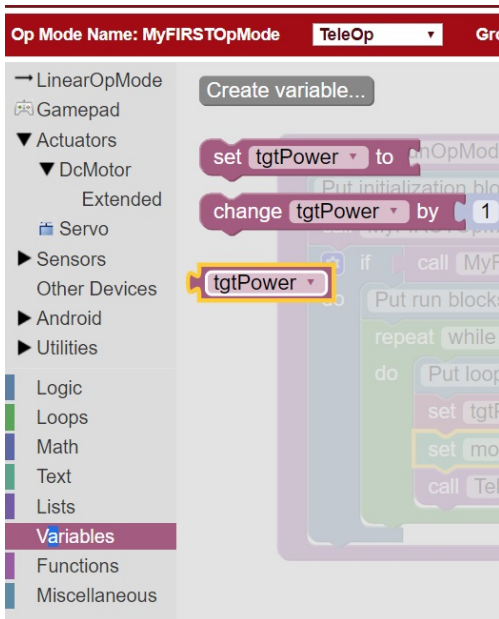
Select the “set motorTest.Power to 1” programming block.



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.

```
to runOpMode
  Put initialization blocks here.
  call MyFIRSTOpMode . waitForStart
  if call MyFIRSTOpMode . opModelsActive
  do
    Put run blocks here.
    repeat while call MyFIRSTOpMode . opModelsActive
    do
      Put loop blocks here.
      set tgtPower to - gamepad1 . LeftStickY
      set motorTest . Power to tgtPower
      call Telemetry . update
```

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

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:

```
1 // run until the end of the match (driver presses STOP)
2 double tgtPower = 0;
3 while (opModeIsActive()) {
4     tgtPower = -this.gamepad1.left_stick_y;
5     motorTest.setPower(tgtPower);
6     telemetry.addData("Target Power", tgtPower);
7     telemetry.addData("Motor Power", motorTest.getPower());
8     telemetry.addData("Status", "Running");
9     telemetry.update();
10
11 }
```

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.

```
1  tgtPower = -this.gamepad1.left_stick_y;
2  motorTest.setPower(tgtPower);
3  telemetry.addData("Target Power", tgtPower);
4  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

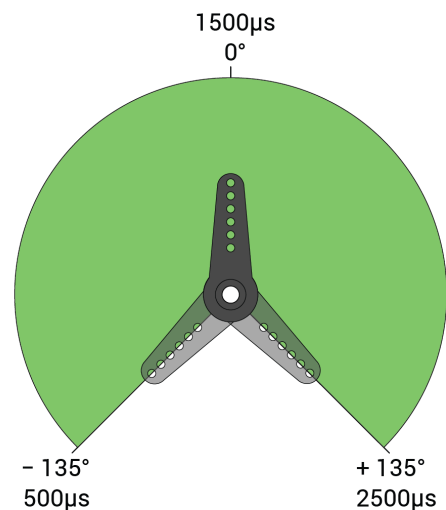
Let's modify your op mode to add the logic required to control a servo motor. For this example, you will use the buttons on the Logitech F310 gamepad to control the position of a REV Robotics Smart Robot Servo ([REV-41-1097](#)).

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.

i 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.



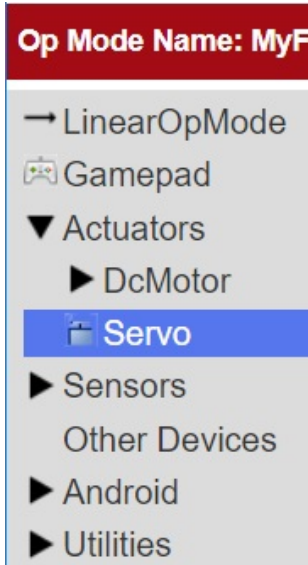
In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway or **neutral position**. If we use the above image of the pulse width range for the servo, this is representative of 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”.



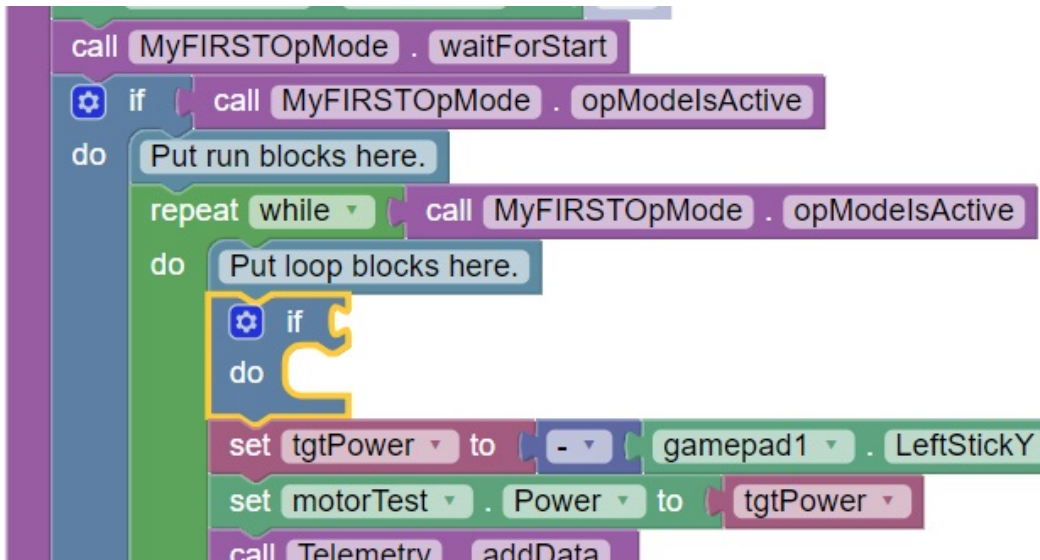
Select the “set servoTest.Position to” block from the list of available Servo blocks.



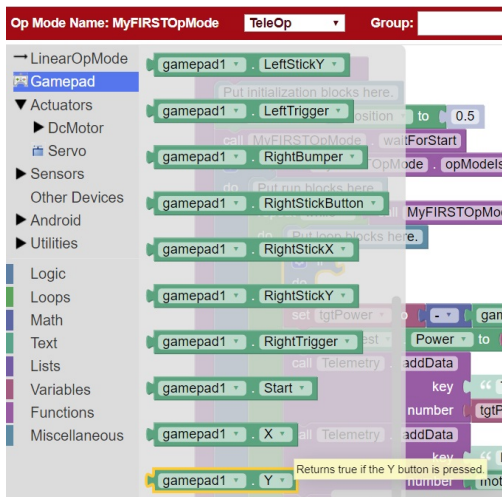
Drag the “set servoTest.Position to” block to the spot just under the comment block that reads “Put initialization blocks here.” The block should click into place. Change the number block to read “0.5” instead of “0”



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.

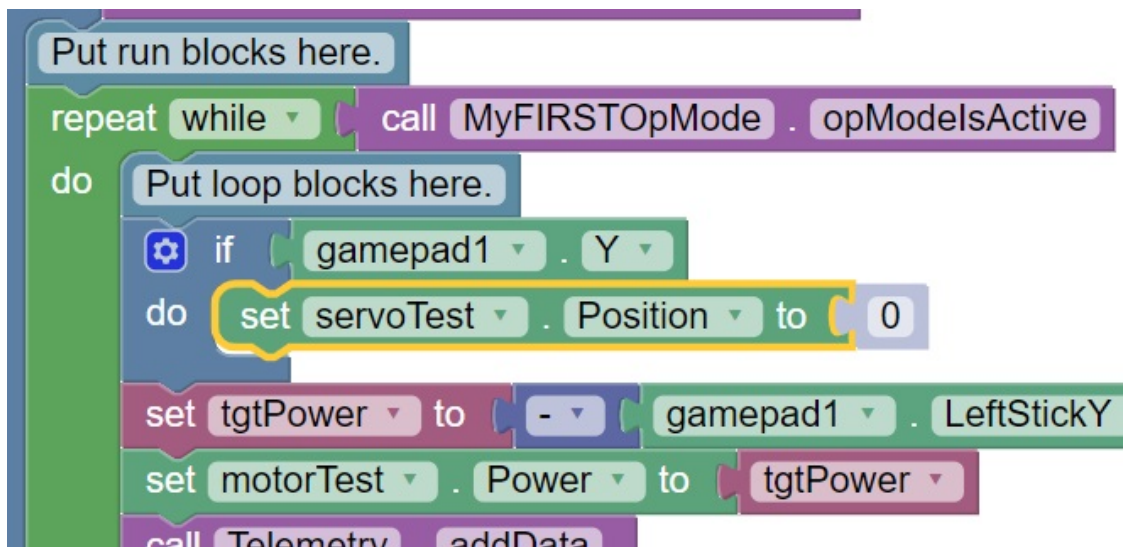


Drag the “gamepad1.Y” block to the right side of the “if do” block. The block should click into place.

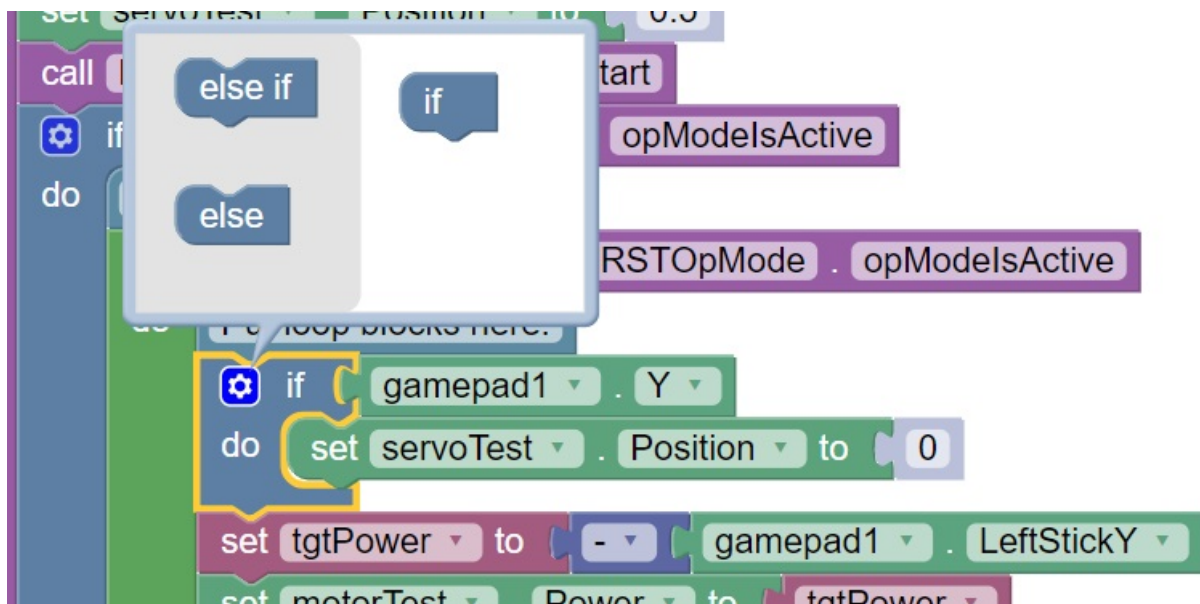


 The “if do” block will use the state of the gamepad1.Y value its test condition. If the “Y” button is pressed, the statements within the “do” portion of the block will be executed.

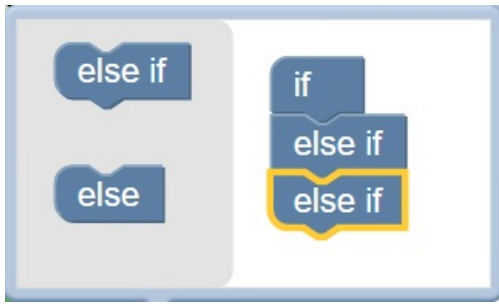
Select the "set servoTest.Position to" block from the list of available Servo blocks. Drag the "set servoTest.Position to" block so that it snaps in place in the do portion of the "if do" block.



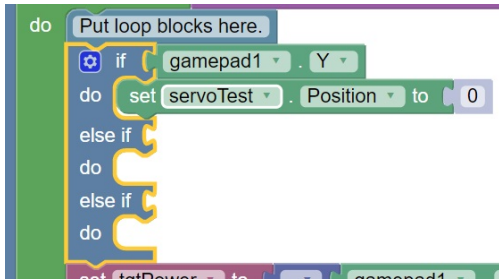
Click on the blue and white Settings icon for the "if do" block. This will display a pop-up menu that lets you modify the "if do" block.



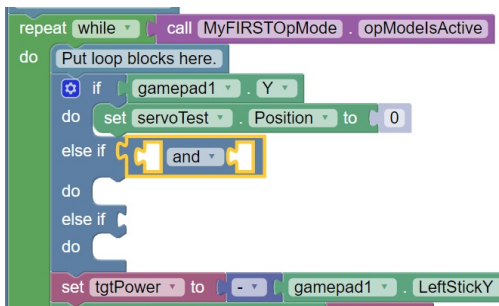
Drag an "else if" block from the left side of the pop-up menu and snap it into place under the "if" block. Drag a second "else if" block from the left side and snap it into place on the right side under the first "else if" block.



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.



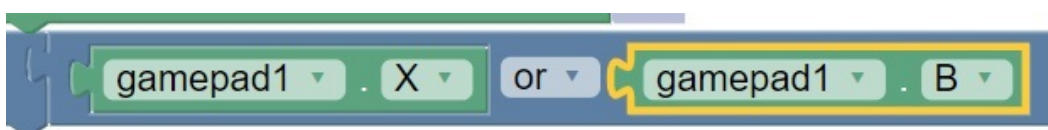
Click on the “Logic” category and select the logical “and” block. Drag the “and” block so it clicks in place as the test condition for the first “else if” block.



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. Then select the “gamepad1.B” and drag it into the second test condition of the “or” block.



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.

```
repeat while (call MyFIRSTOpMode . opModelsActive)
do
  Put loop blocks here.
  if (gamepad1 . Y)
  do
    set servoTest . Position to 0
  else if (gamepad1 . X or gamepad1 . B)
  do
    set servoTest . Position to 0.5
  else if
  do
```

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.

```
repeat while (call MyFIRSTOpMode . opModelsActive)
do
  Put loop blocks here.
  if (gamepad1 . Y)
  do
    set servoTest . Position to 0
  else if (gamepad1 . X or gamepad1 . B)
  do
    set servoTest . Position to 0.5
  else if (gamepad1 . A)
  do
    set servoTest . Position to 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.


```
set tgtPower to gamepad1 . LeftStickY
set motorTest . Power to tgtPower
call Telemetry . addData
  key "Target Power"
  number tgtPower
call Telemetry . addData
  key "Motor Power"
  number motorTest . Power
call Telemetry . addData
  key "Servo Position"
  number servoTest . Position
call Telemetry . update
```

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

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.

```
1  if(gamepad1.y) {
2      // move to -135 degrees.
3      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.

```
1  if(gamepad1.y) {
2      // move to -135 degrees.
3      servoTest.setPosition(0);
4  } else if (gamepad1.x || gamepad1.b) {
5      // move to 0 degrees.
6      servoTest.setPosition(0.5);
7  } else if (gamepad1.a) {
8      // move to + 135 degrees.
9      servoTest.setPosition(1);
10 }
```

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

```
1  // run until the end of the match (driver presses STOP)
2  double tgtPower = 0;
3  while (opModeIsActive()) {
4      tgtPower = -this.gamepad1.left_stick_y;
5      motorTest.setPower(tgtPower);
6      // check to see if we need to move the servo.
7      if(gamepad1.y) {
8          // move to -135 degrees.
9          servoTest.setPosition(0);
10     } else if (gamepad1.x || gamepad1.b) {
```


```
11     // move to 0 degrees.
12     servoTest.setPosition(0.5);
13 } else if (gamepad1.a) {
14     // move to + 135 degrees.
15     servoTest.setPosition(1);
16 }
17 telemetry.addData("Servo Position", servoTest.getPosition());
18 telemetry.addData("Target Power", tgtPower);
19 telemetry.addData("Motor Power", motorTest.getPower());
20 telemetry.addData("Status", "Running");
21 telemetry.update();
22
23 }
```

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.

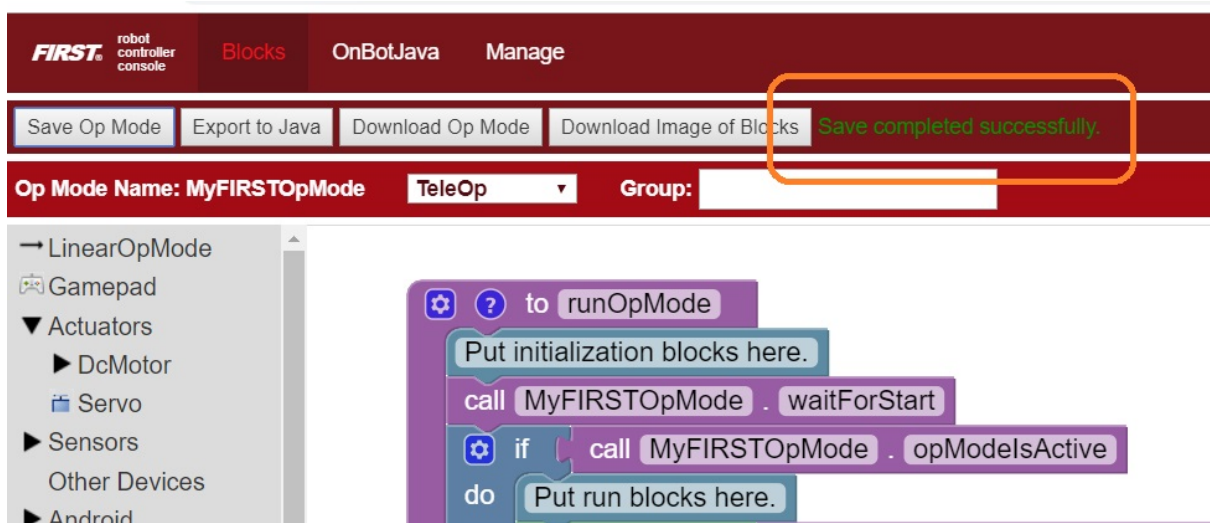
Saving or Building and Op Mode

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.



The screenshot shows the FIRST robot controller console interface. At the top, there is a dark red header with the FIRST logo and the text "robot controller console". Below the header, there are several buttons: "Save Op Mode", "Export to Java", "Download Op Mode", and "Download Image of Blocks". To the right of these buttons, the text "Save completed successfully" is displayed in green. Below the buttons, there is a section for "Op Mode Name: MyFIRSTOpMode", a dropdown menu for "TeleOp", and a "Group:" field. On the left side, there is a tree view of the op mode structure, including "LinearOpMode", "Gamepad", "Actuators" (with sub-items "DcMotor" and "Servo"), "Sensors", "Other Devices", and "Android". On the right side, there is a code editor showing a "to runOpMode" block with the following code:

```
to runOpMode
  Put initialization blocks here.
  call MyFIRSTOpMode . waitForStart
  if call MyFIRSTOpMode . opModelsActive
  do Put run blocks here.
```

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.

```
21 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
22 import com.qualcomm.robotcore.hardware.Gyroscope;
23 import com.qualcomm.robotcore.hardware.DigitalChannel;
24 import com.qualcomm.robotcore.hardware.DistanceSensor;
25 import com.qualcomm.robotcore.hardware.Servo;
26
Build started at Wed Sep 06 2017 08:21:12 GMT-0400 (Eastern Daylight Time)

Build finished in 1.1 seconds
Build succeeded!
```

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

Troubleshooting Common Issues

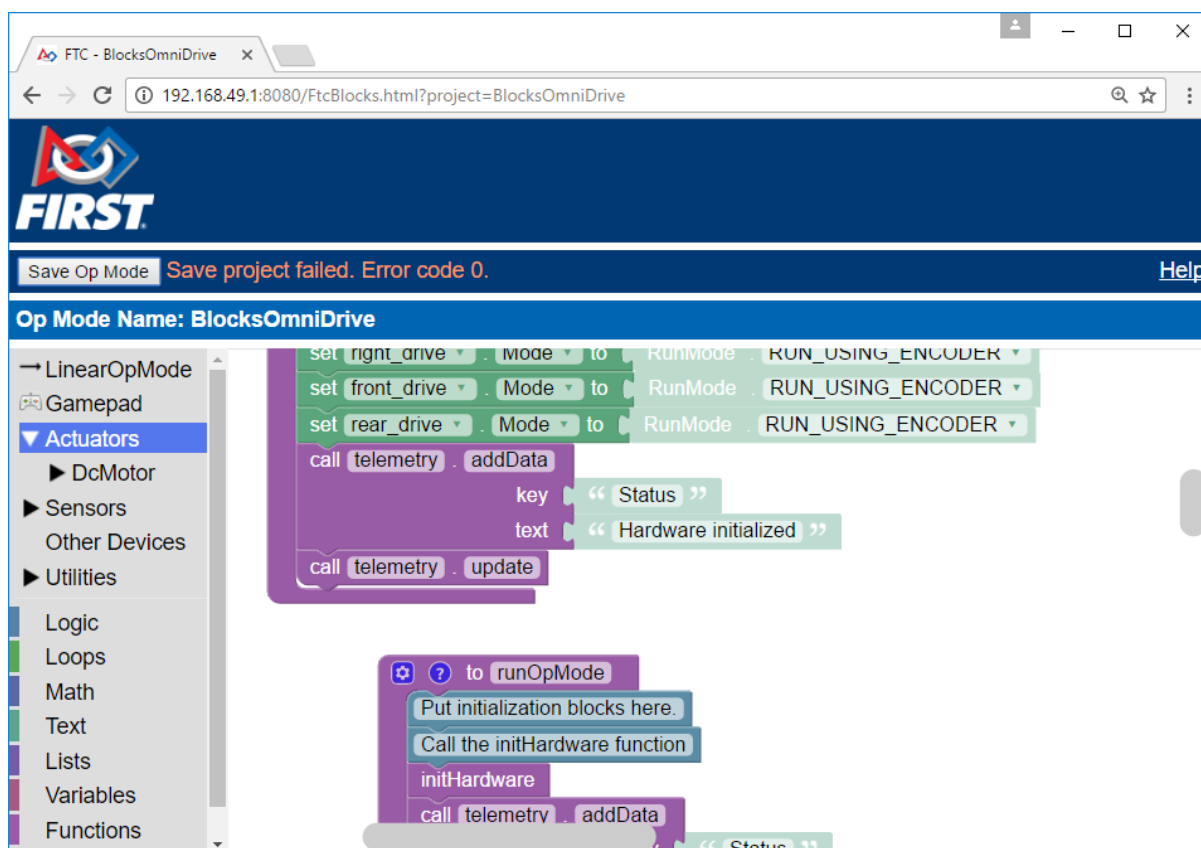
One of the key aspects of troubleshooting is understanding the most common issues that occur in a system. Because of the structural differences of the languages; the common issues in Blocks and OnBot Java differ.

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.

"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 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.

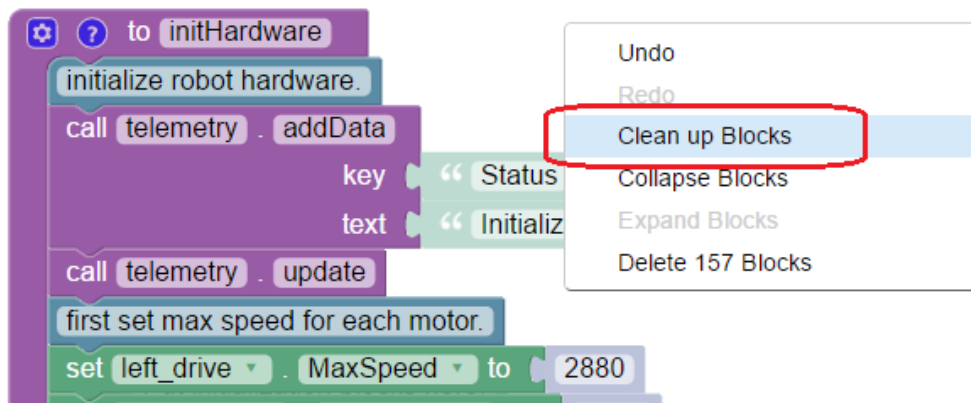
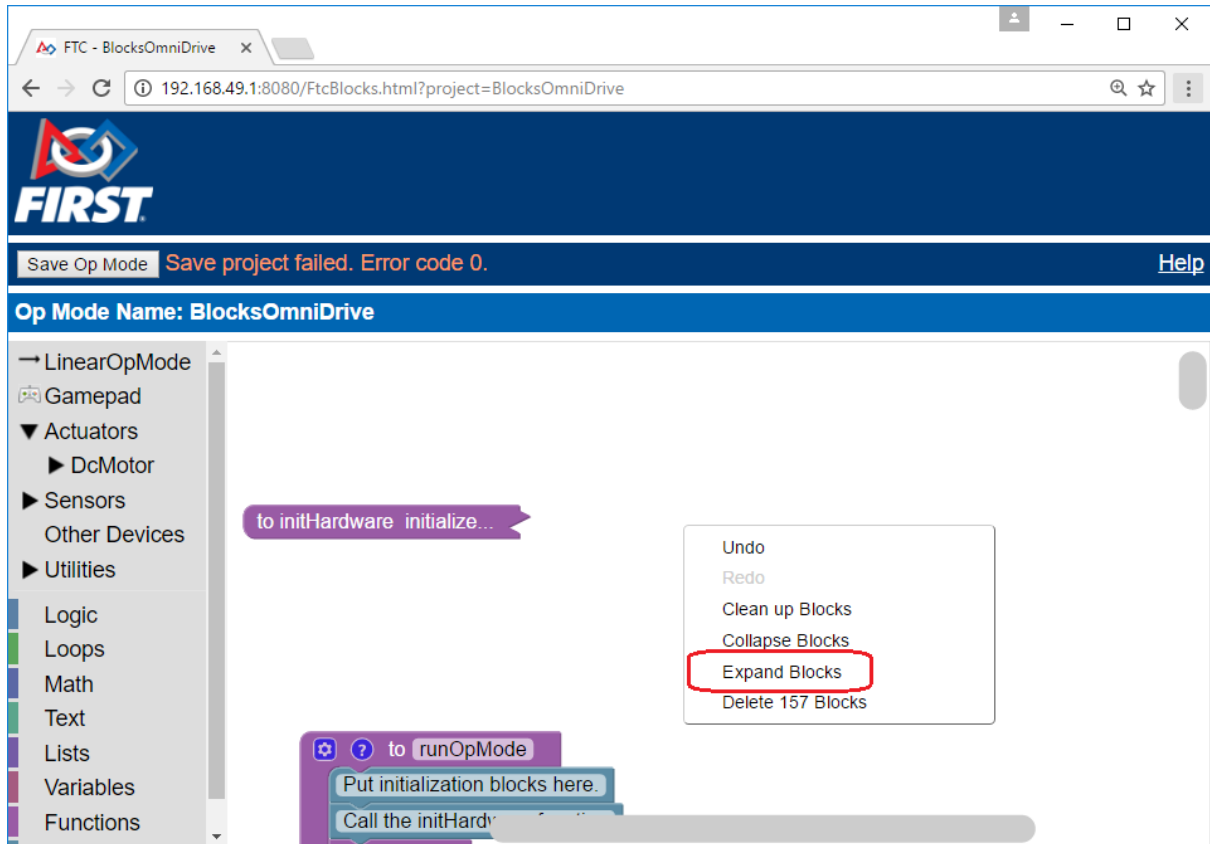
1. Make sure that your laptop is connected to the blocks programming mode Wi-Fi network
2. Press the "Save Op Mode" button again to re-attempt the save operation.

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.



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.

```
58 private Servo servoTest = hardwareMap.get(Servo.class, "servoTest");
59 sensorColorRange = hardwareMap.get(DistanceSensor.class, "s
60 servoTest = hardwareMap.get(Servo.class, "servoTest");
61
62 x telemetry.addData("Status", );
63 telemetry.update();
64 // Wait for the game to start (driver presses PLAY)
65 waitForStart();
66
```

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;
```

Using Encoders

Basic Encoder Concepts


Each motor designed by REV has an encoder built into it that keeps track of its rotation. To use it, you must have a **4-pin JST PH** cable connecting the motor to the Control Hub (**REV-31-1595**) or Expansion Hub (**REV-31-1153**), next to the **2-pin JST VH** cable used to provide power to the motor.

Encoder values are measured in “ticks.” Different motors have different numbers of ticks per rotation of the output shaft based on the gear ratio of the motor. When the Control Hub is turned on, all of its encoder ports are at 0 ticks. As a motor moves forward, its encoder value increases. As a motor moves backwards, its encoder value decreases.

For more information see the [section on encoders](#).

Choosing a Motor Mode

Your programs can always access the encoder values directly, but you can also direct the Control Hub to use the encoder values to maintain a motor’s speed, or maintain a particular position. You do this by changing the motor’s mode.


 It is recommended to use the latest Control Hub and Expansion Hub firmware before using `RUN_USING_ENCODER` mode or `RUN_TO_POSITION` mode.

STOP_AND_RESET_ENCODER Mode

Place a motor in this mode when you want to set its encoder position back to 0. The motor will stop. To start it again, you need to place the motor into one of the other three modes. It is recommended to place each motor you will be using encoders with into this mode at the start of each program, so that you know what position the motor is starting out in.


RUN_WITHOUT_ENCODER Mode

Use this mode when you don’t want the Control Hub to attempt to use the encoders to maintain a constant speed. You can still access the encoder values, but your actual motor speed will vary more based on external factors such as battery life and friction. In this mode, you provide a power level in the -1 to 1 range, where -1 is full speed backwards, 0 is stopped, and 1 is full speed forwards. Reducing the power reduces both torque and speed.

 This mode is a good choice for drivetrain motors driven by joysticks on the gamepad.


RUN_USING_ENCODER Mode

In this mode, the Control Hub will use the encoder to take an active role in managing the motor's speed. Rather than directly applying a percentage of the available power, RUN_USING_ENCODER mode targets a specific velocity (speed). This allows the motor to account for friction, battery voltage, and other factors.

 This mode is a good choice for operations, like a flywheel, that require a specific speed and can use buttons on a gamepad for control.

RUN_TO_POSITION Mode

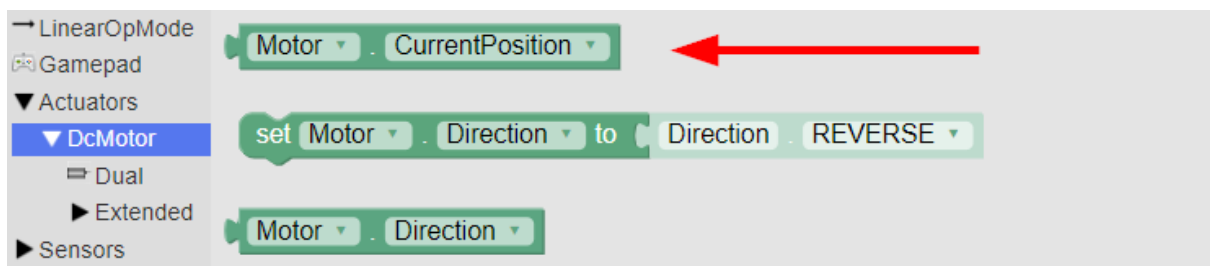
In this mode, the Control Hub will target a specific position, rather than a specific velocity. You still set a velocity, but it is only used as the maximum velocity. The motor will continue to hold its position even after it has reached its target.

 This mode is a good choice for operations, like an arm, that require a specific position and can use buttons on a gamepad for control.

Reading the Encoder Value

Blocks

In Blocks, you access the current encoder value by using the DcMotor CurrentPosition block.



Java

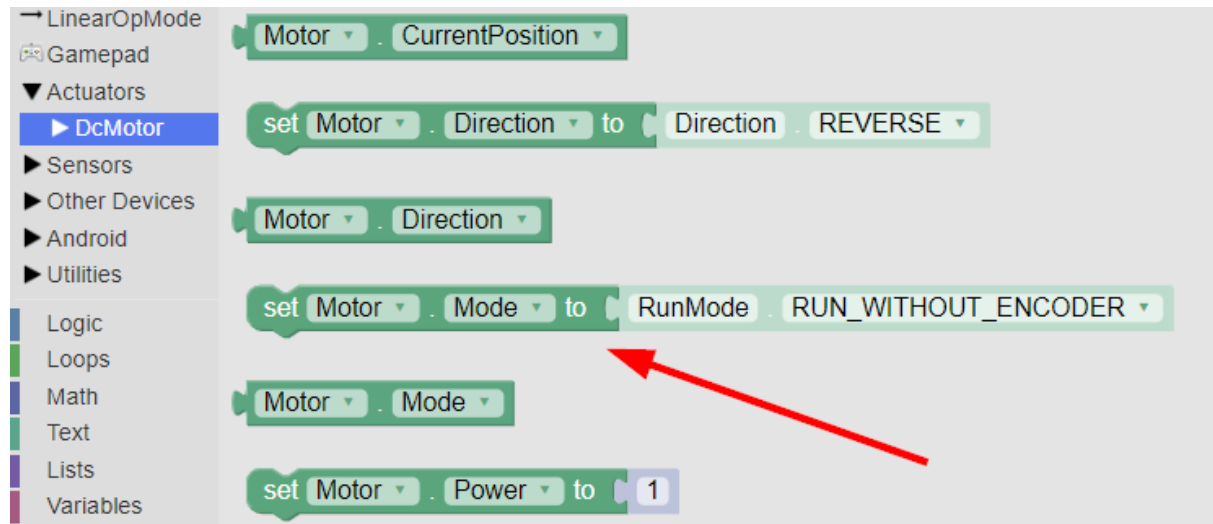
In Java, you access the current encoder value by calling `getCurrentPosition()` on a `DcMotor` or `DcMotorEx` object. This sample program prints the encoder value for a motor configured with the name "Motor" to telemetry:

```
1 package org.firstinspires.ftc.teamcode;
2 // import lines were omitted. OnBotJava will add them automatically.
3
4 @TeleOp
5 public class JavaEncoderTest extends LinearOpMode {
6     DcMotorEx motor;
7
8     @Override
9     public void runOpMode() {
10         motor = hardwareMap.get(DcMotorEx.class, "Motor");
11         waitForStart();
12         while (opModeIsActive()) {
13             telemetry.addData("Encoder value", motor.getCurrentPosition());
14             telemetry.update();
15         }
16     }
17 }
```

Setting the Motor Mode

Blocks

In Blocks, you set the motor's mode with this block. You can select different modes from its dropdown menu.



Java

Here is a snippet of code that demonstrates how to do the same thing in Java. You can skip the first line if you already have retrieved the motor object from hardwareMap. Change RUN_WITHOUT_ENCODER to the desired motor mode (STOP_AND_RESET_ENCODER, RUN_WITHOUT_ENCODER, RUN_USING_ENCODER, or RUN_TO_POSITION).

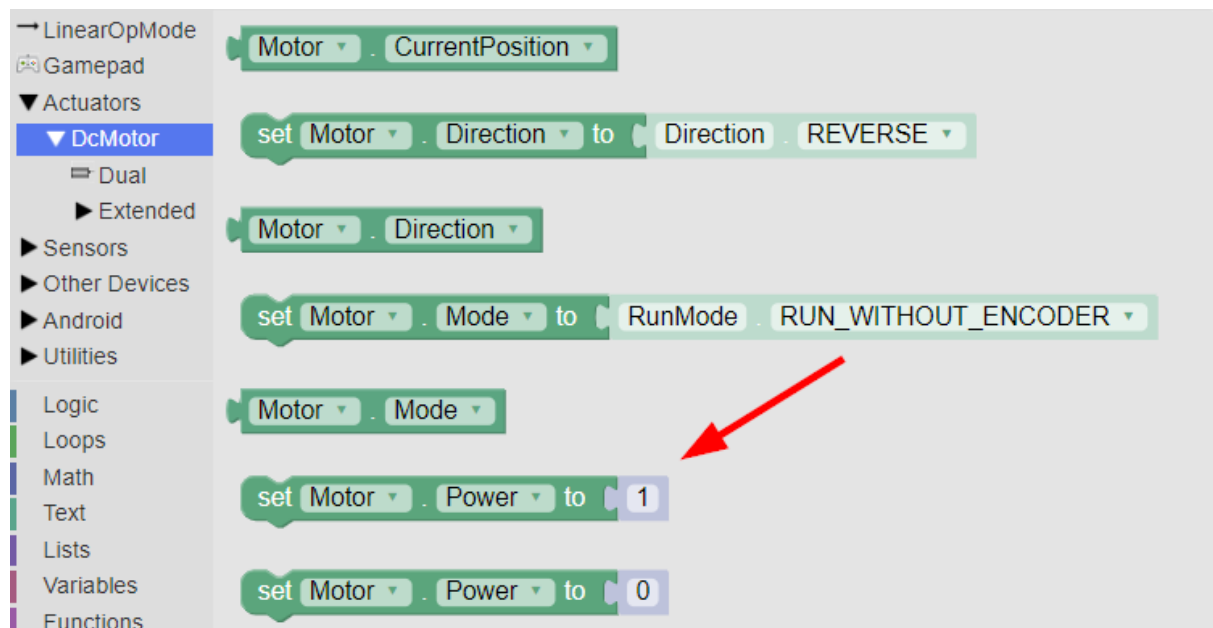
```
1 DcMotorEx motor = hardwareMap.get(DcMotorEx.class, "Motor");
2 motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
```

Using RUN_WITHOUT_ENCODER

The RUN_WITHOUT_ENCODER motor mode is very straightforward, you simply set a power in the range of -1.0 to 1.0. However, if you try to set a velocity (which will be covered later on), the motor will automatically be switched into RUN_USING_ENCODER mode.

Blocks

The power level is set in Blocks mode using this block:



The screenshot shows the Blocks editor interface. On the left is a category list with 'DcMotor' expanded. The main workspace contains the following blocks from top to bottom:

- Motor . CurrentPosition
- set Motor . Direction to Direction . REVERSE
- Motor . Direction
- set Motor . Mode to RunMode . RUN_WITHOUT_ENCODER
- Motor . Mode
- set Motor . Power to 1 (highlighted with a red arrow)
- set Motor . Power to 0

Java

The power level is set in Java by calling `setPower()` on a `DcMotor` or `DcMotorEx` object, as is shown in this snippet. You can skip the first two lines if you already have retrieved the motor object from `hardwareMap` and set the mode to `RUN_WITHOUT_ENCODER`.

```
1 DcMotorEx motor = hardwareMap.get(DcMotorEx.class, "Motor");
2 motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
3 // This will run the motor forward at half-power
4 double motorPower = 0.5;
5 motor.setPower(motorPower);
```

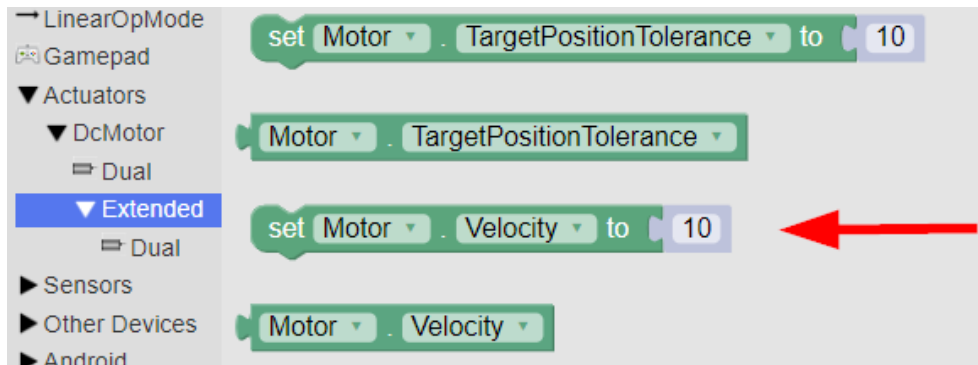
Using RUN_USING_ENCODER

In `RUN_USING_ENCODER` mode, you should set a velocity (measured in ticks per second), rather than a power level. You can still provide a power level in `RUN_USING_ENCODER` mode, but this is not recommended,

as it will limit your target speed significantly. Setting a velocity from RUN_WITHOUT_ENCODER mode will automatically switch the motor to RUN_USING_ENCODER mode. You should pick a velocity that the motor will be capable of reaching even with a full load and a low battery.

Blocks

Providing a velocity is an extended motor feature, which means that the block for it is located under DcMotor > Extended. You can see it here:



Java

The velocity is set in Java by calling `setVelocity()` on a `DcMotorEx` object, as is shown in this snippet. You can skip the first two lines if you have already retrieved the motor object as a `DcMotorEx` from `hardwareMap` and set the mode to `RUN_USING_ENCODER`.

```
1 DcMotorEx motor = hardwareMap.get(DcMotorEx.class, "Motor");
2 motor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
3 // This will turn the motor at 200 ticks per second
4 double motorVelocity = 200;
5 motor.setVelocity(motorVelocity);
```

Using RUN_TO_POSITION

To use `RUN_TO_POSITION` mode, you need to do the following things in this order:

1. Set a target position (in ticks)

2. Switch to RUN_TO_POSITION mode
3. Set the maximum velocity

You should reset the encoders (switch to STOP_AND_RESET_ENCODER mode) during initialization when you use RUN_TO_POSITION mode. If you are using it with a mechanism such as a lift, you have to be careful to make sure that you always have the motor in the same physical location when you reset the encoders, or else your target position won't mean the same thing between runs.

The motor will continue to hold its position even after it has reached its target, unless you set the velocity or power to zero, or switch to a different motor mode.

The following examples assume that the motor used is a Core Hex Motor. If it is a motor that has a more precise encoder, such as an HD Hex Motor, higher velocity and target position values will be more appropriate.

Blocks

Here is a complete Blocks program that uses RUN_TO_POSITION.

```
to runOpMode
  Reset the encoder during the init phase
  set Motor . Mode to RunMode . STOP_AND_RESET_ENCODER
  call RunToPositionBlocks . waitForStart
  if call RunToPositionBlocks . opModelsActive
  do
    Set the motor's target position to 300
    set Motor . TargetPosition to 300
    Switch to RUN_TO_POSITION mode
    set Motor . Mode to RunMode . RUN_TO_POSITION
    Start the motor moving by setting the max velocity
    to 200 ticks per second
    set Motor . Velocity to 200
    repeat while call RunToPositionBlocks . opModelsActive
    do
      Show the motor's status via telemetry
      call Telemetry . addData
      key " velocity "
      number Motor . Velocity
      call Telemetry . addData
      key " position "
      text Motor . CurrentPosition
      call Telemetry . addData
      key " is at target "
      text not call Motor . isBusy
      call Telemetry . update
```

If you want to wait for the motor to reach its target position before continuing in your program, you can use a while loop that checks if the motor is busy (not yet at its target):

```
Loop while the motor is moving to the target
repeat while call Motor . isBusy
do
  Display telemetry while we wait
  call Telemetry . addData
  key " Status "
  text " Waiting for the motor to reach its target position "
  call Telemetry . update
The motor has reached its target position
and the program will continue
```

Java

```
1 package org.firstinspires.ftc.teamcode;
2 // import lines were omitted. OnBotJava will add them automatically.
3
4 @TeleOp
5 public class JavaRunToPositionExample extends LinearOpMode {
6     DcMotorEx motor;
7
8     @Override
9     public void runOpMode() {
10         motor = hardwareMap.get(DcMotorEx.class, "Motor");
11
12         // Reset the encoder during initialization
13         motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
14
15         waitForStart();
16
17         // Set the motor's target position to 300 ticks
18         motor.setTargetPosition(300);
19
20         // Switch to RUN_TO_POSITION mode
21         motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
22
23         // Start the motor moving by setting the max velocity to 200 ticks per second
24         motor.setVelocity(200);
25
26         // While the Op Mode is running, show the motor's status via telemetry
27         while (opModeIsActive()) {
28             telemetry.addData("velocity", motor.getVelocity());
29             telemetry.addData("position", motor.getCurrentPosition());
30             telemetry.addData("is at target", !motor.isBusy());
31             telemetry.update();
32         }
33     }
34 }
```

If you want to wait for the motor to reach its target position before continuing in your program, you can use a while loop that checks if the motor is busy (not yet at its target):

```
1 // Loop while the motor is moving to the target
2 while(motor.isBusy()) {
3     // Let the drive team see that we're waiting on the motor
4     telemetry.addData("Status", "Waiting for the motor to reach its target");
5     telemetry.update();
6 }
7 // The motor has reached its target position, and the program will continue
```


Sensors

Introduction to Sensors

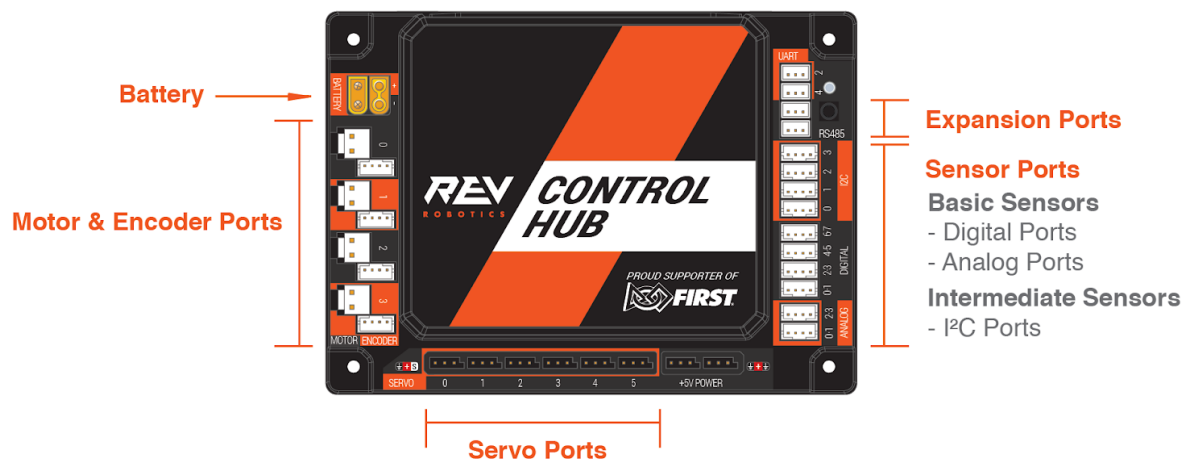
Sensor Basics

Aptly named, sensors are integral to how your robot understands the world around it. For instance, maybe you have noticed your teammates lunch box is green, a color sensor would allow your robot to notice the same thing. Color isn't the only sense your robot has! Robots use sensors to collect various types of information about their environment. The following list is just some of the scenarios where sensors are needed.

Scenarios where a sensor is needed:

- The robot needs to autonomously move to a specific location and stop there.
- The robot needs to move forward at a green signal and stop moving at a red signal.
- The robot has an arm that needs to be prevented from rotating too far or it may damage other parts of the robot.
- The robot needs to stop 1 meter away from an opaque wall.
- The robot needs to be able to tell how many game objects it is currently holding inside it's hopper.


Different Sensor Types and Uses



In The REV Robotics Control System, sensors are classified as **beginner**, **intermediate**, or **advanced**. This division among sensors is based on programming complexity. **Beginner sensors** can typically be coded using an if/else statement. **Intermediate sensors**, like the IMU, require a higher level understanding of programming. **Advanced sensors** require advanced knowledge of programming. Vision sensors and encoders are considered advanced.

Beginner

In the REV Robotics Control System, both **Analog** and **Digital** sensors are considered beginner sensors. Analog and digital sensors log changes in state as changes in Voltage. Digital sensors, like the touch sensor, report voltage changes along a binary. With digital sensors the voltage is typically 0v or 3.3v. Whereas analog sensors are similar to an adjustable resistor and they report a range of voltages between 0v to 3.3v.

 *Some sensors in the REV Control System are capable of running up to 5v. To learn more about sensor voltage visit the pages of the individual sensors!*

The table below gives the basic usage scenarios for analog and digital sensors

Digital	Analog
Gives feedback as either on or off. This type of sensor is ideal for setting limits of a mechanism.	Gives feedback as a proportional voltage range. This type of sensor is ideal for knowing exactly where a mechanism is, like a dial on a radio.

Digital Sensors

- **Touch Sensor**: A sensor with a button. The button press can be used to trigger actions like stopping motors.
- **Magnetic Limit Switch**: A sensor that detects magnetic fields. When there is sufficient field strength of either magnetic pole detected the sensor is triggers and a limit of movement can be established.

Analog Sensors

- **Potentiometer**: The Potentiometer senses the angular position of a shaft.

Intermediate

I2C sensors are considered intermediate because they give feedback through two-way communication with a robot controller. These types of sensors allow for more complex data to communicate to the robot, such as color values of an object.

- **IMU**: The IMU incorporates three sensors: a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis geomagnetic sensor. This sensor can be used to determine orientation and location of the robot.
- **Color Sensor**: A sensor capable of sensing colors and proximity of objects.
- **2m Distance Sensor**: It is typically used to detect the distance from the sensor to other opaque objects.

Advanced

The two types of advanced sensors in the REV Control system are **Vision** sensors and **Encoders**.

Vision

Gives feedback as images to the robot controller. These types of sensors require the use of image processing software, like VuForia, to use to their full potential.

Encoders

An Encoder, in the context of robotics, is a type of digital sensor that converts rotary motion into digital signal. These type of sensors require "decoding" to get this information into a usable form. The Control Hub and Expansion Hub have built in decoding through the "Encoder Ports" under the motor ports.

Integrated Sensors


The REV Robotics Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) integrate a number of feedback sensors. Some of these are user accessible in the latest FTC Android Studio SDK, but others are not yet directly user accessible. These sensors are in some cases also used by the Control Hub and Expansion Hub for internal safety monitoring.

- Battery Voltage Monitoring [**Accessible**]
- Integrated 9-axis IMU [**Accessible**]
 - Bosch BNO055 9-axis absolute orientation sensor
 - Internally connected to I2C port 0 and configured to address 0x28
- Current Monitoring
 - Battery [**Accessible**]
 - I2C Bus [**Accessible**]
 - Digital Power Bus [**Accessible**]
 - Servo Power Bus [Not Accessible]
- Per Motor Channel Current Monitoring [**Accessible**]

Digital

Digital Sensor Basics

Digital sensors give digital feedback to your robot controller along a type of **binary**. In the case of digital sensors the binary is on/off, somewhat similar to a light switch.


 True/False is another well known **binary** in programming. Binary information can be thought of as either/or. A statement is either true or it is false. A limit switch is either on or it is off.

The main difference between a light switch and a digital sensor is that a digital sensor has a **default state**. When an action outside of the default state occurs, like a touch sensor being pressed, an event within the Op Mode is triggered and the robot reacts accordingly.

There are two digital sensors available through REV: Touch ([REV-31-1425](#)) and Magnetic Limit Switch ([REV-31-1462](#))

Configuring Digital Sensors

Each Digital Port on the Control Hub ([REV-31-1595](#)) or Expansion Hub ([REV-31-1153](#)) has two channels for communication. This allows for wiring of two sensors per port when using the [sensor splitter cable](#).

 The touch sensor has certain specifications that affect how you use the sensor splitter cable. Please see the [touch sensor page](#) for more information.

Both the touch sensor and the magnetic limit switch are configured as a “REV Touch Sensor or Magnetic Limit Switch” on the Digital Devices configuration screen. While the magnetic limit switch can be configured on any port, a touch sensor will only work if it is configured on port 1, 3, 5, or 7.

The Touch Sensor is attached to digital port 1, and the Magnetic Limit Switch is attached to digital port 2.

Active Configuration: (unsaved) TestConfiguration

Done Cancel

Port Attached

0 Nothing

NO DEVICE ATTACHED

Device name

1 REV Touch Sensor or Magnetic Limit Switch

Touch

Device name

2 REV Touch Sensor or Magnetic Limit Switch

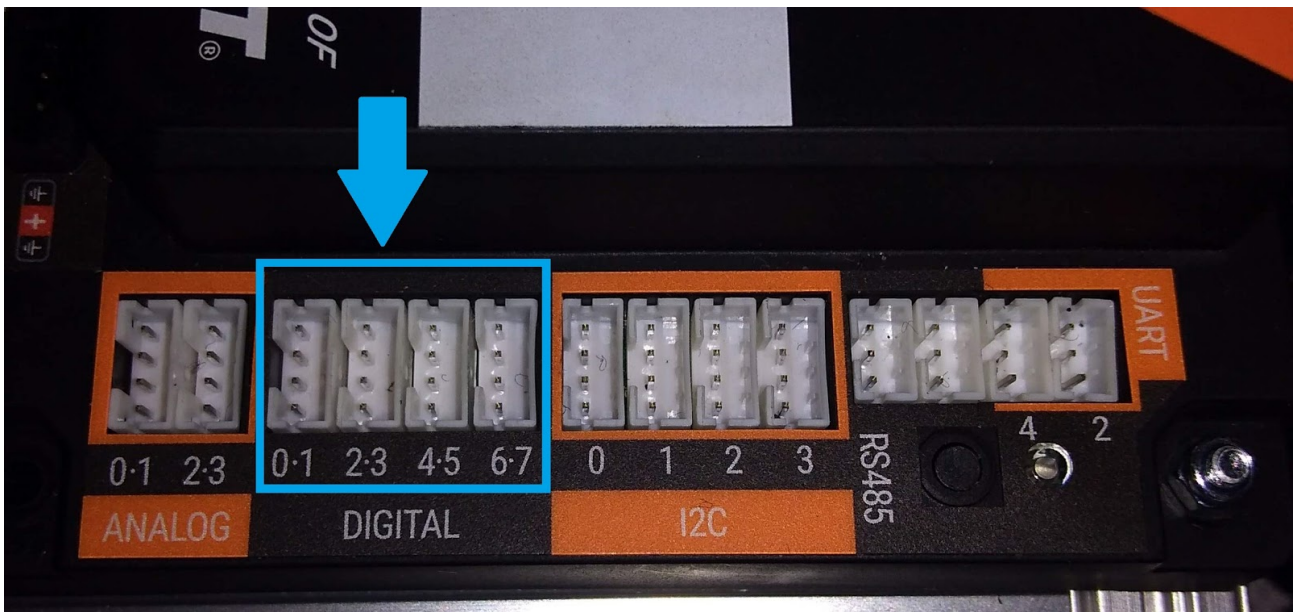
Magnetic

Device name

3 Nothing

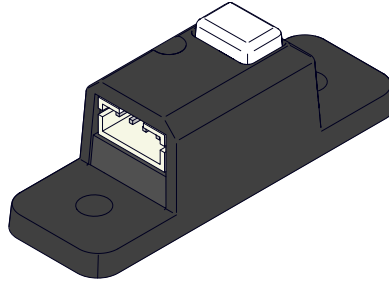
NO DEVICE ATTACHED

The digital ports on the Control hub look like this:



Touch

Touch Sensor Basics



The REV Robotics Touch Sensor ([REV-31-1425](#)) is a digital sensor that can act as a button input or as a basic mechanical limit switch. The touch sensor is similar to a keyboard button, when the button is pressed the touch sensor notifies the Control ([REV-31-1595](#)) or Expansion Hub ([REV-31-1153](#)) and an action in the code is triggered. Sometime this action may stop the motors or reset the encoder angle, depending on use case.

Like all digital sensors, the Touch Sensor acts on a binary. When the button is not pressed, the LED light remains unlit and the value read by the Expansion Hub is 3.3V (high) and when the button is pressed the LED will light and the Expansion Hub will read 0V (Low).

Button	n+1 Voltage	LED State
Not Pressed	3.3V	Off
Pressed	0V	On

How to Use?

Application

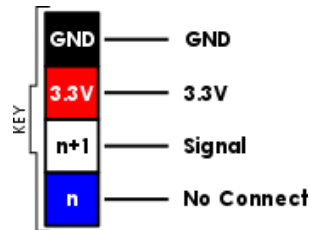
The touch sensor applications are very straight forward and easy to use. Two common examples are a bumper and a limit switch. In the bumper scenario, the sensor is mounted behind a hinged plate. When the plate is bumped the touch sensor is pressed and an action, set in the code, is triggered.

The limit switch is slightly different. The touch sensor can be placed in relation to an arm, or other mechanism, to act as a limit, or stopping point. The limit switch function can help mechanism from over extending/breaking or can act as reset point for encoders so that the point where the arm and the touch sensor meet is always 0.

Installation and Configuration

The sensor can be attached to channel or extrusion using 8mm M3 Hex Cap Screws ([REV-41-1359](#)).

In the image below, is the key for the wired connection between the touch sensor and the robot controller. The touch sensor does not use or pick up a signal from the **n** (blue) wire. This is not a problem if there is only one digital sensor per port. However, if you intend to connect more than one digital sensor to the same port using the sensor splitter cable, make sure that the **n+1** (white) wire portion of the splitter cable is plugged into the touch sensor.




Programming Example

The code blocks below give a basic example of how to use the touch sensor using if/else logic. If the touch sensor is pressed then the motor stops. Otherwise the motor moves.

Blocks

The screenshot shows a programming environment with a sidebar on the left containing a tree view of blocks: LinearOpMode, Gamepad, Actuators (CRServo, DcMotor, Servo), Sensors (IMU-BNO055, IMU-BNO055.Parameters, DistanceSensor, REV Color/Range Sensor, TouchSensor, VoltageSensor), Other Devices, Android, Utilities, Logic, Loops, Math, and Text. The main workspace displays a code block for 'runOpMode' with the following logic:

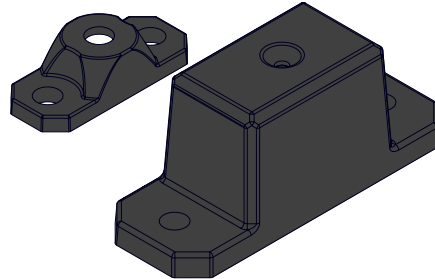
```
to runOpMode
  Setup that runs after the INIT button
  set arm . Mode to RunMode . RUN_WITHOUT_ENCODER
  call TestSensor . waitForStart
  Put run blocks here.
  repeat while call TestSensor . opModelsActive
  do
    if Touch . IsPressed
    do
      If Touch sensor is pressed stop motor
      set arm . Power to 0
    else
      set arm . Power to 0.3
    Put loop blocks here.
    call Telemetry . addData
      key " Arm Motor Power "
      number arm . Power
    call Telemetry . update
```

 The code assumes the sensor has been named "Touch" and the motor has been named "Motor" in configuration.

```
1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4 import com.qualcomm.robotcore.hardware.TouchSensor;
5 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
6 import com.qualcomm.robotcore.hardware.DcMotor;
7
8 @TeleOp
9 public class TouchTest extends LinearOpMode {
10     // Define variables for our touch sensor and motor
11     TouchSensor touch;
12     DcMotor motor;
13
14     @Override
15     public void runOpMode() {
16         // Get the touch sensor and motor from hardwareMap
17         touch = hardwareMap.get(TouchSensor.class, "Touch");
18         motor = hardwareMap.get(DcMotor.class, "Motor");
19
20         // Wait for the play button to be pressed
21         waitForStart();
22
23         // Loop while the Op Mode is running
24         while (opModeIsActive()) {
25             // If the touch sensor is pressed, stop the motor
26             if (touch.isPressed()) {
27                 motor.setPower(0);
28             } else { // Otherwise, run the motor
29                 motor.setPower(0.3);
30             }
31         }
32     }
33 }
34
```

Magnetic Limit Switch

Magnetic Limit Switch Basics



The REV Robotics Magnetic Limit Switch ([REV-31-1462](#)) is a three-sided digital **hall effect** switch. The three internal hall effect elements (one on top, two on the sides) are connected in parallel so if any one of them is triggered the sensor will report as triggered.

Hall effect sensors detect the presence of a magnetic fields. The REV Magnetic Limit Switch is an omnipolar momentary switch; it will trigger when there is sufficient field strength of either magnetic pole detected.

i Nearly all motorized mechanisms (such as arms and elevators) in robotics should be given some form of "limit switch" to prevent them from damaging themselves at the end of their range of motions.

When the sensor gets close to one of the poles the signal voltage is zero, which means the sensor is "pressed". When it reports 3.3v its is outside of the magnetic field and is "not pressed."

Magnetic Field	Output Voltage		LED State
	n	n+1	
North Pole	0V	0V	On
South Pole	0V	0V	On
Insufficient Magnetic Field	V _{IN}	V _{IN}	Off

Product Specs

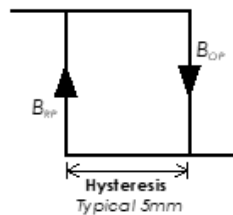
- **Sensor Type:** Digital, Active-low
- **Voltage Range:** 3.3V - 5.0V
- **Signal:** n & n+1

- **Magnetic Polarity:** Omnipolar (both north & south)
 - **Typical Trigger Distance***
 - **Top:** 10mm
 - **Side:** 5mm
 - **Typical Hysteresis:** 5mm
 - **Typical Included Magnet Strength:** 4300G (0.43T)
-

How to Use?

Application

When designing a system using the REV Magnetic Limit Switch it is important to consider in the impact of **hysteresis**. When the magnetic field approaches the Magnetic Limit Switch, after the field strength increases enough that it crosses the rising trigger point (B_{OP}) the sensor triggers. As the magnet is then moved away from the sensor, the magnetic field strength falls but the sensor remains in the triggered state until the field falls below the falling trigger level (B_{RP}). The difference between these two points is the **hysteresis**.

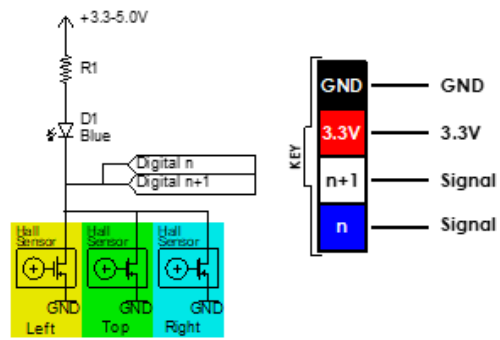


The strength of the magnetic field determines the maximum distance the magnet can be from the sensor and still be detected. Alternate (stronger or weaker) magnets can easily be used to change the trigger range of this sensor.

Installation and Configuration

The sensor portion of the Magnetic Limit switch and the two magnets and **M3 Hardware** compatible and can be attached to any extrusion or channel structural element.

The Magnetic Limit Switch can send signal from either the **n+1** or **n** ports. The REV Magnetic Limit Switch comes with two mountable magnets. Because this sensor does not require a contact interface, the magnet can also be soft mounted almost anywhere with just tape or glue.



- ⚠ For a simple system like stopping an arm at the end of range of motion, the hysteresis might not play much of a role, but for creating one or more stop points on a linear elevator, this may factor into the software design

Analog

Analog Sensor Basics

Analog sensors act as a type of adjustable resistor. As the state of the sensor changes, the voltage reporting back to the robot changes as well. Think of a dimmer switch, the brightness of the lights in the room depends on the where the setting of the knob of slider is. As the knob is adjusted the resistance level adjusts proportionally and the light continuously changes to the output from the knob.

✔ Can you think of anything that act like analog sensors around your household? Here are some we thought of: scale, thermometer, volume knob

Unlike the binary (either/or) status of digital sensors, analog sensors consider all numbers within a specific, given range. When using an analog sensor the actionable trigger will typically be some range like, between 1.2 and 2.2 volts or greater than 0.5 volts.

REV Robotics offers one analog sensor, known as a Potentiometer ([REV-31-1155](#)). The Potentiometer can be used to sense or measure the angular position of an shaft.

i To learn more about using the potentiometer check the [potentiometer](#) page.

Configuring Analog Sensors

Each Analog Port on the Control Hub ([REV-31-1595](#)) or Expansion Hub ([REV-31-1153](#)) has two channels for communication. However, a potentiometer can only be configured to channels 0 or 2; so it is not possible to connect two potentiometers on the same port.

i Some analog sensors from other companies will work with with the Control and Expansion Hub Analog Ports with the use of a custom wiring harness. Check out the [Using 5v Sensor](#) section for more information.

Configuring the potentiometer is simple. In the active configuration select the port being utilized, in this case it is port 0, and choose "Analog Input."

Done

Cancel

Port Attached

0

Analog Input ▼

Potentiometer

Device name

1

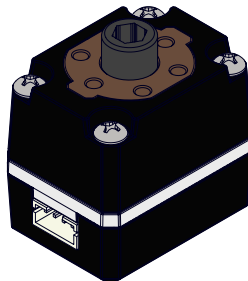
Nothing ▼

NO DEVICE ATTACHED

Device name

Potentiometer

Potentiometer Basics



The Potentiometer (REV-31-1155) senses the angular position of a shaft. It allows information about position to be converted into an analog voltage signal. This signal can be read by the Control Hub (REV-31-1595) or Expansion Hub (REV-31-1153) to control whatever device is attached. A potentiometer is essentially an adjustable resistor that fluctuates resistance as the shaft is turned. When connected to the Control Hub or Expansion Hub, the potentiometer is placed between ground and 3.3V. As the wiper (the knob) moves up and down along the coils of the resistor and the resistance and voltage output change proportionally at each new position.

The Potentiometer has a 270° limit to rotation. The sensor detects how much rotational motion has occurred in a mechanism. A specific limit is set in code to ensure rotation stops at a certain point. This is helpful when building simple arm joints because if properly applied it can prevent a mechanism from damaging itself or other parts of the robot.

⚠ It is important to install the Potentiometer so that it will not be forced beyond its 270° range of motion.

Product Specs

- **Sensor Type:** Analog
 - **Signal Port Mapping:** n
 - **Total Resistance:** 10kΩ
 - **Range of Motion:** 270°
 - **Electrical Connection:** 4-pin JST PH
 - **Output Shaft:** Female 5mm Hex
 - **Mounting Holes:** 6x M3 Tapped
-

How to Use?

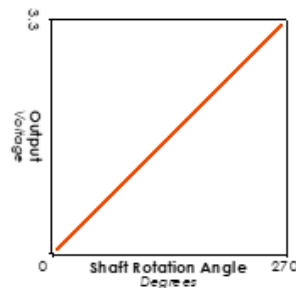
Applications

Potentiometers are most commonly used to measure the angle of an arm type joint. There are two different ways to utilize a potentiometer when using it in conjunction with an arm. One way to use the potentiometer is to directly place it on the shaft being used to pivot the arm. However, placing the potentiometer on an adjacent shaft that connects to the pivot-point shaft, via gears or chain, allows for more design flexibility.

Applying the concept of gear ratios (or sprocket ratios) to the potentiometer; it is possible to manipulate the accuracy/range of motion relationship. When range of motion increases, through changes in gear ratio, accuracy decreases and vice versa.

Calculating the relationship of voltage and angle

Keeping in line with the arm example, it is suggested to use a initialization routine at the start of an Op mode that reads or adjusts the voltage of the potentiometer. This is helpful in scenarios where an arm is in a known starting position or should be.



The general calculation for Degrees/Volts is as follows:

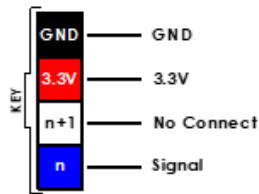
$$270^\circ / 3.3V = 270^\circ / 3300mV = 0.082^\circ / mV \text{ or } 8.18^\circ \text{ per tenth of a volt}$$


Installation and Configuration

The REV Potentiometer mounts to any REV Bracket with the 6 hole motion pattern. The bracket then is mounted to extrusion or channel. The sensor is attached to the bracket and the bracket to the extrusion or channel using 8mm M3 Hex Cap Screws ([REV-41-1359](#)).

This Potentiometer has a 5mm female hex socket input and can be used with any 5mm hex axle, like the ones in the REV Building system. There are six M3 tapped holes around the input shaft on a 16mm circle with will mount to any of the REV Robotics Motion Brackets.

The Potentiometer only sends signal to the hub through the **n** port, which means during configuration the potentiometer will need to be assigned to port 0 or port 2.



 To learn more about configuring the potentiometer head back to the [analog](#) page.

Programming Example


This program has a variable called CurrentVoltage that is used to store the current voltage. CurrentVoltage is updated using the AnalogInput block every time that the program loops. When CurrentVoltage less than the midpoint of 1.65 volts, the motor stops. When the voltage is higher than the midpoint, the motor moves. The potentiometer voltage is also displayed via telemetry.

Blocks

Op Mode Name: PotentiometerTest TeleOp Group:

- LinearOpMode
- Gamepad
 - Actuators
 - Sensors
- Other Devices
 - AnalogInput
 - Android
 - Utilities
- Logic
- Loops
- Math
- Text
- Lists
- Variables
- Functions
- Miscellaneous

```
to runOpMode
  Put initialization blocks here.
  call PotentiometerTest . waitForStart
  if call PotentiometerTest . opModelsActive
  do Put run blocks here.
  repeat while call PotentiometerTest . opModelsActive
  do
    Set CurrentVoltage at the beginning of the loop
    set CurrentVoltage to Potentiometer . Voltage
    Turn the motor on or off depending on the voltage
    if CurrentVoltage < 1.65
    do set motor . Power to 0
    else set motor . Power to 0.3
    Show the Potentiometer voltage via telemetry
    call Telemetry . addData
      key "Potentiometer voltage"
      number CurrentVoltage
    call Telemetry . update
```

 The code assumes that a Potentiometer was configured with the name "Potentiometer", and that a motor was configured with the name "Motor".

```
1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4 import com.qualcomm.robotcore.hardware.AnalogInput;
5 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
6 import com.qualcomm.robotcore.hardware.DcMotor;
7
8 @TeleOp
9 public class PotentiometerTest extends LinearOpMode {
10     // Define variables for our potentiometer and motor
11     AnalogInput potentiometer;
12     DcMotor motor;
13
14     // Define variable for the current voltage
15     double currentVoltage;
16
17     @Override
18     public void runOpMode() {
19         // Get the potentiometer and motor from hardwareMap
20         potentiometer = hardwareMap.get(AnalogInput.class, "Potentiometer");
21         motor = hardwareMap.get(DcMotor.class, "Motor");
22
23         // Loop while the Op Mode is running
24         waitForStart();
25         while (opModeIsActive()) {
26             // Update currentVoltage from the potentiometer
27             currentVoltage = potentiometer.getVoltage();
28
29             // Turn the motor on or off based on the potentiometer's position
30             if (currentVoltage < 1.65) {
31                 motor.setPower(0);
32             } else {
33                 motor.setPower(0.3);
34             }
35
36             // Show the potentiometer's voltage in telemetry
37             telemetry.addData("Potentiometer voltage", currentVoltage);
38             telemetry.update();
39         }
40     }
41 }
42
```


I2C


I2C Sensor Basics

I2C is a common electronic communication standard that allows a leader device, the Hubs, to communicate with multiple devices, followers, attached to the same port. Each connector on a Hub is a separate I2C bus and many different sensors can be connected to each of the four I2C busses available on both the Control Hub and Expansion Hub. Every I2C follower device has a unique address, a number, which is normally fixed by the manufacturer. All of the devices on an individual I2C bus must have a unique address so that the master can communicate with one sensor at a time. If two devices have the same address, such as when using two of the same kind of sensor, they must be used on different I2C busses.

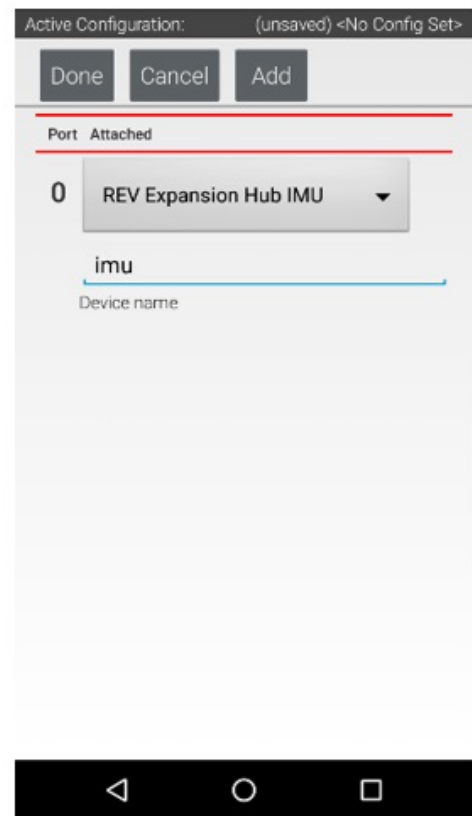
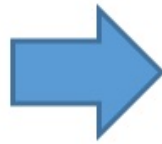
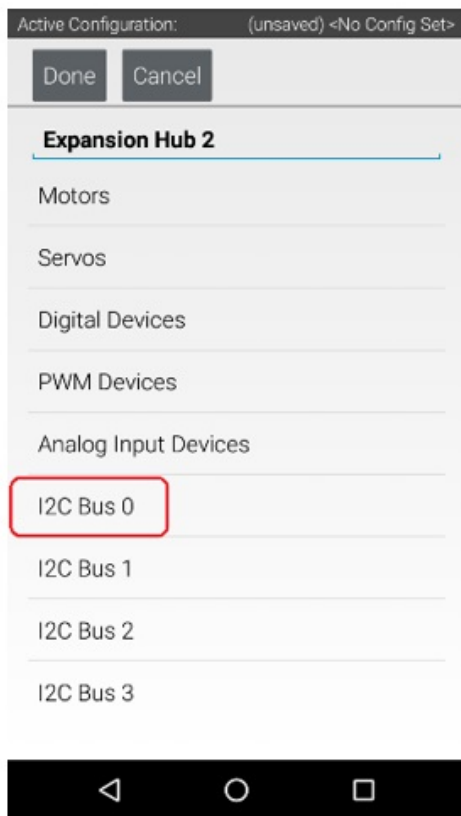
There are three I2C sensors within the REV system: IMU, color, and 2m Distance. The IMU sensor is built in to the Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)).

Configuring I2C Sensors

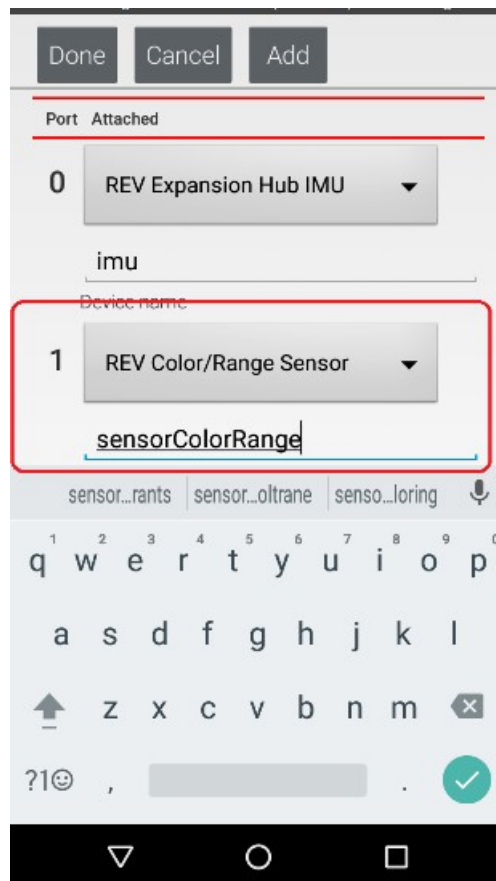
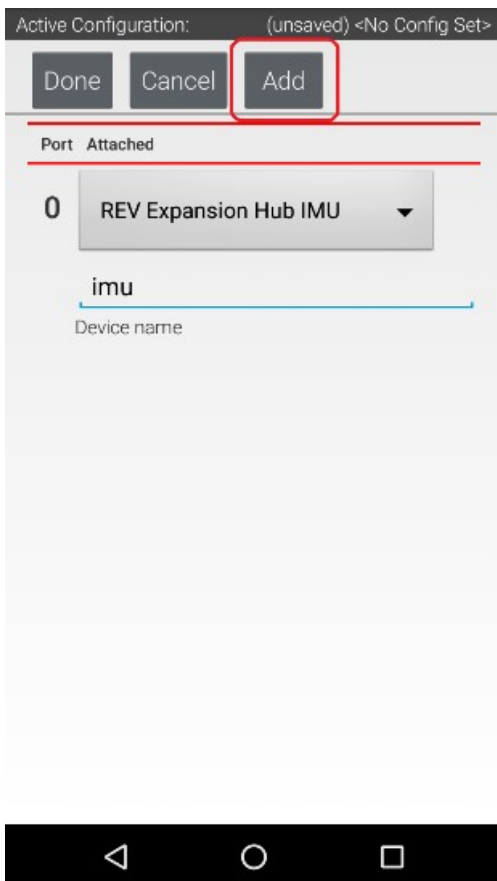
The I2C Bus 0 hosts the internal **IMU** sensor within the Hubs. A REV Robotics Color Sensor ([REV-31-1557](#)) or a 2m Distance Sensor ([REV-31-1505](#)) can be added to I2C Bus 0. The steps below walk through adding a color sensor to Bus 0.

 The REV Robotics Color and 2m Distance Sensors share the same address. In order to have proper functionality of both sensors they should be configured to different buses.

Step 1: Click I2C Bus 0 to launch the configuration screen. As you can see the IMU sensor is already configured to this bus.



Step 2: Press the Add button to add the Color Sensor to this bus. Select "REV Color/Range Sensor" from the drop down menu and name the device.

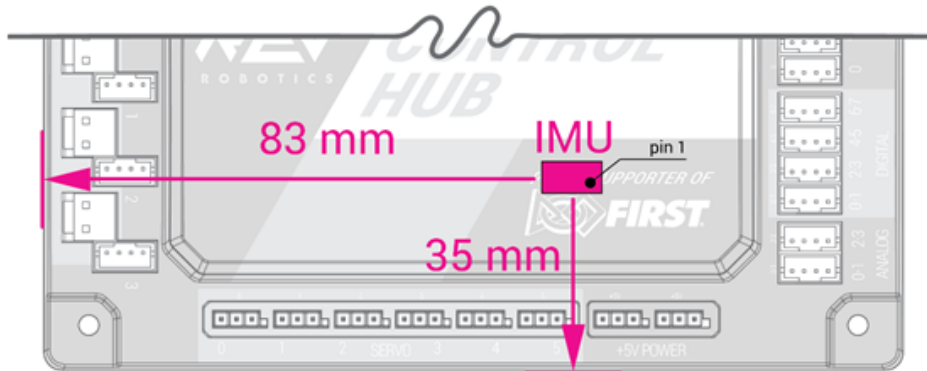


Step 3: When you have finished configuring the sensor hit 'Done.' The app will return to the previous screen.


IMU

IMU Basics

BELOW: IMU Details Shown in Enlarged View



Every REV Robotics Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) have a built in 9-axis IMU, or inertia measurement unit. The IMU incorporates three sensors: a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis geomagnetic sensor. The **accelerometer** measures the affect of forces on acceleration along the three axes. The **gyroscope** measures the rotational location of the the Hubs along the axes. The **geomagnetic** sensor (or magnetometer) uses the Earth's magnetic field to find orientation.

 The accuracy of the magnetometer within the IMU is affected by proximity to surrounding magnetic fields.

The data considered and used by the IMU includes: rotation along each axis, forces of acceleration along each axis, and magnitude of acceleration. The rotational measurements for the gyroscope play an important part in the use of the gyroscope for positioning and location of the robot.

- **Heading** is the measure of rotation along the z-axis. If the Hub is laying flat on a table, the z-axis points upwards through the front plate of the Hub.
- **Pitch** is the measure of rotation along the x-axis. The x-axis is the axis that runs from the bottom of the hub, near the servo ports, to the top of the hub ,where the USB ports are.
- **Roll** is the measure along the y-axis. The y-axis is the axis that runs from the sensor ports on the right to the motor ports on the left.

The orientation of the hub plays a large part into which measurement will be used to determine the orientation of the robot as it moves.

Product Specifications

- I2C Address: 0x28
 - Port: 0
-

How to Use?

Application

There are a multitude of applications for the IMU within autonomous op modes:

- Use the Gyroscope to drive in the straight lines and turn during autonomous
- Use the Accelerometer in conjunction with the gyroscope to avoid drift and give an approximation of position/travel
- Use the IMU with motor encoders to track and determine robot placement on a field

Configuration

Since the IMU is already installed with in the Control or Expansion Hub the main concerns are Hub position and configuration. **The orientation of the Hub affects which axis is getting feedback.**

The IMU always exists on I2C Bus 0.



To learn more on how to configure the IMU check out the [I2C](#) introduction page.

Color Sensor

Color Sensor Basics



The REV Robotics Color Sensor V3 ([REV-31-1557](#)) is a combined color and proximity sensor. From a single sensor you can measure colors and rough distances to various targets. Version 3 introduces a new sensor chip from Broadcom due to the end-of-life of the V1/V2 color sensor chip.

Product Specifications

- **Max. Operating Voltage** 3.3V
- **Sensor Type** I2C
- **I2C Address** 0x52
- **Sensor Part** APDS-9151
- **Measurement Channels** Red, Green, Blue, Alpha, and Proximity
- **Proximity Sensor Range** 1cm – 10cm

Features

- Digital RGB Color Sensing
 - IR Proximity Emitter and Detector
 - Built-in (switchable) white LED
 - Supports Standard (100kHz) or High Speed (400kHz) I2C
-

How to Use?


Application

The REV Robotics Color Sensor has two sensing elements: color and proximity.

Color measurements consist of Red, Green, Blue, and Alpha (clear) values. The white LED on the sensor has a slide switch to turn the LED on or off. Unlit targets are best illuminated with the build-in LED while bright or light-emitting targets may not require the build-in LED. Color data is best collected within 2cm of the target for the strongest color differentiation.

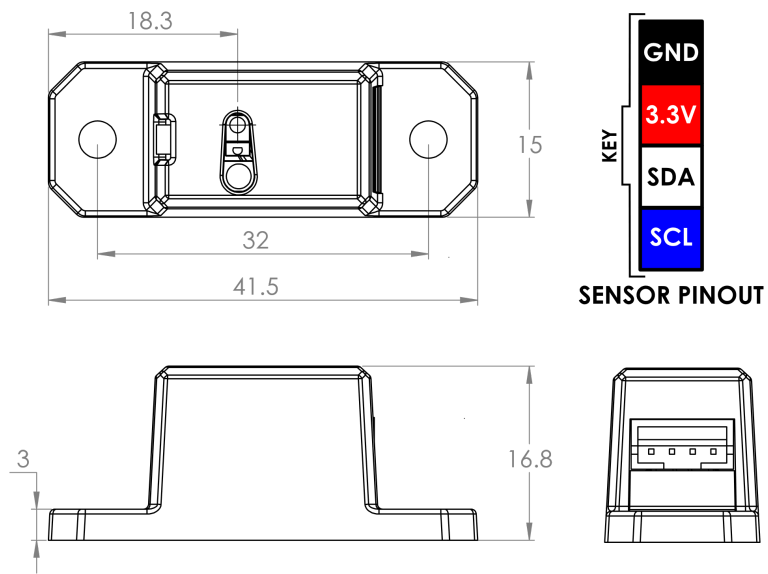
Color sensor applications within FTC vary based on season, although it is typically used to find colored game objects in autonomous mode.

Proximity measurements are based on IR reflectance and can vary depending on lighting conditions and target reflectivity. The proximity sensor is ideally used to determine if something is in front of the sensor. While you can receive rough distance data, we recommend using the 2m Distance Sensor ([REV-31-1505](#)) or similar time-of-flight sensor for accurate distance measurement.

 **Note to users transitioning from Color Sensor V2 to V3:** Color values will not be consistent between V2 and V3 sensors and there are minor changes to the FTC SDK. Be sure to update to the latest SDK and configure your robot to use the "REV Color Sensor V3".

Installation and Configuration

The sensor can be attached to channel or extrusion using 8mm M3 Hex Cap Screws ([REV-41-1359](#)).



i To learn how to configure the color sensor on and I2C Bus check out the [I2C](#) top page!

Programming Example


This program shows the values from the Color Sensor on your phone. Your team will need to figure out the logic to use this information in your program. Below there are three examples of different color modes and their readings. Light Detected mode will read the amount of light on the sensor from 0-1.0. Because the sensor is close to a surface, the LED in the sensor reads 1.0 in the examples.

Blocks

Op Mode Name: TestSensor TeleOp Group:

- LinearOpMode
- Gamepad
- Actuators
- Sensors
- Other Devices
- Android
- Utilities
- Logic
- Loops
- Math
- Text
- Lists
- Variables
- Functions
- Miscellaneous

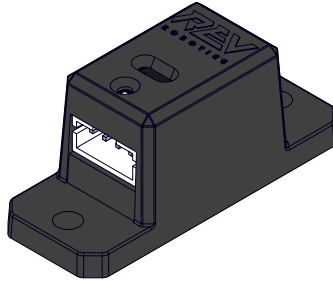
```
to runOpMode
  Setup that runs after the INIT button
  set arm . Mode to RunMode . RUN_WITHOUT_ENCODER
  call TestSensor . waitForStart
  Put run blocks here.
  repeat while call TestSensor . opModelsActive
  do
    call Telemetry . addData
      key " Arm Motor Power "
      number arm . Power
    call Telemetry . addData
      key " Light Detected "
      number Color . LightDetected
    call Telemetry . addData
      key " Red "
      number Color . Red
    call Telemetry . addData
      key " Blue "
      number Color . Blue
    call Telemetry . addData
      key " Green "
      number Color . Green
    call Telemetry . update
```

 The code assumes that the Color Sensor was configured with the name "Color."

```
1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4 import com.qualcomm.robotcore.hardware.ColorSensor;
5 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
6
7 @TeleOp
8 public class TestColorSensor extends LinearOpMode {
9     // Define a variable for our color sensor
10    ColorSensor color;
11
12    @Override
13    public void runOpMode() {
14        // Get the color sensor from hardwareMap
15        color = hardwareMap.get(ColorSensor.class, "Color");
16
17        // Wait for the Play button to be pressed
18        waitForStart();
19
20        // While the Op Mode is running, update the telemetry values.
21        while (opModeIsActive()) {
22            telemetry.addData("Red", color.red());
23            telemetry.addData("Green", color.green());
24            telemetry.addData("Blue", color.blue());
25            telemetry.update();
26        }
27    }
28 }
29
```

2m Distance Sensor

Distance Sensor Basics



The 2m Distance Sensor ([REV-31-1505](#)) measures distances up to 2 meters with millimeter resolution. It is typically used to detect the distance from the sensor to other opaque objects. This sensor can measure how long it takes for the light to bounce off the object it is directed at and return to the sensor. This “time of flight” measurement is more accurate than sensors that rely on the intensity of reflected light.

Product Specifications

- **Measurement Range:** 5cm - 200cm
- **Measurement Resolution:** 1mm
- **Field of View:** 25°
- **Laser Type:** 940 nm (IR) Class 1
- **Sensor Type:** I2C
- **Maximum Bus Frequency:** 400 kHz
- **I2C Address:** 0x52
- **Voltage Range:** 3.3V - 5.0V
- **Max. Operating Current:** 40 mA

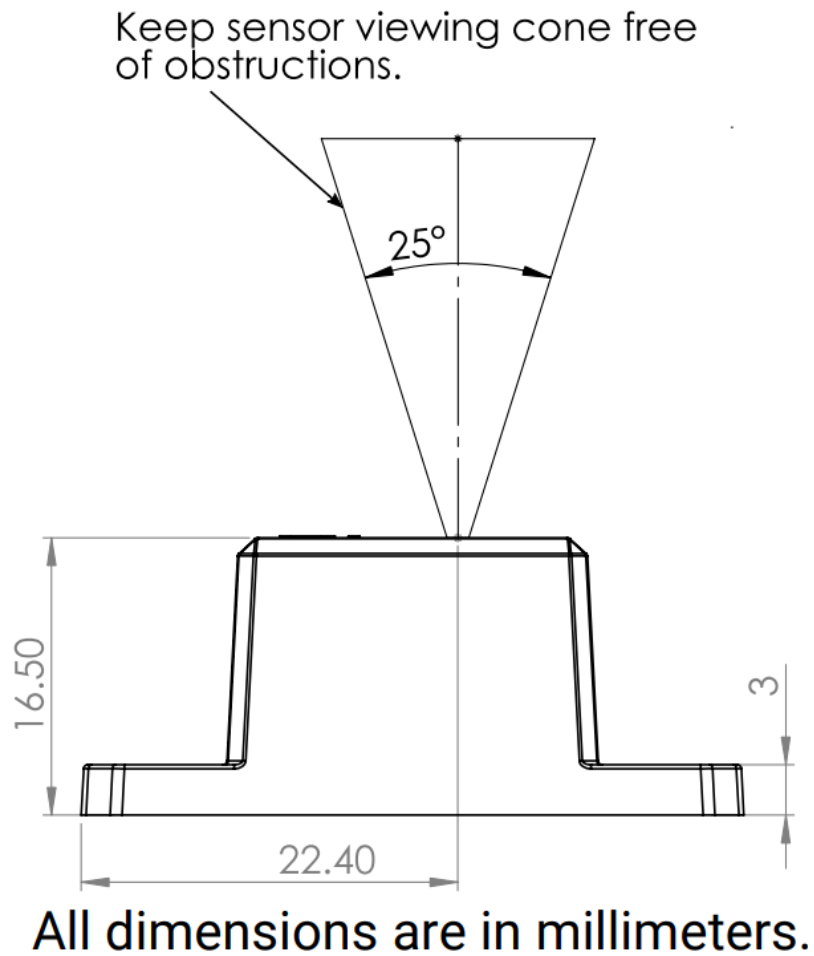
How to Use?

Application

While the REV 2m Distance Sensor produces a significantly more accurate and reliable measurement than other types of ranging sensors, the following tips will help minimize errors.

A major benefit of the time of flight measurement is that the target's surface reflectance does not significantly impact the calculated distance. However, the smallest errors and farthest measurements are achieved with more reflective targets. Similarly, larger targets are easier to detect because they fill more of the sensor's field of view.

Ambient infrared (IR) interference can also affect the measurement distance and quality. The sensor can produce accurate measurements in sunlit environments, but maximum distance will be reduced. The following table outlines the typical ranging capabilities of the sensor.

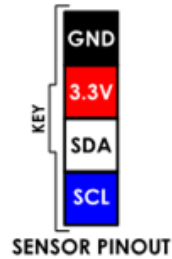
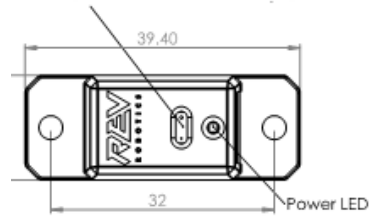


Target Reflectance	Indoor	Outdoor (overcast)
White (88%)	200 cm	80 cm
Grey (17%)	80 cm	50 cm

Installation and Configuration:

The sensor can be attached to channel or extrusion using 8mm M3 Hex Cap Screws ([REV-41-1359](#)).

Caution: Do not touch sensing element.
Touching sensor can result in damage



[Information on how to configure I2C sensors can be found on the I2C page.](#)

Programming Example


This program moves a motor if there is an object less than 10 centimetres from the distance sensor, and stops it if there is no object within that range.

Blocks

Op Mode Name: TestSensor TeleOp Group:

- LinearOpMode
- Gamepad
- Actuators
 - CRServo
 - DcMotor
 - Servo
- Sensors
 - IMU-BNO055
 - IMU-BNO055.Parameters
 - DistanceSensor
 - REV Color/Range Sensor
 - TouchSensor
 - VoltageSensor
- Other Devices
- Android
- Utilities
- Logic
- Loops
- Math
- Text

```
to runOpMode
  Setup that runs after the INIT button
  set arm . Mode to RunMode . RUN_WITHOUT_ENCODER
  call TestSensor . waitForStart
  Put run blocks here.
  repeat while call TestSensor . opModelsActive
  do
    if Touch . IsPressed
    do
      If Touch sensor is pressed stop motor
      set arm . Power to 0
    else
      set arm . Power to 0.3
  Put loop blocks here.
  call Telemetry . addData
  key " Arm Motor Power "
  number arm . Power
  call Telemetry . update
```

 The Java version of this program is pasted below. It assumes that the Distance Sensor was configured with the name "Distance" and that a motor was configured with the name "Motor."

```
1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
4 import com.qualcomm.robotcore.hardware.DcMotor;
5 import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;
6 import com.qualcomm.robotcore.hardware.DistanceSensor;
7 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
8
9 @TeleOp
10 public class DistanceTest extends LinearOpMode {
11     DistanceSensor distance;
12     DcMotor motor;
13
14     @Override
15     public void runOpMode() {
16         // Get the distance sensor and motor from hardwareMap
17         distance = hardwareMap.get(DistanceSensor.class, "Distance");
18         motor = hardwareMap.get(DcMotor.class, "Motor");
19
20         // Loop while the Op Mode is running
21         waitForStart();
22         while (opModeIsActive()) {
23             // If the distance in centimeters is less than 10, set the power to 0.3
24             if (distance.getDistance(DistanceUnit.CM) < 10) {
25                 motor.setPower(0.3);
26             } else { // Otherwise, stop the motor
27                 motor.setPower(0);
28             }
29         }
30     }
31 }
32
```

Encoders

What is an Encoder?

An encoder is anything (device, software, person) that *converts* information from one format(code) into another. Some examples of encoding include:

- A transducer, like a speaker, which *converts* an electrical signal into music
- Software which *encodes* an audio file into an mp3 to decrease file size
- A stenographer (court reporter) takes courtroom dialog and *converts* it into a written record

This section is about rotary encoders which are electro-mechanical devices which convert the angular position of a shaft, like on a motor, to an electronic signal. These signals can be fed into a microcontroller, which controls all robot functions, and then used to provide real world data to make better programming decisions.

There are two main types of encoders: absolute and relative.

Absolute encoders return the actual angle of the rotation (e.g. 30°). Absolute encoders maintain position information if the power is removed, and position data is immediately available when power is reapplied with no rotation needed to read the current angle. The relationship between the encoder value and the motor shaft is set when assembled and will always stay the same. Commonly these encoders use a specially printed pattern disk which are read and converted to a known angle. Generally, absolute encoders are easier to use when programming, but they are more complicated to manufacture so are larger, or more expensive.

Relative encoders, which are also referred to as **incremental encoders**, provide information about the motion of the shaft (e.g. forward at 5 RPM), and only provide data while the shaft is rotating. One way to remember this is that relative encoders return information on the incremental change of the motor output shaft. Relative encoders only provide pulses as the motor turns, and interpreting these pulses into useful information must be done externally. A relative encoder does not know what position it is in at start-up, but it is possible to create a calibration program that must be run at every start-up to obtain reference point to calculate an angle from.

Encoders measure a real world change (shaft rotation) and convert it to an electrical signal. Two common ways to do this are using optical or magnetic feedback:

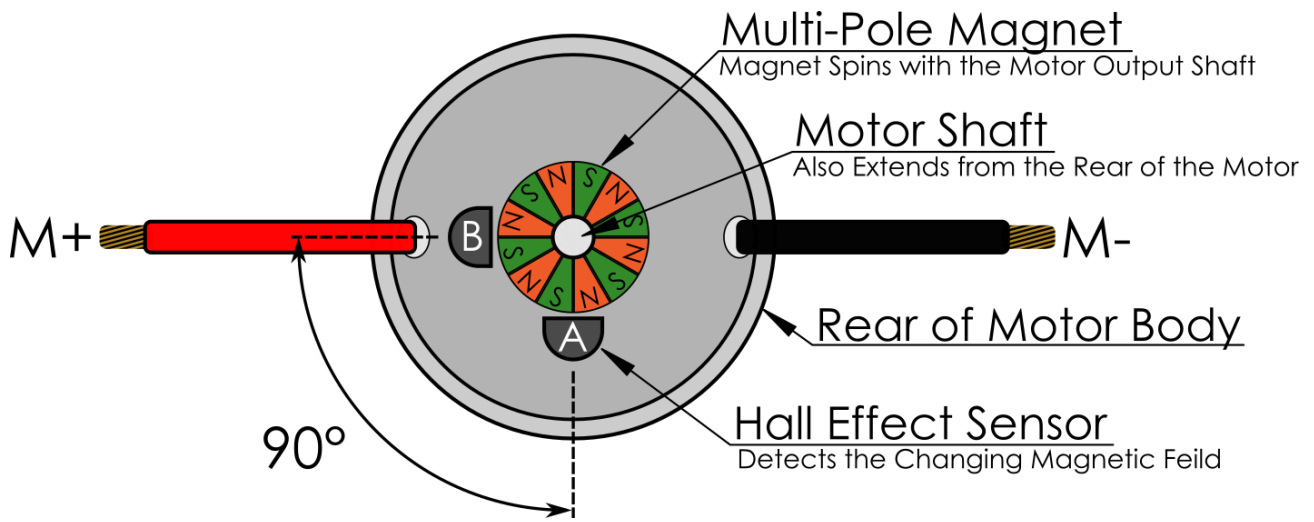
Optical encoders have a disk with a series of either slots or a reflective pattern around the outside which is attached to the motor shaft. A light shines on or through the disk where the light can pass through or reflect onto a photodiode (device which produces an electric signal when light shines on it). These sensors can be very light and compact, but can be very sensitive to anything that might interfere with the light reaching the photodiode. Finger prints on a reflective disk, or dust from a dirty environment can interfere.

Magnetic encoders have a magnet attached to the shaft of a motor and use Hall effect sensors to detect the changing magnetic field as the shaft rotates. Magnetic encoders are able to operate in harsh or dirty

environments.

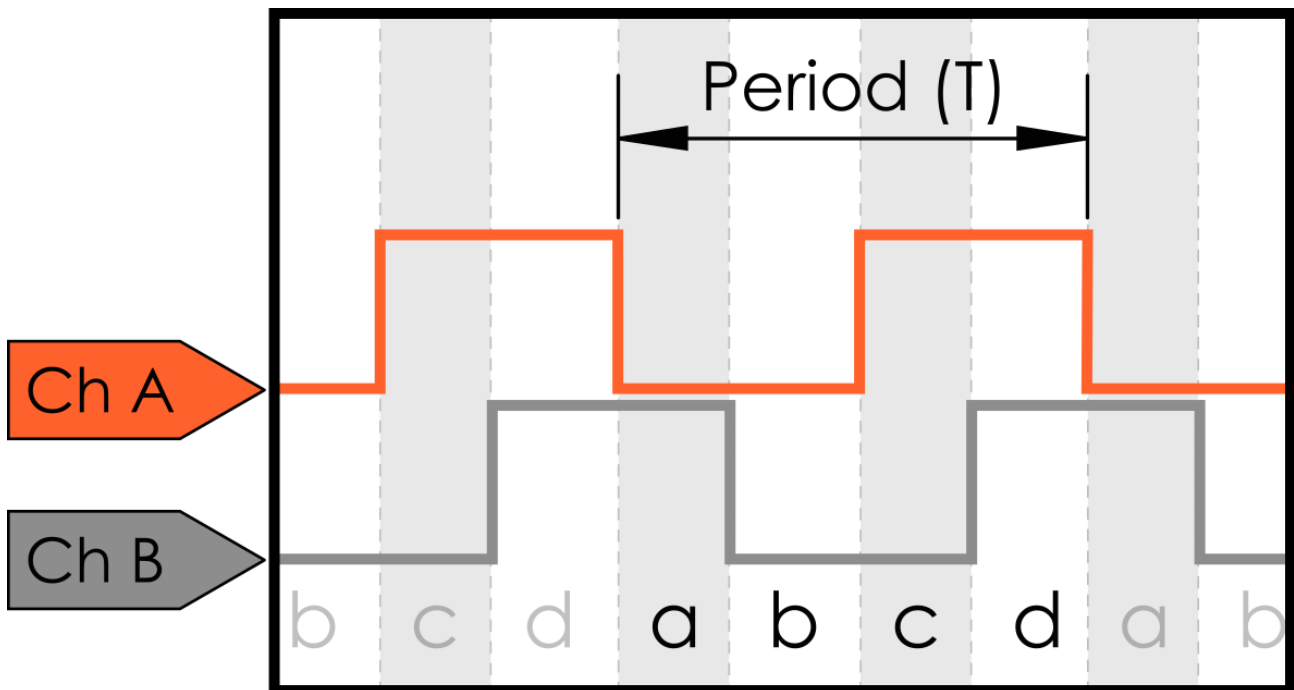
Magnetic Quadrature Encoders

A 12 pole magnetic quadrature encoder is installed on the rear of both the HD Hex Motor and Core Hex Motor. The output shaft of the motor extends from the rear of the motor case and a multi-pole permanent magnet is attached to the shaft. There are two Hall effect sensors, marked 'A' and 'B', mounted next to the magnet at 90° to each other. As each of the 12 poles passes across one of the Hall effect sensors, it creates a change in the magnetic field causing the sensor to generate a measurable voltage signal.



Typical Encoder Configuration Installed on the Rear of a Motor

Quadrature encoders are a specific type of relative encoder that have four different output states. If the root *quad*; means four, but there are only two sensors in this encoder, where does the name come from? The output from the two Hall effect sensors are called "Channel A" and Channel B" respectively; an example of the output is shown below. In a single period (T), defined as the duration of time of one complete cycle in a repeating pattern, the timing diagram has four distinct states (see a, b, c, and d below), hence a quadrature encoder.



Clockwise Quadrature Encoder Output Timing Diagram

The offset from Channel A to Channel B is because the sensors are offset from each other by 90° . As the motor rotates one sensor will see the change before the other. When the motor shaft rotates clockwise (CW), Channel A will lead (the edge will rise before) Channel B. When the motor spins counter clockwise (CCW) Channel A will lag (rise after) Channel B. If there was only one sensor it would still be possible to measure the number of rotations, but not to detect the direction of the motor.

- i On HD Hex and Core Hex motors Channel A leads Channel B when positive voltage is applied to the M+ terminal. However, there are times when this will not hold true in real life. Different reduction gearboxes, or physically swapping the Channel A and Channel B encoder wires into the controller, can reverse the relationship between the channels. Keep this in mind when programming and troubleshooting your robot.

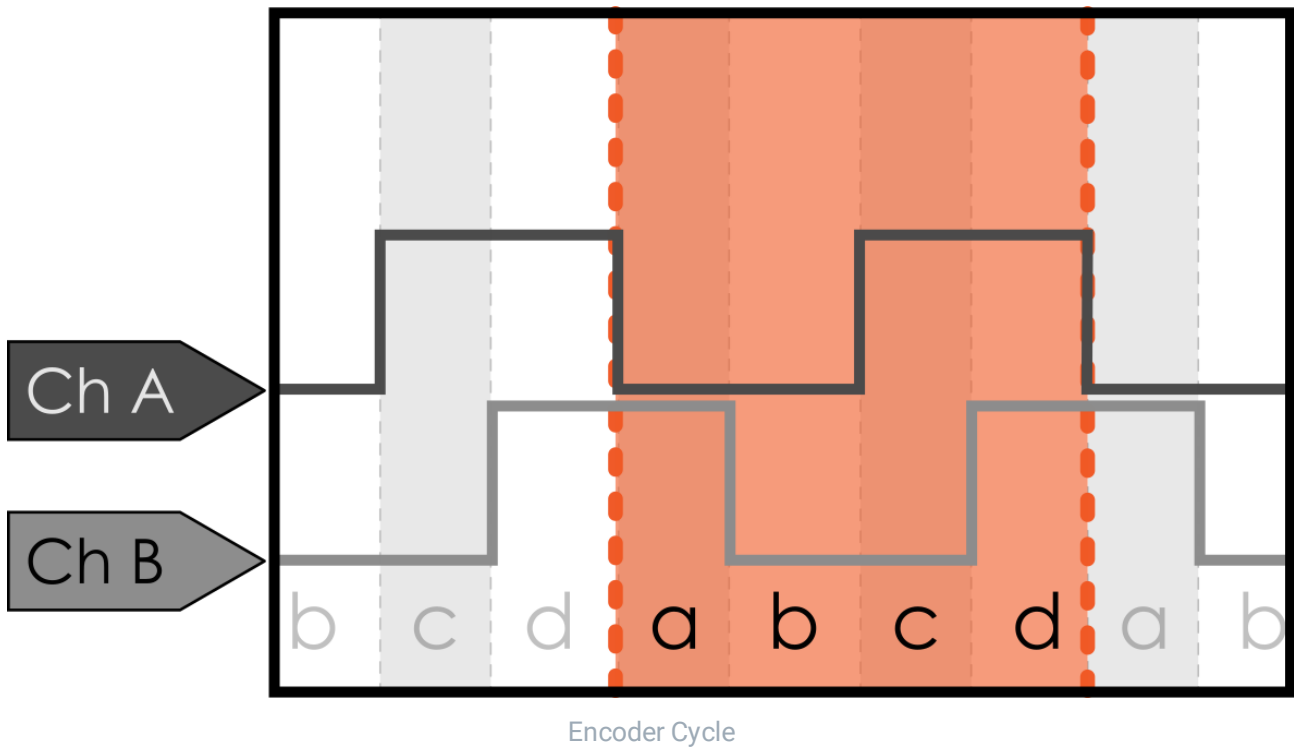
When the encoder is being read by a microcontroller, the two signals are compared to produce a count up pulse or count down pulse. These pulses are counted as steps forward (CW) or backwards (CCW). Using the specifications for the encoder being used, a count can be converted to degrees. This information can be used to drive a robot arm to a specific angle, or tell a robot to drive a certain distance. Both the Control Hub and Expansion Hub communicate to a microcontroller through the encoder ports.

Encoder Technical Specification Definitions

- i There is some conflicting terminology difference between encoder suppliers. This document defines one of the most commonly agreed upon set of terms, however be aware that when

comparing between encoder specifications from different vendor's terms may vary in meaning.

Every time the output goes through all four distinct combinations of output signals, it's called a **cycle** (see a, b, c, and d below). Encoders have a different **cycles-pre-revolution(CPR)** based on the number of poles on the magnet used. The CPR is how many cycles are generated for one complete revolution of the encoder shaft.



An example output from one complete rotation of a 14 CPR encoder is shown in the figure below. A 14 CPR rotation encoder may also be referred to as having 14 rises on channel A. Encoders are mounted to the motor shaft, not the gearbox output shaft, so for a motor with a reduction gearbox attached this is less than one full output shaft rotation.

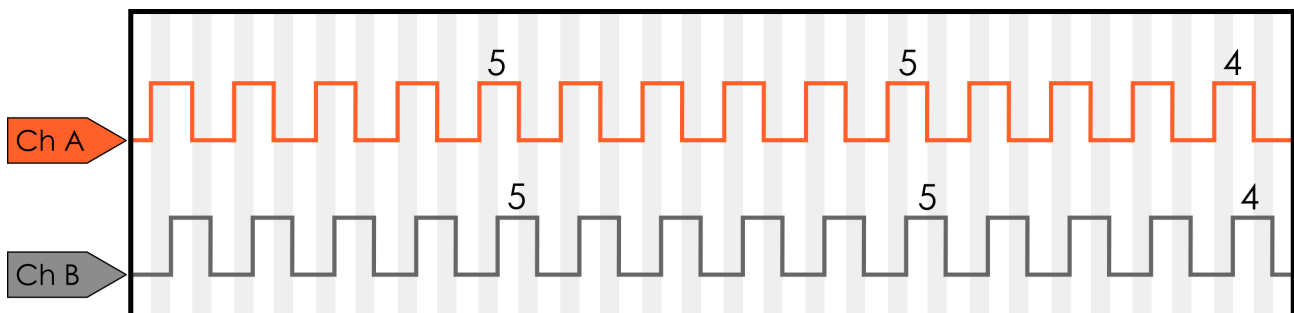


Figure 4: Encoder Output for one Revolution of a 14 CPR Encoder

One reason to use CPR to define an encoder, rather than the commonly used PPR (Pulses per Revolution) is when the encoder signal is decoded by the microcontroller it is possible to do 1x, 2x, or 4x decoding. For 1x

decoding the micro controller would only “count” the rising signal on a single channel, while for 4x decoding each rising or falling edge for both channels is measured as a “count.” Although 4x decoding is one of the most common methods, because it’s based on how the electronics decode the signal from the encoder, and not on the encoder hardware itself, it’s not an ideal method of defining the encoder hardware specifications.

If we assume 4x decoding when each cycle is interpreted, the microcontroller can read the four distinct outputs (a, b, c, and d) as individual steps. So for each CPR, the controller can read four counts/ticks. To calculate the number of counts per rotation of the encoder shaft:

$$COUNTSPERROTATION_{(of\ the\ encoder\ shaft)} = CPR(Cycles\ per\ rotation) \times 4$$

The actual cycles per rotation of the output shaft of the motor is depending on the gearbox that’s attached.

$$COUNTSPERROTATION_{(of\ the\ output\ shaft)} = CPR(Cycles\ per\ rotation) \times 4 \times Reduction$$

This can be calculated into the degrees per count. Assuming no additional reduction is added to the final stage of the motor output (i.e. direct drive) the number of degrees per count is calculated as:

$$DEGREESPERCOUNT = 360^\circ / COUNTSPERROTATION_{(of\ the\ output\ shaft)}$$

REV Motor Encoders

REV Robotics HD Hex Motors ([REV-41-1291](#)) and the Core Hex Motors ([REV-41-1300](#)) come with a magnetic quadrature encoder already installed and an appropriate cable for connecting the encoder output to the REV Robotics Control Hub ([REV-31-1595](#)) or Expansion Hub ([REV-31-1153](#)). See Table 1 and Table 2 for relevant encoder details.

Core Hex Motor (REV-41-1300) Encoder Specifications


Core Hex Motor (REV-41-1300) Reduction	72:1
Free Speed (RPM)	125
Cycles per Rotation of the Encoder Shaft	4 (1 Rise of Channel A)
Counts per Rotation of the Output Shaft	288 (72 Rises of Channel A)

HD Hex Motor (REV-41-1291) Encoder Specifications

HD Hex Motor Reduction	Bare Motor	40:1	20:1
Free Speed (RPM)	6000	150	300
Cycles per Rotation of the Encoder Shaft	28 (7 Rises of Channel A)	28 (7 Rises of Channel A)	28 (7 Rises of Channel A)
Counts per Rotation of the Output Shaft	28 (7 Rises of Channel A)	1120 (280 Rises of Channel A)	560 (140 Rises of Channel A)

Through Bore Encoder

The REV Through Bore Encoder ([REV-11-1271](#)) is specifically designed with the end user in mind, allowing teams to place the sensor in the locations closest to the rotation that they wish to measure. This rotary sensor measures both relative and absolute position through its ABI quadrature output and its absolute position pulse output.

 The FTC Control System (Control Hub and Expansion Hub) only supports incremental encoder input through the motor encoder ports at this time. Absolute pulse input is not supported.

Included with the Through Bore Encoder is a 5mm Hex insert and a 4-Pin JST PH to 6-pin JST PH connector. The 6-pin connector is plugged into the Through Bore Encoder with the 4-pin connector plugging into either the Control Hub ([REV-31-1595](#)) and Expansion Hub ([REV-31-1153](#)) Encoder Port. Both the A and B channels of the encoder are used.

When using the 5mm Hex insert, press the insert into the 1/2" Hex hole before attaching to a mechanism. If you are having difficulty pressing the insert into the encoder, try flipping the insert over and press it in. There is a slight taper in the insert, so it is recommended to press the insert with the smaller end first. When removing, it is recommended to push the insert out in the reverse order (larger end first).

Using 5V Sensors

The Control Hub ([REV-31-1595](#)) is a 3.3V logic level device, but many of the sensors that teams have previously purchased through companies such as Modern Robotics are 5V logic level devices. Many of these previously purchased sensors can be used with the new system by using a logic level converter. REV Robotics offers a Logic Level Converter ([REV-31-1389](#)) and an optional Sensor Adapter Cable ([REV-31-1384](#)) so teams can more easily use their legacy sensors with the REV Control System.

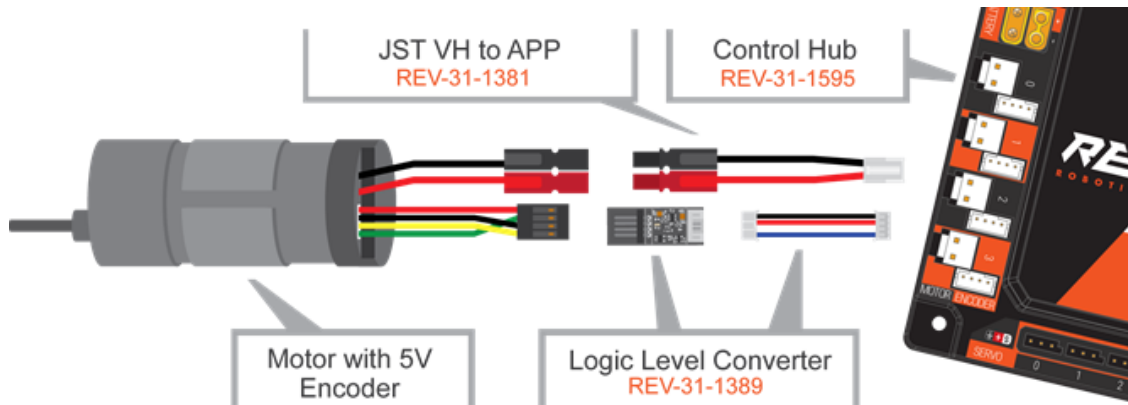
Logic Level Converter

The REV Robotics Logic Level Converter is a PCB which generates a 5V output from the 3.3V input and uses a MOSFET on each signal line to create a bidirectional communication appropriate for a variety of digital signals include I2C communication (Figure 5). For more information on how bidirectional level shifting communication is accomplished, please reference the [NXP Application Note AN10441](#).



Connecting 5V Encoder

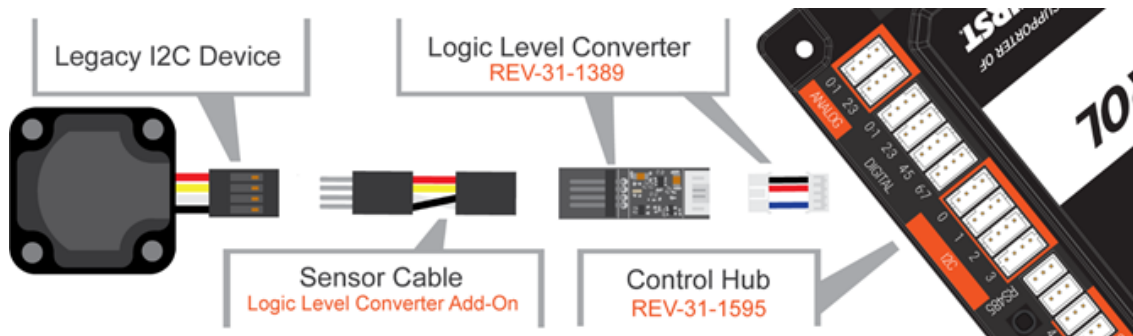
The Logic Level Converter ([REV-31-1389](#)) pin out directly matches the encoder cable pin out for FTC legal motors. Encoder cables plug directly into the Logic Level Converter board and then the 4-pin JST PH Cable ([REV-31-1407](#)), which is included with the Logic Level Converter, is plugged into the appropriate Control Hub ([REV-31-1595](#)) Encoder Port. Motors which are terminated with Anderson Power Pole style connectors use the JST VH to Anderson Power Pole Style ([REV-31-1381](#)) cable to connect to the motor output port on the Control Hub.



- i All REV Robotics Motors work directly with the REV Control and Expansion Hubs. No Logic Level Converter is needed for REV Motors.

Connecting a 5V Sensor

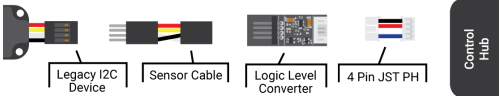
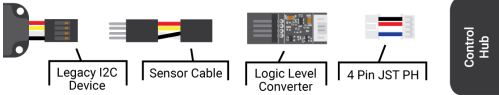
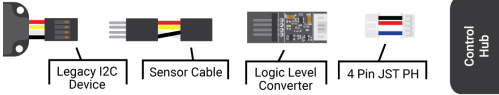
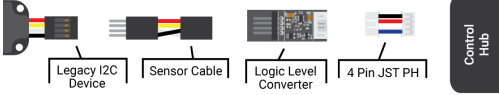
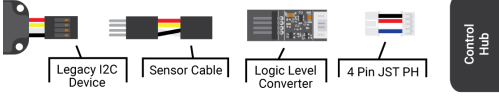
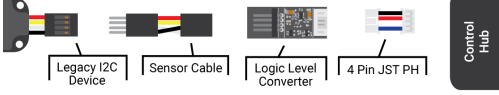
A variety of 5v sensors are usable with the Control Hub (REV-31-1595) when used with a Logic Level Converter (REV-31-1389). For some Modern Robotics I2C sensors a Logic Level Converter, and a change in wiring to match the pin out of the Control Hub are needed. Teams can either purchase a Sensor Cable as an add on to the Logic Level Converter Kit which will cross over the correct wires, or they can carefully rearrange the pin order on the sensor cable. If using the Sensor Cable, connect the sensor to the Control Hub as shown below. It is recommended to zip tie the connection between the sensor and the sensor cable to prevent accidental disconnects. See the [Sensor Compatibility Chart](#) for more information on hardware required for other sensors.



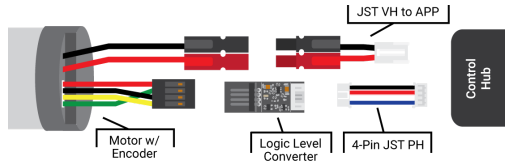
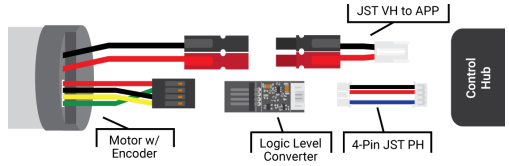
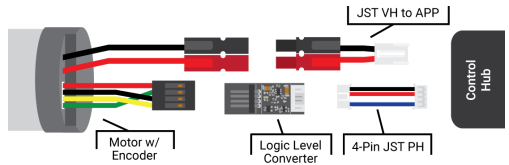
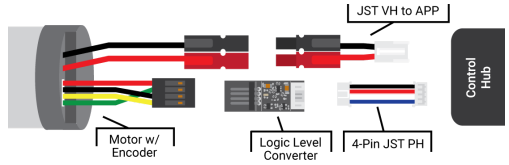
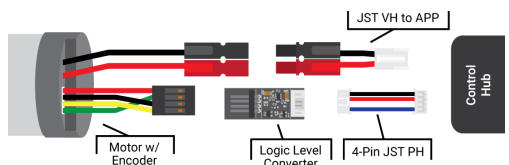
Sensor Compatibility Chart

To determine if your existing sensors can be used with the Control Hub ([REV-31-1595](#)), along with additional hardware needed, see the table below.

Sensor Compatibility Table

Sensor	Type	Compatible	Adapters Needed
Absolute Orientation IMU Fusion Breakout - BNO055 2472 Adafruit	I2C	Yes	3.3V Compatible Custom Wiring Harness Needed
RGB Color Sensor with IR filter and White LED - TCS34725 1334 AdaFruit	I2C	Yes	3.3V Compatible Custom Wiring Harness Needed
Color Sensor 45-2018 Modern Robotics	I2C	Yes	
Compass 45-2003 Modern Robotics	I2C	Yes	
Integrating Gyro 45-2005 Modern Robotics	I2C	Yes	
IR Locator 360 45-2009 Modern Robotics	I2C	Yes	
IR Seeker V3 45-2017 Modern Robotics	I2C	Yes	
Ranger Sensor 45-2008 Modern Robotics	I2C	Yes	



<p>NeveRest Motor AM-3461, AM-3102, AM-2964a, AM-3103, AM-3104 AndyMark</p>	<p>Quad Encoder</p>	<p>Yes</p>	
<p>HD Hex Motor REV-41-1301 REV Robotics</p>	<p>Quad Encoder</p>	<p>Yes</p>	<p>Directly Compatible No Custom Adapters Needed</p>
<p>Core Hex Motor REV-41-1301 REV Robotics</p>	<p>Quad Encoder</p>	<p>Yes</p>	<p>Directly Compatible No Custom Adapters Needed</p>
<p>12v 4mm Motor Kit 50-0119 MATRIX</p>	<p>Quad Encoder</p>	<p>Yes</p>	
<p>12v 6mm Motor Kit 50-0120 MATRIX</p>	<p>Quad Encoder</p>	<p>Yes</p>	
<p>Standard Motor Kit 50-0001 MATRIX</p>	<p>Quad Encoder</p>	<p>Yes</p>	
<p>Max Motor Shaft Encoder Kit W38000 Tetrix</p>	<p>Quad Encoder</p>	<p>Yes</p>	
<p>Limit Switch 45-2401 Modern Robotics</p>	<p>Digital</p>	<p>Yes</p>	<p>No Adapter Needed Custom Wiring Harness Required.</p>
<p>Rate Gyro 45-2004 Modern Robotics</p>	<p>Analog</p>	<p>No</p>	<p>Not Officially Supported</p>
<p>Optical Distance Sensor 45-2006 Modern Robotics</p>	<p>Analog</p>	<p>No</p>	<p>Not Officially Supported</p>



Touch Sensor 45-2007 Modern Robotics	Analog	Yes	No Adapter Needed Custom Wiring Harness Required
Light Sensor 45-2015 Modern Robotics	Analog	No	Not Officially Supported
Magnetic Sensor 45-2020 Modern Robotics	Analog	No	Not Officially Supported
