# SPARK MAX

## SPARK MAX Overview

The REV Robotics SPARK MAX Motor Controller is an all-in-one USB, CAN, and PWM enabled motor controller that can drive both 12 V brushed and 12 V brushless DC motors. SPARK MAX is designed for use in the *FIRST*® Robotics Competition (FRC), incorporating advanced motor control in a small, easy-to-use, and affordable package. Configure and run the SPARK MAX through its built-in USB interface without needing a full control system.

## Feature Summary

- Brushed and sensored-brushless motor control
- PWM, CAN, and USB control interfaces
  - PWM/CAN - Locking and keyed 4-pin JST-PH
  - USB - USB type C
- USB configuration and control
  - Rapid configuration with a PC
- Smart control modes
  - Closed-loop velocity control
  - Closed-loop position control
  - Follower mode
- Encoder port
  - Locking and keyed 6-pin JST-PH
  - 3-phase hall-sensor encoder input
  - Motor temperature sensor input
- Data port
  - Limit switch input
  - Quadrature encoder input with index
  - Multi-function pin
- Mode button
  - On-board motor type and idle behavior configuration
- RGB status LED
  - Detailed mode and operation feedback
- Integrated power and motor wires
  - 12 AWG ultra-flexible silicone wire

- Passive cooling

---

## Kit Contents

The following items are included with each SPARK MAX Motor Controller

- 1 - SPARK MAX Motor Controller
- 1 - USB-A male to USB-C cable
- 1 - 4-pin JST-PH to CAN cable
- 1 - 4-pin JST-PH to single PWM cable
- 1 - PWM/CAN cable retention clip
- 1 - Data port protection cap

Special Thanks

We appreciate the assistance from the community for feedback, contributing, and testing the SPARK MAX, especially Team 195 The CyberKnights.

## SPARK MAX Specifications

The following tables provide the operating and mechanical specifications for the SPARK MAX motor controller.

> ⚠ DO NOT exceed the maximum electrical specifications. Doing so will cause permanent damage to the SPARK MAX and will void the warranty.

#### Main Electrical Specifications

> ⓘ Continuous operation at 60A may produce high temperatures on the heat sink. Caution should be taken when handling the SPARK MAX if it has been running at higher current level for an extended period of time.

> ⓘ If using a battery to power SPARK MAX, make sure the fully charged voltage is below 24V allowing for sustained operation. Some battery chemistries and configurations, including 6S LiPo packs, have a charge voltage above the maximum operating voltage for SPARK MAX.

### PWM Input Specifications

| | |
|---|---|
| † | Brushed: between A and B outputs at 100% duty. Brushless: A->B->C direction at 100% duty. |
| †† | Neutral corresponds to zero output voltage (0 V) and is either braking or coasting depending on th current idle behavior mode. |
| ††† | Brushed: between A and B outputs at 100% duty Brushless: C->B->A direction at 100% duty. |
| ‡ | If a valid pulse isn't received within the timeout period, the SPARK MAX will disable its output. |
| ‡‡ | Input deadband is added to each side of the neut pulse width. Within the deadband, output state is neutral. The deadband value is configurable usin the REV Hardware Client or through the CAN interface. |

### Data Port Specifications

| Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|
| Digital input voltage range † | 0 | - | 5 | V |
| Digital input-high voltage † | 1.85 | - | - | V |
| Digital input-low voltage † | - | - | 1.36 | V |
| Analog input voltage range †† | 0 | - | 3.3 | V |
| Analog input (12bit) | - | 81 | - | µV |
| 5V supply current (I5V) ‡ | - | - | 100 | mA |
| 3.3V supply current (I3.3V) | - | - | 30 | mA |
| Total supply current (I5V + I3.3V) | - | - | 100 | mA |

| | |
|---|---|
| | |
| † | See Data Port for more details on the digital pins on the Data Port. |
| †† | See Data Port for more details on the analog pin the Data Port. |
| ‡ | The 5V supply is shared between the Data Port and Encoder Port. |

**Encoder Port Specifications**

| Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|
| Digital input voltage range † | 0 | - | 5 | V |
| Digital input-high voltage † | 1.85 | - | - | V |
| Digital input-low voltage † | - | - | 1.36 | V |
| Analog input voltage range †† | 0 | - | 3.3 | V |
| 5V supply current (I5V) ‡ | - | - | 100 | mA |
| 3.3V supply current (I3.3V) | - | - | 30 | mA |
| Total supply current (I5V + I3.3V) | - | - | 100 | mA |

| | |
|---|---|
| | |
| † | See Encoder Port for more details on the digital pins on the Data Port. |

| | |
|---|---|
| †† | See Encoder Port for more details on the analog pin on the Data Port. |
| ‡ | The 5V supply is shared between the Data Port and Encoder Port. |

**Mechanical Specifications**

| Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|
| Body length | - | 70 | - | mm |
| Body width | - | 35 | - | mm |
| Body height | - | 25.5 | - | mm |
| Weight | - | 113.3 | - | g |
| Power and motor wire gauge | - | 12 | - | AWG |
| Power and motor wire length | - | 15 | - | cm |

## Getting Started with SPARK MAX

The SPARK MAX is a motor controller that can control both Brushed DC and Brushless DC motors. Out of the box, the MAX defaults to its Brushless Mode and is ready to drive a NEO Brushless Motor with its PWM interface. Included in this section are the basic steps to get a motor spinning using the REV Hardware Client as well as information on how to configure your SPARK MAX.

> ⓘ We recommend following this guide in its entirety at least once to understand the key features of the SPARK MAX. This guide can also serve as a fallback in case of any issues faced.

---

## Before You Start

Before following this guide, the REV Hardware Client should be Installed before continuing. The client is the best way to verify that the device is configured correctly, and is ***required*** before using the CAN interface.

# Wiring the SPARK MAX

## Required Materials

- 12V Battery
- 120A Circuit Breaker
- Power Distribution Panel
- SPARK MAX
- Brushed or Brushless Motor
- USB Type-C Cable

---

## Prepare the Components

### Test Bed

Using a test bed is an easy way to get started with using the SPARK MAX and verify connections and code. For the initial bring up of the SPARK MAX a test bed with a single SPARK MAX, a brushless or brushed motor, and a properly wired Power Distribution Panel with breaker is recommended.

### Electrical Connections

The power and motor wires are permanently connected to the SPARK MAX and are not replaceable. So take care not to cut these wires too short. It is highly recommended to install connectors on these wires to simplify both the power and motor connections. A common connector used for this purpose is the Anderson Power Pole connector. Follow our Anderson Power Pole guide for tips on how to properly crimp these connectors.

> ⓘ Make sure power is disconnected or turned off before making any electrical connections on your test bed or robot.

Connect the integrated SPARK MAX power leads labeled V+ (red) and V- (black) to an available channel on the Power Distribution Panel. If you need to extend the length of the integrated wires, it is recommended to use 12AWG wire or larger (lower gauge number).

---

## Motor Connections

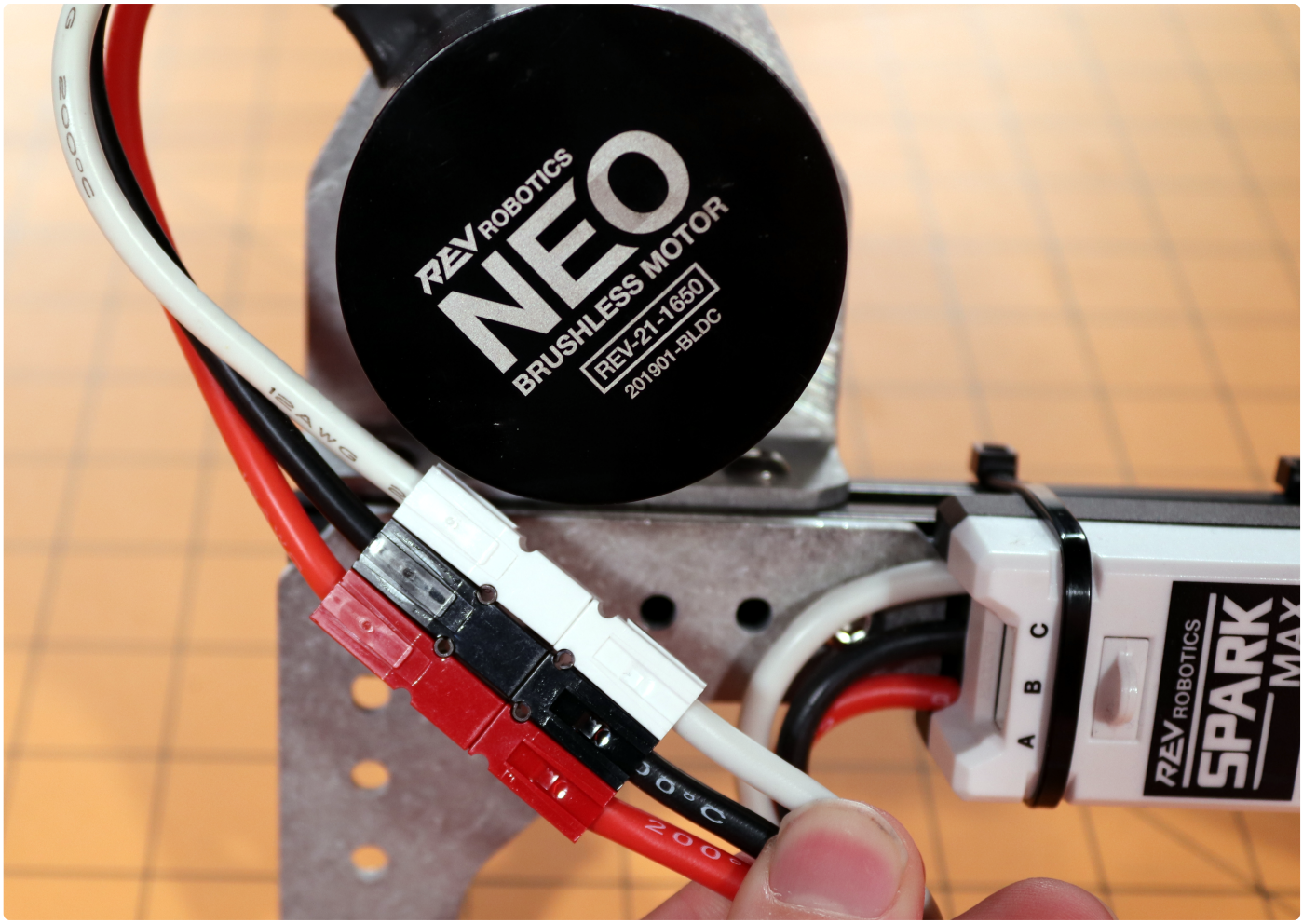
The first step is determining the type of motor you wish to connect. The SPARK MAX supports two types of motors: brushed DC and brushless DC. An easy way to determine the motor type is to look at the number of

primary (larger) motor wires. Brushed motors only have 2 primary motor wires, while brushless motors have 3 primary wires and additional s

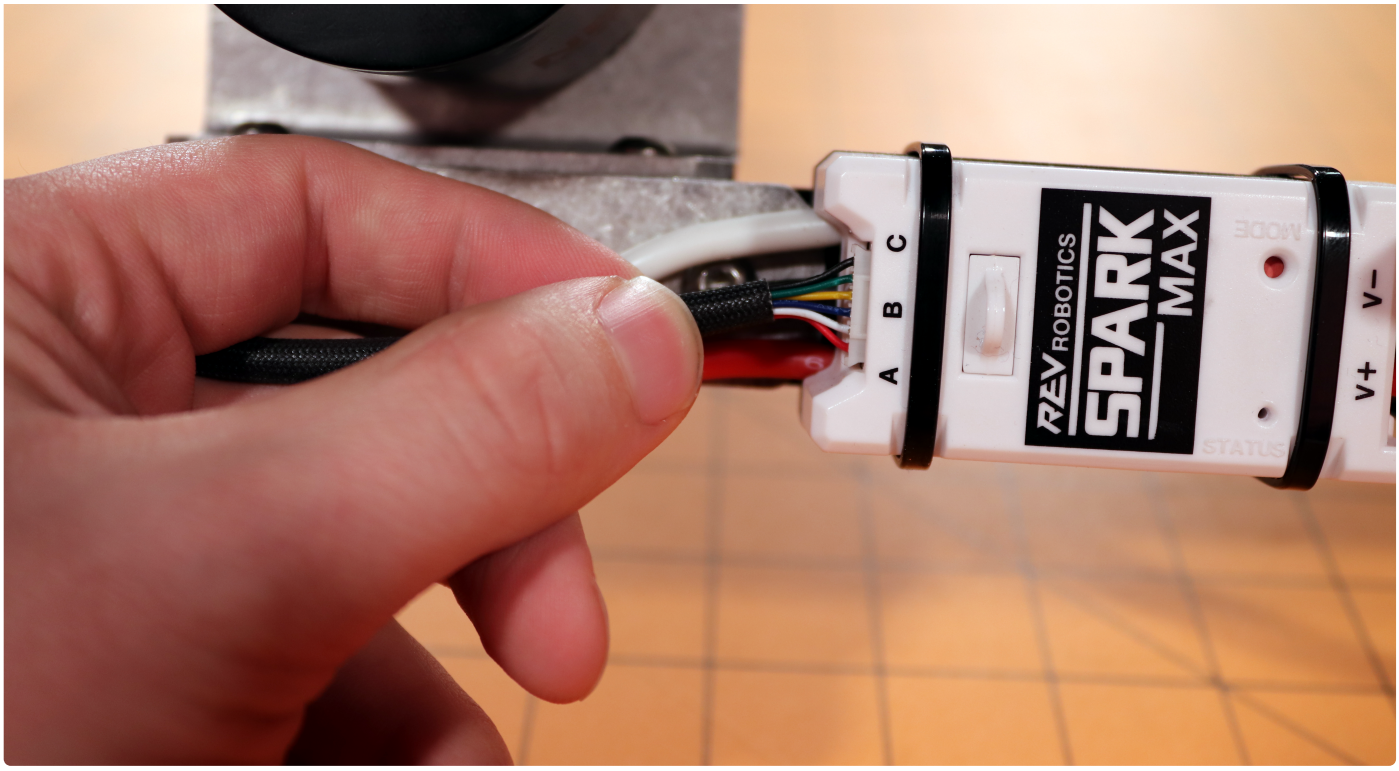
| Brushless | Brushed |
| --- | --- |
| NEO | CIM |
| NEO 550 | Mini CIM |
| | RS-775 |
| | RS-550 |
| | BAG |
| | etc. |

**NEO Brushless Motor Connections**

Connect the three motor wires; red, black, and white, to the matching SPARK MAX output wires labeled A (red), B (black), and C (white).




Next connect the NEO or NEO 550's encoder cable to the port labeled ENCODER just above the output wires.

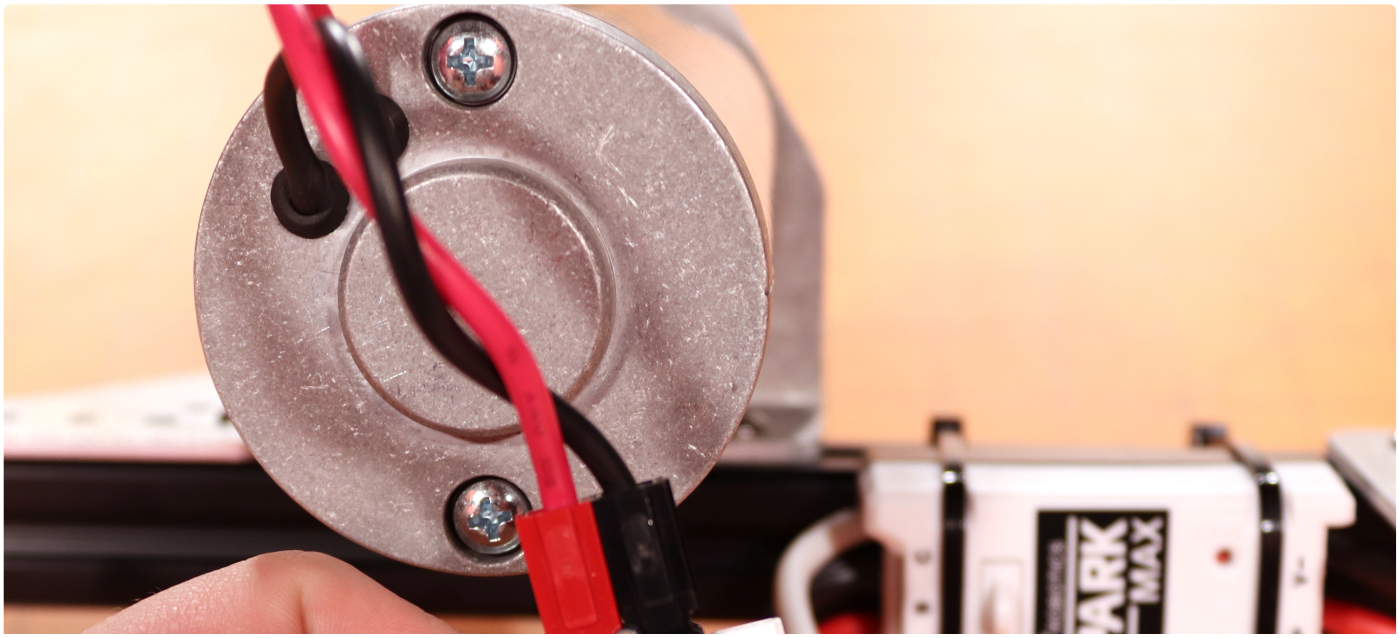



> The encoder sensor cable is ***required*** for the operation of brushless motors with SPARK MAX. The motor will not spin without it.

**Brushed DC Motor Connections**

Connect the two motor wires, M+ (red) and M- (black), to the SPARK MAX output wires labeled A (red) and B (black).

The third output wire, labeled C (white), is not used when driving a brushed motor and should be secured and insulated. We recommend tying it back with a zip-tie and covering the end with a piece of electrical tape. Do not cut this wire in case you wish to use a brushless motor in the future. In the example below the extra unused motor wire is insulated by the white connector and secured in the block.



## Verify Connection

Carefully check all connections before continuing and verify that all colors match. The SPARK MAX can be permanently damaged if the power connection is reversed.

> Leave the CAN cable disconnected for now, we will wiring this up later.

## Make it Spin!

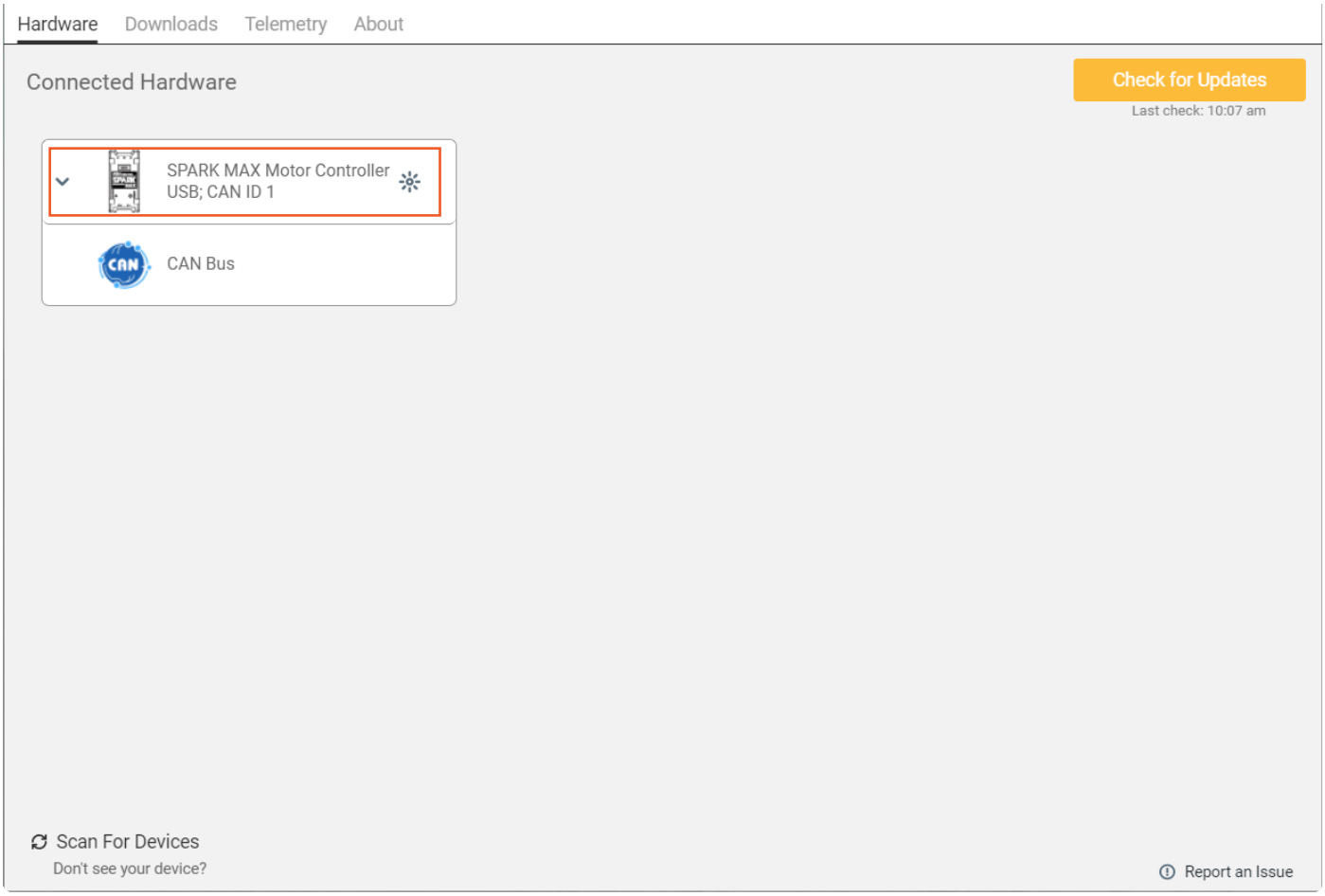
## Power On

Now that the device is wired, and the connections carefully checked, power on the robot. You should see the SPARK MAX slowly blinking its for a new device the color will be Magenta. If the LED is dark, or you see a different blink pattern, refer to the Status LED guide for troubleshooting.

> If you are using a brushed motor, you may see a sensor error. This is expected until you configure the device to accept a brushed motor in the following steps.

## Connect to the SPARK MAX

Plug in the USB cable and start the REV Hardware Client. Select the SPARK MAX from the Connected Hardware

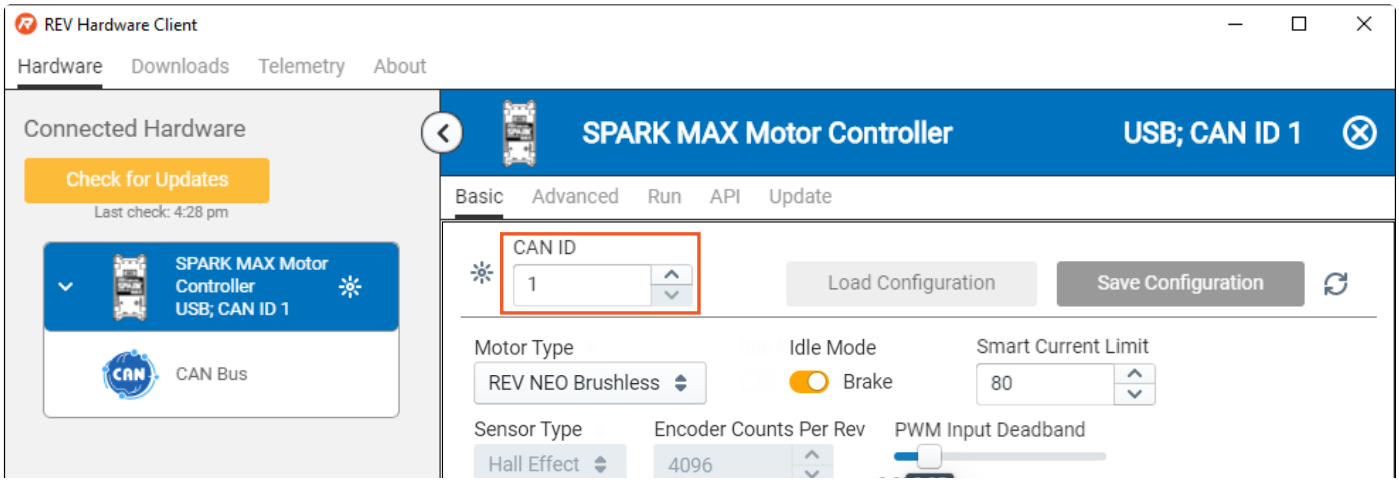


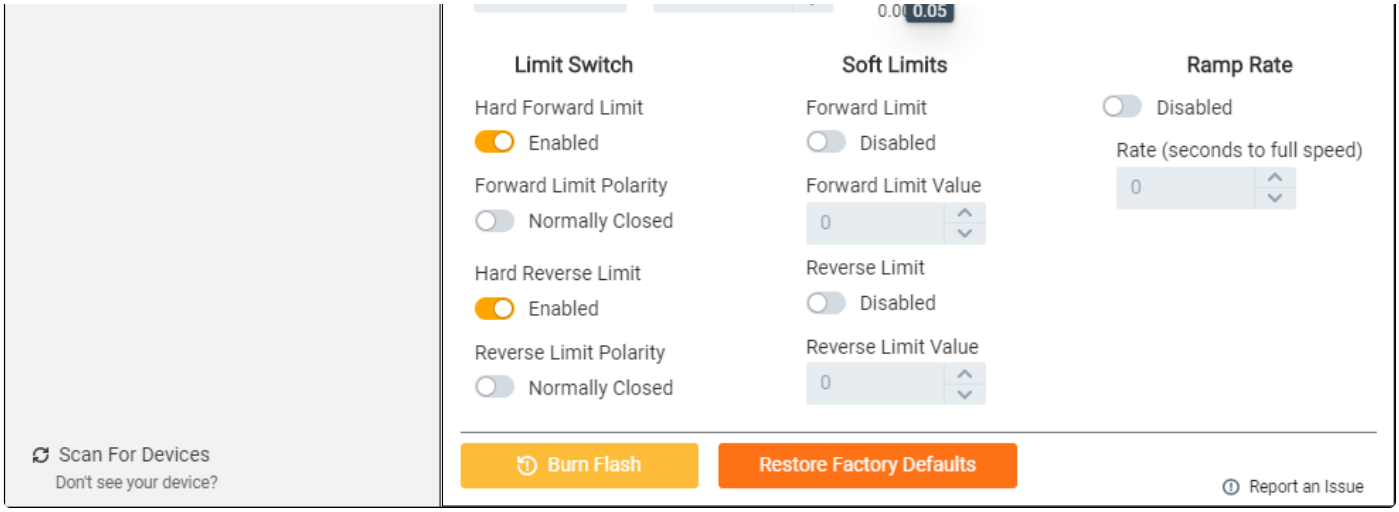

> If you can not see the SPARK MAX, make sure that the SPARK MAX is not being used by another application, such as the REV SPARK MAX Client
> . Then unplug the SPARK MAX from the computer and plug it back in.

---

## Basic Setup and Configuration

Before any parameters can be changed, you **must** first assign a unique CAN ID to the device. This can be any number between 1 and 63. After setting a unique CAN ID, the user interface will refresh and allow you to change other parameters.

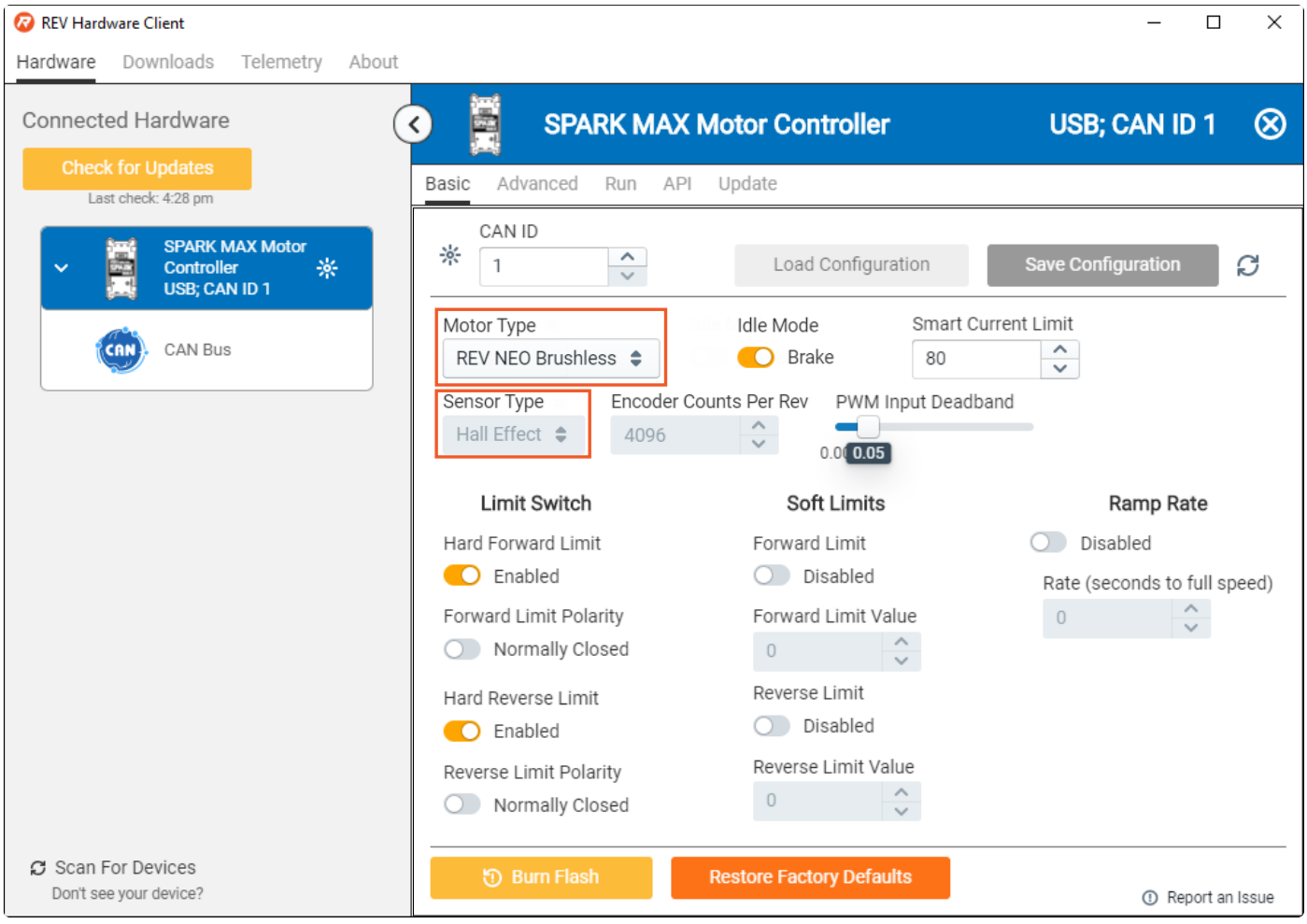





> Eventually you may set up a CAN network on your test bench or robot. Be sure each device on the network has a unique CAN ID. It is helpful to label each device with its ID number to aid in troubleshooting.

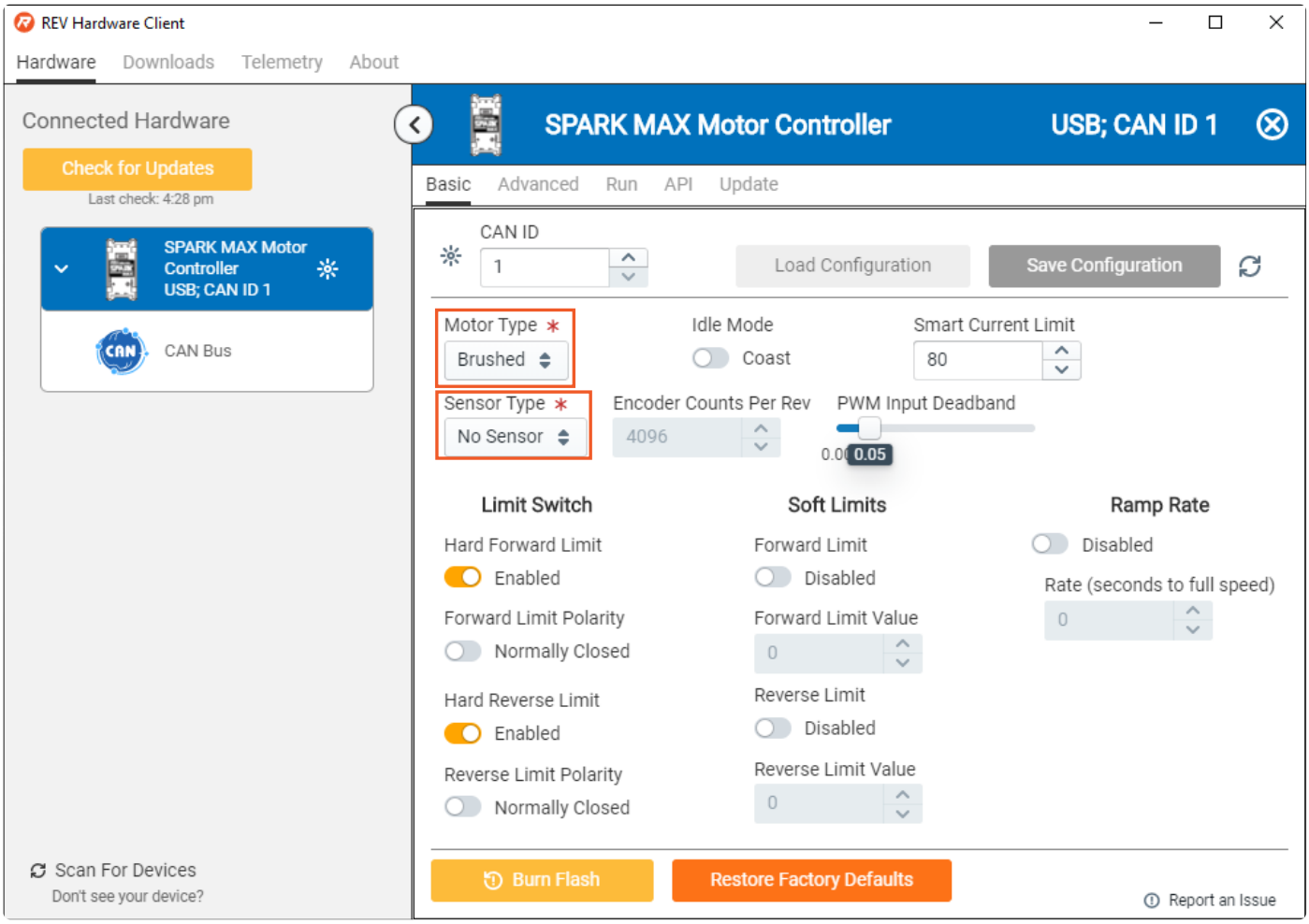
### Set the Motor Type

If you are using a NEO or NEO 550, verify that the motor type is set to **REV NEO Brushless**, Sensor Type is **Hall Effect**, and the LED is blinking Magenta or Cyan.



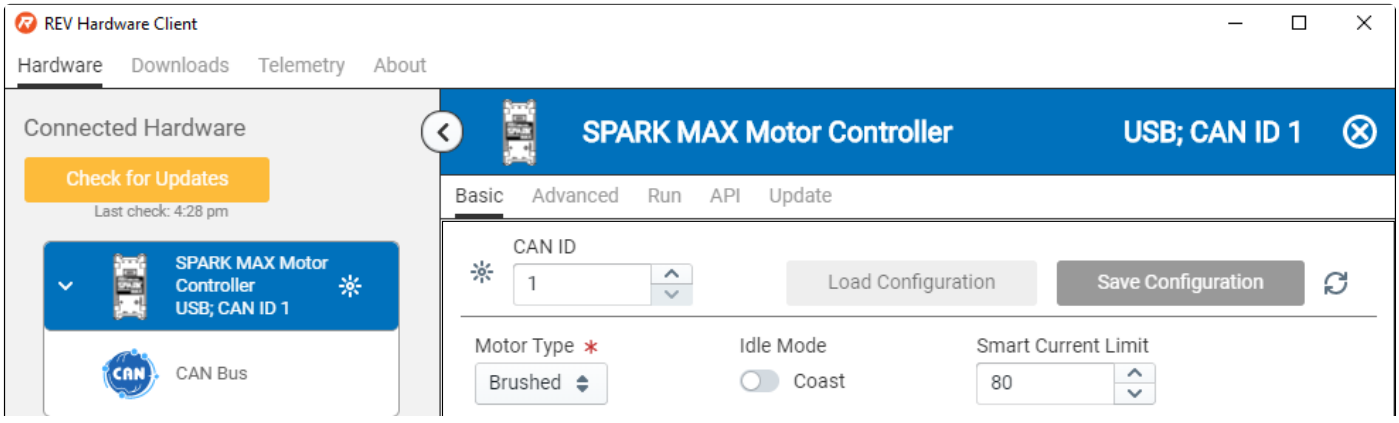
> If you see a *Sensor Fault* blink code, make sure the encoder cable is plugged in completely.

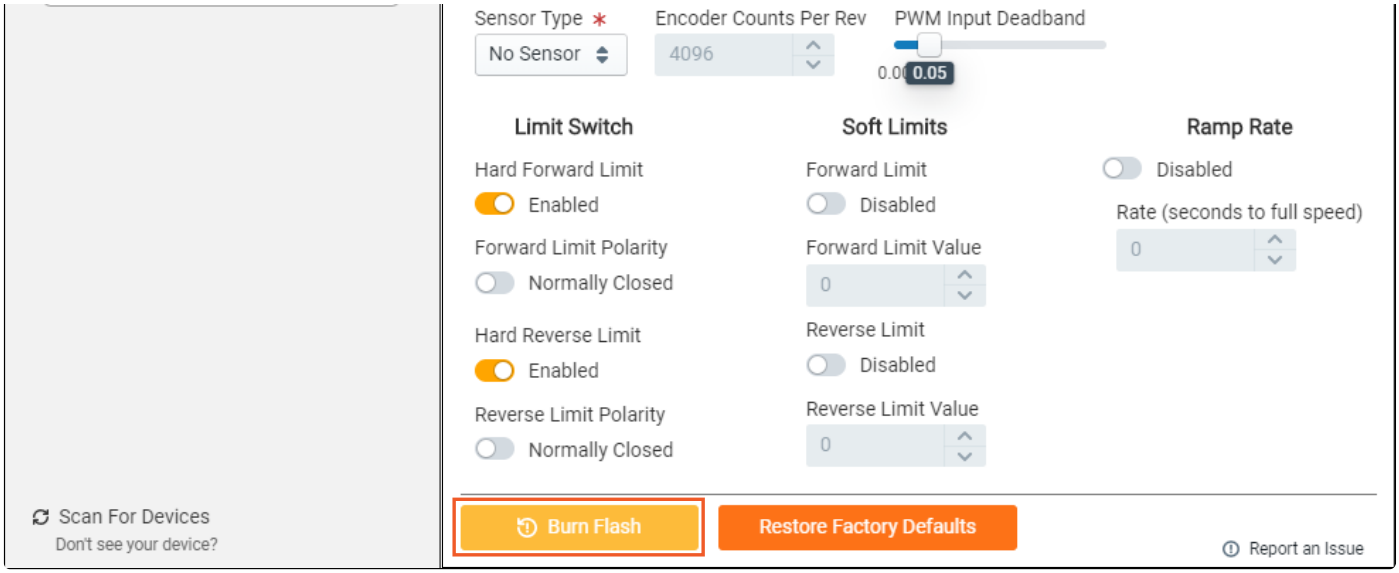
If you are running brushed motor, set the motor type to **Brushed**, sensor type as **No Sensor**, and verify that the LED is blinking Yellow or Blue.




## Save the Settings

For the SPARK MAX to remember its new configuration through a power-cycle, the settings must be saved. To do this, press the *Burn Flash* button on the bottom of the page. It will take a few seconds to save, indicated by the loading symbol on the button.






Any settings saved this way will be remembered when the device is powered back on. You can always restore the factory defaults if you need to reset the device.

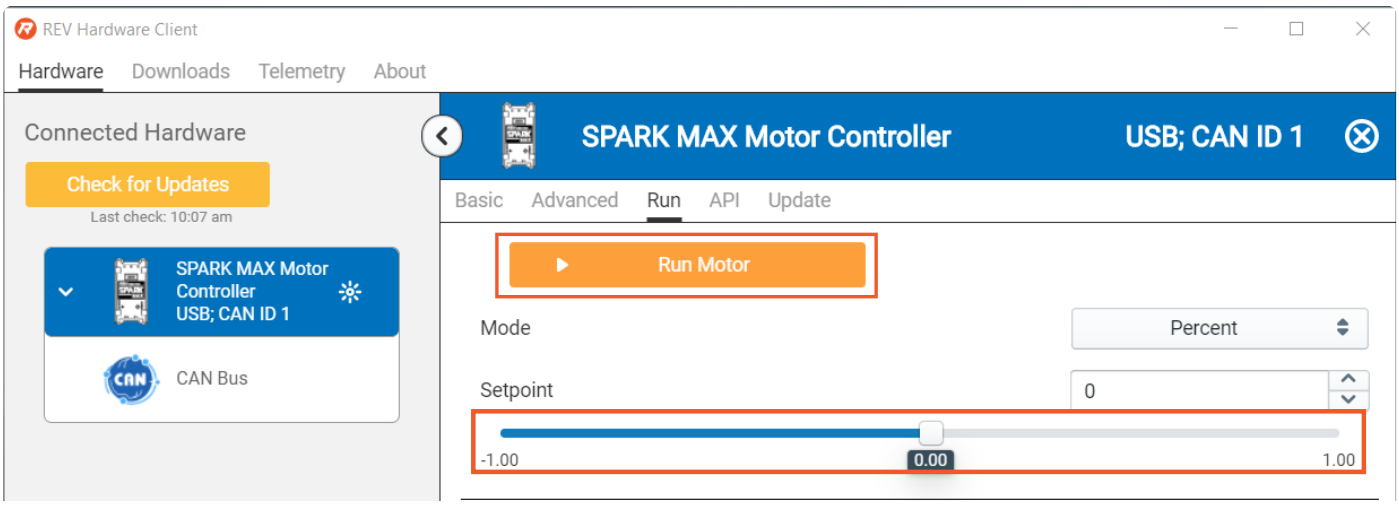
---

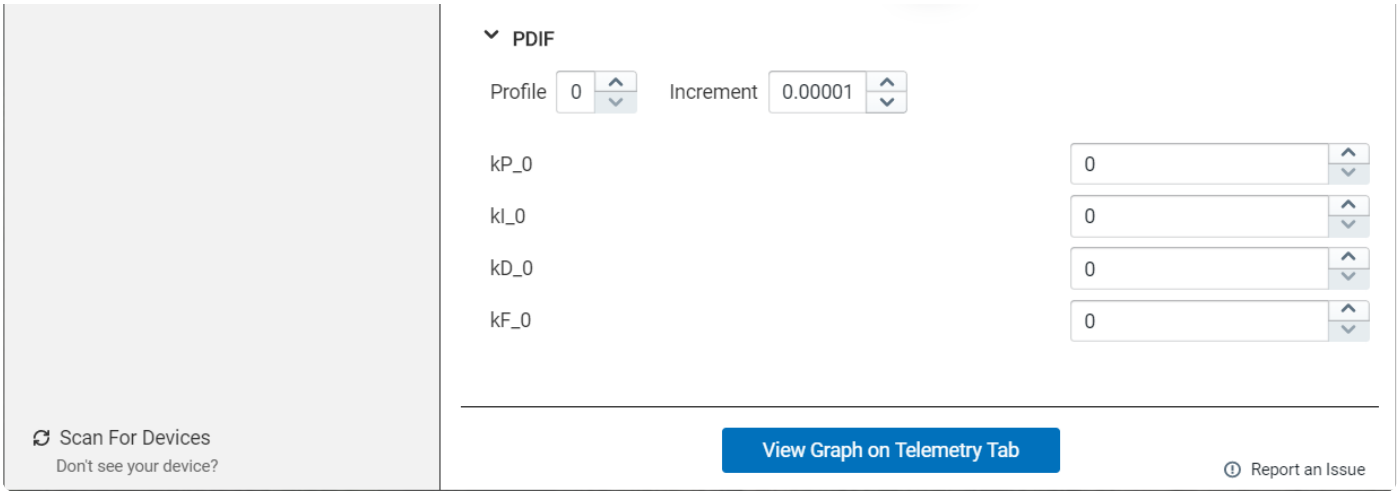
## Spin the Motor

> ⚠ Before running any motor, make sure all components are in a safe state, that the motor is secured, and anyone nearby is aware. FRC motors are very powerful and can quickly cause damage to people and property.

> ⓘ Keep the CAN cable disconnected throughout the test. For safety reasons, the SPARK MAX Client will not run the motor if the roboRIO is connected. If the roboRIO was connected, power cycle the SPARK MAX.

To spin the motor, go to the Run tab, keep all of the default settings and press *Run Motor.* The *setpoint* is 0 by default, meaning that the motor is being commanded to **idle** (0% power). When you press *Run* you should see the LED go from slow blinking to solid, indicating that the motor is idling.






**Slowly** ramp the setpoint slider up. The motor should start to spin and you should see a green blink pattern proportional to the speed you have set to the motor. Slowly ramp the slider down. The motor should spin in reverse, and you should see a red blink pattern proportional to the speed you have set to the motor.

If you are unable to spin the motor, visit our troubleshooting guide.

## Basic Configurations

SPARK MAX has many operating modes that can be configured through its CAN and USB interfaces. Additionally, the following basic operating modes can be configured with the MODE button located on the top of the SPARK MAX:

- Idle Behavior: Brake/Coast
- Motor Type: Brushed/Brushless

**Mode configuration must be done with power applied to the SPARK MAX.**

> Configuring the idle behavior and motor type using the mode button is a quick way to set up a SPARK MAX without using a computer. This is most useful when you are controlling the SPARK MAX through its PWM interface or when testing the affect of Braking or Coasting on a mechanism.

### Idle Behavior

Whenever the SPARK MAX is receiving a neutral signal (no motor movement) or no signal at all (robot disabled), it can either brake the motor or let it coast. When in Brake Mode, MAX will short the motor wires to each other, electrically braking the motor. This slows the motor down very quickly if it was spinning and makes it harder, but not impossible to back-drive the motor when it is stopped.

- With power turned on, press and release the MODE button to switch between Brake and Coast Mode.
- The STATUS LED will indicate which mode it is in. See the STATUS LED Colors and Patterns section for more information.

### Motor Type

It is very important to have the SPARK MAX configured for the appropriate motor type.

> ⚠ Operating in Brushed Mode with a brushless motor connected will permanently damage the motor!

With power turned on, press and hold the MODE button for approximately 3 - 4 seconds.

- The STATUS LED will change and indicate which motor type is selected. See the STATUS LED Colors and Patterns section for more information.
- Release the MODE button.

## Status LED Patterns

SPARK MAX will indicate important status information on its multi-colored STATUS LED located on the top of its case. The following tables shows each state and the corresponding LED color pattern.

### Standard Operation

| **Operating Mode** | Idle Mode | State | Color/Pattern | Graphic |
|---|---|---|---|---|
| Brushed | Brake | No Signal | Blue Blink |  |
| | | Valid Signal | Blue Solid |  |
| | | | | |
| | Coast | No Signal | Yellow Blink |  |
| | | Valid Signal | Yellow Solid |  |
| | | | | |
| Brushless | Brake | No Signal | Cyan Blink |  |
| | | Valid Signal | Cyan Solid |  |
| | | | | |

| | | | | |
|---|---|---|---|---|
| | Coast | No Signal | Magenta Blink |  |
| | | Valid Signal | Magenta Solid |  |
| | | | | |
| Partial Forward | | | Green Blink |  |
| Full Forward | | | Green Solid |  |
| | | | | |
| Partial Reverse | | | Red Blink |  |
| Full Reverse | | | Red Solid |  |
| | | | | |
| Forward Limit | | | Green/White Blink |  |
| Reverse Limit | | | Red/White Blink |  |

**Identification, Updating, and Recovery**

| Mode | Color/Pattern | Graphic |
|---|---|---|
| Device Identify | White/Magenta Fast Blink |  |
| CAN Bootloader Firmware Updating | White/Yellow Blink (v1.5.0)<br>Green/Magenta Blink (v1.4.0) |  |
| CAN Bootloader Firmware Retry | White/Blue Blink |  |
| USB DFU (Device Firmware Update) | Dark (LED off) |  |
| Recovery Mode | Dark (LED off) |  |

**Fault Conditions**

| **Fault Conditions** | Condition | Color/Pattern | Graphic |
|---|---|---|---|
| 12V Missing | The motor will not drive if powered only by USB. This blink code warns the user of this condition. | Orange/Blue Slow Blink |  |
| Sensor Fault | This can occur if the sensor type is misconfigured, the sensor cable is not plugged in or damaged, or if a sensor other than the motor sensor is plugged in. | Orange/Magenta Slow Blink |  |
| Gate Driver Fault | A fault reported by the core internal electronic circuitry. If this code persists after power cycling the controller, contact REV. | Orange/Cyan Slow Blink |  |
| CAN Fault | The CAN fault will be shown after the first time the device is plugged into the CAN port and a fault later occurs. Check your CAN wiring if you see this fault. | Orange/Yellow Slow Blink |  |
| Corrupt Firmware (recover using Recovery Mode) | Firmware failed to load. | Dark (LED off) |  |

## Troubleshooting

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.

---

## General Troubleshooting Tips

The key to effective troubleshooting is isolating the issue. Many issues can show the same symptom, so eliminating failure points one at a time is critical to finding the root cause.

#### Rule Out Issues By Isolation

If possible, try to eliminate a section of the system when troubleshooting. For example:

- Rule out a code or control wiring issue:
  - Use the REV Hardware Client to run the SPARK MAX over USB.
    - **Please be aware of the CAN lockout feature of the SPARK MAX.** If it has been connected to the roboRIO's CAN bus, a safety feature within the SPARK MAX will lock out USB communication. Disconnecting from the CAN bus and power-cycling the MAX will release the lock.
  - If this is your first time running the REV Hardware client, see the Getting Started with the REV Hardware Client for a tour of the software and its features.
- Rule out a code issue:
  - Create a simple test program using our SPARK MAX Example Code.
- Rule out a mechanical issue:
  - Remove the motor from the mechanism or use a different, free spinning motor.

#### Use the Driver Station

An extremely useful set of tools can be found on the Driver Station:

- Use the Driver Station Log File Viewer
  - Look at the PDP channel current draw:
  -
    - Higher than expected current on a channel can indicate both mechanical and electrical issues.
  - Look at the battery voltage:
    - Large dips in the battery voltage around the time of an issue can indicate battery health issues that cause brownouts.
- Use the CAN/Power Tab
  - Look at the CAN Bus Utilization.
  - Look at CAN Faults.
  - Look at Comms Faults:
    -

- Comms faults can affect the SPARK MAX. If it loses communication with the roboRIO, it will go to its safe disabled state. This can look like a momentary glitch in a motor spinning if the comms faults are infrequent and irregular.

### Use the APIs

It is also very useful to log or plot operating values internal to the SPARK MAX. These values can be accessed using the SPARK MAX APIs. Useful values to log:

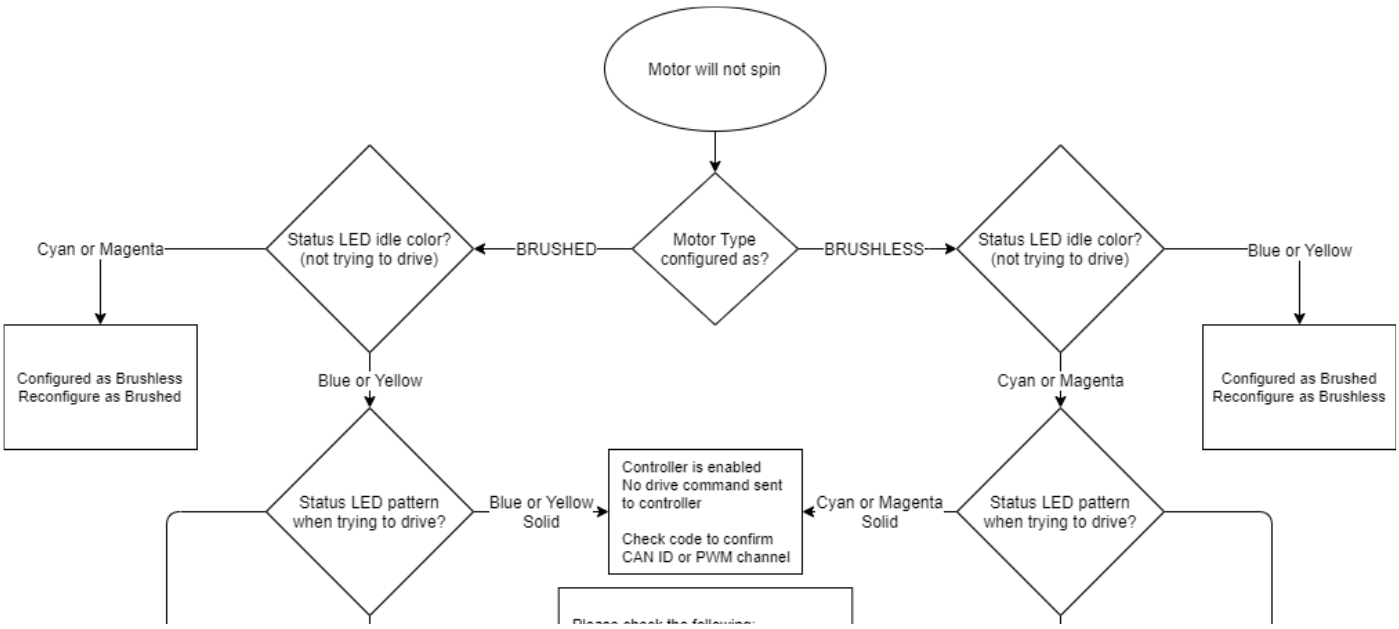
- *getAppliedOutput()*
  - This value will show what the SPARK MAX is actually applying to the motor output. This can illuminate issues with closed loop control tuning.
- *getOutputCurrent()*
  - This value will show the output current going to the phases of the motor. Output current won't always be the same as the Input current measured by the PDP. Knowing the output current is useful to diagnose current-limit issues if motors are overheating.
- *getBusVoltage()*
  - A way to measure the input voltage right at the controller.
- *getStickyFaults()*
  - A sticky fault indicates if a fault has occurred since the last time the faults were reset. Checking these can provide a lot of insight into what the controller is experiencing.

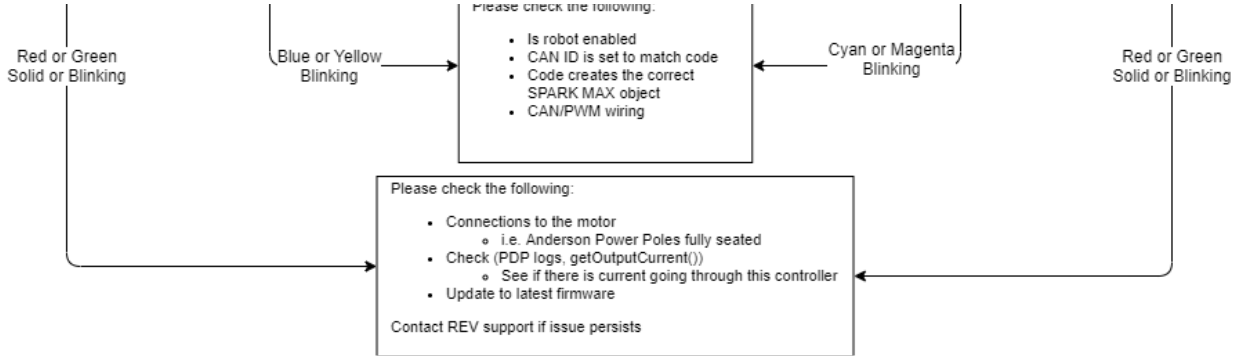
---

## Common Faults and Issues

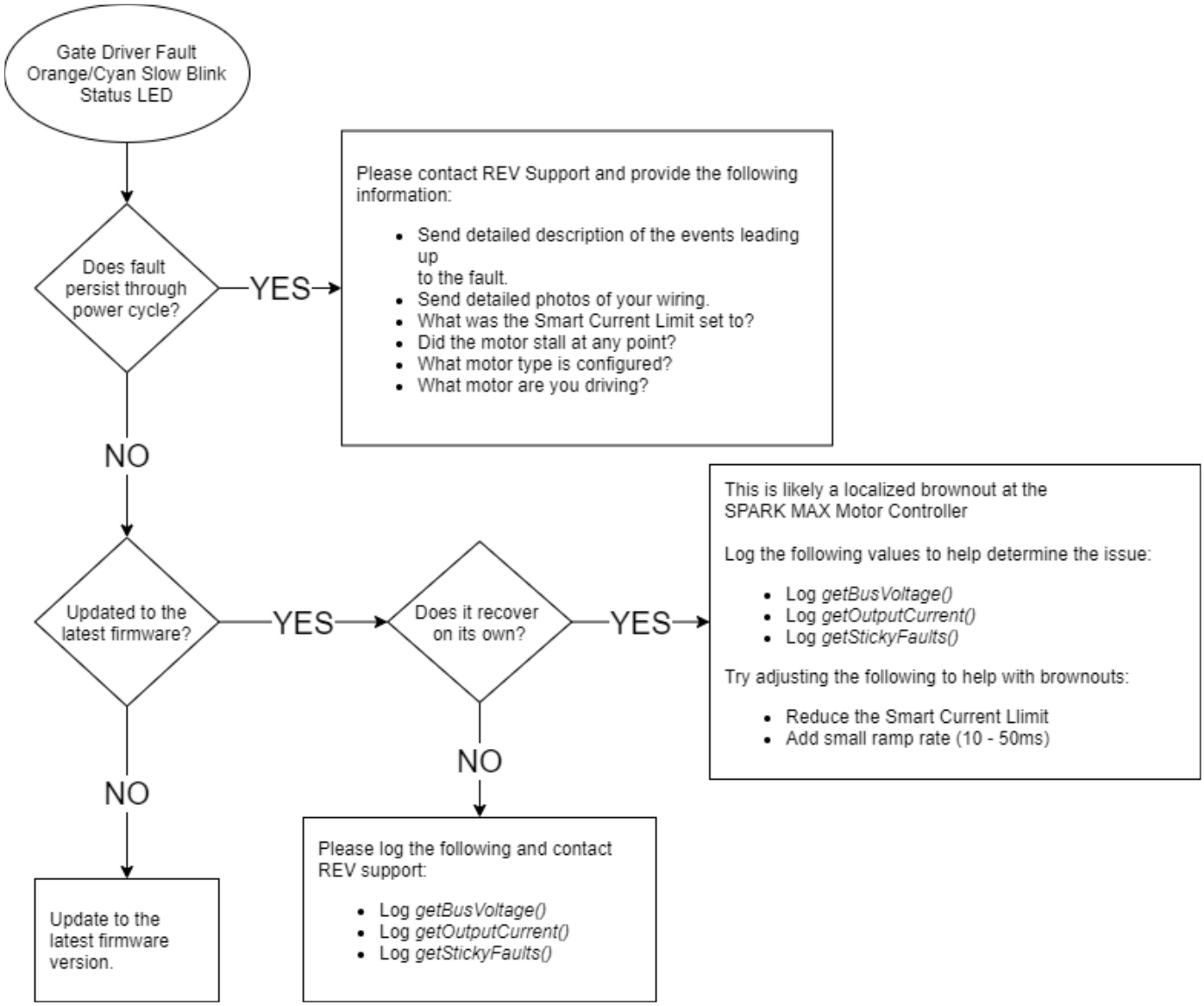
Below you will find some troubleshooting steps for some common faults and issues related to operating the SPARK MAX.

### Motor Not Spinning

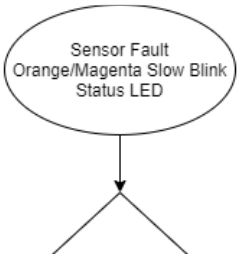


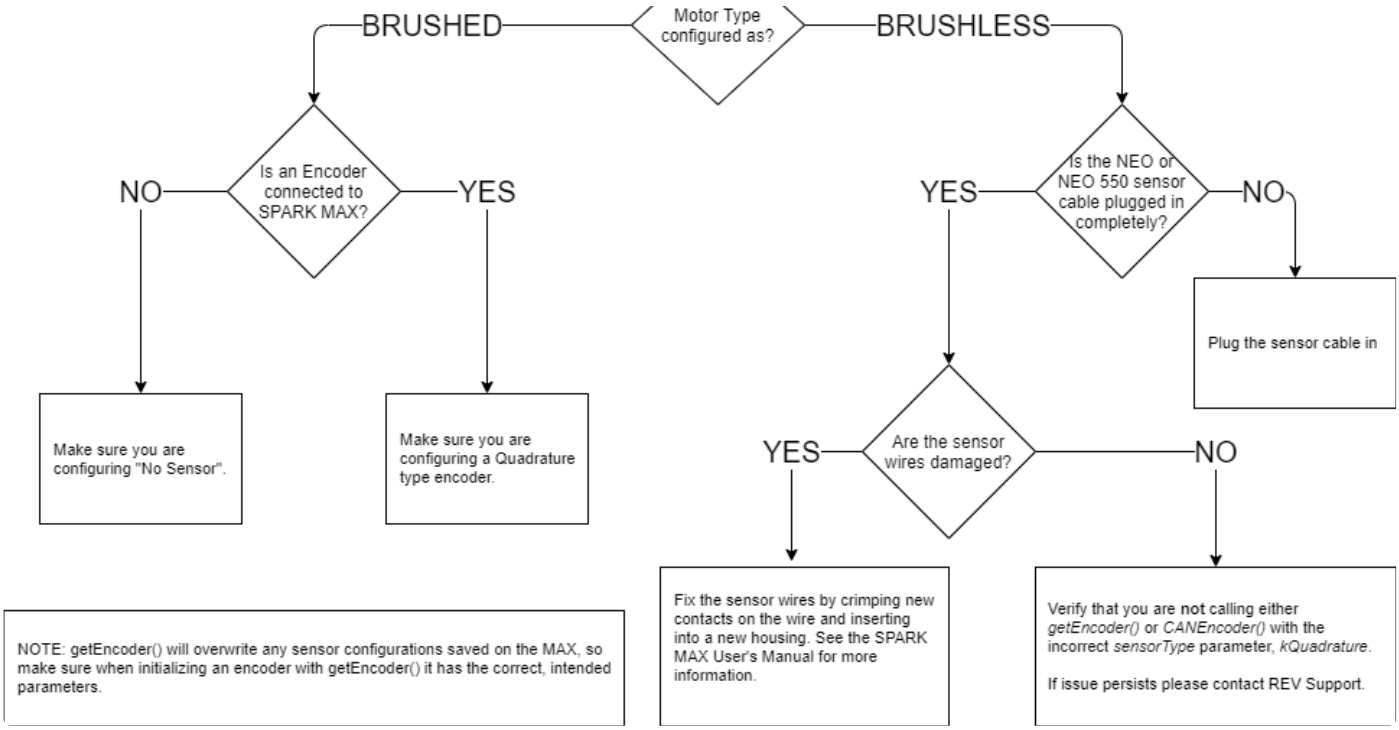



## Gate Driver Fault




## Sensor Fault

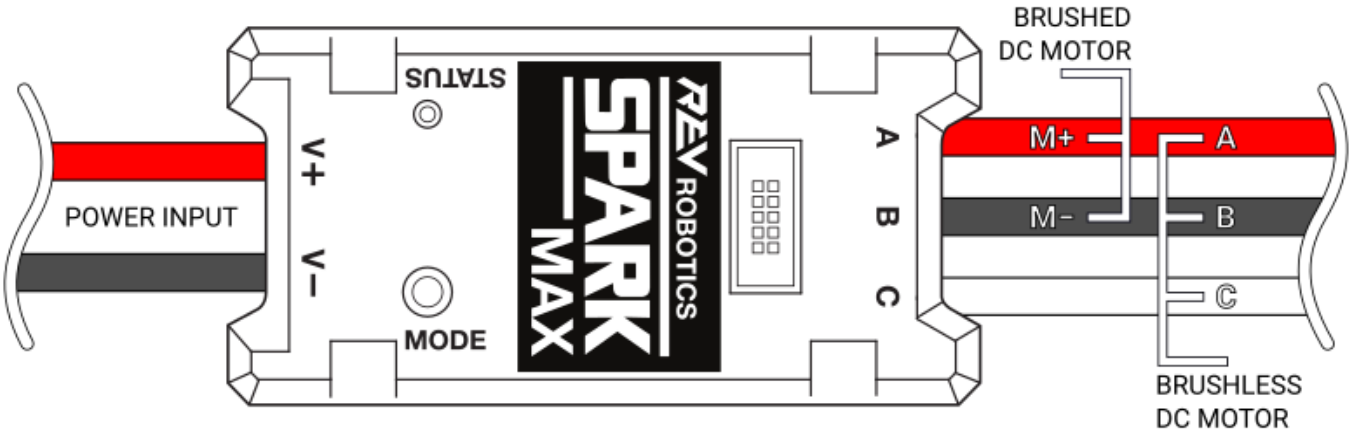





# Feature Description

## Power and Motor Connections

SPARK MAX is designed to drive 12V brushed and brushless DC motors at currents up to 60A continuously. Power and motor connections are made through the two sets of wires built into the SPARK MAX. The wires are 12AWG ultra-flexible silicone-coated wire. Each wire runs approximately 15cm from the end faces of the controller. Be sure to take care when cutting and stripping the wires as not to cut them too short. The figure below shows these connections in detail.




SPARK MAX Motor Controller Power Connections

As with any electrical component, make all connections with the power turned off. Connecting the SPARK MAX to a powered system may result in unexpected behavior an may pose a safety risk.

---

## Motor Output

Motor output wires are labeled as A, B, and C with red, black, and white wires. Brushed motors must be connected to the A and B wires, while brushless motors must be connected to all three. **It is critical that the order of the brushless motor wires match the SPARK MAX or the motor will not spin and could be damaged.** Additional details are below.

Motor Connections

| Motor Type | Motor Wires | SPARK MAX Wires |
|---|---|---|
| Brushed | Red / M+ | Red / A |
| | Black / M- | Black / B |
| | | |
| Brushless (NEO Brushless Motor) | Red / A | Red / A |
| | Black / B | Black / B |
| | White / C | White / C |

SPARK MAX cannot detect which motor type it is connected to. Be sure to configure the SPARK MAX to run the type of motor you have connected. See the Motor Type - Brushed/Brushless Mode section for more details on configuring the appropriate motor type.

---

## Power Input

Power input wires are labeled as V+ and V- with red and black wires. The SPARK MAX is intended to operate in a 12 V DC robot system, however it is compatible with any DC power source between 5.5 V and 24 V.

⚠ DO NOT reverse V+ and V- or swap motor and power connections. Doing so will cause permanent damage to the SPARK MAX and will void the warranty.

⚠ DO NOT exceed the maximum supply voltage of 30V. Doing so will cause permanent damage to the SPARK MAX and will void the warranty.

When using high current motors, it is recommended to use a power source that is capable of handling large surge currents, e.g. a 12V lead-acid battery. If the supply voltage drops below 5.5V the SPARK MAX will brown out, resulting in unexpected behavior. It is also highly recommended to incorporate a fuse or circuit-

breaker in series with the SPARK MAX between it and the power source to prevent exceeding the maximum current rating.

> ⚠ DO NOT exceed the maximum current ratings of 60A or 100A for 2 seconds. Doing so will cause permanent damage to the SPARK MAX and will void the warranty.

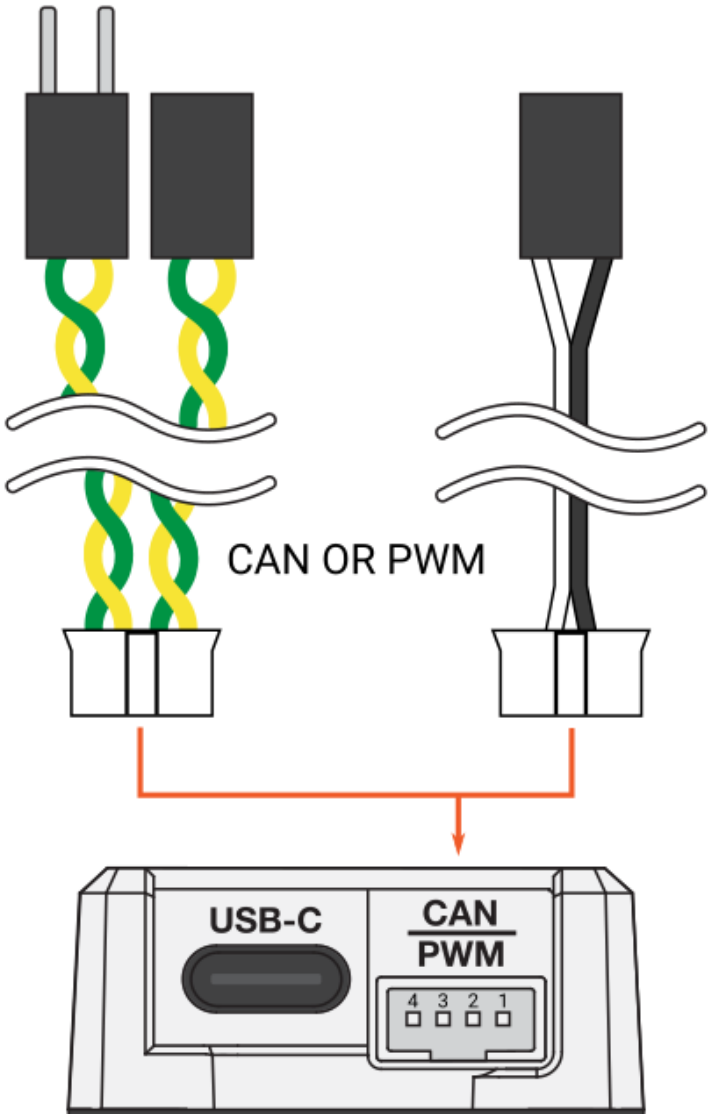
## Control Connections

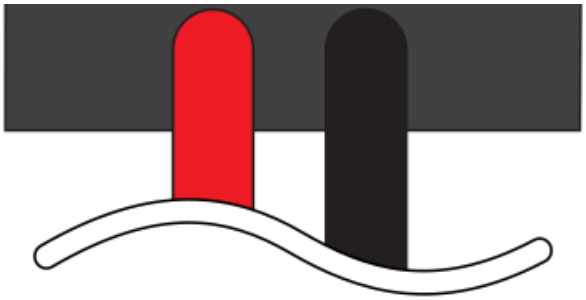
The SPARK MAX can be controlled by three different interfaces, servo-style PWM, controller area network (CAN), and USB. The following sections describe the physical connections to these interfaces in detail. For details on the operation and protocols of the PWM, CAN, and USB interfaces, please see the section on Control Interfaces.

---

## CAN/PWM Port

The CAN/PWM Port is located on the power input side of the SPARK MAX. This port can be connected to either a servo-style PWM signal or a CAN bus with other devices. Connector details can be found below.





**CAN/PWM Port Connector Information**

| Connector Pin | CAN Function | PWM Function |
|---|---|---|
| 1 | CAN High | Signal |
| 2 | CAN Low | Ground |
| 3 | CAN High | Signal |
| 4 | CAN Low | Ground |

**Matting Connector Information**

| Description | Manufacturer | Part Number | Vendor | Vendor P/N |
|---|---|---|---|---|
| JST-PH 4-pin Housing | JST | PHR-4 | DigiKey | 455-1164-ND |
| JST-PH Contact | JST | SPH-002T-P0.5L | DigiKey | 455-2148-1-ND |
| Recommended Crimping Tool | IWISS | SN-2549 | Amazon | SN-2549 |

Identical-function pins are electrically connected inside the SPARK MAX, therefore the CAN daisy-chain is completed internally and any two signal and ground pairs can be used for PWM.

---

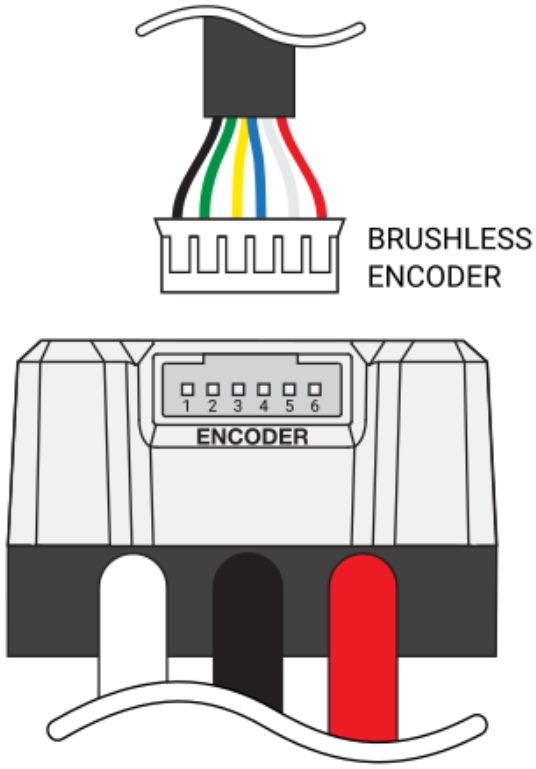
## USB-C Port

The USB-C Port is located on the power input side of the SPARK MAX. It supports USB 2.0 and 5V power for the SPARK MAX's internal microcontroller. While you can configure the SPARK MAX without main power, you will not be able to spin a motor.

## Encoder Port

Located on the motor output side of the SPARK MAX is a 6-pin Encoder Port. This port is designed to accept the built-in hall-encoder from the NEO Brushless Motor, but it can also connect to other external encoders when running in Brushed Mode. The connector details can be found below.

> The SPARK MAX can be configured to run in Alternate Encoder Mode, which reconfigures the Data Port on the top of the controller to accept an alternative quadrature encoder in addition to the Encoder Port.




Encoder Port Connector Information

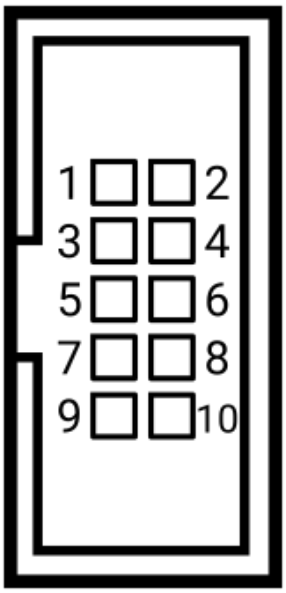
| Connector Pin | Pin Type | Pin Function |
|---|---|---|
| 1 | Power | Ground |
| 2 | Digital | Encoder C / Index |
| 3 | Digital | Encoder B |
| 4 | Digital | Encoder A |
| 5 | Analog | Motor Temperature |
| 6 | Power | +5V |

Mating Connector Information

| Description | Manufacturer | Part Number | Vendor | Vendor P/N |
|---|---|---|---|---|
| JST-PH 6-pin Housing | JST | PHR-6 | DigiKey | 455-1162-ND |
| JST-PH Contact | JST | SPH-002T-P0.5L | DigiKey | 455-2148-1-ND |
| Recommended Crimping Tool | IWISS | SN-2549 | Amazon | SN-2549 |

## Data Port

Located on the top of the SPARK MAX, the Data Port allows for extra sensor input and future feature development. The connector details can be found below.




Data Port Connector Information

| Connector Pin | Pin Type | Pin Function |
|---|---|---|
| 1 | Power | +3.3V |
| 2 | Power | +5V |
| 3 | Analog | Analog Input |
| 4 | Digital | Forward Limit Switch Input |

| | | |
|---|---|---|
| 5 | Digital | Encoder B |
| 6 | Digital | Multi-function Pin |
| 7 | Digital | Encoder A |
| 8 | Digital | Reverse Limit Switch Input |
| 9 | Digital | Encoder C / Index |
| 10 | Ground | Ground |

Using the SPARK MAX Data Port Breakout Board (REV-11-1278) makes interfacing with the SPARK MAX Data Port easier.

---

## Limit Switch Inputs

SPARK MAX has two limit switch inputs that, when triggered, can independently prevent motion in both the forward and reverse directions. By default, when the pin for the corresponding direction is grounded, SPARK MAX will override any input commands for that direction and force the output into the neutral state. Input commands for the opposite direction will still be processed unless the corresponding limit signal is also triggered.

The default polarity is compatible with Normally Open (NO) style limit switches, who's contacts are shorted together when the switch is pressed. The Limit Switch Inputs can be configured for the opposite polarity using the USB or CAN interfaces. When configured for the opposite polarity, Normally Closed (NC), the limit will be triggered when the pin is left disconnected from ground. In other words, connecting the pin to ground will release the limit. The following table shows these configurations in detail:

Limit Switch Operation

| Connected Switch Type | Configured Limit Switch Polarity | Switch State | Limit Triggered (preventing motion) | Recommended Combinations |
|---|---|---|---|---|
| Normally Open | Normally Open (default) | Released | Normal operation | * |
| | | Pressed | Triggered | |
| | Normally Closed | Released | Triggered | |
| | | Pressed | Normal operation | |
| Normally Closed | Normally Open (default) | Released | Triggered | |
| | | Pressed | Normal operation | |
| | Normally Closed | Released | Normal operation | * |
| | | Pressed | Triggered | |

---

## Quadrature Encoder Input

The Quadrature Encoder Input on the Data Port is compatible with standard quadrature encoder signals, usually labeled as channel A, channel B, and Index. SPARK MAX shares these signals with the Encoder Port on the output side of the controller, therefore the Index signal is shared with the third brushless encoder signal C. When in Brushless Mode, these Data Port pins cannot be used with an external encoder. See Alternate Encoder Mode for information on how to configure the SPARK MAX to accept an alternative encoder source when running in Brushless Mode.
When in Brushed Mode, an external encoder can be connected through either the Data Port or the Encoder Port.

The SPARK MAX encoder signals are not pulled high internally. This is to ensure the maximum compatibility with different types of encoders.

---

## Analog Input

The Analog Port on the SPARK MAX can measure voltages up to 3.3V with 12-bit resolution. The SPARK MAX Data Port Breakout includes a 5V to 3.3V amplifier circuit so that 5V signals can be sensed with the Analog Input pin.

Analog input is supported on firmware versions 1.4.0 and newer.

---

## Multi-function Pin

This pin is reconfigured when the SPARK MAX is configured in Alternate Encoder Mode.

---

## Power Rails

The SPARK MAX Data Port can provide both 3.3V and 5V power to connected devices. Please check Data Port Specifications for details on the supply current capabilities of both rails.

---

## SPARK MAX Data Port Breakout Board - Features

The SPARK MAX Data Port Breakout Board (REV-11-1278) makes it easy to connect external sensors to the SPARK MAX Data Port.

- Solder pads for every Data Port pin
- Analog input 5V to 3.3V converter
  - Built-in amplifier maps 0V - 5V analog signals to the native 0V - 3.3V range of the SPARK MAX Analog Input
  - Configurable resistors can bypass the amplifier (move R3 to R4 position)

- Pass-through Data Port connector
  - Connect other sensors with data port compatible cables while using this breakout
- Mounts directly to SPARK MAX
  - No need for a data port cable
  - Securely mounts to the SPARK MAX zip-tie notches

# Operating Modes

## Motor Type - Brushed/Brushless Mode

Brushed and brushless DC motors require different motor control schemes based on the differences in their technology. It is possible to damage the SPARK MAX, the motor, or both if the appropriate motor type isn't configured properly. At the moment, the NEO Brushless Motor and the NEO 550 Brushless Motor are the only FRC-legal brushless motors compatible with the SPARK MAX, so choosing the correct operating mode should be straightforward.

Brushed or brushless motor types can be configured using the Mode Button, CAN, and USB interfaces.

---

## Configuration with Mode Button

Follow the steps below to switch motor types with the Mode Button. It is recommended that the motor be left disconnected until the correct mode is selected.

> ⓘ Use a small screwdriver, straightened paper clip, pen, or other small implement to press the button. Do not use any type of pencil as the pencil lead can break off inside the SPARK MAX.

1. Connect the SPARK MAX to main power, not just USB Power.
2. The Status LED will indicate which motor type is configured by blinking yellow or blue for Brushed Mode, or blinking magenta or cyan for Brushless Mode.
3. Press and hold the Mode Button for approximately 3 seconds.
4. After the button has been held for enough time, the Status LED will change and indicate the different motor type.
5. Release the mode button.

> ⓘ Please see the Status LED Colors and Patterns in for more details on the Brushed and Brushless Mode colors.

## Configuration with USB

Follow the steps below to switch motor types with the USB and the REV Hardware Client application. Be sure to download and install the REV Hardware Client application before continuing.

1. Connect the SPARK MAX to your computer using a USB-C cable.
2. Open the REV Hardware Client and verify that the application is connected to your SPARK MAX.
3. On the **Basic** tab, select the appropriate motor type under the **Select Motor Type** menu.
4. Click **Update Configuration** and confirm the change.

## Configuration with CAN

Please see the API Information for information on how to configure the SPARK MAX using the CAN interface.

## Idle Mode - Brake/Coast Mode

When the SPARK MAX is receiving a neutral command the idle behavior of the motor can be handled in two different ways: **Braking** or **Coasting**.

When in **Brake Mode**, the SPARK MAX will effectively short all motor wires together. This quickly dissipates any electrical energy within the motor and brings it to a quick stop.

When in **Coast Mode**, the SPARK MAX will effectively disconnect all motor wires. This allows the motor to spin down at its own rate.

The Idle Mode can be configured using the Mode Button, CAN, and USB interfaces.

#### Configuration with Mode Button

Follow the steps below to switch the Idle Mode between Brake and Coast with the Mode Button.

> ⓘ Use a small screwdriver, straightened paper clip, pen, or other small implement to press the button. Do not use any type of pencil as the pencil lead can break off inside the SPARK MAX.

1. Connect the SPARK MAX to main power, not just USB Power.
2. The Status LED will indicate which Idle Mode is currently configured by blinking blue or cyan for Brake and yellow or magenta for Coast depending on the motor type.
3. Press and release the Mode Button

4. You should see the Status LED change to indicate the selected Idle Mode.

> Please see the Status LED Colors and Patterns for more details on the Brushed and Brushless Mode colors.

#### Configuration with USB

Follow the steps below to switch the Idle Mode between Brake and Coast with the USB and the REV Hardware Client application. Be sure to download and install the REV Hardware Client application before continuing.

1. Connect the SPARK MAX to your computer using a USB-C cable.
2. Open the REV Hardware Client application and verify that the application is connected to your SPARK MAX.
3. On the **Basic** tab, select the desired mode with the **Idle Mode** switch.
4. Click **Update Configuration** and confirm the change.

#### Configuration with CAN

Please see the API Information for information on how to configure the SPARK MAX using the CAN interface.

## Control Interfaces

The SPARK MAX can be controlled by three different interfaces, servo-style PWM, controller area network (CAN), and USB. The following sections describe the operation and protocols of these interfaces. For more details on the physical connections, see Control Connections.

---

## PWM Interface

The SPARK MAX can accept a standard servo-style PWM signal as a control for the output duty cycle. Even though the PWM port is shared with the CAN port, SPARK MAX will automatically detect the incoming signal type and respond accordingly. For details on how to connect a PWM cable to the SPARK MAX, see CAN/PWM Port.

The SPARK MAX responds to a factory default pulse range of 1000µs to 2000µs. These pulses correspond to full-reverse and full-forward rotation, respectively, with 1500µs (±5% default input deadband) as the neutral position, i.e. no rotation. The input deadband is configurable with the REV Hardware Client or the CAN interface. The table below describes how the default pulse range maps to the output behavior.

PWM Pulse Mapping

| | Output duty cycle and direction (default deadband) | | | | |
|---|---|---|---|---|---|
| | **Full Reverse** | **Proportional Reverse** | **Neutral** | **Proportional Forward** | **Full Forward** |
| **Output duty cycle, D (%)** | $D = 100$ | $100 < D < 0$ | $D = 0$ | $0 < D < 100$ | $D = 100$ |
| **Input pulse width, p (µs)** | $p \leq 1000$ | $1000 < p < 1475$ | $1475 \leq p \leq 1525$ | $1525 < p < 2000$ | $2000 \leq p$ |
| **Maximum pulse range, p (µs)** | $500 \leq p \leq 2500$ | | | | |
| **Valid pulse frequency, f (Hz)** | $50 \leq f \leq 200$ | | | | |

> If a valid signal isn't received within a 60ms window, the SPARK MAX will disable the motor output and either brake or coast the motor depending on the configured Idle Mode. For details on the Idle Mode, see Idle Mode - Brake/Coast Mode.

---

## CAN Interface

The SPARK MAX can be connected to a robot CAN network. CAN is a bi-directional communications bus that enables advanced features within the SPARK MAX. SPARK MAX must be connected to a CAN network that has the appropriate termination resistors at both endpoints. Please see the FIRST Robotics Competition Robot Rules for the CAN bus wiring requirements. Even though the CAN port is shared with the PWM port, SPARK MAX will automatically detect the incoming signal type and respond accordingly. SPARK MAX uses standard CAN frames with an extended ID (29 bits), and utilizes the FRC CAN protocol for defining the bits of the extended ID:

CAN Packet Structure

| ExtID [28:24] | ExtID [23:16] | ExtID [15:10] | ExtID [9:6] | ExtID [5:0] |
|---|---|---|---|---|
| Device Type | Manufacturer | API Class | API Index | Device ID |

Each device on the CAN bus must be assigned a unique CAN ID number. Out of the box, SPARK MAX is assigned a device ID of 0. It is highly recommended to change all SPARK MAX CAN IDs from 0 to any unused ID from 1 to 62. CAN IDs can be changed by connecting the SPARK MAX to a Windows computer and using the REV Hardware Client. For details on other SPARK MAX configuration parameters, see Configuration Parameters.

Additional information about the CAN accessible features and how to access them can be found in the SPARK MAX API Information section.

### Periodic Status Frames

The SPARK MAX sends data periodically back to the roboRIO. Frequently accessed data, like motor position and temperature, can be accessed using several APIs. Data is broken up into several CAN "frames" which are sent at a periodic rate. This rate can be changed manually in code, but unlike other parameters, this setting **does not persist** through a power cycle. The rate can be set anywhere from a

minimum 1ms to a maximum 65535ms period. The table below describes each status frame and its available data.

Periodic Status 0 - Default Rate: 10ms

| Available Data | Description |
|---|---|
| Applied Output | The actual value sent to the motors from the moto controller. The frame stores this value as a 16-bit signed integer, and is converted to a floating poin value between -1 and 1 by the roboRIO SDK. Thi value is also used by any follower controllers to s their output. |
| Faults | Each bit represents a different fault on the controller. These fault bits clear automatically whe the fault goes away. |
| Sticky Faults | The same as the Faults field, however the bits do not reset until a power cycle or a 'Clear Faults' command is sent. |
| Is Follower | A single bit that is true if the controller is configure to follow another controller. |

Periodic Status 1 - Default Rate: 20ms

| Available Data | Description |
|---|---|
| Motor Velocity | 32-bit IEEE floating-point representation of the motor velocity in RPM using the selected sensor. |
| Motor Temperature | 8-bit unsigned value representing:<br>Firmware version 1.0.381 - Voltage of the temperature sensor with 0 = 0V and 255 = 3.3V.<br>Current firmware versions - Motor temperature in for the NEO Brushless Motor. |
| Motor Voltage | 12-bit fixed-point value that is converted to a floating point voltage value (in Volts) by the roboRIO SDK. This is the input voltage to the controller. |
| Motor Current | 12-bit fixed-point value that is converted to a floating point current value (in Amps) by the roboRIO SDK. This is the raw phase current of th motor. |

Periodic Status 2 - Default Rate: 20ms

| Available Data | Description |
| --- | --- |
| Motor Position | 32-bit IEEE floating-point representation of the motor position in rotations. |

#### Use-case Examples

#### Position Control on the roboRIO

A user wants to implement their own PID loop on the roboRIO to hold a position. They want to run this loop at 100Hz (every 10ms), but the motor position data in Periodic Status 2 is sent at 20Hz (every 50ms).

The user can change this rate to 10ms by calling:

*Pseudocode*

```
setPeriodicFrameRate(PeriodicFrame.kStatus2, 10);
```

#### High CAN Utilization

A user has many connected CAN devices and wishes to minimize the CAN bus utilization. They do not need any telemetry feedback, and have several follower devices that are only checked for faults.

The user can set the telemetry frame rates low, and set the Periodic Status 0 frame rate low on the follower devices:

*Pseudocode*

```
leader.setPeriodicFrameRate(PeriodicFrame.kStatus1, 500);
leader.setPeriodicFrameRate(PeriodicFrame.kStatus2, 500);
follower.setPeriodicFrameRate(PeriodicFrame.kStatus0, 100);
follower.setPeriodicFrameRate(PeriodicFrame.kStatus1, 500);
follower.setPeriodicFrameRate(PeriodicFrame.kStatus2, 500);
```

#### Faster Follower Bandwidth

The user wants the follower devices to update at a faster rate: 200Hz (every 5ms).

The Periodic Status 0 frame can be increased to achieve this.

*Pseudocode*

```
leader.setPeriodicFrameRate(PeriodicFrame.kStatus0, 5);
```

---

## USB Interface

The SPARK MAX can be configured and controlled through a USB connection to a computer running the REV Hardware Client. The USB interface utilizes a standard CDC (USB to Serial) driver. The command interface is similar to CAN, using the same ID and data structure, but always sends and receives a full 12-byte packet. The CAN ID is omitted (DNC) when talking directly to the device. However, the three MSB of the ID allow selection of alternate commands:

- 0b000 - Standard command - CAN ID omitted (DNC)
- 0b001 - Extended command - USB specific

All commands sent over USB receive a response. In the case that the corresponding CAN command does not receive a response, the USB interface receives an Ack command.

USB Packet Structure

| ExtID [31:29] | ExtID [28:24] | ExtID [23:16] | ExtID [15:10] | ExtID [9:6] | ExtID [5:0] |
|---|---|---|---|---|---|
| USB Command Type | Device Type (2) | Manufacturer (0x15) | API Class | API Index | Device ID |

USB Non-Standard Commands

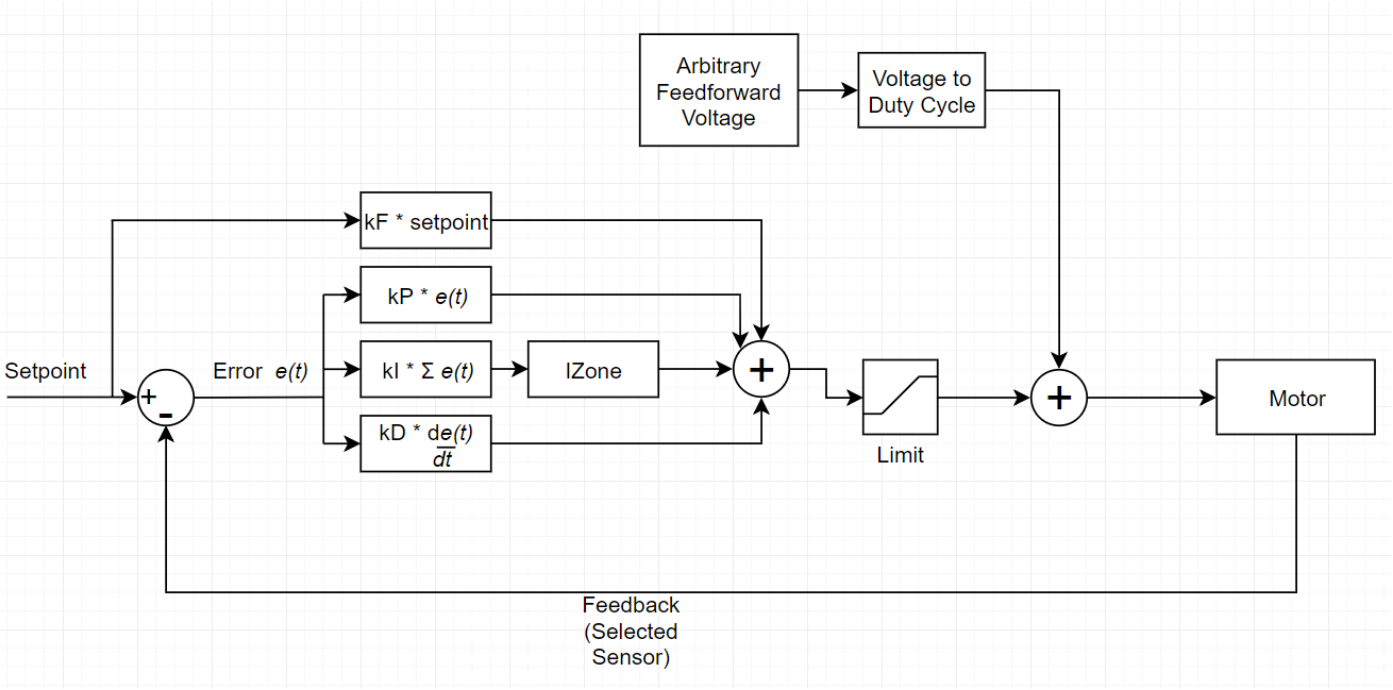
| Command | API Class | API Index |
|---|---|---|
| Enter DFU Bootloader (will also disconnect USB interface) | 0 | 1 |

## Closed Loop Control

SPARK MAX can operate in several closed-loop control modes, using sensor input to tightly control the motor velocity, position or current. The internal control loop follows a standard PID algorithm with a feed-forward (F) term to compensate for known system offsets.

Additionally, an arbitrary feedforward signal is added to the output of the control loop after *all* calculations are done. The units for this signal can be selected as either *voltage* or *duty cycle.* This feature allows more advanced feedforward calculations to be performed by the controller. More details about the types of feedforward calculations can be found on the WPILib documentation. Using the *voltage* units for arbitrary feedforward allows the user to send the calculated feedforward voltage from the WPILib API directly to the control loop.

Below is a diagram and the firmware implementation of the internal SPARK MAX PIDF.



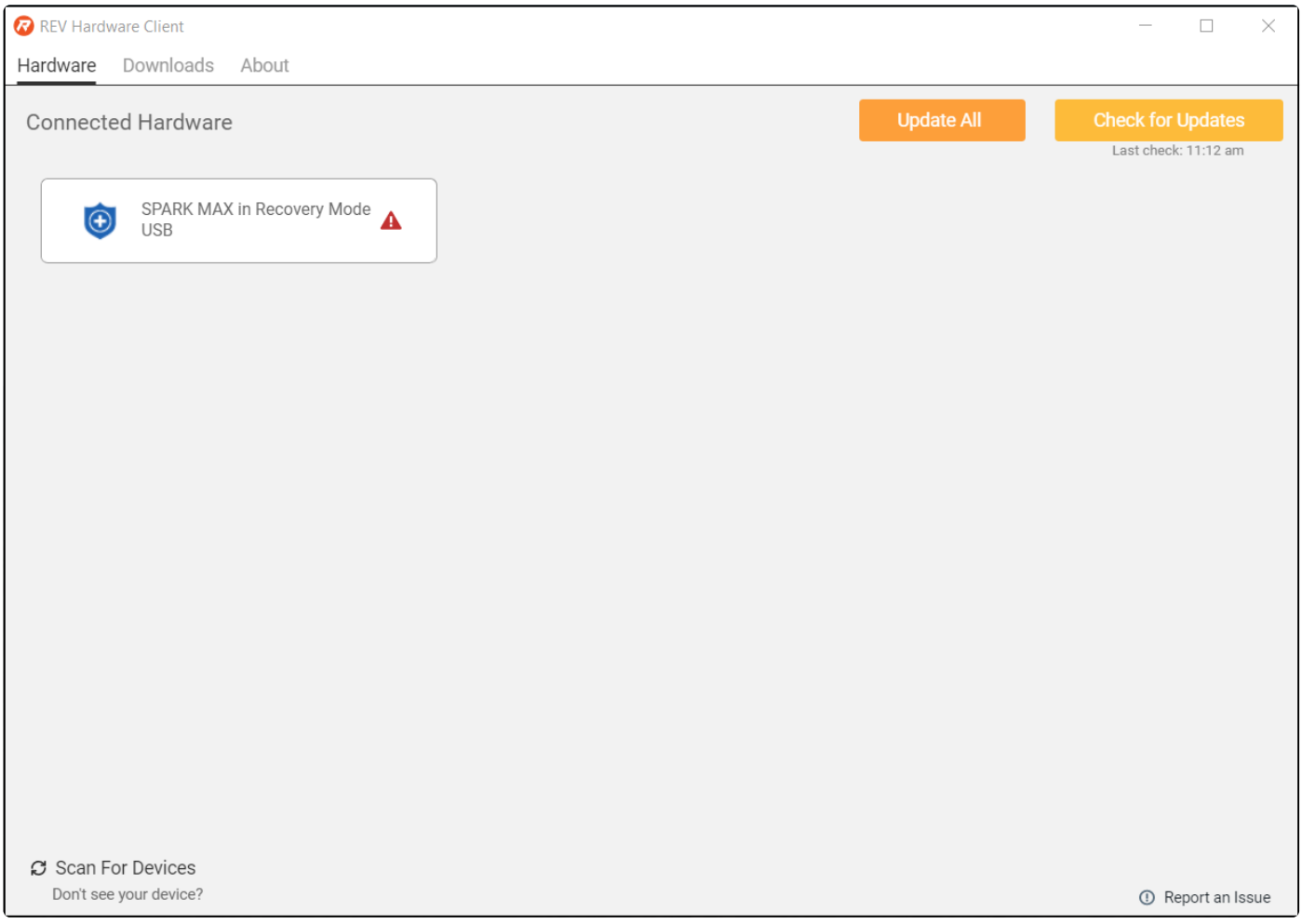

```
//Synchronous PID, call at desired frequency
float pid_run(pid_instance_t* pid, float setpoint, float pv,
        const pid_constants_t* constants)
{
    float error = setpoint - pv;

    float p = error * constants->kP;

    if(fabsf(error) <= constants->iZone || constants->iZone == 0.0f) {
        pid->iState = pid->iState + (error * constants->kI);
    } else {
        pid->iState = 0;
    }

    float d = (error - pid->prev_err);
    pid->prev_err = error;
    d *= constants->kD;

    float f = setpoint * constants->kF;

    float output = p + pid->iState + d + f;
    pid->output = fminf(fmaxf(output,constants->kMinOutput),constants->kMaxOutput);

    return output;
}
```

For more information on utilizing the built-in closed-loop control modes, please take a look at our SPARK MAX Code Examples.

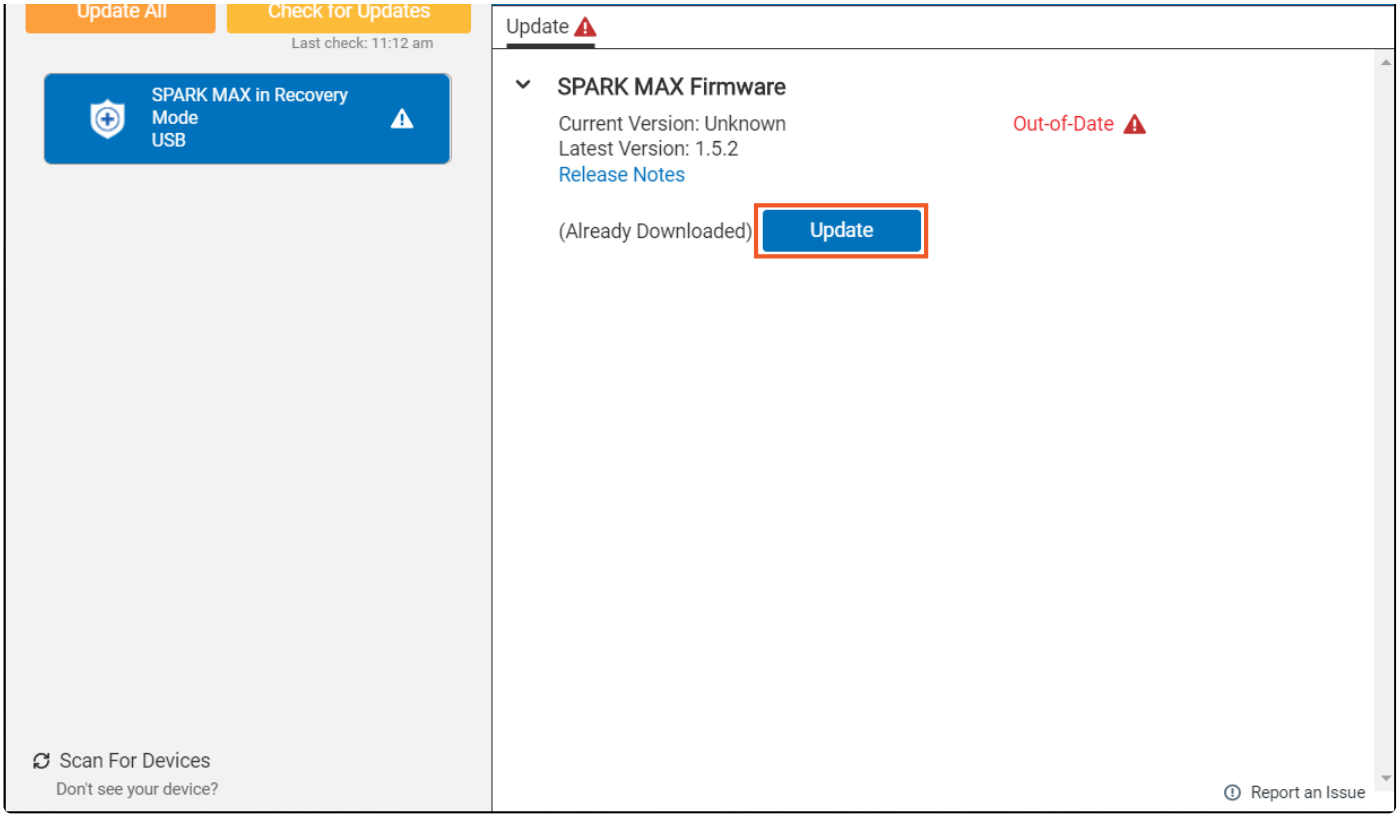
## Recovery Mode

When updating the firmware on the SPARK MAX, it is possible for the process to be interrupted or for the firmware to be corrupted by a bad download. In this state, the Status LED will be dark and the SPARK MAX will fail to operate. SPARK MAX has a built-in recovery mode that can force it to accept new firmware even if the controller seems to be bricked. The following procedure requires a small tool, like a straightened paper clip, to press the Mode Button, a USB C cable, and a computer with the REV Hardware Client installed:

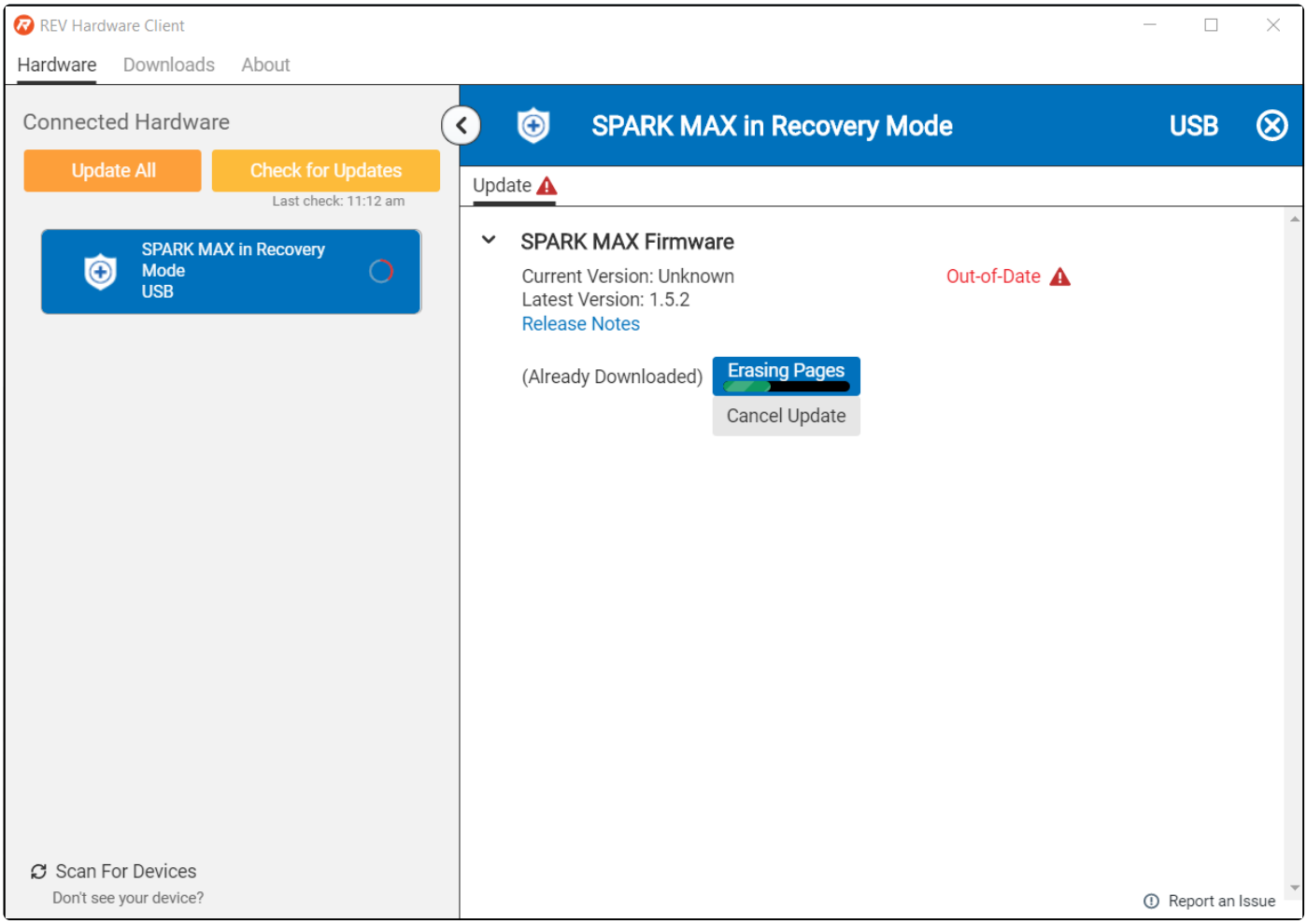
- With the SPARK MAX powered off completely, press and hold the Mode Button.
- While still holding the Mode Button, connect the SPARK MAX to the computer using the USB cable. The Status LED **will not** illuminate, this is expected.
- Wait a few seconds for the computer to recognize the connected device, then release the Mode Button.
- Open the REV Hardware Client. The SPARK MAX will remain dark and it **will not** connect to the Client, this is expected.
- Select the SPARK MAX in Recovery Mode from the REV Hardware Client




- Press the Update Button

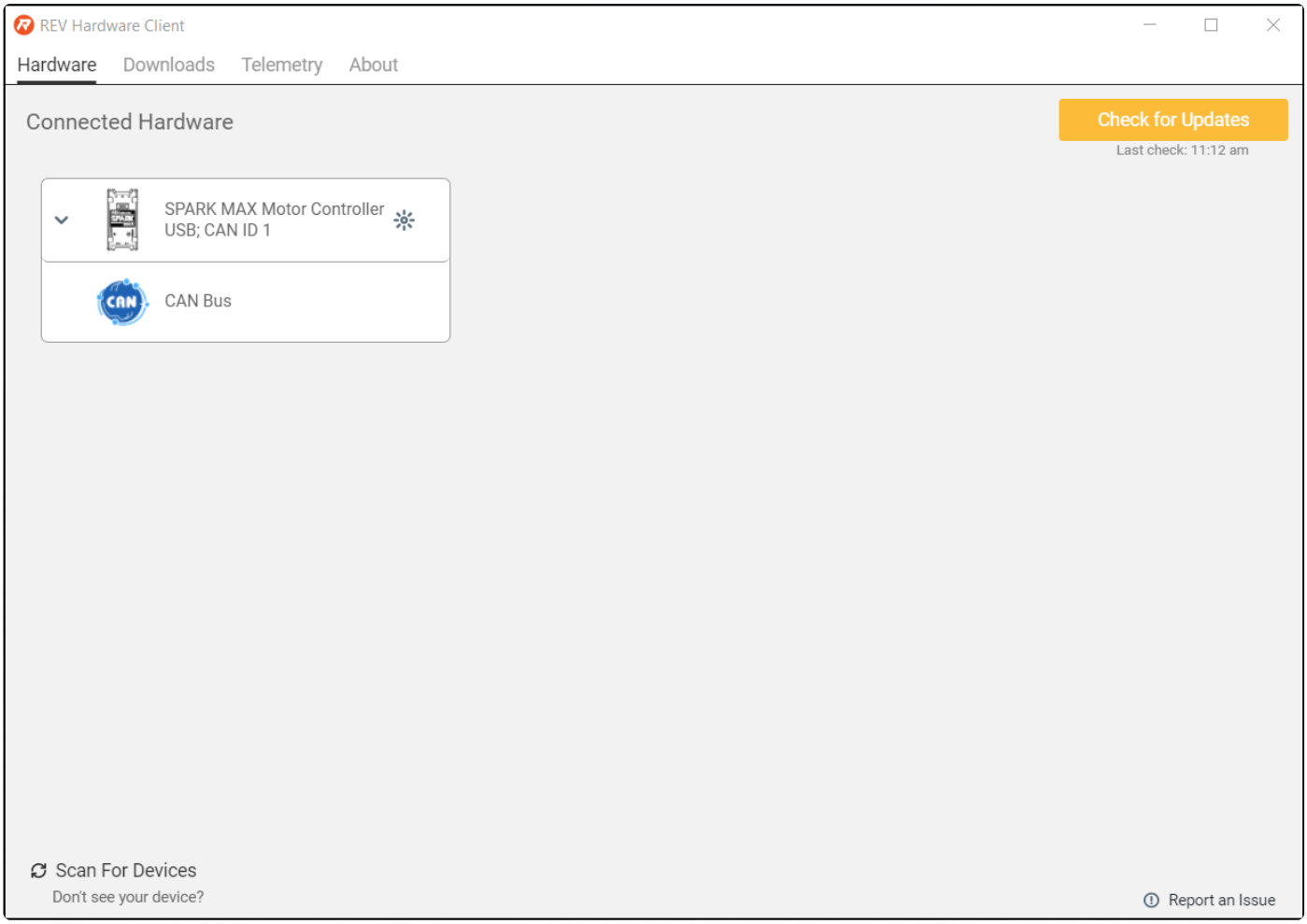



- Wait for the Firmware Update to complete




- Once completed the SPARK MAX will disconnect then reconnect to the REV Hardware Client. Select

the SPARK MAX to start the configuration process




ⓘ The Recovery Mode is also available with the Legacy SPARK MAX Client

## Alternate Encoder Mode

The SPARK MAX can be configured to run in Alternate Encoder Mode, which reconfigures the Data Port on the top of the controller to accept an alternative quadrature encoder, separate from the default encoder inputs shared between the front Encoder Port and the default quadrature encoder Data Port pins. Analog input is not affected by Alternate Encoder Mode.

ⓘ **This feature is designed for use in low-RPM mechanisms such as drivetrains, arms, and other manipulators.** For high RPM applications it is recommended to use the built-in motor sensor for brushless motors or the default encoder inputs for brushed motors.

Alternate Encoder Specifications

| Parameter | Specification |
| --- | --- |

| | |
|---|---|
| Encoder Output Voltage Level | 3.3V or 5.0V |
| Encoder Type Supported | Quadrature† |
| Maximum Counts per Second | 165000 |

| | |
|---|---|
| † | Index pulses are not currently supported |

> ⚠ **Before connecting a sensor with 5V output**, the SPARK MAX **must first be configured** into Alternate Encoder Mode and its configuration **must be saved** on the SPARK MAX. This can be done through the REV Hardware Client or the software APIs.

Maximum RPM with Common Quadrature Encoders

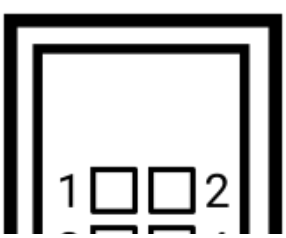
| Encoder | Counts per Revolution | Max RPM |
|---|---|---|
| REV Through Bore Encoder | 8192 | 1200 |
| CTRE SRX Mag Encoder | 4096 | 2400 |
| Greyhill 63R256 | 1024 | 9600 |

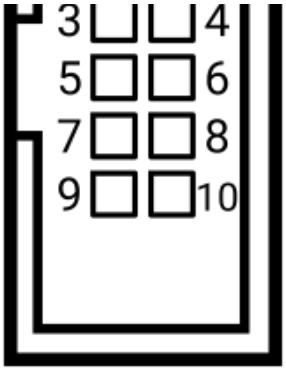
When configured for Alternate Encoder Mode, a quadrature encoder connected to the reconfigured Data Port pins can be used as a feedback device by the SPARK MAX. **Please note, the limit switch inputs cannot be used at the same time as an alternate encoder.** The limit switch pins are repurposed for the alternate encoder and are thus disabled. Please see Connecting an Alternate Encoder for for more information.

---

## Connecting an Alternate Encoder

Connecting an alternate encoder will likely require a custom wiring harness to connect the necessary encoder power, ground, and signals to the reconfigured Data Port. When configured in Alternate Encoder Mode, the Data Port has the following pinout:






Data Port Pinout in Alternate Encoder Mode

| Connector Pin | Pin Type | Pin Function |
|---|---|---|
| 1 | Power | +3.3V |
| 2 | Power | +5V |
| 3 | Analog | Analog Input |
| 4 | Digital | **Alternate Encoder Index†** |
| 5 | Digital | Encoder B |
| 6 | Digital | **Alternate Encoder A** |
| 7 | Digital | Encoder A |
| 8 | Digital | **Alternate Encoder B** |
| 9 | Digital | Encoder C / Index |
| 10 | Ground | Ground |

| | |
|---|---|
| † | The Alternate Encoder Index pin is reserved but r currently supported |

Use an Alternate Encoder Adapter (REV-11-1881) to connect a REV Through Bore Encoder directly to the SPARK MAX Data Port. This adapter has a JST PH 6-pin connector that is compatible with the Through Bore Encoder's pinout and a selection switch to change the signal that is connect to pin 4 of the data port.

Another option is the SPARK MAX Data Port Breakout Board. This board can be used to wire an alternate encoder to the Data Port. The following table describes which pads on the breakout should be used for which signals coming from the alternate encoder.

Alternate Encoder Pin-mapping for SPARK MAX Data Port Breakout Board

| Breakout Board Pad Label | Alternate Encoder Function |
|---|---|
| Limit - F | Index† |
| P6 (P5 in older batches) | A |
| Limit - R | B |
| 3.3V or 5.0V | Encoder Power |
| GND | Encoder Ground |

| | |
|---|---|
| † | The Alternate Encoder Index pin is reserved but r currently supported |

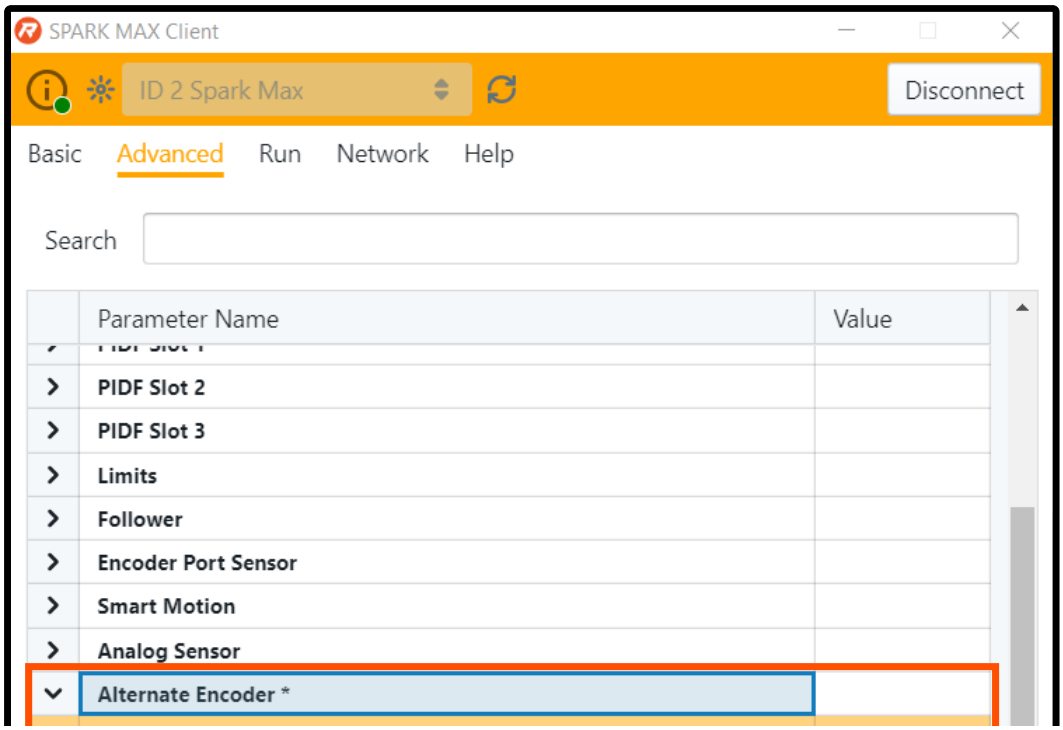
---

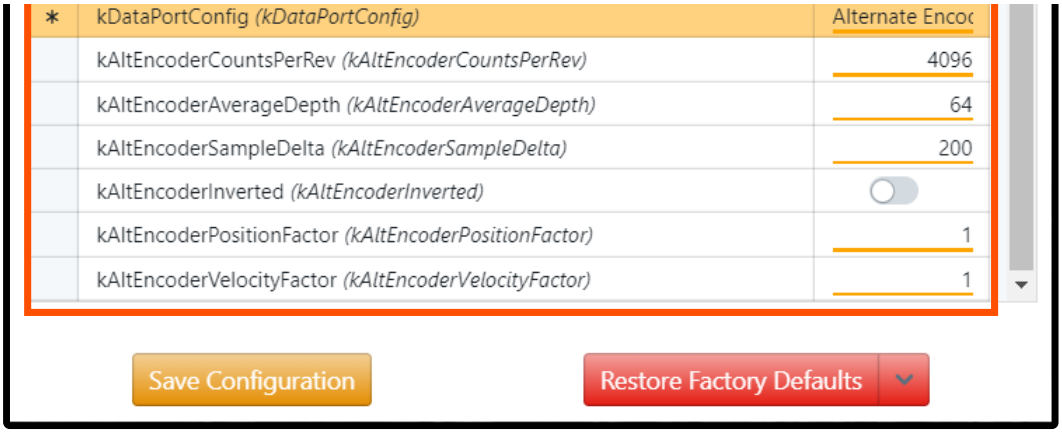
## Configuring and Using the Alternate Encoder Mode

Below you will find the steps required to set up and use the Alternate Encoder Mode on the SPARK MAX, starting with configuration through either the SPARK MAX Client or the SPARK MAX APIs.

### Configuration Using the SPARK MAX Client

Using the SPARK MAX Client, navigate to the Advanced Tab and scroll to the Alternate Encoder parameter section. Enable the alternate encoder by setting the *kDataPortConfig* parameter to '1'. You can also set the other Alternate Encoder parameters at this time.






### Configuring Using the SPARK MAX APIs

If using the SPARK MAX APIs, the Alternate Encoder is automatically configured when the Alternate Encoder object is instantiated. An Alternate Encoder is created the same as a *CANEncoder*, either by directly using the constructor or calling *GetAlternateEncoder()* on a previously constructed *CANSparkMax*.

```
static constexpr int kCanId = 1;
static constexpr auto kMotorType = rev::CANSparkMax::MotorType::kBrushless;
static constexpr auto kAltEncType = rev::CANEncoder::AlternateEncoderType::kQuadrature;
static constexpr int kCPR = 8192;

// initialize SPARK MAX with CAN ID
rev::CANSparkMax m_motor{kCanID, kMotorType};

/**
* An alternate encoder object is constructed using the GetAlternateEncoder()
* method on an existing CANSparkMax object. If using a REV Through Bore
* Encoder, the type should be set to quadrature and the counts per
* revolution set to 8192
*/
rev::CANEncoder m_alternateEncoder = m_motor.GetAlternateEncoder(kAltEncType, kCPR);
```

Currently, quadrature is the only available type of configuration for an alternate encoder. This is differentiated from the other types of encoder configurations available for an encoder connected through the front facing Encoder Port on the SPARK MAX.

### Configuration Conflicts

Since the alternate encoder inputs and the default digital inputs are shared on the Data Port, the user cannot use both the alternate encoder and a digital inputs in code. Therefore, a **std::invalid_argument** (C++), **IllegalArgumentException** (Java), or an **Error on the Error Out terminal** (LabVIEW) will be thrown if a user tries to construct both types objects in code simultaneously.

### Closed-Loop Control

The alternate encoder can be used with the different closed-loop control modes available on the SPARK MAX. The feedback device used by a *CANPIDController* must be set to use the alternate encoder through

*SetFeedbackDevice().*

```
/**
* By default, the PID controller will use the Hall sensor from a NEO or NEO 550 for
* its feedback device. Instead, we can set the feedback device to the alternate
* encoder object
*/
m_pidController.SetFeedbackDevice(m_alternateEncoder);
```

**Initial Bring-up**

Unlike the built-in sensor on the NEO Brushless motors, the 'phase' of the alternate encoder is unknown to the SPARK MAX. Before enabling any closed-loop control, it is critical that the phase is configured correctly. To verify:

1. Configure and connect the sensor as a quadrature alternate encoder, but **do not** run a closed-loop mode.
2. Plot the output signal of the motor using *GetAppliedOutput()* and the output of the encoder using *altEncoder.GetVelocity()*. Confirm that the sensor is behaving as expected. This can be done on the SmartDashboard:
   `frc::SmartDashboard::PutNumber("Alt Encoder Velocity", m_alternateEncoder.GetVelocity());`
   `frc::SmartDashboard::PutNumber("Applied Output", m_motor.GetAppliedOutput());`
3. Verify that the sign of the sensor is correct relative to the motor direction when driving it forward and backward. If it is not, the sensor must be inverted by calling *altEncoder.SetInverted(true).*

# REV Hardware Client

## Getting Started with the REV Hardware Client

The REV Hardware Client is software designed to make managing REV devices easier for the end user. This Client automatically detects connected device(s), downloads the latest software for those device(s), allows for seamless updating of the device(s), and allows for device configuration and management.

For more information on the REV Hardware Client, including a full list of supported devices, see the User's Manual.

| Latest REV Hardware Client - Version 1.3.0 |
| --- |
| Download Latest REV Hardware Client |

> The REV Hardware Client **will not work with SPARK MAX beta units** distributed by REV to the SPARK MAX beta testers. It is only compatible with units received after 12/21/2018.

#### System Requirements

- Operating System: Windows 7 (64-bit) or newer
- Processor: 64-bit

#### Installation Instructions

- Download the REV Hardware Client Installer above.
- Run the Installer
- Run the REV Hardware Client from the Windows Start Menu or a desktop shortcut

## Navigating the REV Hardware Client

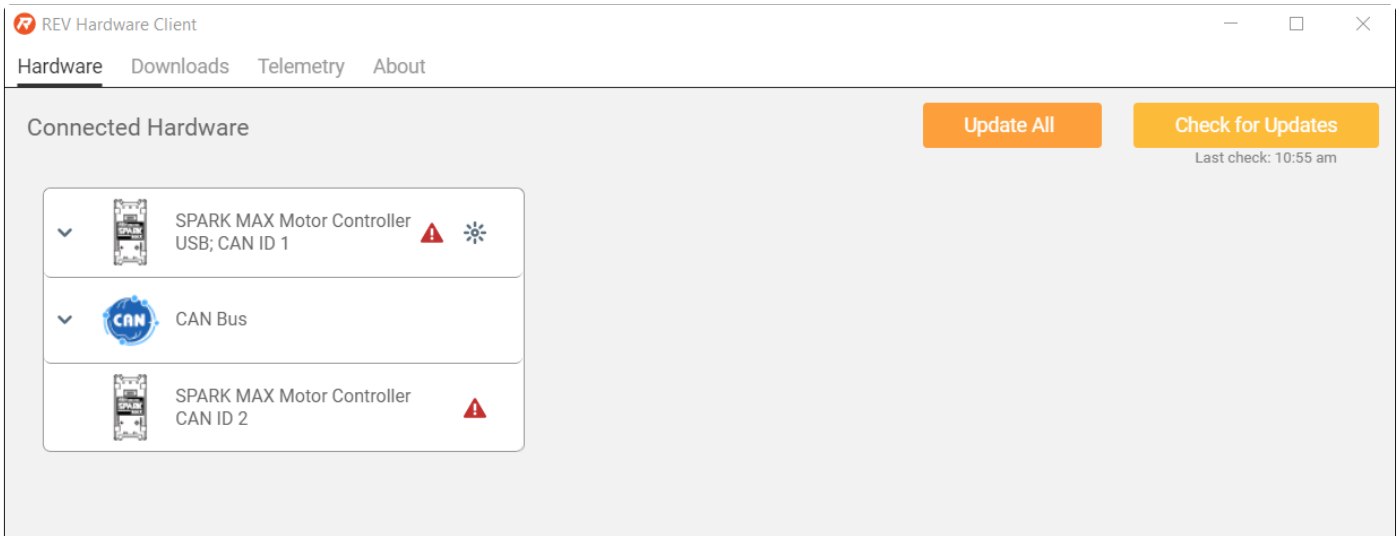
The REV Hardware Client has three tabs to manage different features of the Client. The Hardware Tab is where supported hardware devices are managed. The Downloads Tab allows for the downloading of supported device software for updating when offline. The About Tab has information on what devices are supported, updating the REV Hardware Client, and having issue reporting.

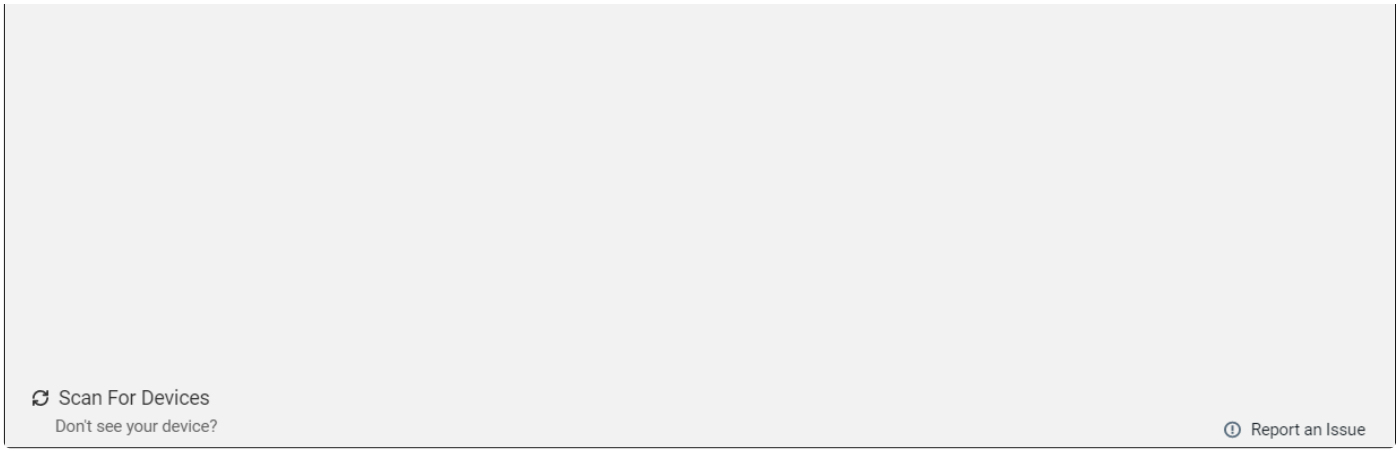
Individual devices, like the SPARK MAX, have additional tabs available when the device is selected. For a full overview of the default navigation features of the REV Hardware Client see the User's Manual. Below is more information on using the specific features for the SPARK MAX.

---

## Hardware Tab

The Hardware Tab is used to select devices connected via USB or the USB to CAN bridge for configuration, updates, and more.



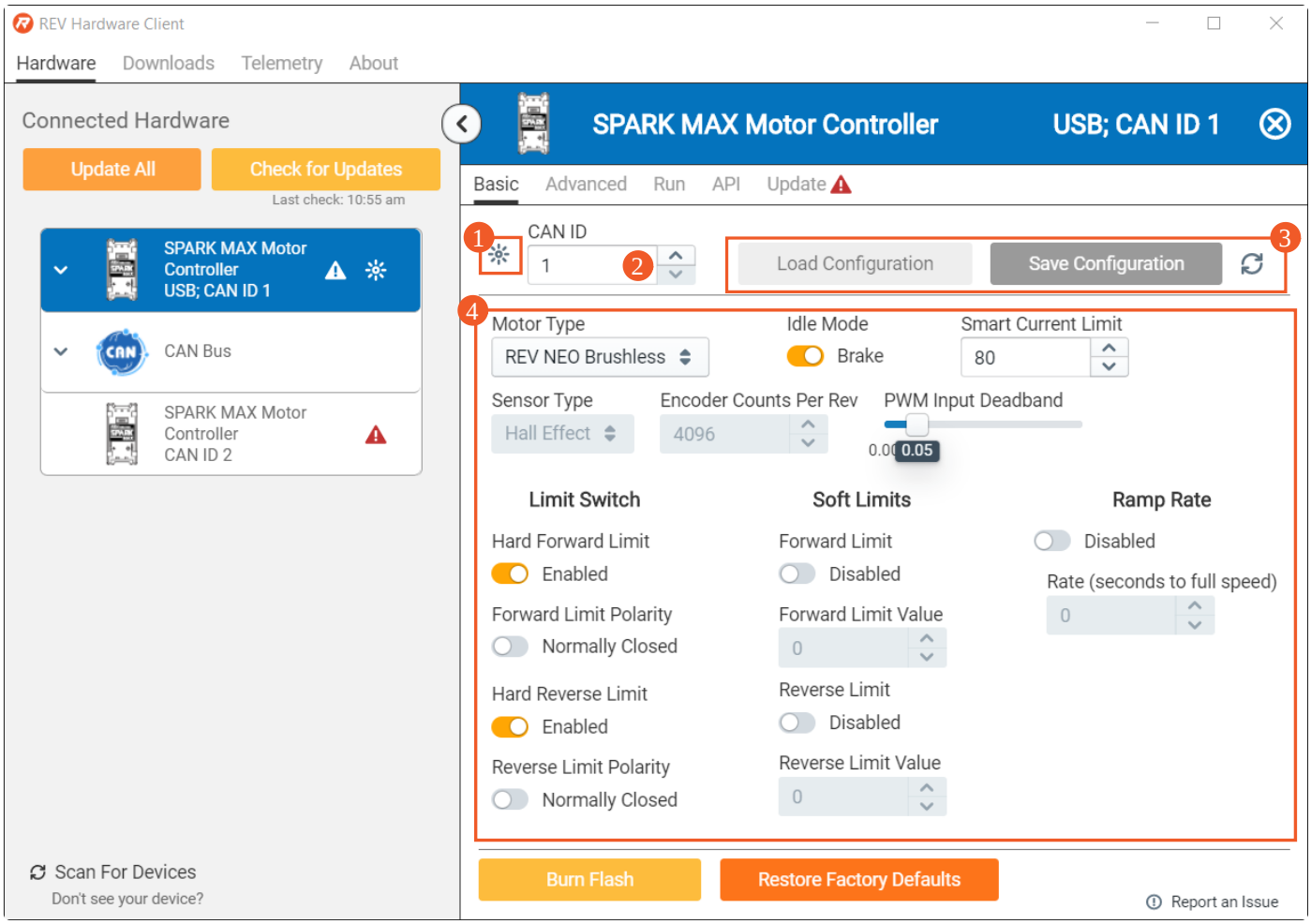



Once a SPARK MAX is selected from the Hardware tab a number of device specific tabs will show.

---

## Basic Tab

The Basic Tab is used to set the most common parameters for the SPARK MAX.



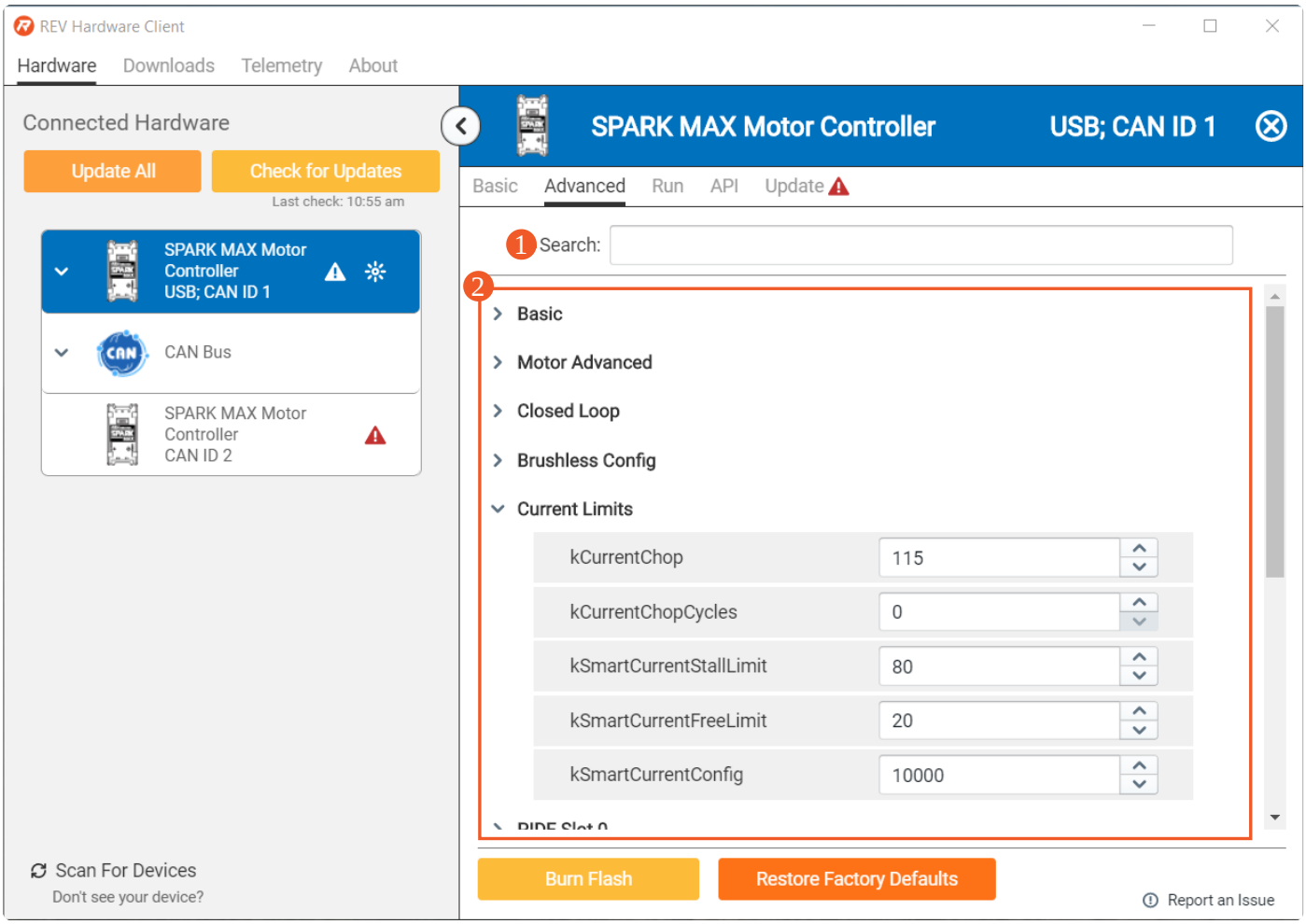

1. **Device Identify:** Blink the selected SPARK MAX's LED for identification.
2. **CAN ID:** This assigns a SPARK MAX a CAN ID for identification over the CAN BUS. Any configured SPARK MAX **must have** a CAN ID.

3. **Configurations:** This drop down allows you to select pre-existing configurations store on the Windows machine running the SPARK MAX Client or to pull the existing parameters stored on in RAM on the SPARK MAX. This is helpful when configuring multiple motor controllers to the same settings.
4. **Configured Parameters:** Change the motor type, sensor type, idle mode behavior, and more.

---

## Advanced Tab

The Advanced Tab allows for changing all configurable parameters of the SPARK MAX without needing to set them in code.



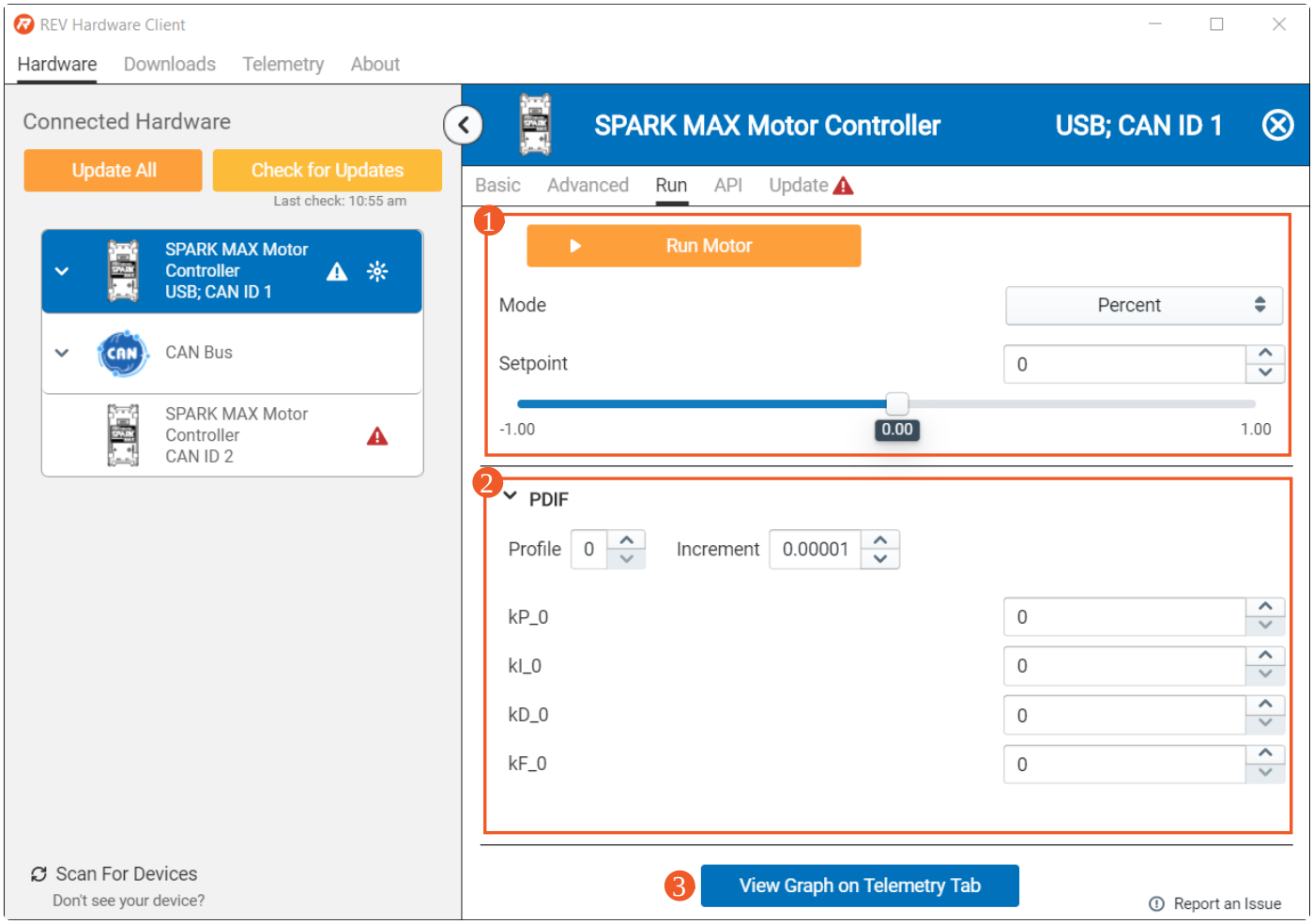

1. **Search Parameters:** Allows for easy look up of a specific parameter for editing.
2. **Parameter Table:** Select the arrow to show all configurable parameters within a specific group. For more information on each parameter type see Configuration Parameters.

> Remember to burn the parameters to flash memory before disconnecting the SPARK MAX

---

## Run Tab

The Run Tab allows for the SPARK MAX to operate over USB or a USB to CAN Bridge without the need for a full control system. This is helpful for testing mechanisms and tuning their control loops.

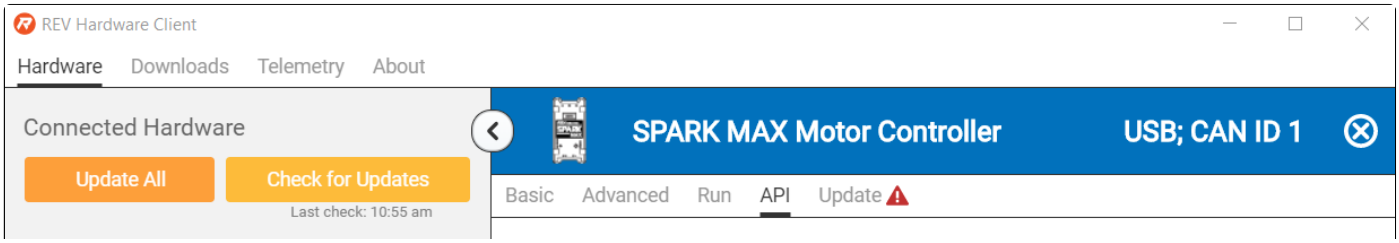


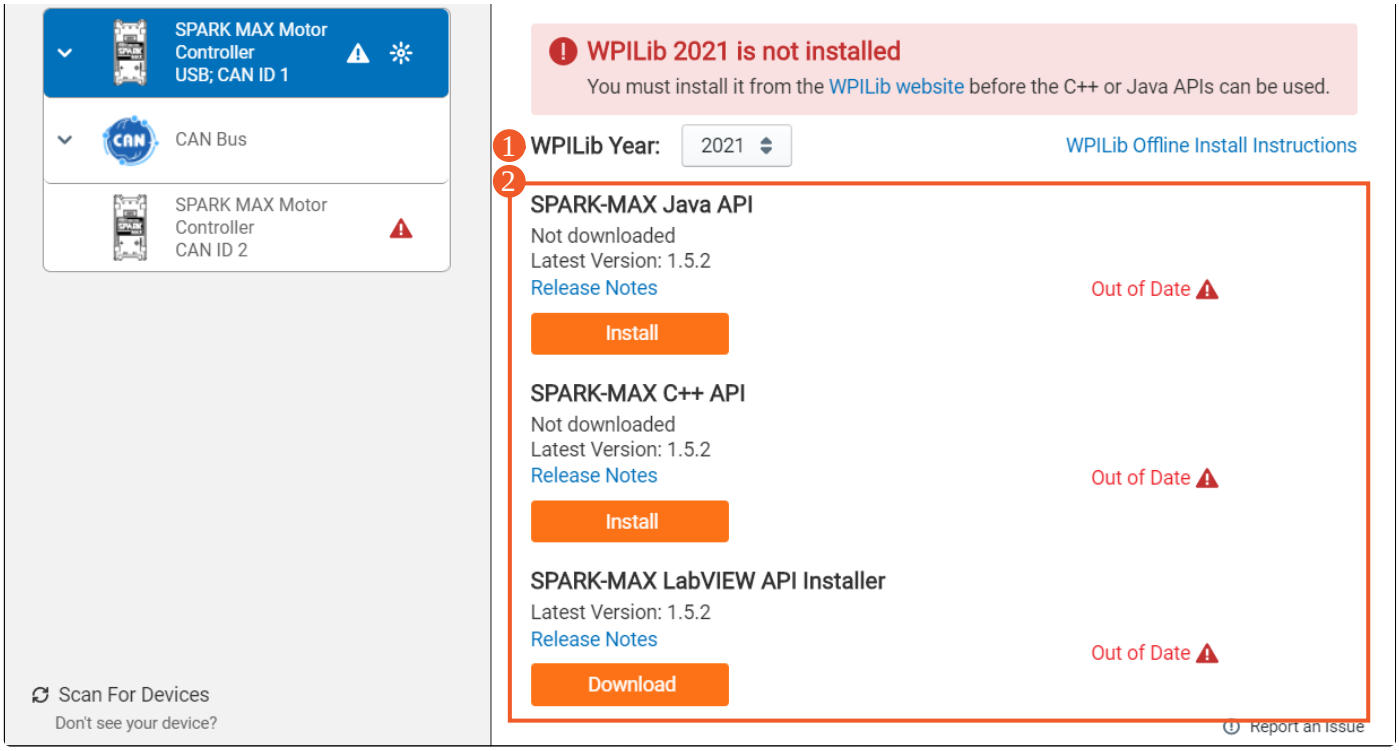

1. **Run:** Choose setpoints to run a motor connected to a SPARK MAX using various modes, including position, velocity, and duty cycle.
2. **PIDF:** Update PIDF parameters on the fly to tune control loops on the SPARK MAX.
3. **View Graph:** Moves the Client over to the Telemetry Tab to show any added signals in graph form when running a SPARK MAX. This is helpful when tuning control loops.

---

## API Tab

The API tab shows what the current version and latest version of the SPARK MAX APIs are. There are links to installing the current version of WPILib and for installing the SPARK MAX APIs properly.



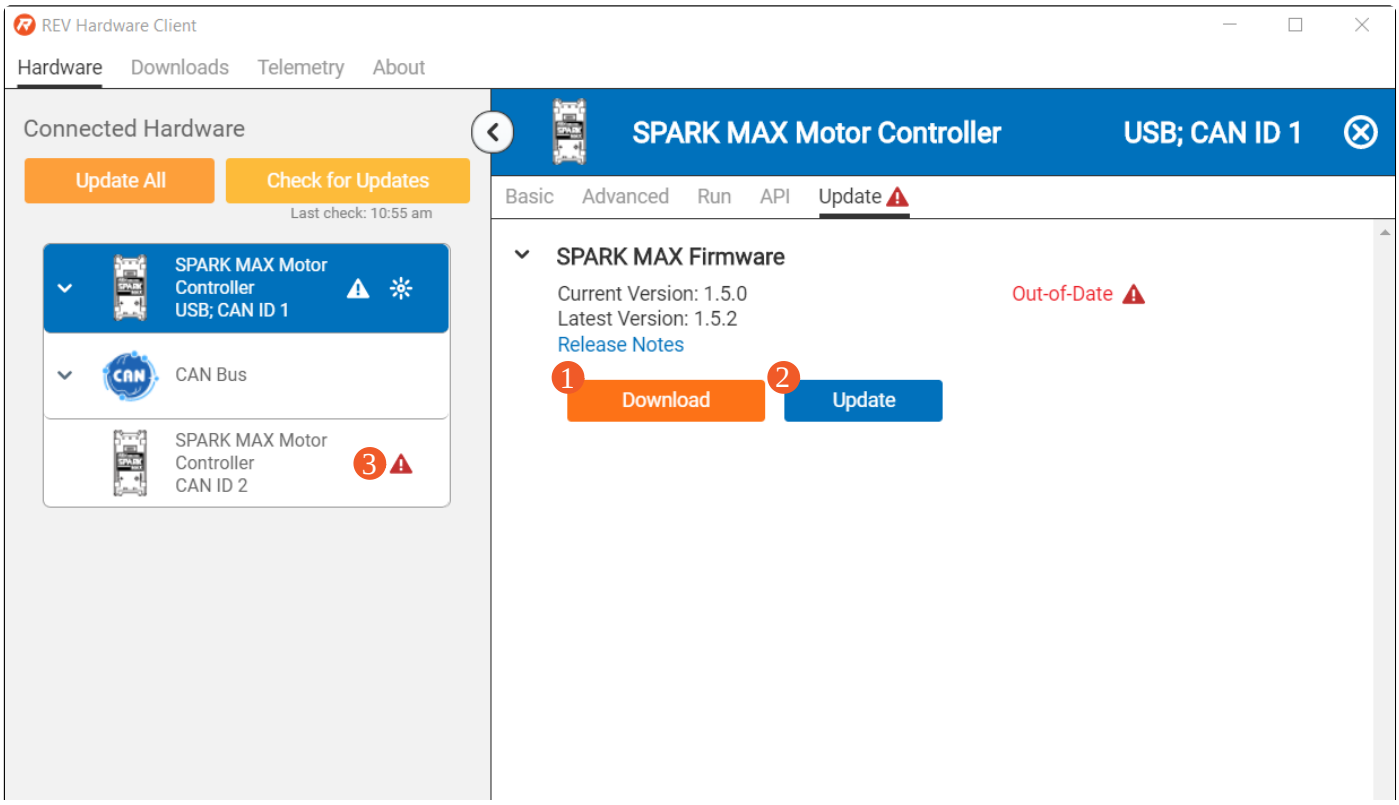



1. **Select Year:** Selects the WPILib release year that is installed for API compatibility.
2. **API Selection:** Ability to choose which language API to download.

---

## Update Tab

The Update tab shows what version of firmware is on the selected device, if that device is up to date, and update the firmware of the selected device.




1. **Download Latest Firmware:** Downloads latest firmware onto the local machine running the Client.
2. **Update Firmware:** Updates the selected device with the latest firmware.
3. **Out-of-date Firmware Warning:** Warning to alert the user there is new firmware available for any connected device.

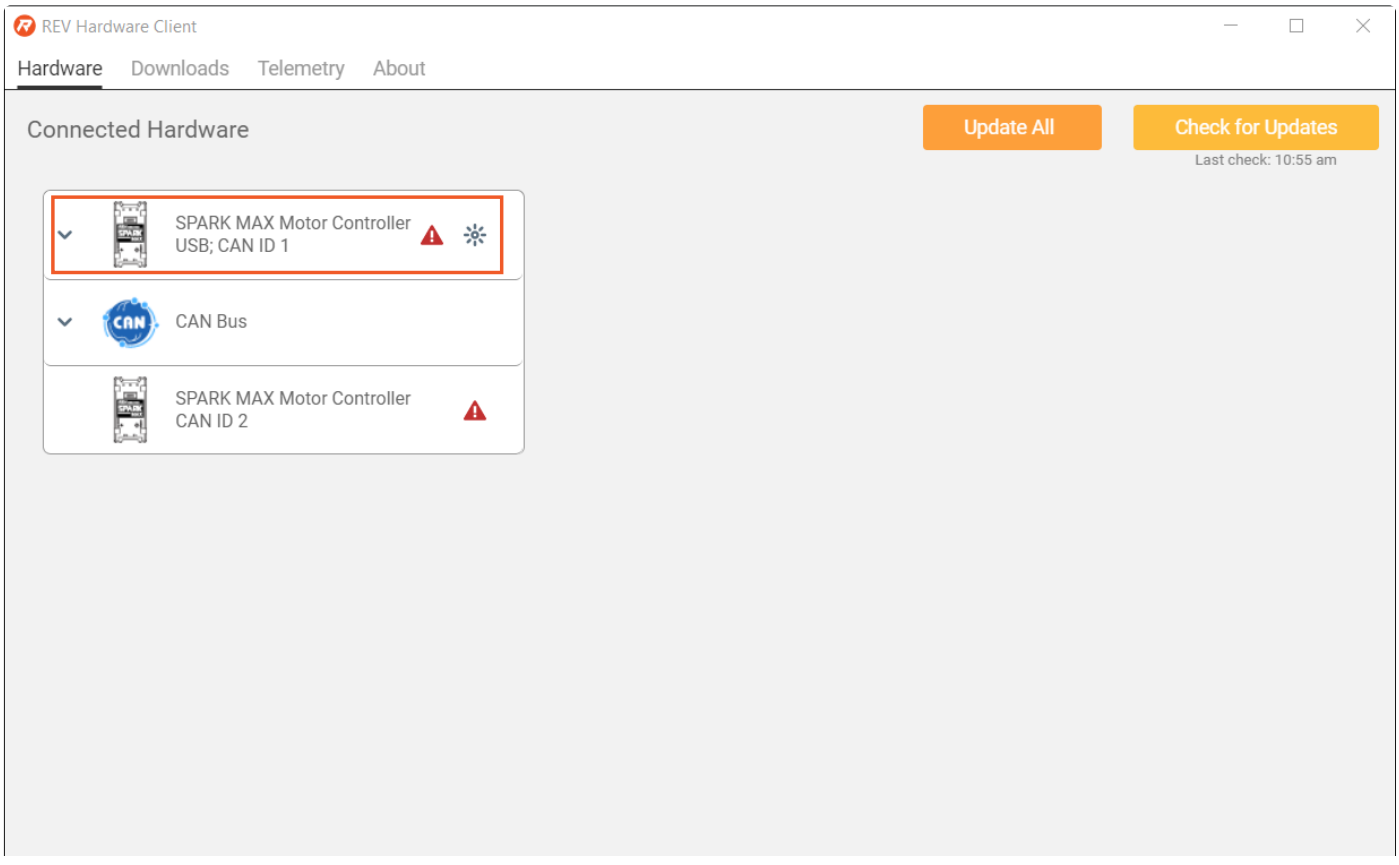
> For more information on the firmware updating process see Updating Device Firmware

## Updating Device Firmware

### Updating a Single SPARK MAX

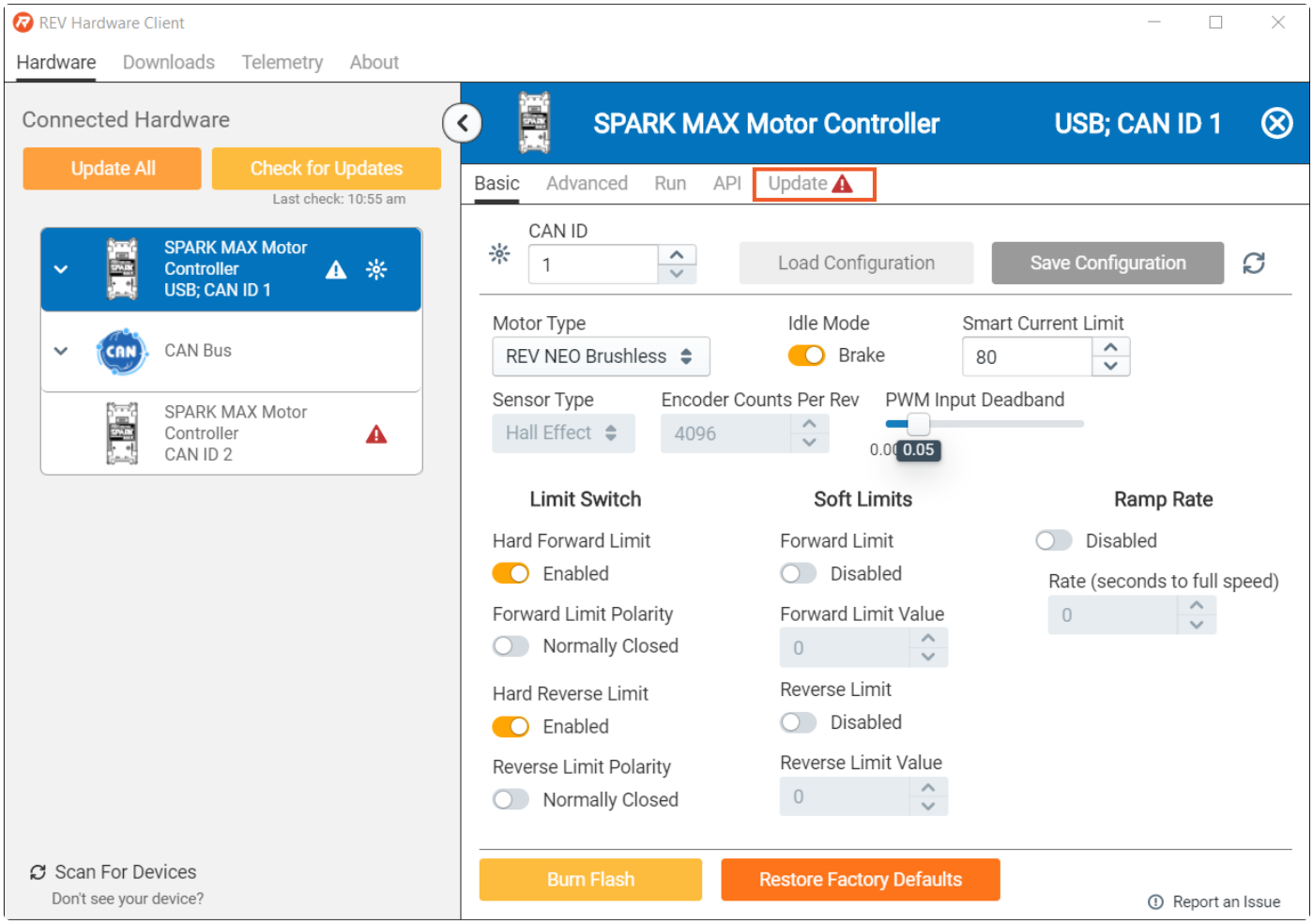
- Connect your SPARK MAX Motor Controller to your computer with a USB-C cable.
- Open the REV SPARK MAX Client application.
- The Client should automatically scan and connect to your SPARK MAX.

Once the SPARK MAX is connected via USB-C select it within the **Connected Hardware.**

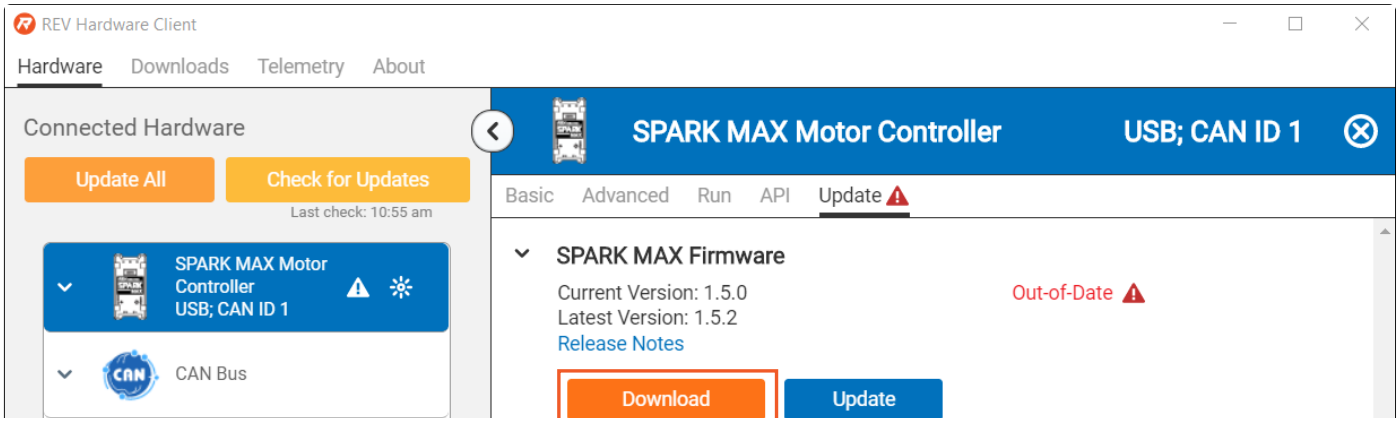


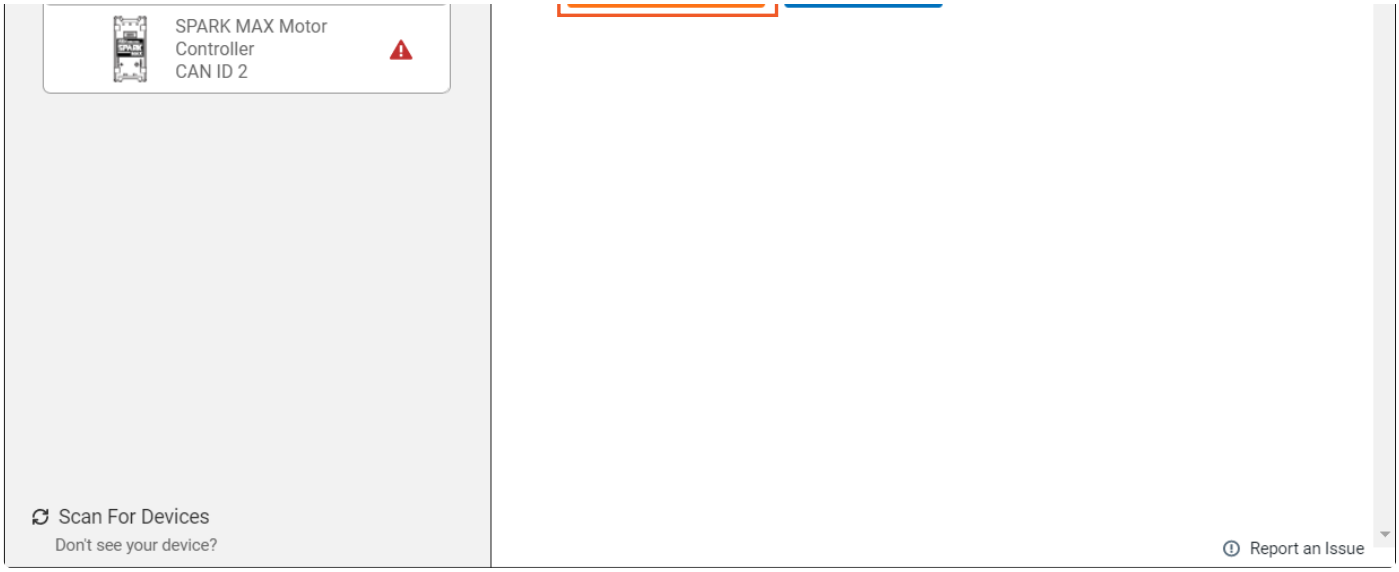

> If the SPARK MAX connected via USB-C is running firmware version 1.5.0 or later allows the SPARK MAX to work as a USB to CAN Bridge. Other CAN connected SPARK MAXs running version 1.4.0 can be selected for firmware updates over CAN.

Within the Hardware Client, for the SPARK MAX, there are 5 tabs. The Hardware Client will open up on the **Basic** tab. To update firmware select the **Update** tab.

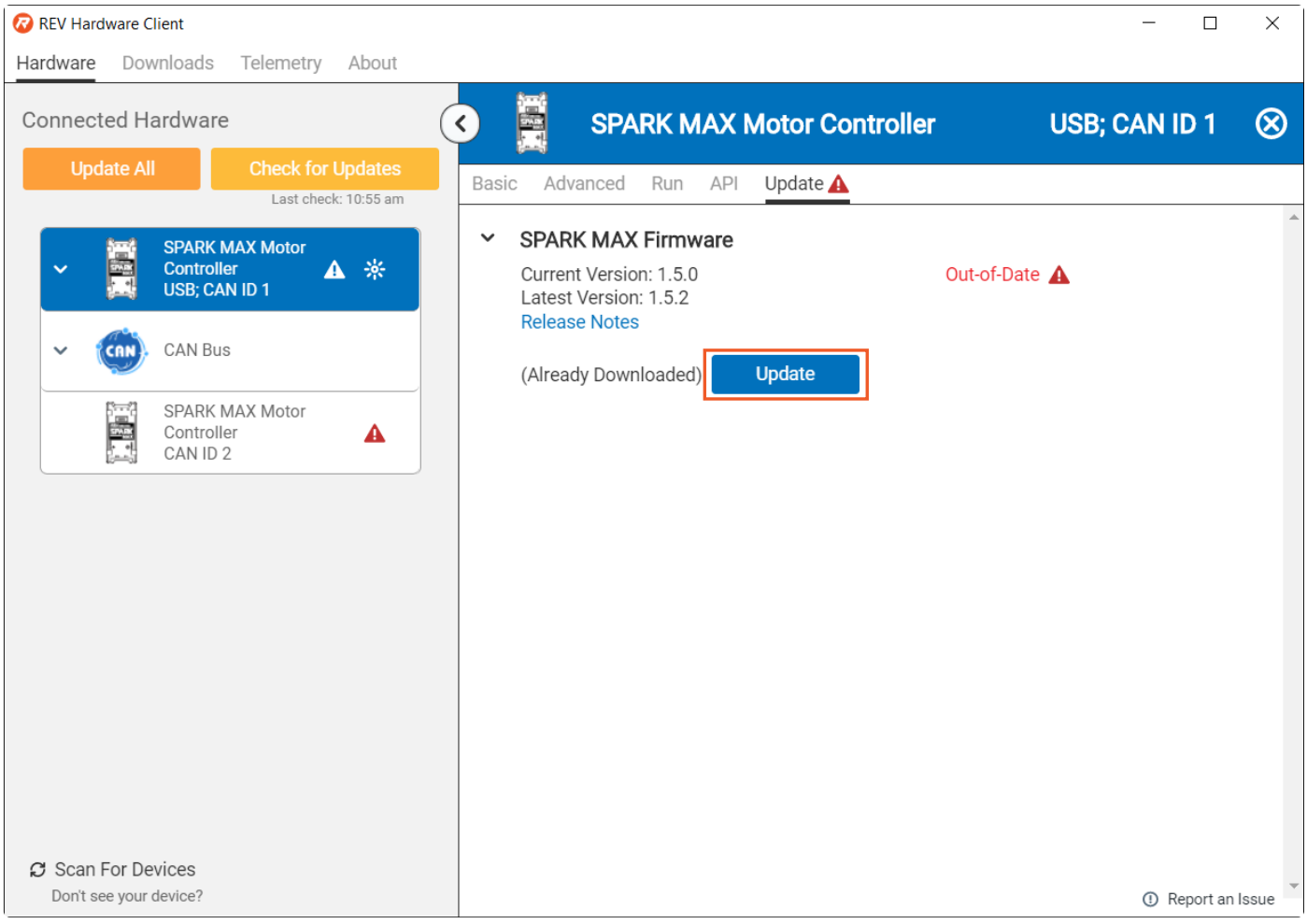



Under **SPARK MAX Firmware**, select download to download the latest version of the firmware.

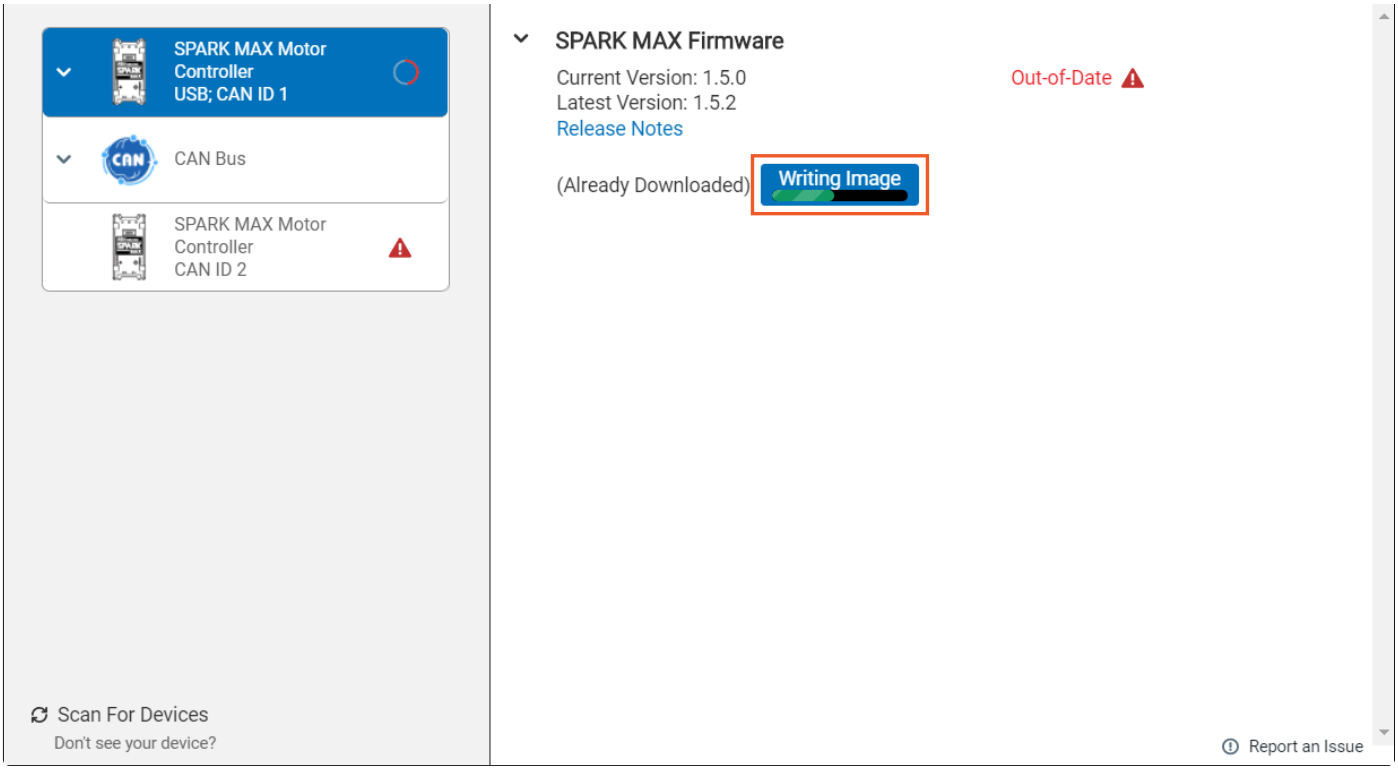




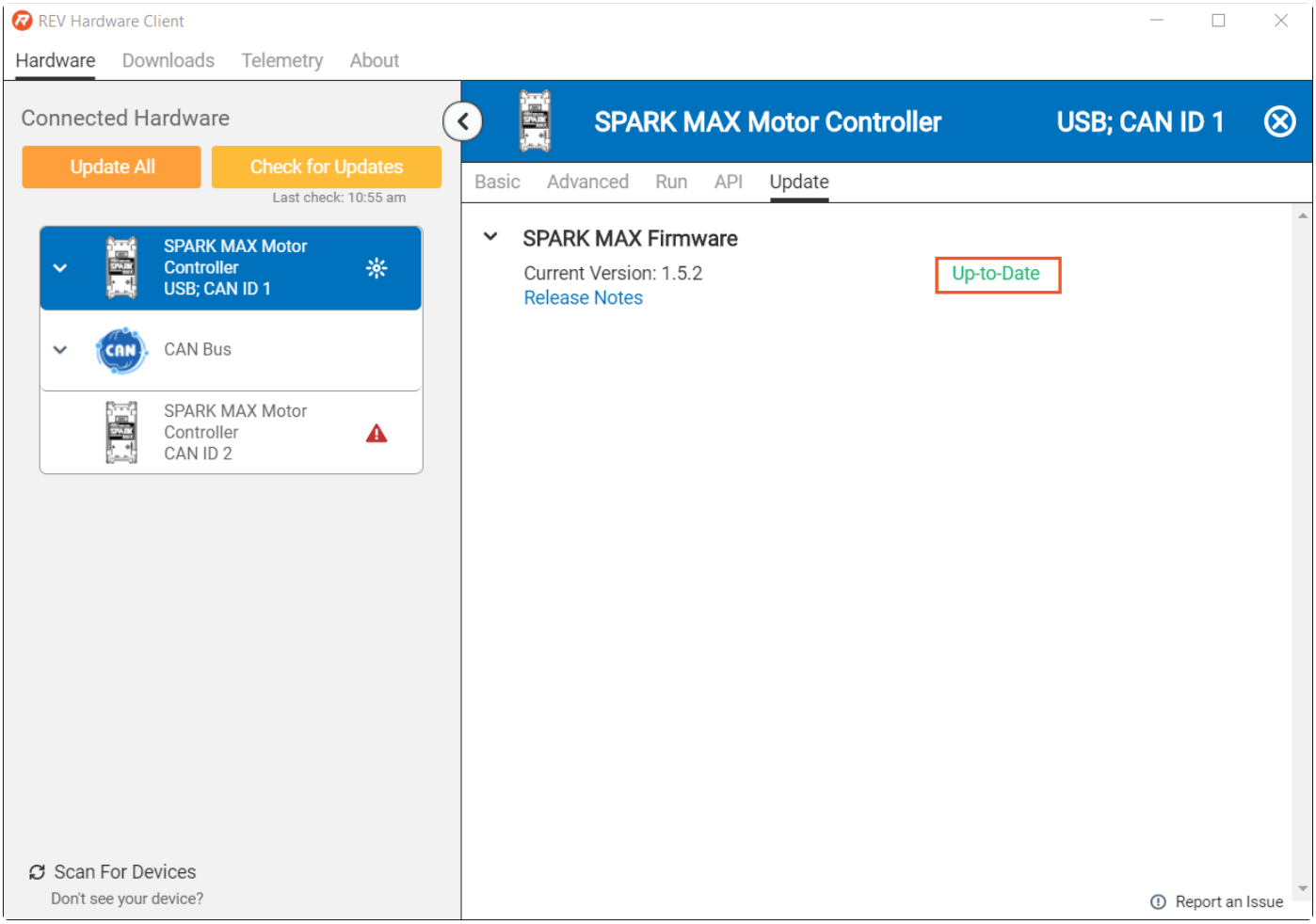

Once the firmware has downloaded select update.




The update process will flash the firmware image onto the SPARK MAX. The status bar will show the progress of the process.




Once the firmware update is done your SPARK MAX will show a new status of **Up-to-Date.**



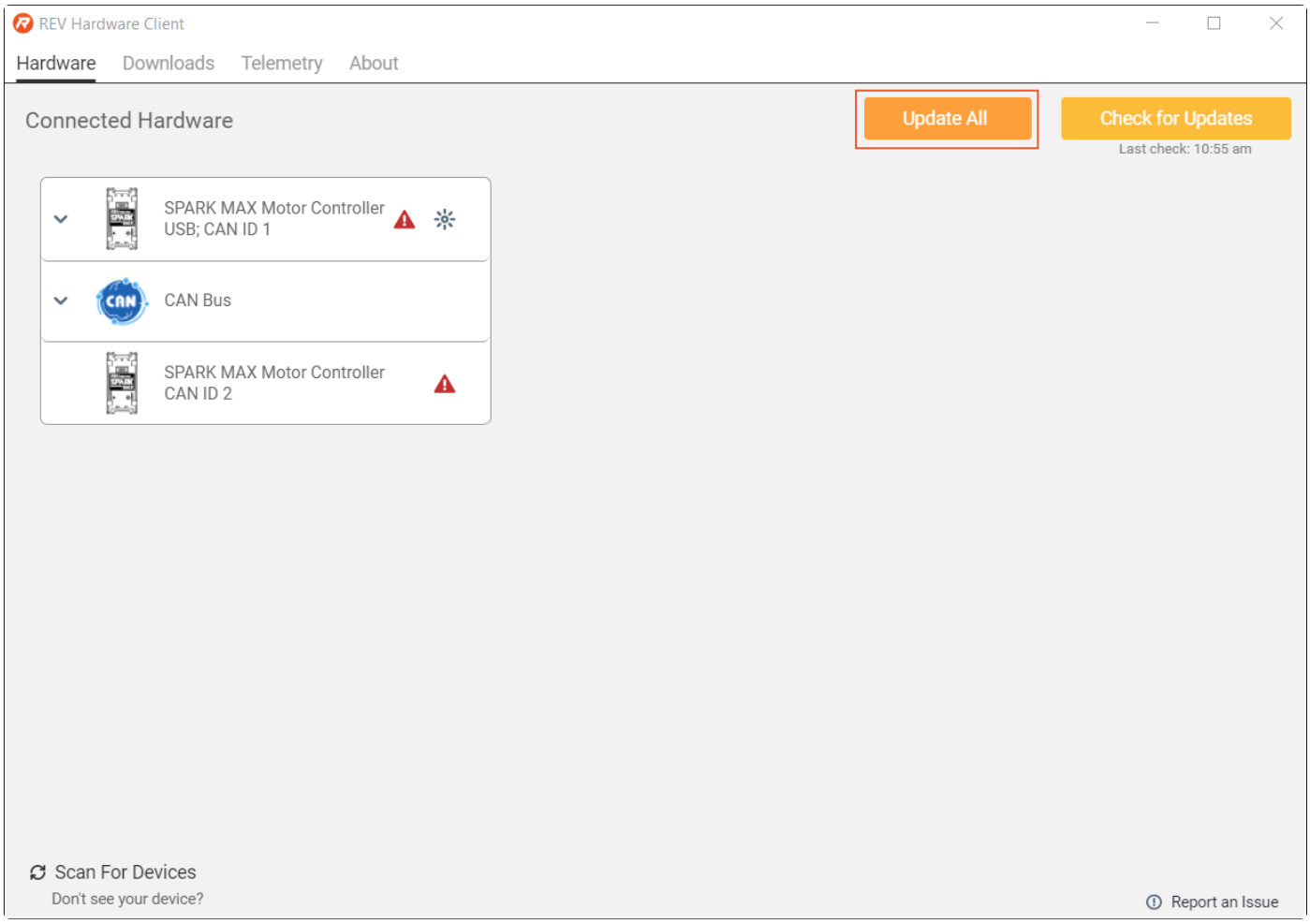

> If your SPARK MAX is running firmware older than 1.4.0, you may need to unplug and replug the USB-C cable into the SPARK MAX for it to reconnect to the Client.

### Updating Multiple Devices with the USB-to-CAN Bridge

SPARK MAX Firmware Version 1.5.0 includes a USB-to-CAN Bridge feature that allows a single USB-connected SPARK MAX to act as a bridge to the entire CAN bus it is connected to. This allows for configuration and simultaneous updating of multiple SPARK MAX controllers without having to connect to each one individually. Using this feature requires the following:

- A USB-connected SPARK MAX that is updated to firmware version 1.5.0 or newer to act as the Bridge.
- Other SPARK MAXs connected on the CAN bus must be individually updated to firmware version 1.4.0 before they are able to receive mass-updates from the Bridging SPARK MAX.

Once these requirements are satisfied, navigate to the **Hardware** tab, select the **Update All** button.




Each device with the Out-of-Date warning will update with the latest version of the firmware.

# Software Resources

## SPARK MAX Firmware Change Log

It is recommended to keep your SPARK MAX up-to-date with the latest firmware. The REV Hardware Client application will automatically download the latest firmware, but you can also download the firmware manually below:

| Latest Firmware - Version 1.5.2 |
| --- |
| Download Latest Firmware |

> ⓘ This firmware **will not work with SPARK MAX beta hardware units** distributed by REV to the SPARK MAX Beta testers. It is only compatible with units received after 12/20/2018.

Latest Firmware Change Log - Version 1.5.2

- Fixes issue with the *kDataPortConfig* parameter not enabling Alternate Encoder Mode on a power cycle after it has been configured and saved to flash.
- Improves Gate Driver Fault recovery.

Previous Firmware Change Log - Version 1.5.1

- Improvements to BLDC commutation timing.
  - Most noticeable at high RPMs only achievable by the NEO 550.
- Fixes rare case where the SPARK MAX Status LED shows normal driving but no actual output is occurring until a reboot of the controller.

Previous Firmware Change Log - Version 1.5.0

- Version 1.5.0 Changes
  - Adds a unique hash key to the firmware. This key is a hashed value based on the unique 96-bit device ID guaranteed to be unique for every STM32 device. The hash value itself is 32-bits to make transmission/reception and on-board comparison easier. It is unlikely for a key collision especially since most buses have only a small number of devices.
  - Adds ID Query command to have all devices which match the CAN ID sent to return their unique hash key. This is typically used to identify all devices with CAN ID 0 but can be used to identify devices with conflicting CAN IDs.
  - Adds ID Assign command, which sends a unique hash key and a CAN ID, which is received by any device whose CAN ID matches the ID field. The device whose unique hash matches has their own CAN ID set to the desired value. This is for initial or automatic provisioning, or to re-address devices with the same CAN ID.
  - Adds identify command which flashes the LED blue/magenta. This command uses the CAN ID, or the unique hash if CAN ID = 0.
  -

Complete overhaul of USB interface, creating a generic USB-to-CAN driver connection to act as the bridge to the interface.
  - Adds Alternate Encoder Mode for BLDC external encoder support.
  - Adds device manufacturing info to firmware frame.
  - Adds retry frame to CAN bootloader if there is an issue.
- Version 1.4.1 Changes
  - Fixes issue where creating a CANSparkMax object would cause the SensorType parameter to be set to NoSensor.

Previous Firmware Change Log - Version 1.4.0

- **Version 1.4.0 Changes**
  - Adds non-competition heartbeat command when not used with a roboRIO.
  - Adds Raspbian support using official WPILib tools.
  - Adds locking mechanism to prevent simultaneous USB and CAN commands.
  - Proportional blink codes now match inversion settings (i.e. positive input is always green, and negative input is always red).
  - Limit switch and soft limits now follow inversion settings.
  - Motor controller inversion now in the firmware instead of the API.
  - Adds a configurable range for absolute feedback devices (currently only applicable to an absolute mode analog sensor), which also prevents users from setting a setpoint out of range.
  - Adds ability to use both absolute and relative analog sensors as feedback devices
  - Adds filtering for Analog sensor with settings for velocity moving average filter
  - Adds ability to configure how errors are tracked and handled by the user.
    - Calls can be automatically registered and tracked, with any errors displayed to the DriverStation or users can use the *GetLastError()* after calls to determine if an error has been thrown. This is done by changing the error timeout through *SetCANTimeout()*, where a timeout of 0 means that the calls are non-blocking, and errors are checked in a separate thread and sent through to the driver station.
  - Other minor improvements and bug fixes.
- **Version 1.3.0 Changes**
  - Ability to configure the feedback device for the PIDController.
  - Addition of CANAnalog which will function as a possible feedback device.
  - Added API for using encoders with brushed DC motors.
  - Adds API to enable and set soft limits.
  - All control modes now reset integrator on limit switch activation for long as the limit switch is held.
  - Smart Motion now honors acceleration rate after limit switch changes from triggered to not triggered.
  - Fixes issue where sticky faults can be cleared incorrectly when a new fault is set and the old fault is no longer present.
  -

    - Adds status 3 periodic frame for analog sensor.
    - Other minor improvements and bug fixes.
- **Version 1.2.1 Changes**
    - Adds initial CAN bootloading functionality.
        - Requires continuous power during update and requires USB recovery if the update fails after erasing the flash (i.e. power is lost during update).
        - This is not yet integrated into a formal tool, but is exposed in the API. Future versions will allow recovery over CAN.
    - Fix to bus voltage measurement, previous version reported a lower value.
    - Additional accuracy improvements to all ADC measurements.
    - Adds additional filtering to bus voltage.
    - Fix for Follow settings not being reapplied after power cycle.
    - Adds ability to change units for arbFF between voltage and percent bus voltage (or percent compensated voltage).

Previous Firmware Change Log - Version 1.1.33

- Fixes issue that causes a Gate Driver Fault if the NEO is spinning at a certain speed during power up.
    - e.g. while still moving after a breaker trips for more than ~1.4s after breaker recovery.
- Fixes issue where configuration data can be corrupted under a unique set of conditions, including loss of power while configuration settings are being saved to flash memory.
    - Issue symptoms: Controller cannot be controlled by a normal address and would report a firmware version of 0.0.0 in the Client, even after a successful firmware update. **Updating a controller that is in this state to 1.1.33 will recover the controller.**

Previous Firmware Change Log - Version 1.1.31

- Adds new Smart Velocity mode - uses same constrains as the Smart Motion mode but outputs velocity instead of position. This is not a motion profiling mode, but rather provides acceleration limiting as opposed to simple ramp rates.

- Improvements to heartbeat implementation.
- CAN ID of 0 now considered 'unconfigured' and will not be enabled.
- Changes the default output range for all PID slots to be [-1,1] instead of [0,0]
- Changes setpoint commands to 'stick' instead of relying on periodic frames from the controller.
- Follow mode can now persists through a power-cycle.
- Periodic Status 2 (position data) frame rate default changed from 50ms to 20ms.
- Watchdog changed to 220ms.
- Proportional blink codes while driving now have a minimum blink rate (minimum is same as 10% applied output blink rate).
- Fault flag 'over voltage' replaced with 'IWDT Reset'.

- Velocity and Position conversion factors are now used for all internal PID calculations.

This firmware update requires an API update. Please see the API Information section for the latest updates. The table below outlines the compatibility between firmware versions and API versions:

| | | API | |
|---|---|---|---|
| | | LabVIEW - 1.0.26<br>Java - 1.0.28<br>C++ - 1.0.27 | LabVIEW - 1.1.8 (or newer)<br>Java - 1.1.8 (or newer)<br>C++ - 1.1.8 (or newer) |
| **Firmware** | 1.1.26 (or newer) | Not Compatible | Compatible |
| | 1.0.385 (or older) | Compatible | Not Compatible |

Previous Firmware Change Log - Version 1.1.26 (beta)

- Adds new Smart Motion mode utilizing trapezoidal motion profiling
- Improvements to velocity decoding for low speeds and associated PID control loops requiring control at low speeds
- Improvements to gate driver fault handling
- Adds voltage compensation mode
- Adds ability to set the stored sensor position
- Adds ability to set the maximum PID integral accumulator value to prevent integral windup
- Adds ability to set the PID I accumulator value
- Adds filter function to PID derivative value
- Adds closed-loop current control
- Adds closed-loop ramp rate separate from open-loop
- Adds ability to follow Phoenix 4.11 motor controllers
  - *Note:* This feature depends on the firmware of the Phoenix controllers, and is not controlled by REV. We will do our best to build this functionality, but this is not an officially supported feature.
- Adds ability to set a unit conversion for both velocity and position units
- Adds ability to reset to factory defaults
- Improvements to smart current limits
- Improvements to current data sent over CAN
- Improvements to internal settings table implementation
- Calling the constructor no longer resets the encoder count. This must be done manually by running CANEncoder.setPosition()

Previous Firmware Change Log - Version 1.0.385

- Fixes PWM control issue introduced by version 1.0.384.

Previous Firmware Change Log - Version 1.0.384

- Fixes GetPostition() randomly returning 0.
- Fixes momentary velocity spike when accelerating from 0 speed.
- Fixes issue related to rapid control mode switching by the user.
- Fixes issue related to extreme low voltage CAN bus operation.
- Adds conversion for motor temperature.
- Improvement to "Has Reset" flag.
- Improvements to brownout fault indicator.
- Improvements to brownout behavior.

---

## Factory Images

When updating the SPARK MAX Firmware we recommend using the REV Hardware Client and the latest firmware file listed in the firmware changelog. When updating normally, key configuration parameters, such as the CAN ID, will be preserved through an update, preventing the need to reconfigure every time you update. In some rare instances it may be beneficial to completely reinstall the factory firmware and reset all configuration parameters. Factory images can be found below:

- Factory Image Version 1.1.33
- Factory Image Version 1.1.31
- Factory Image Version 1.0.385
- Factory Image Version 1.0.381

## SPARK MAX API Information

Below you will find information on how to download and install the SPARK MAX APIs for LabVIEW, Java, and C++.

| **Language** | Current API Version | Documentation |
|---|---|---|
| LabVIEW | 1.5.2 | Embedded (Press Ctrl-H) |
| Java | 1.5.4 | Java Docs |
| C++ | 1.5.4 | C++ Docs |

Latest API Change Log - Version 1.5.4 - Java/C++ Only

- Updated dependencies for WPILib 2021.1.2
- Adds basic simulation support

- Adds MacOS X support
- Fixes issue effecting some C++ projects when using static initialization for the SPARK MAX object

Latest API Change Log - Version 1.5.2 - Java/C++ Only

- Fixes possible crash when no CAN bus connected when code is first run.
- Adds the CAN ID of the device to every error message.
- Reduces the timing for error messages, and removes repeated messages to make error diagnostics more clear.
- Makes the base class rev::CANSensor object public to help with some Kotlin wrappers.
- Sends an error to the console if two CANSparkMax objects are created with the same CAN ID.

Latest API Change Log - Version 1.5.2 - LabVIEW Only

- Fixes issue with Follow.vi not deploying correctly.

Previous API Change Log - Version 1.5.1 - Java/C++

- Updated dependencies for WPILib 2020.1.2
- Removed extra thread for setpoint calls.
    - *Set* calls now directly call *CAN,* reducing CAN latency.
- Adds *SetVoltage()* override for the WPILib SpeedController class.

Previous API Change Log - Version 1.5.1 - LabVIEW

- Update to LabVIEW 2019 (FRC 2020)

Previous API Change Log - Version 1.5.0 - All Languages

- Adds the ability to use an alternate encoder as a feedback device when connected to the SPARK MAX Data Port. When using an alternate encoder, Hard Limit Switches cannot be used on that SPARK MAX.
- Adds the ability to send telemetry data back from a SPARK MAX. This is done by opening a Telemetry Stream for a particular subset of telemetry data (categorized by TelemetryIDs). A TelemetryMessage contains the ID, value, timestamp, name, units, and bounds of a particular TelemetryID. Additionally, users can list what subset of telemetry data is available for each SPARK MAX.
- Adds a DeviceScanner that will scan the CANBus for other CAN Devices. Will filter what devices to look for, and users can specify filters to let through different devices.
- Addresses bug with CANEncoder backwards compatibility.
- Addresses bug with the Java Control Frame Period not being properly set.

Previous API Change Log - Version 1.4.1 - Java/C++ Only

- Fixes issue in Java/C++ where creating a *CANSparkMax* object causes the Sensor Type parameter to be set to *NoSensor*. Desired behavior is to leave the existing setting.

Previous API Change Log - Version 1.4.0 - All Languages

- **Version 1.4.0 Changes**
  - Adds non-competition heartbeat command when not used with a roboRIO.
  - Adds Raspbian support using official WPILib tools.
  - Adds locking mechanism to prevent simultaneous USB and CAN commands.
  - Motor controller inversion now in the firmware instead of the API.
  - Adds a configurable range for absolute feedback devices (currently only applicable to an absolute mode analog sensor), which also prevents users from setting a setpoint out of range.
  - Adds ability to use both absolute and relative analog sensors as feedback devices
  - Adds filtering for Analog sensor with settings for velocity moving average filter
  - Adds ability to configure how errors are tracked and handled by the user.
    - Calls can be automatically registered and tracked, with any errors displayed to the DriverStation or users can use the *GetLastError()* after calls to determine if an error has been thrown. This is done by changing the error timeout through *SetCANTimeout()*, where a timeout of 0 means that the calls are non-blocking, and errors are checked in a separate thread and sent through to the driver station.
  - Other minor improvements and bug fixes.
- **Version 1.3.0 Changes**
  - Ability to configure the feedback device for the PIDController.
  - Addition of CANAnalog which will function as a possible feedback device.
  - Added API for using encoders with brushed DC motors.
  - Vendor dependencies now allows building Windowsx86-64 based build target.
  - Moves entire implementation to C level, C++ and Java libraries are now simple wrappers.
  - Adds API to enable and set soft limits.
  - Fixes possible driver set out of bounds if given a CAN ID less than 0.
  - *CANSparkMaxLowLevel::SetEncPosition()* and *CANSparkMaxLowLevel::SetIAccum()* have been moved to protected and should be accessed via their respective objects: *CANEncoder.SetEncoderPosition()* and *CANPIDController::SetIAccum()*.
  - *CANSparkMaxLowLevel::SetParameter** and G*etParameter** are changed to private.
  - Adds status 3 periodic frame for analog sensor.
  - Other minor improvements and bug fixes.
- **Version 1.2.1 Changes**
  - Adds initial CAN bootloading functionality.
    - Requires continuous power during update and requires USB recovery if the update fails after

erasing the flash (i.e. power is lost during update).
    - This is not yet integrated into a formal tool, but is exposed in the API. Future versions will allow recovery over CAN.
- Adds ability to change units for arbFF between voltage and percent bus voltage (or percent compensated voltage).
- API adds check if its own version is too old based on major and minor revision numbers.
- Adds version number of API.

Previous API Change Log - Version 1.1.9 - LabVIEW

- Fixes potential hang in Follow.vi in the case that an error is on the input terminal. Java/C++ unaffected.

Previous API Change Log - Version 1.1.9 - Java/C++ Only

- Fixes loop overrun issues in Java/C++. LabVIEW unaffected.

Previous API Change Log - Version 1.1.8 - All Languages

- Adds Smart Velocity Mode
- Fixes inversion issues for IMaxAccum and Output Range

- Updates heartbeat implementation
- Adds default slotID for setDFilter() functions
- Update heartbeat rate from 50ms to 25ms
- Changes fault flag for Overvoltage to IWDTReset

This API update also requires a firmware update. Please see the SPARK MAX Firmware Updates section for the latest updates. The table below outlines the compatibility between firmware versions and API versions:

| | | **API** | |
|---|---|---|---|
| | | LabVIEW - 1.0.26<br>Java - 1.0.28<br>C++ - 1.0.27 | LabVIEW - 1.1.8 (or newer)<br>Java - 1.1.8 (or newer)<br>C++ - 1.1.8 (or newer) |
| **Firmware** | 1.1.26 (or newer) | Not Compatible | Compatible |
| | 1.0.385 (or older) | Compatible | Not Compatible |

Previous API Change Log - Version 1.1.5 (beta) - Java/C++ - Version 1.1.4 (beta) - LabVIEW

- Version 1.1.5
    - Adds method to stop any REV periodic frames before enabling heartbeat
    - Improvements to heartbeat protocol
    - Moved all heartbeat implementation into its own dependency called by C++ and Java through JNI

    - Removed setControlFrameRate() call and removed repeating packets
        - Updates to control frames are now as fast/slow as the user calls Set() or SetReference()
    - Updates to doxygen/javadocs comments
    - Maven artifacts now include source for both Java and C++
- Version 1.1.3
    - Improvements to setInverted()
    - New API calls for new firmware features
    - Better error message in Java when CAN is not connected
    - Documentation updates are in progress.

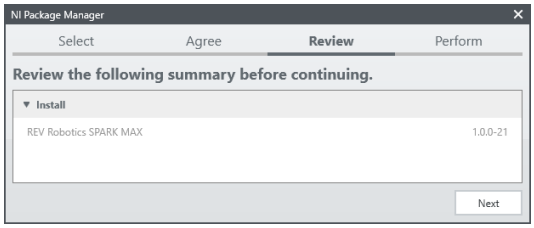
---

# LabVIEW API

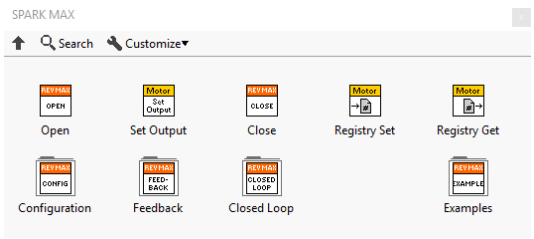
| Latest LabVIEW API - Version 1.5.2 |
| --- |
| Download Latest LabVIEW API |

### LabVIEW API Installation Instructions

1. Download the latest API package from the download link above.
2. Make sure LabVIEW for FRC 2021 is installed and updated.
3. Open the REV SPARK MAX LabVIEW Package. The NI Package Manager should automatically open.
4. Click **Next**:




5. Once the installation is complete, you will be able to access the REV SPARK MAX VIs in the **LabVIEW Functions Pallet -> WPI Robotics Library -> Third Party -> REV Robotics -> SPARK MAX**.




---

# Java API

| Latest JAVA API - Version 1.5.4 |
| --- |
| Download Latest JAVA API |

**Java API Installation Instructions**

**Online Installation**

You can use the online method to install the REV Robotics Java API if your development machine is connected to the internet:

1. Open your robot project in VSCode.
2. Click on the WPI icon in the corner to open the WPI Command Pallet.
3. Select **Manage Vendor Libraries**.
4. Select **Install new library (online)**.
5. Enter the following installation URL and press ENTER:

   https://www.revrobotics.com/content/sw/max/sdk/REVRobotics.json

**Offline Installation**

Latest Java API - Version 1.5.4

1. Download and unzip the latest SPARK MAX Java API into the *C:\Users\Public\wpilib\2021* directory on windows and *~/wpilib/2021* directory on Unix systems.
2. Follow the Adding an offline-installed Library instructions from WPI.

Java API Documentation

For a list and description of all classes:

- Online SPARK MAX Java Documentation
- Offline SPARK MAX Java Documentation (pdf)

---

# C++ API

| Latest C++ API - Version 1.5.4 |
| --- |
| Download Latest C++ API |

**C++ API Installation Instructions**

**Online Installation**

You can use the online method to install the REV Robotics C++ API if your development machine is connected to the internet:

1. Open your robot project in VSCode.
2. Click on the WPI icon in the corner to open the WPI Command Pallet.
3. Select **Manage Vendor Libraries**.
4. Select **Install new library (online)**.
5. Enter the following installation URL and press ENTER:

   https://www.revrobotics.com/content/sw/max/sdk/REVRobotics.json

**Offline Installation**

Latest C++ API - Version 1.5.4

1. Download and unzip the latest SPARK MAX C++ API into the *C:\Users\Public\wpilib\2021* directory on windows and *~/wpilib/2021* directory on Unix systems.
2. Follow the Adding an offline-installed Library instructions from WPI.

C++ API Documentation

For a list and description of all classes:

- Online SPARK MAX C++ Documentation
- Offline SPARK MAX C++ Documentation (pdf)

## SPARK MAX Code Examples

At the links below you will find code examples in LabVIEW, Java, and C++ for common SPARK MAX control modes. We will be adding examples as we develop them, so please check back regularly. Examples included as of 2/6/19:

- LabVIEW Code Examples
  - Alternate Encoder
  - Arcade Drive with CAN
  - Arcade Drive with PWM
  - Bus Measurements
  - Get and Set Parameters

    - Limit Switch
    - Position Closed Loop Control
    - Read Encoder Values
    - Velocity Closed Loop Control
- Java Code Examples
    - Alternate Encoder
    - Analog Feedback Device
    - Bus Measurements
    - Encoder Feedback Device
    - Get and Set Parameters
    - Limit Switch
    - Motor Follower
    - Position Closed Loop Control
    - Read Encoder Values
    - Smart Motion Example
    - Soft Limits
    - Tank Drive With CAN
    - Velocity Closed Loop Control
- C++ Code Examples
    - Alternate Encoder
    - Analog Feedback Device
    - Arcade Drive With CAN
    - Arcade Drive With PWM
    - Bus Measurements
    - Encoder Feedback Device
    - Get and Set Parameters
    - Limit Switch
    - Position Closed Loop Control
    - Read Encoder Values
    - Smart Motion Example
    - Soft Limits
    - Velocity Closed Loop Control

Even though some examples may only exist in a particular language, they can be a good place to start as the APIs are very similar between languages.

SPARK MAX Code Examples

## Configuration Parameters

Below is a list of all the configurable parameters within the SPARK MAX. Parameters can be set through the CAN or USB interfaces. The parameters are saved in a different region of memory from the device firmware and persist through a firmware update.

| Name | ID | Type | Unit | Default | Description |
|---|---|---|---|---|---|
| kCanID | 0 | uint | - | 0 | CAN ID<br><br>This parameter persists through a normal firmware update. |
| kInputMode | 1 | Input Mode | - | 0 | Input mode, this parameter i read only ar the input mode is detected by the firmware automatical 0 - PWM 1 - CAN 2 - USB |
| kMotorType | 2 | Motor Type | - | BRUSHLES S | Motor type:<br><br>0 - Brushed 1 - Brushles<br><br>This parameter persists through a normal firmware update. |

| | | | | | |
|---|---|---|---|---|---|
| Reserved | 3 | - | - | | Reserved |
| kSensorType | 4 | Sensor Type | - | HALL_EFFECT | Sensor type<br>0 - No Sens<br>1 - Hall Sensor<br>2 - Encoder<br>This parameter persists through a normal firmware update. |
| kCtrlType | 5 | Ctrl Type | - | CTRL_DUTY_CYCLE | Control Typ<br>this is a rea only parameter c the currentl active contr type. The control type changed by calling the correct API.<br>0 - Duty Cycle<br>1 - Velocity<br>2 - Voltage<br>3 - Position |
| kIdleMode | 6 | Idle Mode | - | IDLE_COAST | State of the half bridge when the motor controller commands zero output is disabled.<br>0 - Coast<br>1 - Brake<br>This parameter persists through a normal firmware update. |

| | | | | | |
|---|---|---|---|---|---|
| kInputDeadband | 7 | float32 | % | 0.05 | Percent of t input which results in ze output for PWM mode This parameter persists through a normal firmware update. |
| Reserved | 8 | - | - | - | Reserved |
| Reserved | 9 | - | - | - | Reserved |
| kPolePairs | 10 | uint | - | 7 | Number of pole pairs fo the brushles motor. This the number poles/2 and can be determined by either counting the number of magnets or counting the number of windings ar dividing by This is an important term for speed regulation t properly calculate th speed. |
| | | | | | If the half bridge detects this current limit will disable the motor driver for a |

| | | | | | |
|---|---|---|---|---|---|
| kCurrentChop | 11 | float32 | Amps | 115 | fixed amour of time set b kCurrentCh pCycles. Th is a low sophisticati 'current control'. Set to 0 to disable. The max value i 125. |
| kCurrentChop Cycles | 12 | uint | - | 0 | Number of PWM Cycle for the h-bridge to be off in the ca that the current limit set. Min = 1 multiples of PWM perio (50µs). During this time the current will recirculating through the low side MOSFETs, instead of 'freewheelin the diodes, the bridge w be in brake mode durin this time. |
| kP_0 | 13 | float32 | - | 0 | Proportiona gain consta for gain slot |
| kI_0 | 14 | float32 | - | 0 | Integral gai constant for gain slot 0. |
| kD_0 | 15 | float32 | - | 0 | Derivative gain consta for gain slot |

| | | | | | |
|---|---|---|---|---|---|
| kF_0 | 16 | float32 | - | 0 | Feed Forward ga constant for gain slot 0. |
| kIZone_0 | 17 | float32 | - | 0 | Integrator zone consta for gain slot The PIDF loop integrator w only accumulate while the setpoint is within IZone of the target |
| kDFilter_0 | 18 | float32 | - | 0 | PIDF derivative filter consta for gain slot |
| kOutputMin_0 | 19 | float32 | - | -1 | Max output constant for gain slot 0. This is the max output the controlle |
| kOutputMax_0 | 20 | float32 | - | 1 | Min output constant for gain slot 0. This is the min output the controlle |
| kP_1 | 21 | float32 | - | 0 | Proportiona gain consta for gain slot |
| kI_1 | 22 | float32 | - | 0 | Integral gai constant for gain slot 1. |
| kD_1 | 23 | float32 | - | 0 | Derivative gain consta for gain slot |
| | | | | | Feed |

| | | | | | |
|---|---|---|---|---|---|
| kF_1 | 24 | float32 | - | 0 | Forward ga constant for gain slot 1. |
| kIZone_1 | 25 | float32 | - | 0 | Integrator zone consta for gain slot The PIDF loop integrator w only accumulate while the setpoint is within IZone of the target |
| kDFilter_1 | 26 | float32 | - | 0 | PIDF derivative filter consta for gain slot |
| kOutputMin_1 | 27 | float32 | - | -1 | Max output constant for gain slot 1. This is the max output the controlle |
| kOutputMax_1 | 28 | float32 | - | 1 | Min output constant for gain slot 1. This is the min output the controlle |
| kP_2 | 29 | float32 | - | 0 | Proportiona gain consta for gain slot |
| kI_2 | 30 | float32 | - | 0 | Integral gai constant for gain slot 2. |
| kD_2 | 31 | float32 | - | 0 | Derivative gain consta for gain slot |
| kF_2 | 32 | float32 | - | 0 | Feed Forward ga |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | constant for gain slot 2. |
| kIZone_2 | 33 | float32 | - | 0 | integrator zone consta for gain slot The PIDF loop integrator w only accumulate while the setpoint is within IZone of the target |
| kDFilter_2 | 34 | float32 | - | 0 | PIDF derivative filter consta for gain slot |
| kOutputMin_2 | 35 | float32 | - | -1 | Max output constant for gain slot 2. This is the max output the controlle |
| kOutputMax_2 | 36 | float32 | - | 1 | Min output constant for gain slot 2. This is the min output the controlle |
| kP_3 | 37 | float32 | - | 0 | Proportiona gain consta for gain slot |
| kI_3 | 38 | float32 | - | 0 | Integral gai constant for gain slot 3. |
| kD_3 | 39 | float32 | - | 0 | Derivative gain consta for gain slot |
| kF_3 | 40 | float32 | - | 0 | Feed Forward ga constant for gain slot 3. |
| | | | | | Integrator |

| | | | | | |
|---|---|---|---|---|---|
| kIZone_3 | 41 | float32 | - | 0 | zone consta for gain slot The PIDF loop integrator w only accumulate while the setpoint is within IZone of the target |
| kDFilter_3 | 42 | float32 | - | 0 | PIDF derivative filter consta for gain slot |
| kOutputMin_3 | 43 | float32 | - | -1 | Max output constant for gain slot 3. This is the max output the controlle |
| kOutputMax_3 | 44 | float32 | - | 1 | Min output constant for gain slot 3. This is the min output the controlle |
| Reserved | 45 | - | - | - | Reserved |
| Reserved | 46 | - | - | - | Reserved |
| Reserved | 47 | - | - | - | Reserved |
| Reserved | 48 | - | - | - | Reserved |
| Reserved | 49 | - | - | - | Reserved |
| kLimitSwitchFwdPolarity | 50 | bool | - | 0 | Forward Lir Switch polarity. 0 - Normally Open 1 - Normally Closed |
| | | | | | Reverse Lir Switch |

| | | | | | |
|---|---|---|---|---|---|
| kLimitSwitchRevPolarity | 51 | bool | - | 0 | polarity.<br>0 - Normally Open<br>1 - Normally Closed |
| kHardLimitFwdEn | 52 | bool | - | 1 | Limit switch enable, enabled by default |
| kHardLimitRevEn | 53 | bool | - | 1 | Limit switch enable, enabled by default |
| Reserved | 54 | - | - | - | Reserved |
| Reserved | 55 | - | - | - | Reserved |
| kRampRate | 56 | float32 | V/s | 0 | Voltage ram rate active f all control modes in % output per second, a value of 0 disables thi feature. All APIs take th reciprocal to make the ur 'time from 0 full'. |
| kFollowerID | 57 | uint | - | 0 | CAN EXTID of the message w data to follo |
| | | | | | Special configuratio register for setting up to follow on a repeating message (follower mode). CFG[0] to CFG[3] whe |

| | | | | | |
|---|---|---|---|---|---|
| kFollowerConfig | 58 | uint | - | 0 | CFG[0] is th motor outpu start bit (LSB), CFG[1] is th motor outpu stop bit (MSB). CFG[0] - CFG[1] determines endianness CFG[2] bits determine sign mode and inverte CFG[3] sets preconfigur controller (0x1A = RE 0x1B = Talon/Victo style as of 2018 seaso |
| kSmartCurrentStallLimit | 59 | uint | A | 80 | Smart Curre Limit at stall or any RPM less than kSmartCurr tConfig RPM |
| kSmartCurrentFreeLimit | 60 | uint | A | 20 | Smart curre limit at free speed |
| kSmartCurrentConfig | 61 | uint | - | 10000 | Smart curre limit RPM value to sta linear reduction of current limit Set this > fr speed to disable. |
| Reserved | 62 | - | - | - | Reserved |
| Reserved | 63 | - | - | - | Reserved |
| | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reserved | 64 | - | - | - | Reserved |
| Reserved | 65 | - | - | - | Reserved |
| Reserved | 66 | - | - | - | Reserved |
| Reserved | 67 | - | - | - | Reserved |
| Reserved | 68 | - | - | - | Reserved |
| kEncoderCountsPerRev | 69 | uint | - | 4096 | Number of encoder counts in a single revolution, counting every edge on the A an B lines of a quadrature encoder. (Note: This different tha the CPR sp of the encoder which is 'Cycles per revolution'. This value i 4 * CPR. |
| kEncoderAverageDepth | 70 | uint | - | 64 | Number of samples to average for velocity dat based on quadrature encoder input. This value can b between 1 and 64. |
| | | | | | Delta time value for encoder velocity measureme in 500µs increments. |

| | | | | | |
|---|---|---|---|---|---|
| kEncoderSampleDelta | 71 | uint | per 500us | 200 | The velocity calculation will take del the current sample, and the sample * 500µs behind, and divide by th the sample delta time. Can be any number between 1 and 255 |
| Reserved | 72 | - | - | - | Reserved |
| Reserved | 73 | - | - | - | Reserved |
| Reserved | 74 | - | - | - | Reserved |
| kCompensatedNominalVoltage | 75 | float32 | V | 0 | In voltage compensation mode mode, this i the max scaled voltage. |
| kSmartMotionMaxVelocity_0 | 76 | float32 | - | 0 | |
| kSmartMotionMaxAccel_0 | 77 | float32 | - | 0 | |
| kSmartMotionMinVelOutput_0 | 78 | float32 | - | 0 | |
| kSmartMotionAllowedClosedLoopError_0 | 79 | float32 | - | 0 | |
| kSmartMotionAccelStrategy_0 | 80 | float32 | - | 0 | |
| kSmartMotionMaxVelocity_1 | 81 | float32 | - | 0 | |

| | | | | | |
|---|---|---|---|---|---|
| kSmartMotionMaxAccel_1 | 82 | float32 | - | 0 | |
| kSmartMotionMinVelOutput_1 | 83 | float32 | - | 0 | |
| kSmartMotionAllowedClosedLoopError_1 | 84 | float32 | - | 0 | |
| kSmartMotionAccelStrategy_1 | 85 | float32 | - | 0 | |
| kSmartMotionMaxVelocity_2 | 86 | float32 | - | 0 | |
| kSmartMotionMaxAccel_2 | 87 | float32 | - | 0 | |
| kSmartMotionMinVelOutput_2 | 88 | float32 | - | 0 | |
| kSmartMotionAllowedClosedLoopError_2 | 89 | float32 | - | 0 | |
| kSmartMotionAccelStrategy_2 | 90 | float32 | - | 0 | |
| kSmartMotionMaxVelocity_3 | 91 | float32 | - | 0 | |
| kSmartMotionMaxAccel_3 | 92 | float32 | - | 0 | |
| kSmartMotionMinVelOutput_3 | 93 | float32 | - | 0 | |
| kSmartMotionAllowedClosedLoopError_3 | 94 | float32 | - | 0 | |
| kSmartMotionAccelStrategy_3 | 95 | float32 | - | 0 | |

| | | | | | |
|---|---|---|---|---|---|
| kIMaxAccum_0 | 96 | float32 | - | 0 | |
| kSlot3Placeholder1_0 | 97 | float32 | - | 0 | |
| kSlot3Placeholder2_0 | 98 | float32 | - | 0 | |
| kSlot3Placeholder3_0 | 99 | float32 | - | 0 | |
| kIMaxAccum_1 | 100 | float32 | - | 0 | |
| kSlot3Placeholder1_1 | 101 | float32 | - | 0 | |
| kSlot3Placeholder2_1 | 102 | float32 | - | 0 | |
| kSlot3Placeholder3_1 | 103 | float32 | - | 0 | |
| kIMaxAccum_2 | 104 | float32 | - | 0 | |
| kSlot3Placeholder1_2 | 105 | float32 | - | 0 | |
| kSlot3Placeholder2_2 | 106 | float32 | - | 0 | |
| kSlot3Placeholder3_2 | 107 | float32 | - | 0 | |
| kIMaxAccum_3 | 108 | float32 | - | 0 | |
| kSlot3Placeholder1_3 | 109 | float32 | - | 0 | |
| kSlot3Placeholder2_3 | 110 | float32 | - | 0 | |
| kSlot3Placeholder3_3 | 111 | float32 | - | 0 | |
| kPositionConversionFactor | 112 | float32 | - | 1 | |
| kVelocityConversionFactor | 113 | float32 | - | 1 | |

| | | | | | |
|---|---|---|---|---|---|
| kClosedLoopRampRate | 114 | float32 | DC/sec | 0 | |
| kSoftLimitFwd | 115 | float32 | - | 0 | Soft limit forward valu |
| kSoftLimitRev | 116 | float32 | - | 0 | Soft limit reverse valu |
| Reserved | 117 | - | - | - | Reserved |
| Reserved | 118 | - | - | - | Reserved |
| kAnalogPositionConversion | 119 | float32 | rev/volt | 1 | Conversion factor for position fro analog sensor. This value is multiplied b the voltage give an output value |
| kAnalogVelocityConversion | 120 | float32 | vel/v/s | 1 | Conversion factor for velocity fro analog sensor. This value is multiplied b the voltage give an output value |
| kAnalogAverageDepth | 121 | uint | - | 0 | Number of samples in moving average of velocity. |
| | | | | | 0 Absolute: this mode th sensor position is always read as voltage * conversion factor and reads the |

| | | | | | |
|---|---|---|---|---|---|
| kAnalogSensorMode | 122 | uint | - | 0 | absolute position of t sensor. In th mode setPosition( does not have an effect.<br><br>1 Relative: this mode th voltage difference is summed to calculate a relative position. |
| kAnalogInverted | 123 | bool | - | 0 | When inverted, the voltage is calculated a (ADC Full Scale - ADC Reading). This means that for absolute mode, the sensor valu is 3.3V - voltage. In relative mod the direction is reversed. |
| kAnalogSampleDelta | 124 | uint | - | 0 | Delta time between samples for velocity measureme |
| Reserved | 125 | - | - | - | Reserved |
| Reserved | 126 | - | - | - | Reserved |
| | | | | | 0: Default configuratio using limit switches |

| | | | | | |
|---|---|---|---|---|---|
| kDataPortConfig | 127 | uint | - | 0 | 1: Alternate Encoder Mode - limit switches ar disabled an alternate encoder is enabled. This parameter persists through a normal firmware update. |
| kAltEncoderCountsPerRev | 128 | uint | - | 4096 | Number of encoder counts in a single revolution, counting every edge on the A an B lines of a quadrature encoder. (Note: This different tha the CPR sp of the encoder which is 'Cycles per revolution'. This value i 4 * CPR. |
| kAltEncoderAverageDepth | 129 | uint | - | 64 | Number of samples to average for velocity dat based on quadrature encoder input. This value can b |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | between 1 and 64. |
| kAltEncoderSampleDelta | 130 | uint | - | 200 | Delta time value for encoder velocity measureme in 500µs increments. The velocity calculation will take del the current sample, and the sample * 500µs behind, and divide by th the sample delta time. Can be any number between 1 and 255. |
| kAltEncoderInverted | 131 | bool | - | 0 | Invert the phase of the encoder sensor. This is useful when the motor direction is opposite of the motor direction. |
| kAltEncoderPositionFactor | 132 | float32 | - | 1 | Value multiplied b the native units (rotations) o the encoder for position. |
| kAltEncoderVelocityFactor | 133 | float32 | - | 1 | Value multiplied b the native units (rotations) o |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | the encoder for velocity. |

## Migrating from CTRE Phoenix to SPARK MAX

Many teams have been using various CTRE motor controllers such as the Talon SRX and Talon SPX, and have concerns about porting software between platforms. Fortunately the feature set and code required is similar between the two, and porting from one to the other is easy. Below shows the common tasks and the changes required to convert from the code for CTRE Phoenix devices to the SPARK MAX.

---

## JAVA

### Include Library

REV SPARK MAX

```
1 import com.revrobotics.CANEncoder;
2 import com.revrobotics.CANPIDController;
3 import com.revrobotics.CANSparkMax;
4 import com.revrobotics.ControlType;
5 import com.revrobotics.CANSparkMaxLowLevel.MotorType;
```

Phoenix Framework

```
1 import com.ctre.phoenix.motorcontrol.can.*;
2 import com.ctre.phoenix.motorcontrol.*;
```

### Create Object CAN ID 1

REV SPARK MAX

```
1 private CANSparkMax sparkMax = new CANSparkMax(deviceID, MotorType.kBrushless);
```

```
1 TalonSRX talonSRX = new TalonSRX(1);
```

### Reset Factory Defaults

REV SPARK MAX

```
1 sparkMax.restoreFactoryDefaults();
```

Phoenix Framework

```
1 talonSRX.configFactoryDefault();
```

### Select Encoder

This is a brushed motor feature and not needed for using brushless motors with SPARK MAX

REV SPARK MAX

```
1 encoder = sparkMax.getEncoder();
2 encoder.setInverted(false);
3 sparkMax.setFeedbackDevice(encoder);
```

Phoenix Framework

```
1 talonSRX.configSelectedFeedbackSensor(FeedbackDevice.CTRE_MagEncoder_Relative, kPIDL
2 talonSRX.setSensorPhase(true);
```

### Set closed loop constants

```
pidController.setP(kP);
pidController.setI(kI);
pidController.setD(kD);
pidController.setIZone(kIz);
pidController.setFF(kFF);
pidController.setOutputRange(kMinOutput, kMaxOutput);
```

Phoenix Framework

```
talonSRX.configNominalOutputForward(0, kTimeoutMs);
talonSRX.configNominalOutputReverse(0, kTimeoutMs);
talonSRX.configPeakOutputForward(kMaxOutput, kTimeoutMs);
talonSRX.configPeakOutputReverse(kMinOutput, kTimeoutMs);

talonSRX.config_kF(kPIDLoopIdx, kP, kTimeoutMs);
talonSRX.config_kP(kPIDLoopIdx, kI, kTimeoutMs);
talonSRX.config_kI(kPIDLoopIdx, kD, kTimeoutMs);
talonSRX.config_kD(kPIDLoopIdx, kF, kTimeoutMs);
```

### Run Closed Loop Velocity Control

For this example velocity is set at 500 RPM

REV SPARK MAX

```
pidController.setReference(500.0, ControlType.kVelocity);
```

Phoenix Framework

```
// Convert from 500 RPM to 'native units'
double VelocityinUnitsPer100ms  = 500.0 * 4096 / 600;
talonSRX.set(ControlMode.Velocity, VelocityinUnitsPer100ms );
```

### Read RPM of Motor

REV SPARK MAX

```
encoder.getVelocity();
```

Phoenix Framework

```
talonSRX.getSelectedSensorVelocity(kPIDLoopIdx) * 600 / 4096;
```

### Read Applied Output Percent

REV SPARK MAX

```
sparkMax.getAppliedOutput();
```

Phoenix Framework

```
talonSRX.getMotorOutputPercent();
```

### Run Closed Loop Position Control

This example runs a position control loop for 10 rotations.

REV SPARK MAX

```
pidController.setReference(10.0, ControlType.kPosition);
```

Phoenix Framework

```
targetPositionRoations = 10.0 * 4096;
talonSRX.set(ControlMode.Position, targetPositionRotations);
```

### Change Units from 'rotations' to 'inches'

This example assumes a 4" wheel on a 15:1 reduction

REV SPARK MAX

```
encoder.setPositionFactor(M_PI * 4/15);
```

Phoenix Framwork

```
// No Equivellant
```

### Run Closed Loop Position Control for a set distance

This example assumes a 4" wheel on a 15:1 reduction to move 2 feet (24 inches).

REV SPARK MAX

```
pidController.setReference(24.0, ControlType.kPosition);
```

Phoenix Framework

```
targetPositionRotations = (M_PI * 4 / 15 * 4096) * 24;
talonSRX.set(ControlMode.Position, targetPositionRotations);
```

### Save Parameters

REV SPARK MAX

```
// Run after all parameters are set in RobotInit()
sparkMax.burnFlash();
```

Phoenix Framework

```
1 //Automatic
```

### Follow another device

REV SPARK MAX

```
1 sparkMax1.follow(sparkMax2);
```

Phoenix Framework

```
1 talonSRX1.follow(talonSRX2);
```

### Invert a device

REV SPARK MAX

```
1 sparkMax.setInverted(true);
```

Phoenix Framework

```
1 talonSRX.setInverted(true);
```

### Configure a limit switch

REV SPARK MAX

```
forwardLimit = sparkMax.getForwardLimitSwitch(LimitSwitchPolarity.kNormallyOpen);
```

Phoenix Framework

```
talonSRX.configForwardLimitSwitchSource(
                                        LimitSwitchSource.LimitSwitchSource_Feedback
                                        LimitSwitchNormal.LimitSwitchNormal_Normally
                                        kTimeoutMs);
```

### Disable a limit switch

REV SPARK MAX

```
forwardLimit.EnableLimitSwitch(false);
```

Phoenix Framework

```
talonSRX.overrideLimitSwitchesEnable(true);
```

---

## C++

### Include Library

REV SPARK MAX

```
#include "rev/CANSparkMax.h"
```

Phoenix Framework

```
#include "ctre/Phoenix.h"
```

### Create Object CAN ID 1

REV SPARK MAX

```
rev::CANSparkMax sparkMax{1, rev::CANSparkMax::MotorType::kBrushless};
```

Phoenix Framework

```
TalonSRX* talon = new TalonSRX(1);
```

### Reset Factory Defaults

REV SPARK MAX

```
sparkMax.RestoreFactoryDefaults();
```

Phoenix Framework

```
talon->ConfigFactoryDefault();
```

### Select Encoder

This is a brushed motor feature and not needed for using brushless motors with SPARK MAX.

REV SPARK MAX

```
rev::CANEncoder encoder = sparkMax.GetEncoder(rev::CANEncoder::EncoderType::kQuadrat
encoder.SetInverted(true);
sparkMax.SetFeedbackDevice(encoder);
```

Phoenix Framework

```
talonSRX->ConfigSelectedFeedbackSensor(FeedbackDevice::CTRE_MagEncoder_Relative, 0,
talonSRX->SetSensorPhase(true);
```

### Set closed loop constants

REV SPARK MAX

```
rev::CANPIDController pidController = sparkMax.GetPIDController();
pidController.SetFeedbackDevice(encoder);

pidController.SetP(kP);
pidController.SetI(kI);
pidController.SetD(kD);
pidController.SetIZone(kIz);
pidController.SetFF(kFF);
pidController.SetOutputRange(kMinOutput, kMaxOutput);
```

Phoenix Framework

```
talonSRX->ConfigNominalOutputForward(0, kTimeoutMs);
talonSRX->ConfigNominalOutputReverse(0, kTimeoutMs);
talonSRX->ConfigPeakOutputForward(1, kTimeoutMs);
talonSRX->ConfigPeakOutputReverse(-1, kTimeoutMs);

 talonSRX->Config_kF(kPIDLoopIdx, 0.1097, kTimeoutMs);
talonSRX->Config_kP(kPIDLoopIdx, 0.22, kTimeoutMs);
talonSRX->Config_kI(kPIDLoopIdx, 0.0, kTimeoutMs);
talonSRX->Config_kD(kPIDLoopIdx, 0.0, kTimeoutMs);
```

### Run Closed Loop Velocity Control

For this example velocity is set at 500 RPM

REV SPARK MAX

```
pidController.SetReference(500.0, rev::ControlType::kVelocity);
```

Phoenix Framework

```
"// Convert from 500 RPM to 'native units'
double VelocityinUnitsPer100ms  = 500.0 * 4096 / 600;
talonSRX->Set(ControlMode::Velocity, VelocityinUnitsPer100ms );
```

### Read RPM of Motor

REV SPARK MAX

```
encoder.GetVelocity();
```

Phoenix Framework

```
talonSRX->GetSelectedSensorVelocity(kPIDLoopIdx) * 600 / 4096;
```

### Read Applied Output Percent

REV SPARK MAX

```
sparkMax.GetAppliedOutput();
```

Phoenix Framework

```
talonSRX->GetMotorOutputPercent();
```

### Run Closed Loop Position Control
This example runs a position control loop for 10 rotations.

REV SPARK MAX

```
pidController.SetReference(10.0, rev::ControlType::kPosition);
```

Phoenix Framework

```
targetPositionRotations = 10.0 * 4096;
talonSRX->Set(ControlMode::Position, targetPositionRotations);;
```

### Change Units from 'rotations' to 'inches'

This example assumes a 4" wheel on a 15:1 reduction

REV SPARK MAX

```
encoder.SetPositionFactor( M_PI * 4 / 15 );
```

Phoenix Framwork

```
// No Equivellant
```

### Run Closed Loop Position Control for a set distance

This example assumes a 4" wheel on a 15:1 reduction to move 2 feet (24 inches).

REV SPARK MAX

```
pidController.SetReference(24.0, rev::ControlType::kPosition);
```

Phoenix Framework

```
targetPositionRotations = (M_PI * 4 / 15 * 4096) * 24;
talonSRX->Set(ControlMode::Position, targetPositionRotations);
```

### Save Parameters

REV SPARK MAX

```
// Run after all parameters are set in RobotInit()
sparkMax.BurnFlash();
```

Phoenix Framework

```
//Automatic
```

### Follow another device

REV SPARK MAX

```
sparkMax1.Follow(sparkMax2);
```

Phoenix Framework

```
talonSRX1->Follow(*talon2);
```

### Invert a device

REV SPARK MAX

```
sparkMax.SetInverted(true);
```

Phoenix Framework

```
talonSRX->SetInverted(true);
```

### Configure a limit switch

REV SPARK MAX

```
rev::CANDigitalInput forwardLimit = sparkMax.GetForwardLimitSwitch(rev::CANDigitalIn
```

Phoenix Framework

```
talonSRX->ConfigForwardLimitSwitchSource(
                                        LimitSwitchSource::LimitSwitchSource_Feedbac
                                        LimitSwitchNormal::LimitSwitchNormal_Normall
                                        kTimeoutMs);
```

### Disable a limit switch

REV SPARK MAX

```
forwardLimit.EnableLimitSwitch(false);
```

Phoenix Framework

```
talonSRX->OverrideLimitSwitchesEnable(true);
```

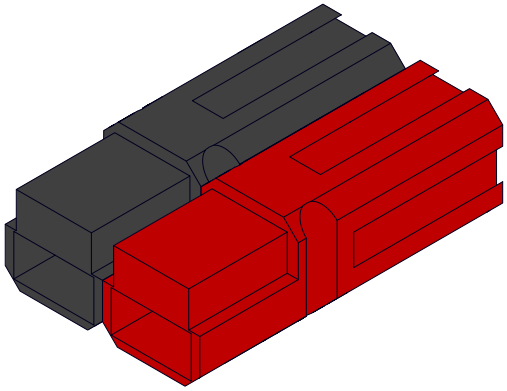
# Tips and Tricks

## Anderson Powerpole Connectors

Anderson Powerpole connectors are a popular choice in the FIRST community for electrical connections. Ensuring that these connectors are crimped properly and the contact is fully inserted into the housing is key to having a good electrical connection.

Anderson Powerpole Connectors consist of two major parts: the **housing** and **contact.** There are a number of different housings and contacts depending on the power requirements of the system. The most common housing and contact used with the SPARK MAX is the 1327 series housing paired with the 45 amp contacts.

### Anderson Connector Housing

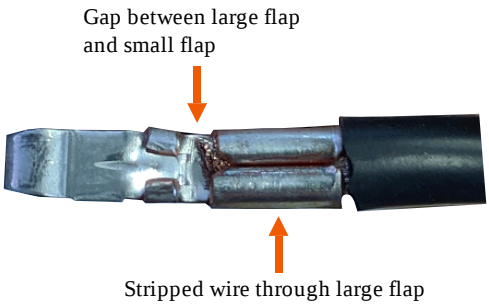


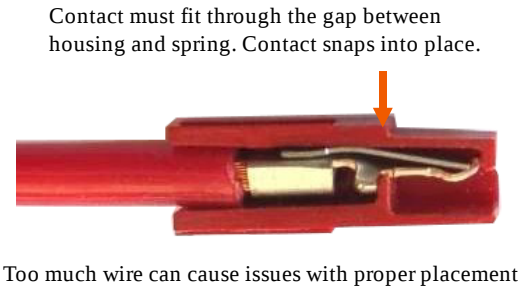
The housings for the connectors are genderless allowing all powerpole connectors to mate with themselves. The 1327 series housing can utilize contacts rated for 15-45 amps. Housings come in a variety of colors allowing for easy pairings with the wire color.

Dovetails on each housing allow them to slide together. Do not attempt to snap the housings together as they can break. After the housings are mated together adding a roll pin prevents the housings from become detached during operation. Each housing is fitted with a spring to retain the contacts after they are inserted

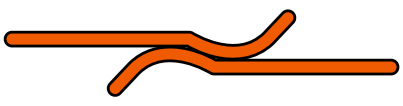
**Anderson Contacts**

1327 series housings can use contacts rated for between 15 and 45 amps. The 45 amp contacts are used with the SPARK MAX. When striping wire for the contact, make sure the stripped wire is the length of the large flap. No wire should extend past the large flap into the gap between the large and small flaps on the contact. Utilize a proper crimping tool when crimping on the connector.




Having too much wire exposed can cause issues with proper placement of the contact in the housing. This can lead to bad connections. If the contacts are not fully inserted into the housing the contact connection issues can arise. The images above are examples of good crimps with the proper amount of wire inserted into the contact.



Good Connection

Bad Connection

Having improper placement of the contact in the housing can lead to intermittent brownouts of the SPARK MAX, the contact dislodging from the housing, or have cause problems with one or more of the phases of a brushless motor.
For more information on powerpole assembly see the Powerwerx instructions.

# SPARK MAX Client - Legacy

## Getting Started with the SPARK MAX Client

> This is **legacy documentation** for our discontinued SPARK MAX Client Software. If you are interested in running a SPARK MAX via a computer, please see our newer documentation: Getting Started with the REV Hardware Client.

Update, configure, and test your SPARK MAX Motor Controller with the SPARK MAX Client application.

| Latest SPARK MAX Client - Version 2.1.1 |
| --- |
| Download Latest SPARK MAX Client |

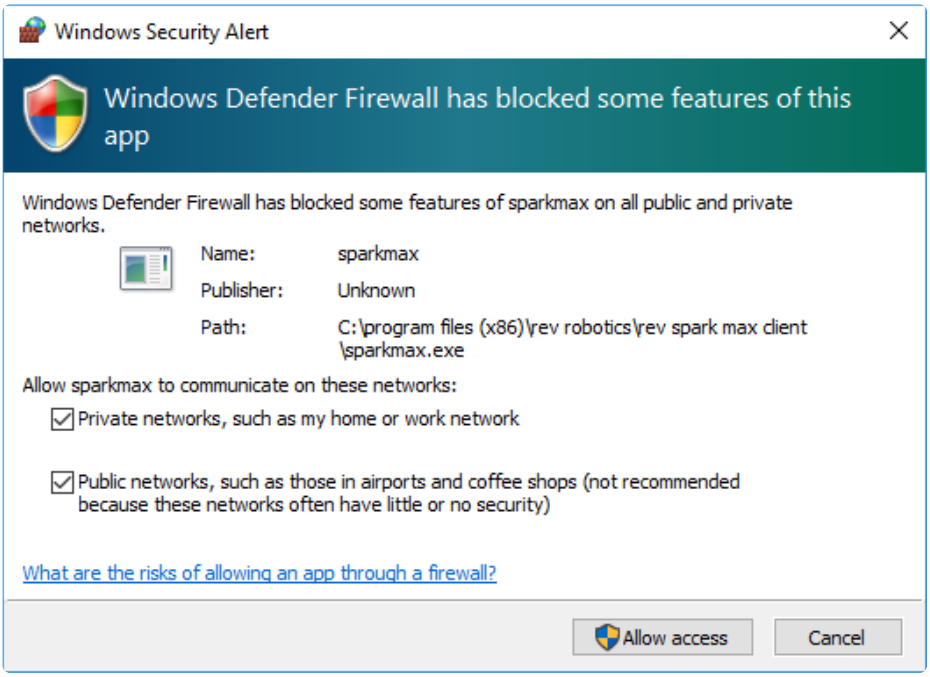
The SPARK MAX Client **will not work with SPARK MAX beta units** distributed by REV to the SPARK MAX beta testers. It is only compatible with units received after 12/21/2018.

#### System Requirements

- Windows 10 64-bit
  - Windows 7 64-bit might work but it is not supported.
- Internet connection for automatic updates

#### Installation Instructions

1. Download the SPARK MAX Client installer above.
2. Run the installer. Windows may require approval to install the application.
3. During the installation process, separate driver installation windows may appear. Some driver installations may fail if you already have the driver installed from a previously installed Client, this is expected.
4. Once installed, run the application. If prompted, be sure to grant network access. Without network access, the client software won't be able to download the latest SPARK MAX firmware and client updates.



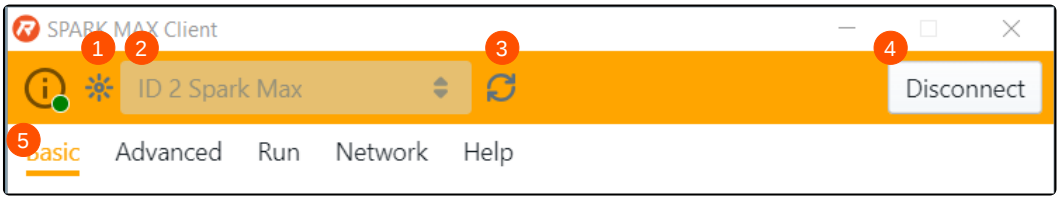

## Navigating the SPARK MAX Client

> This is **legacy documentation** for our discontinued SPARK MAX Client Software. If you are interested in running a SPARK MAX via a computer, please see our newer documentation: Getting Started with the REV Hardware Client.

---

## Navigation Bar

The Navigation bar is visible on all tabs of the SPARK MAX Client and allows you to select with SPARK MAX the Client is connected to.



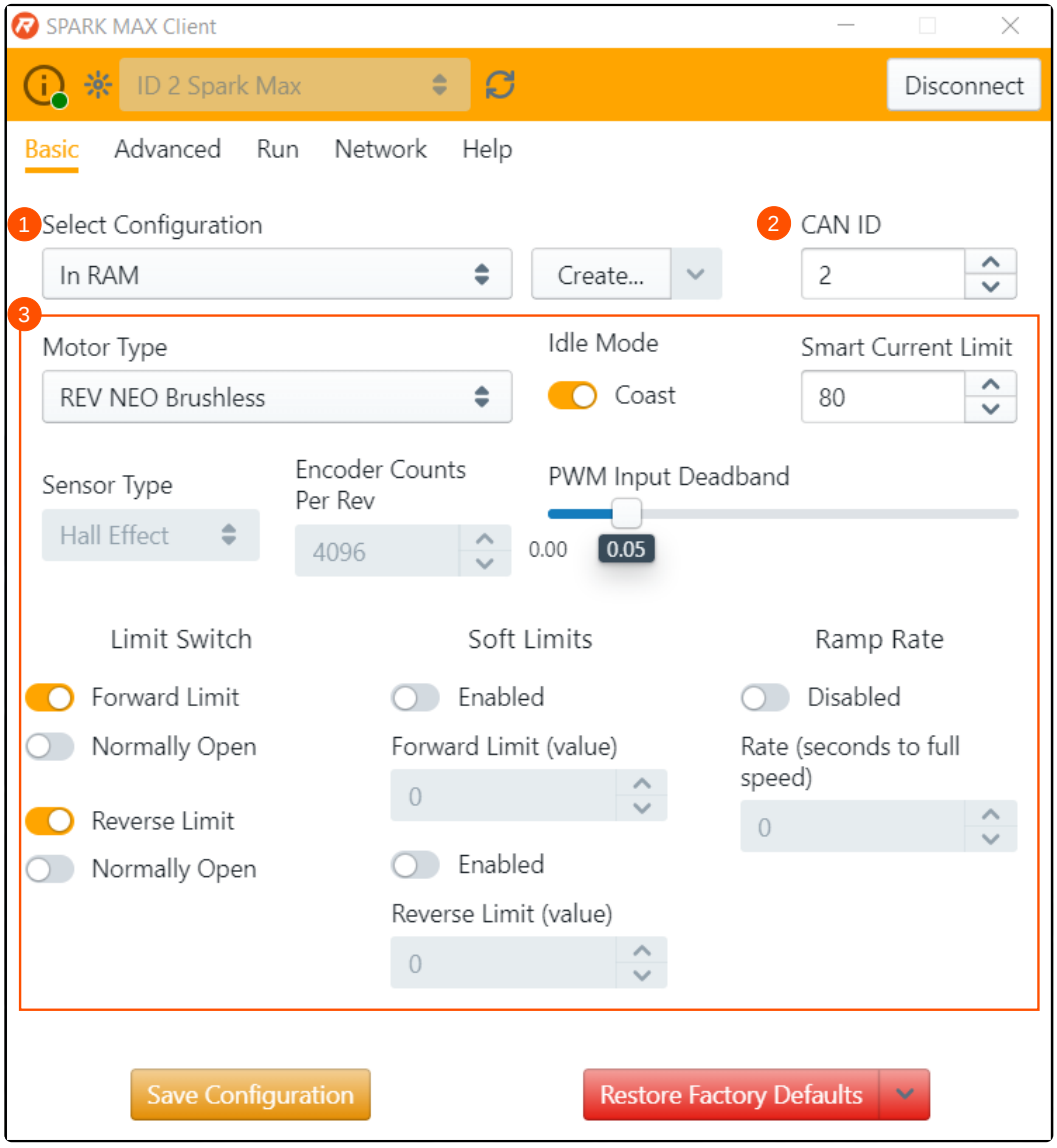

1. **Identify Device:** The status LED of a selected device will blink. This is helpful when troubleshooting or configuring multiple devices.
2. **Device Selection:** See each SPARK MAX connected to the SPARK MAX Client. This includes other devices connected via CAN if running firmware 1.4.0 or later.

3. **Rescan:** Looks for additional SPARK MAX devices connected to the SPARK MAX Client. This includes other devices connect via CAN if running firmware 1.4.0 or later.
4. **Connect/Disconnect:** After selecting a device connecting to the device pulls all the configuration parameters set on the device.
5. **Tabs**: Select one of the five tabs to gain access to configure, update, and run SPARK MAX.

---

## Basic Tab

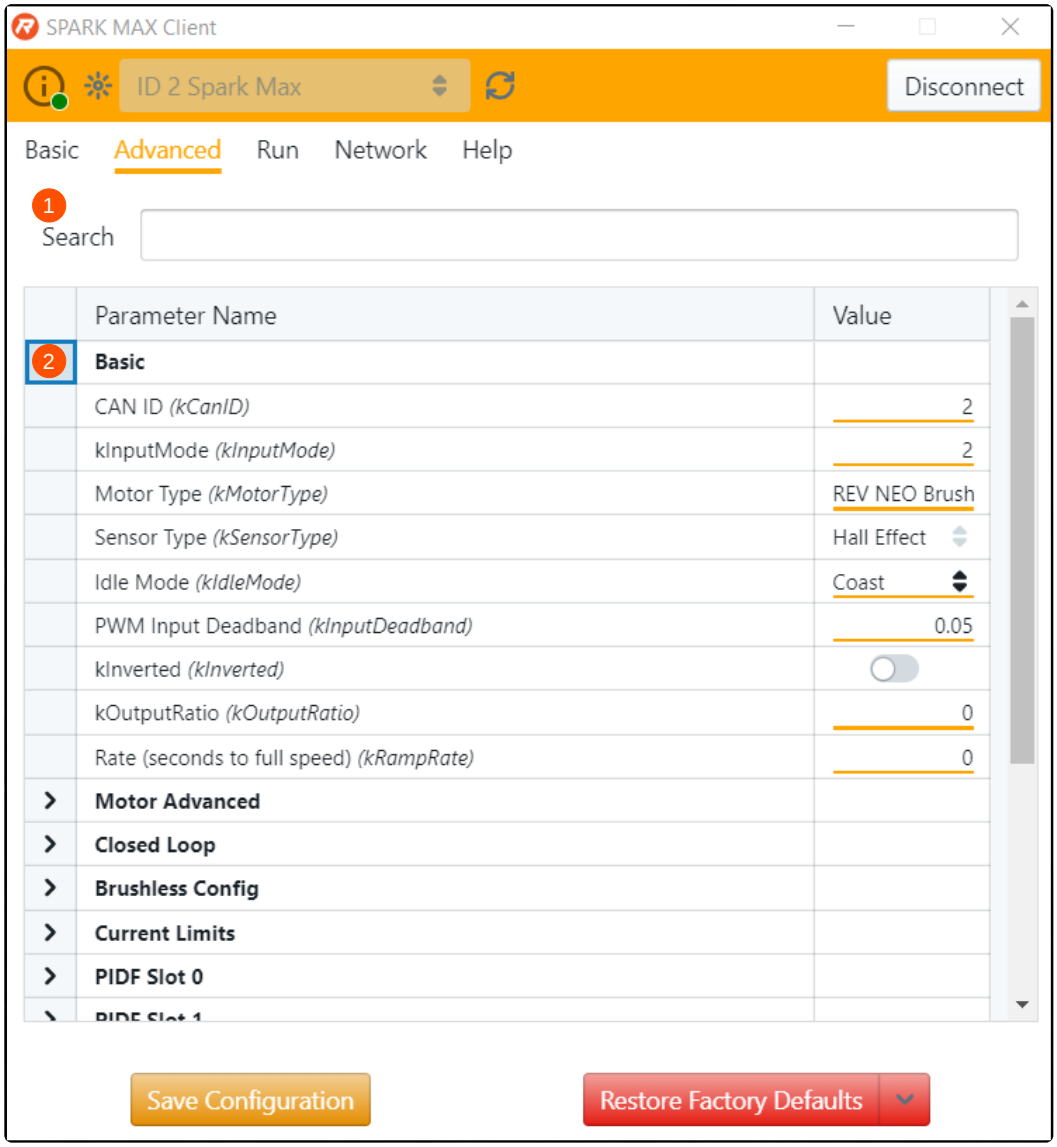
The Basic Tab is used to set the most common parameters for the SPARK MAX




1. **Configurations:** This drop down allows you to select pre-existing configurations store on the Windows machine running the SPARK MAX Client or to pull the existing parameters stored on in RAM on the SPARK MAX. This is helpful when configuring multiple motor controllers to the same settings.
2. **CAN ID:** This assigns a SPARK MAX a CAN ID for identification over the CAN BUS. Any configured SPARK MAX **must have** a CAN ID.
3. **Configured Parameters:** Change the motor type, sensor type, idle mode behavior, and more.

## Advanced Tab

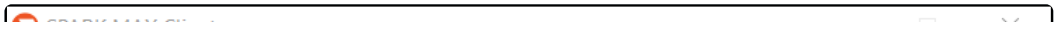
The Advanced Tab allows for changing all configurable parameters of the SPARK MAX without needing to set them in code.

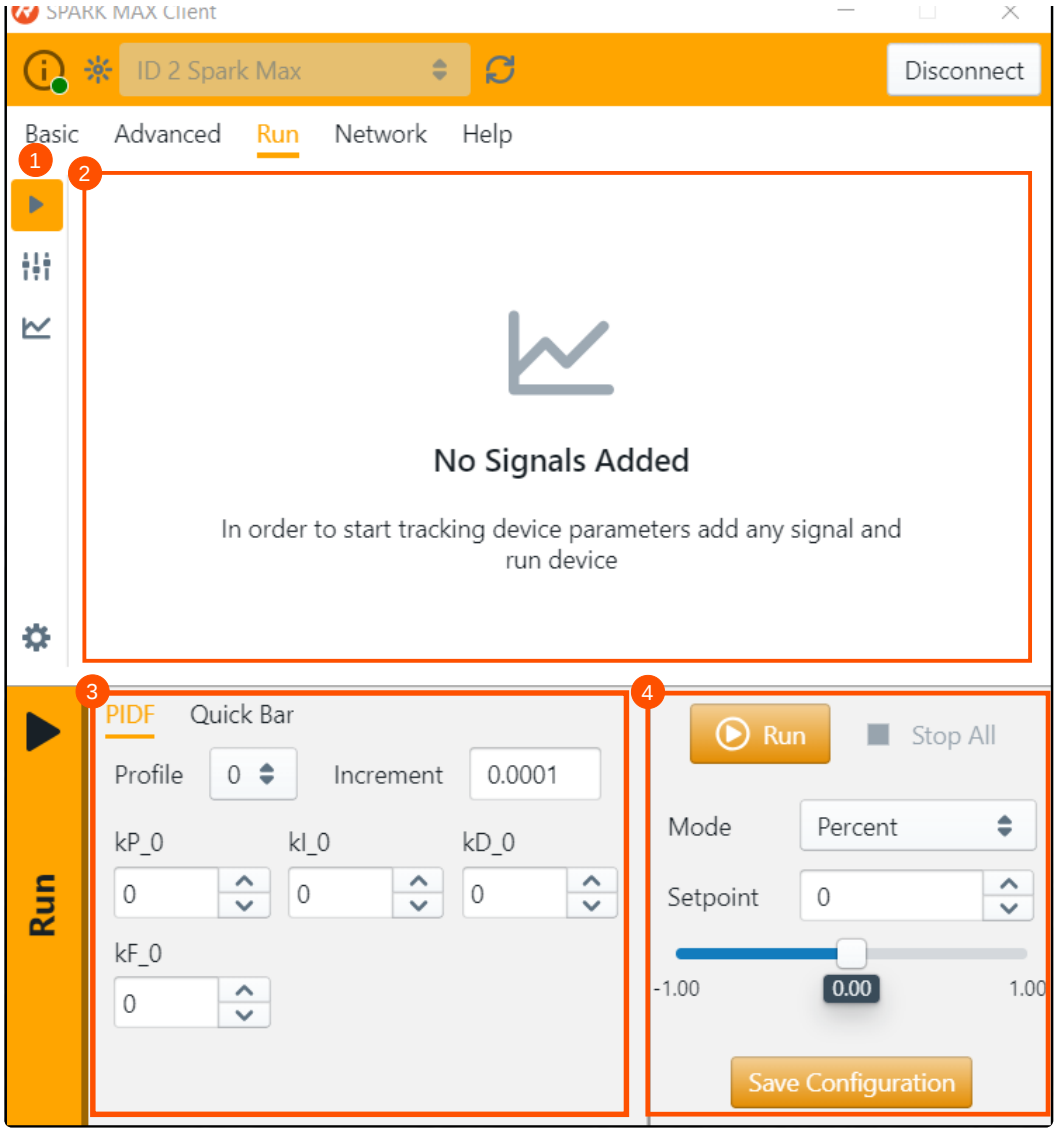



1. **Search Parameters:** Allows for easy look up of a specific parameter for editing.
2. **Parameter Table:** Select the arrow to show all configurable parameters within a specific group. For more information on each parameter type see Configuration Parameters.

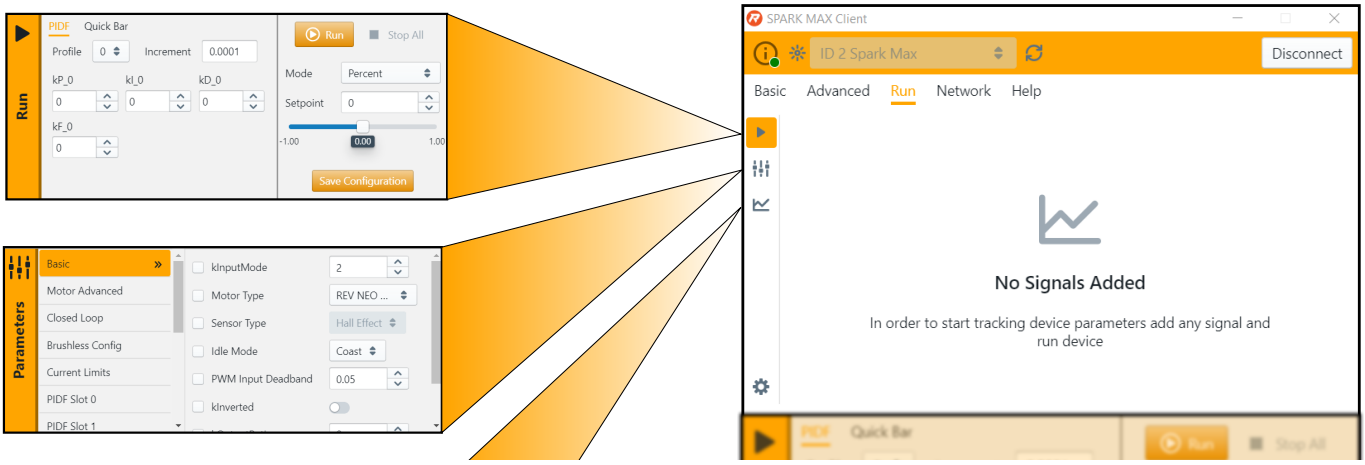
## Run Tab

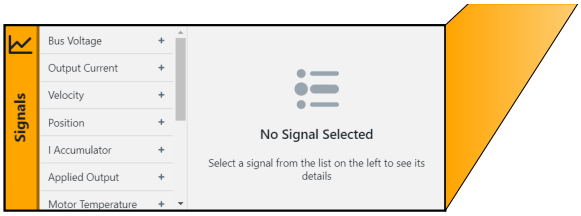
The Run Tab allows for the SPARK MAX to operate over USB or a USB to CAN Bridge without the need for a full control system. This is helpful for testing mechanisms and tuning their control loops.






1. **Bar Select:** Select from either run, parameters, or signals to provide information and feedback when operating SPARK MAX.
2. **Signal Chart:** Shows any added signals in graph form when running a SPARK MAX. This is helpful when tuning control loops.
3. **PIDF:** Update PIDF parameters on the fly to tune control loops on the SPARK MAX.
4. **Run:** Choose setpoints to run a motor connected to a SPARK MAX using various modes, including position, velocity, and duty cycle.





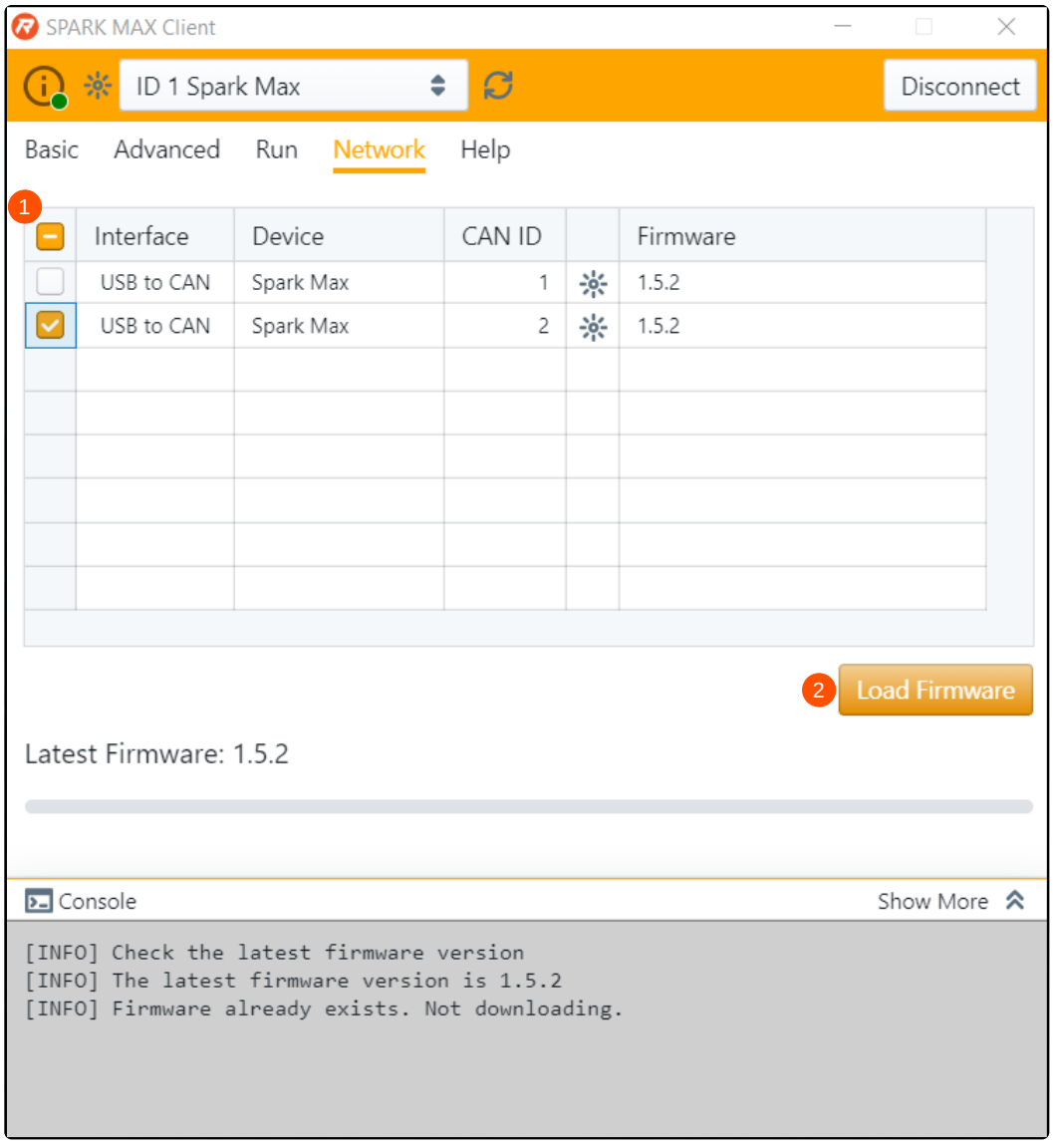


The three icons for bar select change the bottom third of the Run Tab for configuration. Once signals and other parameters are configured selecting the run bar icon will allow for running of a motor with the SPARK MAX Client.

---

## Network Tab

The Network Tab shows all connected devices via USB and the USB to CAN interface. From the Network Tab each device can be identified and firmware updated.



1. **Device Select:** Select a device to update firmware.
2. **Load Firmware:** Select what firmware to update onto selected devices.

> For more information on the firmware updating process see Updating Device Firmware for both single device and multiple device updates.

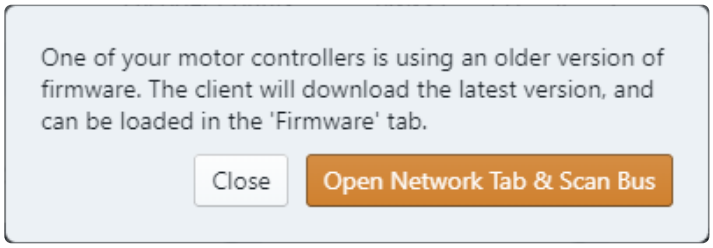
## Updating Device Firmware

> This is **legacy documentation** for our discontinued SPARK MAX Client Software. If you are interested in running a SPARK MAX via a computer, please see our newer documentation: Getting Started with the REV Hardware Client.

### Updating a Single Device

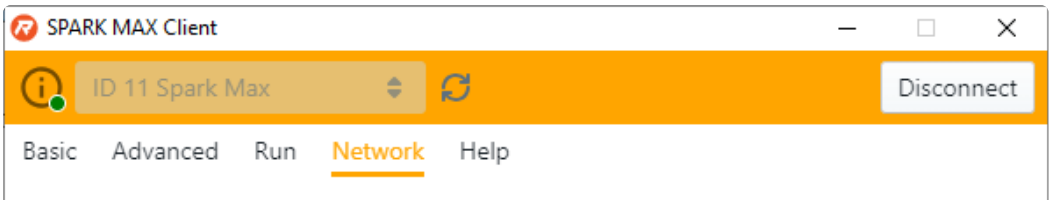
Follow the steps below to update the firmware on your SPARK MAX:

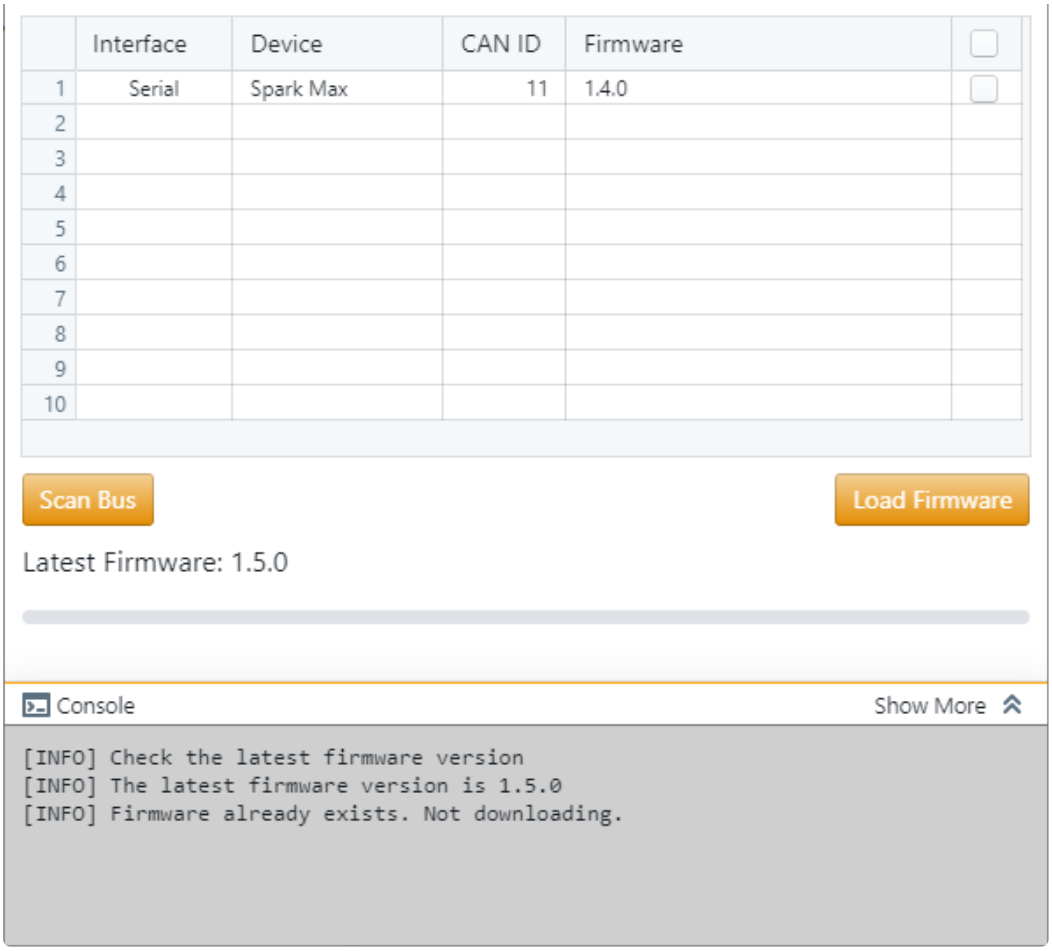
- Connect your SPARK MAX Motor Controller to your computer with a USB-C cable.
- Open the REV SPARK MAX Client application.
- The Client should automatically scan and connect to your SPARK MAX. If your SPARK MAX is running outdated firmware, you will be notified with a pop-up window like the one pictured below:




> If your SPARK MAX is running firmware older than 1.4.0, you may not see a pop-up and will need to proceed directly to the **Network** tab and click **Scan Bus** manually.

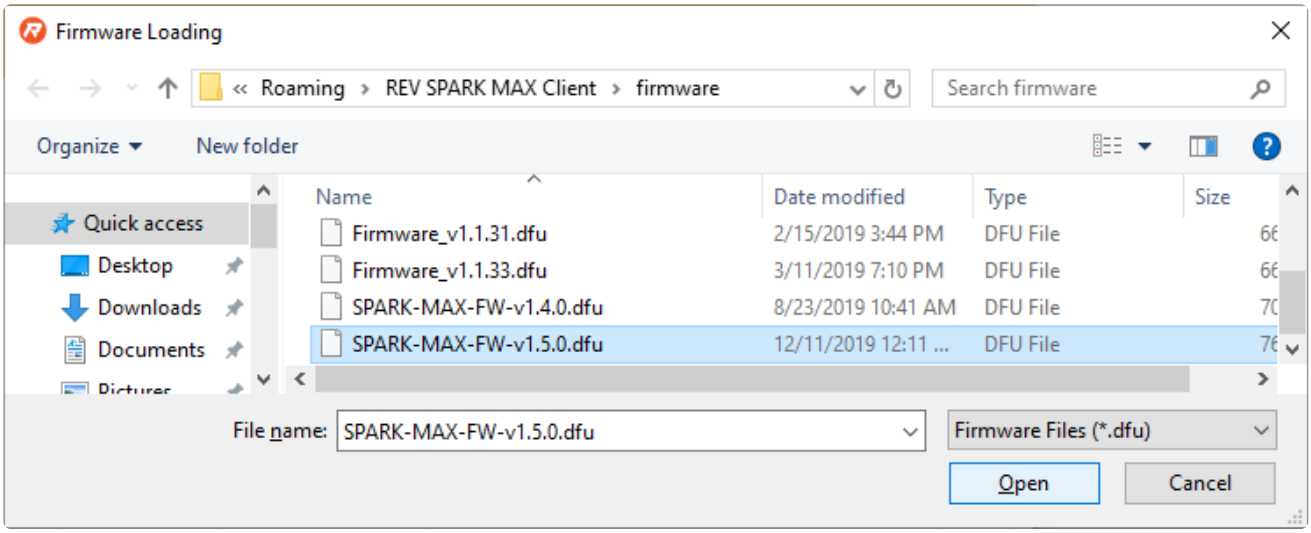
- Click **Open Network Tab & Scan Bus** and proceed to the next step.
- Your SPARK MAX should now be listed in the device list. Click the checkbox next to the SPARK MAX you wish to update, and click **Load Firmware**.



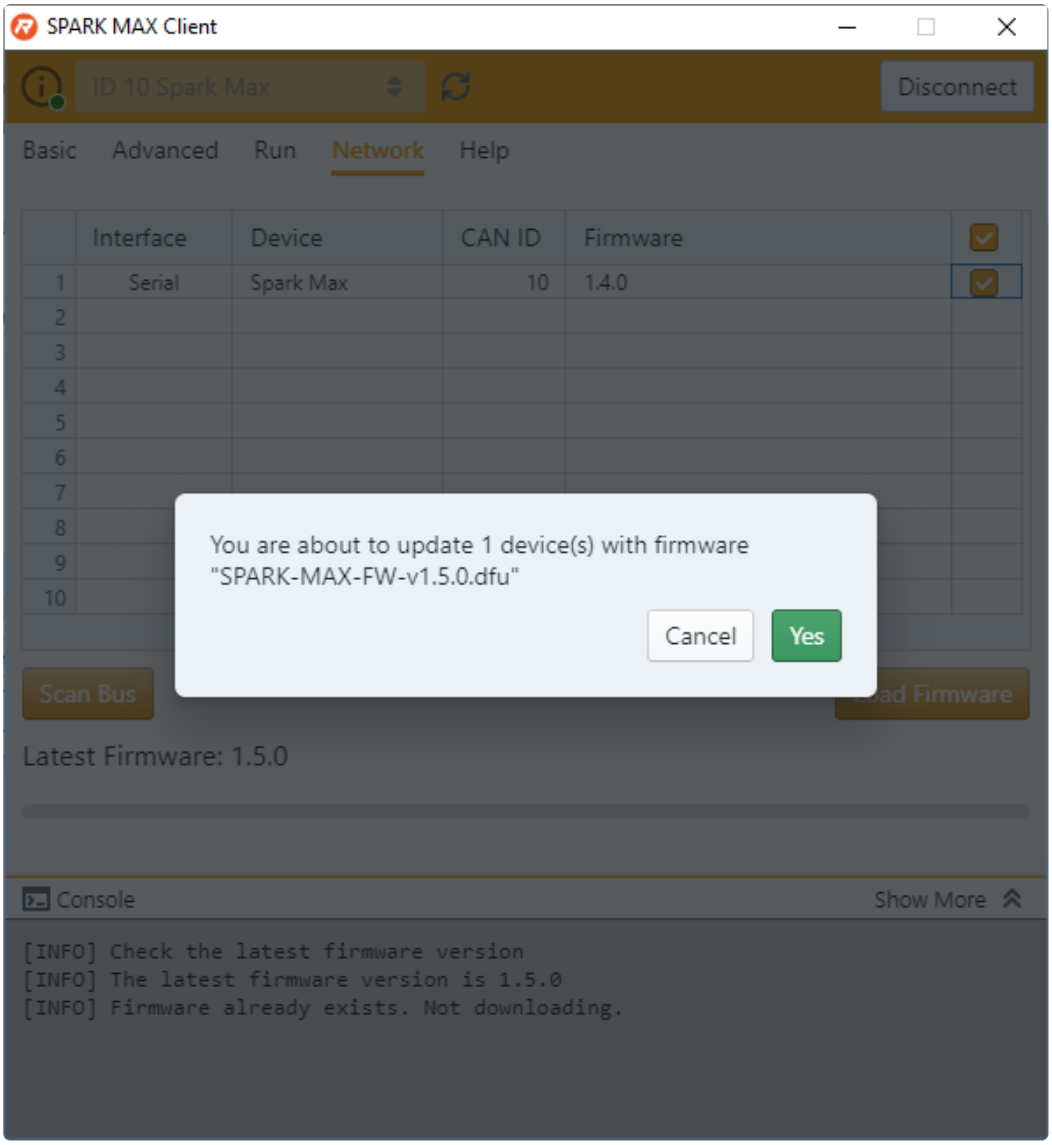



> If your SPARK MAX is listed and you are unable to click the checkbox next to it, you must put your SPARK MAX into Recovery Mode.

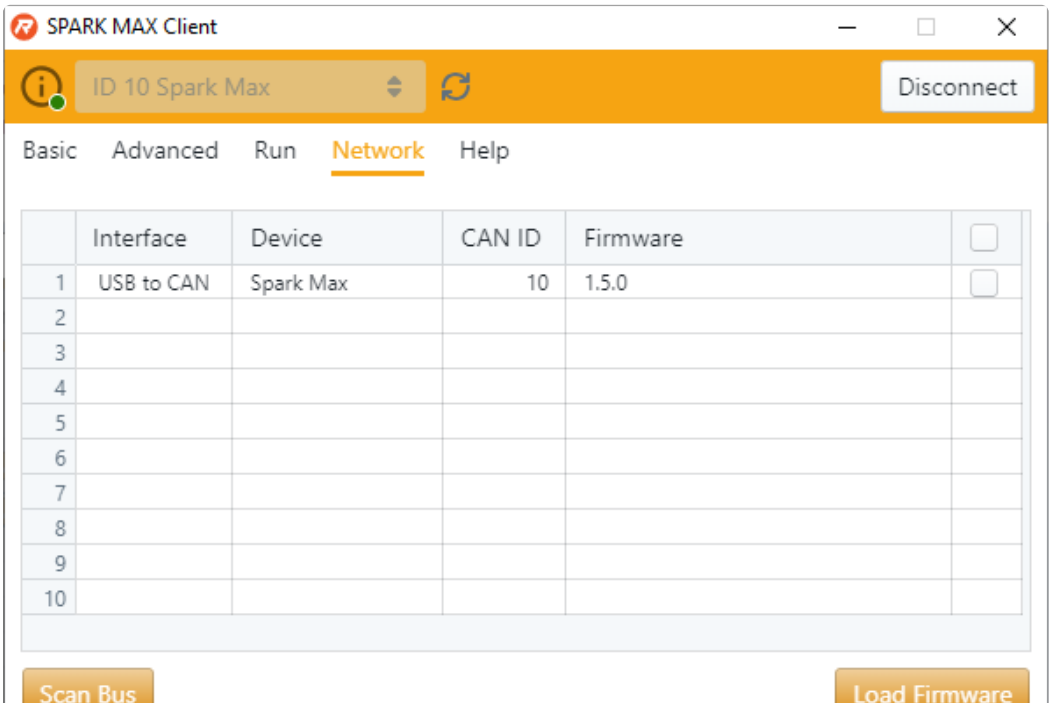
- Select the latest firmware file in the firmware directory that the client created on startup. If there isn't a firmware directory, you can also navigate to a file that was downloaded manually. Click **Open** once the appropriate firmware file is selected:

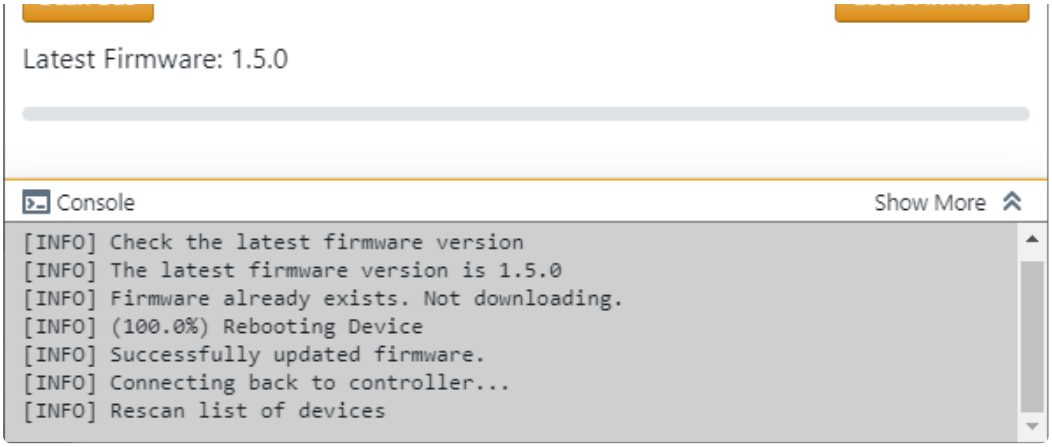



- Click **Yes** to confirm the update.




- Once complete, the Client will rescan the bus and display the updated controllers.



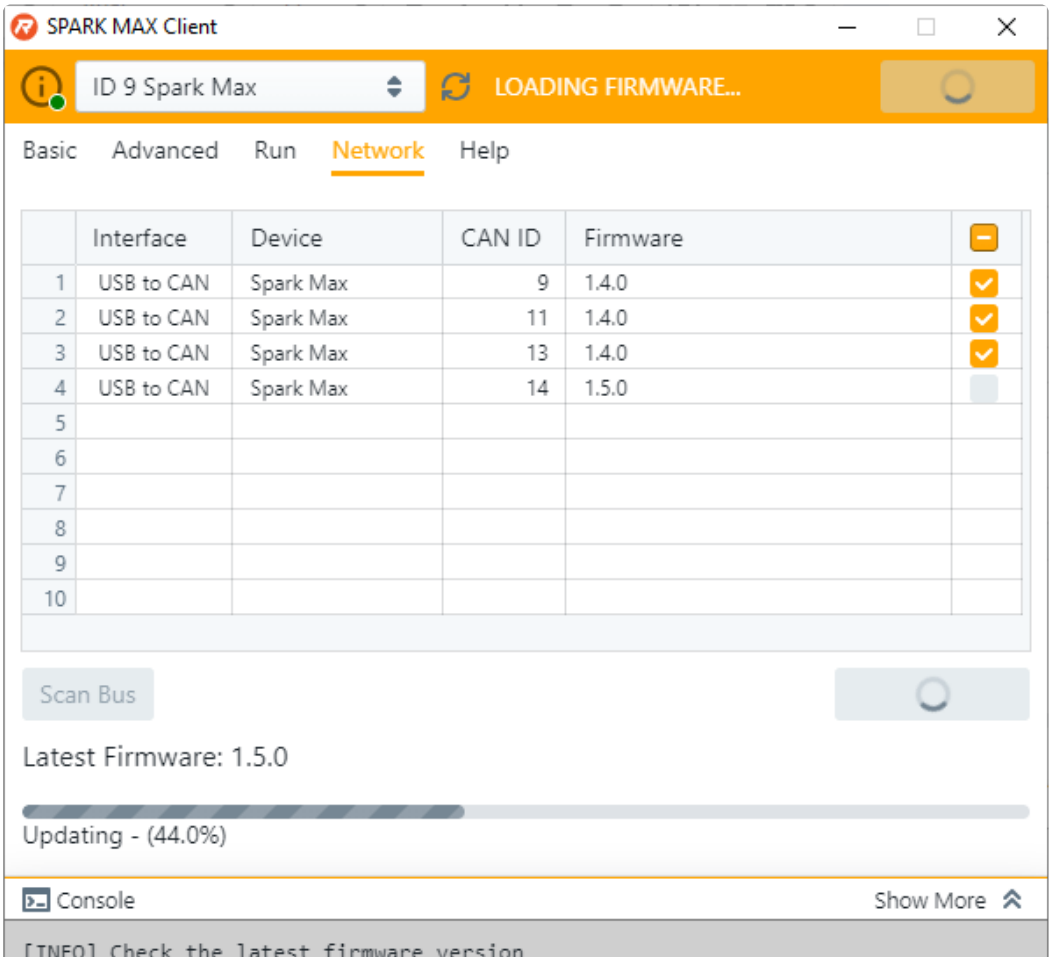


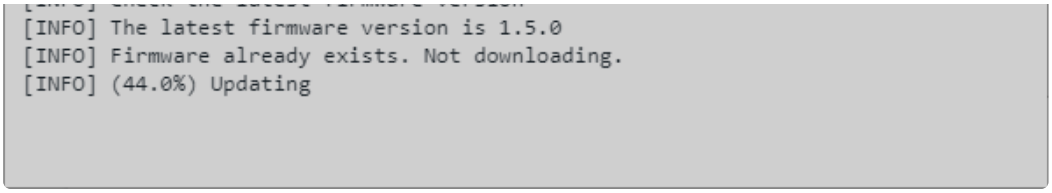

**Updating Multiple Devices with the USB-to-CAN Bridge**

SPARK MAX Firmware Version 1.5.0 includes a USB-to-CAN Bridge feature that allows a single USB-connected SPARK MAX to act as a bridge to the entire CAN bus it is connected to. This allows for configuration and simultaneous updating of multiple SPARK MAX controllers without having to connect to each one individually. Using this feature requires the following:

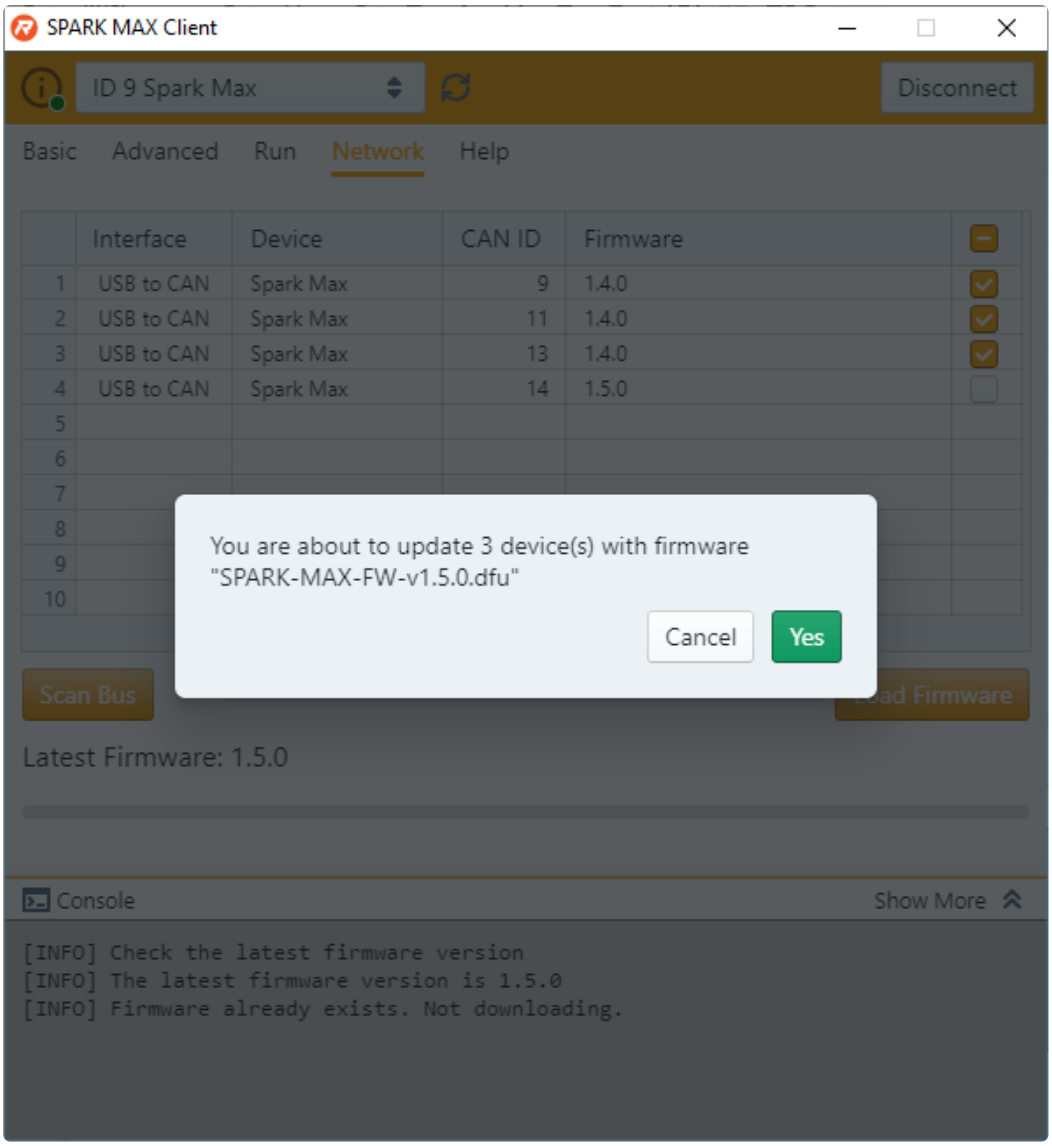
- A USB-connected SPARK MAX that is updated to firmware version 1.5.0 or newer to act as the Bridge.
- Other SPARK MAXs connected on the CAN bus must be individually updated to firmware version 1.4.0 before they are able to receive mass-updates from the Bridging SPARK MAX.

Once these requirements are satisfied, navigate to the **Network** tab, select the controllers you wish to update, and follow the same firmware update procedure described above starting at Step 4.






When complete, the Client will display the number of successfully updated controllers.




If a controller fails to update it is usually due to the process being interrupted by a bad power or CAN connection. Severe interruptions can cause the firmware update to be corrupted. A corrupted controller can no longer be updated over the USB-to-CAN Bridge, however, it can be recovered by connecting to the controller directly over USB and putting it in Recovery Mode.

## Recovery Mode with the SPARK MAX Client

> This is **legacy documentation** for our discontinued SPARK MAX Client Software. If you are interested in running a SPARK MAX via a computer, please see our newer documentation: Getting Started with the REV Hardware Client.

When updating the firmware on the SPARK MAX, it is possible for the process to be interrupted or for the firmware to be corrupted by a bad download. In this state, the Status LED will be dark and the SPARK MAX will fail to operate. SPARK MAX has a built-in recovery mode that can force it to accept new firmware even if the controller seems to be bricked. The following procedure requires a small tool, like a straightened paper clip, to press the Mode Button, a USB C cable, and a computer with the SPARK MAX Client Application installed:

1. With the SPARK MAX powered off completely, press and hold the Mode Button.
2. While still holding the Mode Button, connect the SPARK MAX to the computer using the USB cable. The Status LED **will not** illuminate, this is expected.
3. Wait a few seconds for the computer to recognize the connected device, then release the Mode Button.
4. Open the SPARK MAX Client Application. The SPARK MAX will remain dark and it **will not** connect to the Client, this is expected.
5. Navigate to the **Network** tab and click the **Rescan** arrows at the top of the window.
6. The SPARK MAX will be listed under *Devices in Recovery Mode.* Click the checkbox next to the device.
7. Click the **Load Firmware** button.
8. Select the latest firmware file and click **Open**.
9. The firmware should load successfully and the SPARK MAX will now connect to the Client.

## SPARK MAX Client Troubleshooting

> This is **legacy documentation** for our discontinued SPARK MAX Client Software. If you are interested in running a SPARK MAX via a computer, please see our newer documentation: Getting Started with the REV Hardware Client.

#### Error During First-time Firmware Update

If this is the first time installing the SPARK MAX Client or connecting a SPARK MAX in Recovery Mode, you may see an error the first time you try to update firmware on your computer. The DFU driver is one of two drivers installed by the Client and is used for updating firmware. It may not install completely until a SPARK MAX in DFU Mode (Recovery Mode) is plugged in to the computer.

If you see an error during your first firmware update, please do the following:

1. Close the Client application.
2. Unplug the SPARK MAX from the computer.
3. Plug the SPARK MAX back into the computer.
4. Open the Client application.

Alternatively, you can preemptively finalize the DFU driver installation by following the Recovery Mode steps before using the Client for the first time.
We are aware of this issue and will be releasing a fix in a future update of the SPARK MAX Client.

### Troubleshooting

As we get feedback from users and identify exact causes for issues, please look back here for troubleshooting help. If you are running into issues running the SPARK MAX Client try the following **BEFORE** contacting support@revrobotics.com:

- Try running the SPARK MAX Client as an Administrator
- Make sure that Windows is fully up-to-date. Some computers have Windows Update disabled and need to be updated manually.
- Check the Device Manager and verify that the SPARK MAX shows up as one of the following two devices with no caution symbols:
  - Normal operating mode: Device Manager -> Ports (COM & LPT) -> USB Serial Device (COMx)
  - Recovery mode: Device Manager -> Universal Serial Bus Controllers -> STM Device in DFU Mode
    - If the device shows up with errors or as STM32 BOOTLOADER, try installing the DFU drivers separately.