Phoenix Documentation

CTRE

Jan 17, 2019

Contents

Below is the latest documentation for CTR-Electronics Phoenix software framework. This includes...

- **Class library** for **Talon SRX, Victor SPX, CANifier and Pigeon-IMU** (C++/Java/LabVIEW for FRC, C# for HERO)
- **Phoenix Tuner** Graphical Interface - provides configuration options, diagnostics, control and plotting.
- **Phoenix Diagnostic Server** - install on to roboRIO for Tuner, and to perform HTTP API requests for diagnostic information.

CHAPTER 1

Blog Entries

CHAPTER 2

Follow these instructions in order!

2.1 FRC 2019 Blogs

2.1.1 BLOG: FRC 2019 Week 1

Hello FRC Community,

It's the end of "week 1" of the build season, and a good time for an update from CTRE.

C++/Java teams appear to generally be successful becoming familiar with this year's new software collection (VS Code + Phoenix + Tuner). If your team has not updated yet, we recommend starting with WPI Screensteps, then running through Phoenix Framework Documentation for CTRE devices.

LabVIEW teams appear to be up and running as well with our kickoff software library. We've encountered a single report of an install related issue, which has been reproduced and fixed. As a reminder to teams, please **use Tuner to install the Phoenix library** into your roboRIO, and not last year's HERO LifeBoat.

See the sections below for more information and good luck this build season!

Omar Zrien

Co-owner CTR-Electronics

Updated API Docs

This week we went through our C++/Java API **filling in any missing comment block sections**. The online API doc has been updated for C++ and Java. Of course if you still have questions, feel free to contact our support directly.

Remember to update the firmware!

We got a few support calls/emails this week from teams reporting that their Talon SRX motor controllers would not enable as expected. Here were the root-causes we found:

- Wrong firmware - Didn't update.

- Still wrong firmware, where is the web-dash? What's Phoenix Tuner?
- Didn't set device IDs.
- We are supposed to use VS Code now? When did that happen?
- No we didn't install anything called Phoenix. What's that?

But more often than not, **the root-cause was not updating to this year's firmware (4.11)**. Once updated, teams were ready to rock and roll.

So as a reminder, teams should ensure that their motor controllers are up to date for proper functionality with 2019 software.

Phoenix on other platforms?

Some teams are already aware that *Phoenix isn't just a library targeted for the roboRIO*. For those of you perusing our maven site you may have noticed *several Linux binaries*. These have been used for leveraging Talons/Victors/CANifiers/Pigeon on third party hardware such as:

- Raspberry PI + SocketCAN interface
- NVIDIA Jetson TX2 + native CAN
- Even Linux Desktop with SocketCAN device plugged in.

After reviewing the FRC rules, it appears that leveraging these CAN devices from third-party CAN hardware is **now officially FRC legal**. This opens up an entirely new way to develop robots in FRC.

- Want to update your set-points from your Raspberry PI vision system?
- Want to run closed-loops in your NVIDIA Jetson?
- Do you prefer a development platform/environment outside of the roboRIO control system?

For the first time ever, you can now update your control outputs *outside of the roboRIO*, while the *roboRIO safely manages the enable/disable* of your Talon SRXs and Victor SPXs.

The requirements for this are to link the appropriate Phoenix-targeted library into your application (typically Linux-amd64, Linux-armhf, Linux-aarch64), allowing you to use the same Phoenix API on your platform.

Additionally you must provide a CAN bus interface. A popular method to do this is with through SocketCAN.

And finally make sure Phoenix loads on the roboRIO, either by creating a dummy Phoenix CAN device that is not on the bus, or calling the new loadPhoenix() routine.

We have a GitHub example demonstrating this type of control on an FRC robot with a Raspberry PI + CANable(USB). Driver station is used to enable/disable the robot, and the rest is done in the Raspberry PI C++ application.

Note: Reading gamepads inputs on a raspberry pi is typically not FRC legal, but controlling the Talons from the pi now is.

Note: This applies to CAN-bus control of Talon SRX/Victor SPX, and **not PWM**.

Phoenix Tuner 1.0.2

We released a minor update to Tuner today. Tuner 1.0.2 is packaged with today's installer (see below) but also available as a separate download...

https://github.com/CrossTheRoadElec/Phoenix-Releases/releases

Fixes:

- Improved the form response to high-dpi monitors. Note that the plotter features still work best at 1920 X 1080 (otherwise Windows may auto-scale the application on plotter-enable).
- Improved mDNS resolving. This improves discovery of the roboRIO when user enters team number instead of a static-IP or 172.22.11.2 (USB) host name.

Phoenix Framework 5.12.1

We also released a minor Phoenix update today.

https://github.com/CrossTheRoadElec/Phoenix-Releases/releases

The same API comment-block updates that were applied on our site are now in our library so that VS Code Intellisense can display more information.

Tip: C++ teams may have to invoke "./gradlew clean" and/or "WPILib C++: Refresh C++ Intellisense" for Intellisense to update.

Our maven site has also been updated with the 5.12.1 libraries.

We also added more firmware version checking (we report a DriverStation message already, but we now do it as soon as the Phoenix object is created, instead of waiting for you to call certain routines). We were motivated to do this due to the support calls we got this week mentioned earlier :)

And finally we fixed the context help for SetInverted.vi (LabVIEW), this was reported by a team.

This minor update also provides an opportunity for *C++/Java teams to become familiar with the "Update" instructions* for third-party libraries. Be sure to review the update instructions

Balance Bot

Last year during the Worlds Championships, we demoed a small 2-wheeled balance bot using our HERO control system. No, I don't expect any competition robots to employ the same drive train :)

But during the off-season, we redesigned it to be easier to 3D print, assemble, and support. Earlier this week we were asked about the demo, only to realize we never posted the files!

The CAD and source is now available on GitHub.

2.1.2 BLOG: FRC 2019 Kickoff

Once again it is time to kick off a new FRC season!

The Phoenix installer and non-Windows binary kit is now available.

You can also find the latest firmware CRFs at each product page (the installer also installs them).

The new features included in this season's release are listed below. What's great about this list is that every single feature was a request from an FRC student or mentor. Also this year's API release is **almost entirely backwards compatible**.

New features below

- Motor controller followers can use SetInverted(enum) to match or oppose master motor controller. SetInverted(bool) still exists and works exactly the same as last season.
- Motion Profile and Motion Profile Arc have an explicit feedforward term (allows for kS, kV, kA, etc..) for both primary and aux PIDs.
- Motion Profile (and Arc) use a linear interpolator to adjust the targets every 1ms. This means you can send less points, reducing CAN bandwidth, but still have resolute control.
- Motion Profile API has a simpler mode where you can simply call Start/IsFinished. Legacy API still exists and is supported.
- Phoenix Tuner and Phoenix Diagnostics Server replaces the silverlight based diagnostics from past seasons.
- Phoenix Tuner provides the means of controlling Talons for simple testing.
- Phoenix Tuner includes a plotter.
- Phoenix Tuner provides import and export of all config settings.
- Phoenix Tuner can be used to factory default config settings (as well as API).
- Phoenix Tuner can field-upgrade all same-model devices with one click.
- Phoenix Tuner can be used to perform Pigeon IMU temperature calibration.
- New config routines to simplify management of persistent settings: configFactoryDefault and configAllSettings
- Compatibility with WPI distributed Visual Studio Code interface and GradleRIO build system.
- Maven hosted API provides an alternative to using the Phoenix installer to get API binaries (however the offline install is highly recommended).
- No more FRC versus nonFRC firmware, see "FRC Lock" features for explanation.
- Phoenix C++ API is portable to RaspPI, Linux-Desktop, NVIDIA Jetson TX2, etc.
- Performance improvements of general status CAN get routines (get routines take far less time to execute than previous seasons).
- Talon SRX: Maximum reportable velocity increased. New maximum RPM is 38400 RPM (@ 4096 untis per rotation).
- C++/Java: Added default parameter values for pidIdx, slotIdx, and timeoutMs where appropriate.

Note: Installing Phoenix on another Linux device and controlling Talon SRX / Victor SPXs may or may not be FRC legal depending on 2019 rules.

Fixes

- Current Limit Errata fixed from last year on Talon SRX.
- SetYaw fixed from last year to be in degrees on Pigeon IMU.

Back-breaking API changes

- If using Motion Profile Arc, be sure to set the bAuxPID flag to true in the trajectory points. This member variable did not exist before.

- Motion Profile Trajectory point duration is now integer (milliseconds) with [0,127] ms range.
- MotionMagicArc enum removed from LabVIEW. This enum was never used. Arc'ing with Motion Magic is accomplished with the four-parameter set() and MotionMagic enum.

Good luck this build season! - Omar Zrien

2.1.3 Site Created

This blog entry indicates the original creation date.

2.2 Phoenix Software Reference Manual

This is the latest documentation for CTR-Electronics Phoenix software framework. This includes:

1) Class library for Talon SRX, Victor SPX, CANifier and Pigeon IMU (C++/Java/LabVIEW for FRC, C# for HERO).
2) Phoenix Tuner GUI - provides configuration options, diagnostics, control and plotting.
3) Phoenix Diagnostic Server - install on to RIO for Tuner, and to perform HTTP API requests for diagnostic information.

Be sure to follow the following instructions in order to ensure success in developing your robot platform.

2.3 Primer: CTRE CAN Devices

CTR-Electronics has designed many of the available CAN bus devices for FRC-style robotics. This includes:

- Talon SRX and Victor SPX motor controllers (PWM and CAN)
- Pigeon IMU
- CANifier
- Pneumatics Control Mode (PCM)
- Power Distribution Panel (PDP)

These devices have similar functional requirements, specifically every device of a given model group requires a unique device ID for typical FRC use (settings, control and status). The device ID is usually expressed as a number between '0' and '62', allowing use for up to 63 Talon SRXs, 63 Victors, 63 PDPs, etc. at once. This range does not intercept with device IDs of other CAN device types. For example, there is no harm in having a Pneumatics Control Module (PCM) and a Talon SRX both with device ID '0'. However, having two Talon SRXs with device ID '0' will be problematic.

These devices are field upgradable, and the firmware shipped with your devices will predate the "latest and greatest" tested firmware intended for use with the latest API release. Firmware update can be done easily using Phoenix Tuner.

The Talon SRX and Victor SPX provide two pairs of twisted CANH (yellow) and CANL (green) allowing for daisy chaining. Other devices such as the PDP and PCM have weidmuller connectors that accept twisted pair cabling. Often you will be able to use your Talons and Victors to connect together your PCM and PDP to each other.

The CAN termination resistors are built into the FRC robot controller (roboRIO) and in the Power Distribution Panel (PDP) assuming the PDP's termination jumper is in the ON position.

More information on wiring and hardware requirements can be found in the user manual of each device type.

2.4 Primer: What is Phoenix Software

Phoenix is a package that targets LabVIEW, C++, and Java for the FRC Robotics Controller platform, i.e. the NI roboRIO robot controller.

It includes the Application Programming Interface (API), which are the functions you call to manipulate the CTRE CAN bus devices: Talon SRX, Victor SPX, CANifier, and Pigeon IMU.

NOTE: PCM and PDP API are built into the core WPI distribution.

The C++ and Java APIs are very similar, typically only differing on the function name case (configAllSettings in Java versus ConfigAllSettings in C++). Because Java is more widely used in FRC than C++, this document will reference the Java routine names. C++ users should take note that the leading character of every function is UPPERCASE in C++.

Additionally, Phoenix shared libraries are also targeted for C++ on Linux (amd64, armhf, aarch64) and typically available on our maven repository. The example support libraries use socket-can for CANBus access, however custom drivers can be provided by the end user for alternative CANBus solutions (NVIDEA TX2 native CAN bus for example).

Phoenix also includes a NETMF (C#) class library for the non-FRC HERO Robot Controller. This can replace the roboRIO in use cases that don't require the full features of the FRC control system, and are not in use during competition.

Note: Phoenix framework is designed so that it is possible to control/leverage Talons, Victors, Pigeons, CANifiers outside of the roboRIO (Rasp Pi + socket CAN for example), and use the roboRIO/DriverStation for safely enable/disable the actuators. Whether this is FRC legal will depend on the 2019 FRC rules.

There are tons of examples in all languages at CTRE's GitHub account:

- https://github.com/CrossTheRoadElec/Phoenix-Examples-Languages
- https://github.com/CrossTheRoadElec/Phoenix-Examples-LabVIEW

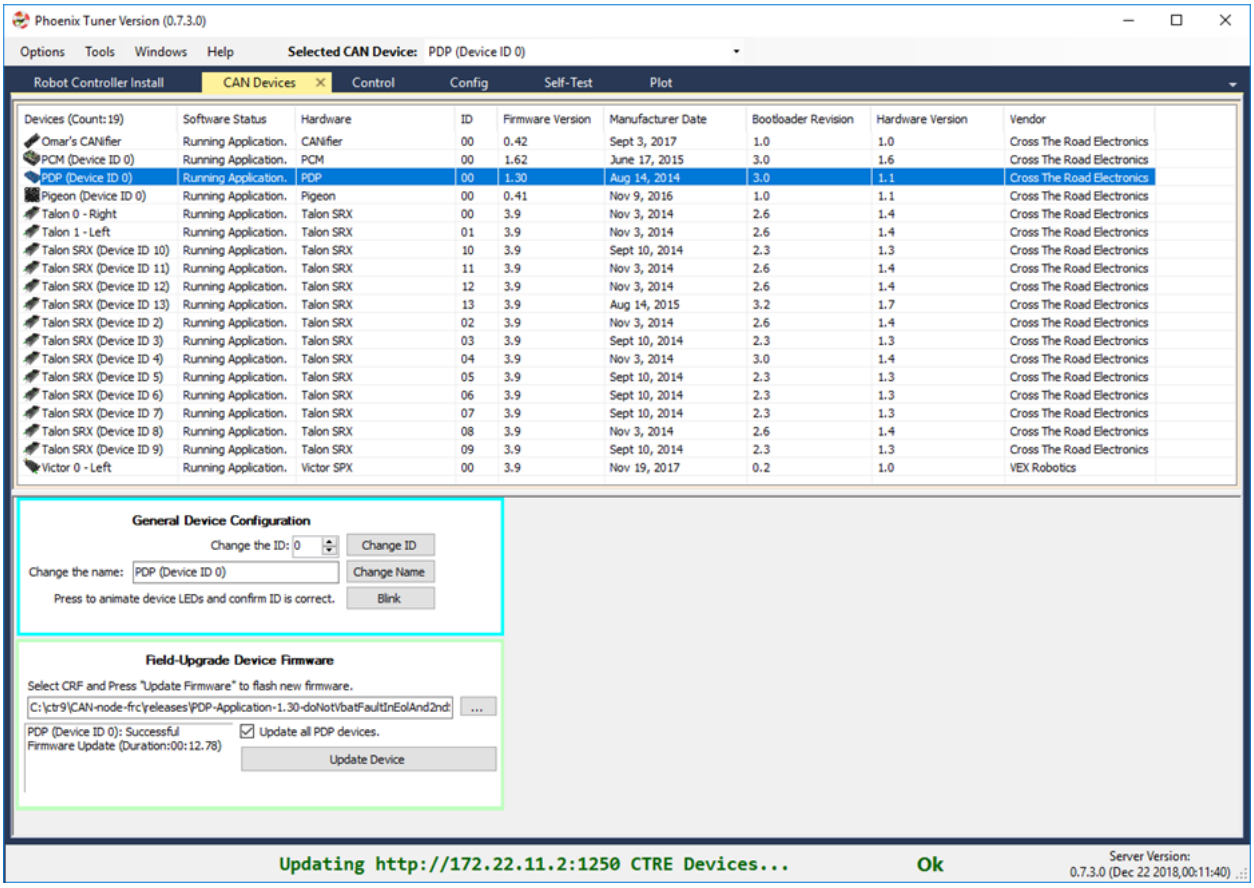
Entire GitHub organization: https://github.com/CrossTheRoadElec/

Phoenix-Examples-Languages and Phoenix-Examples-LabVIEW are specifically tested on the FRC RoboRIO control system.

Phoenix-Linux-SocketCAN-Example demonstrates control of Talons from a Raspberry Pi. https://github.com/CrossTheRoadElec/Phoenix-Linux-SocketCAN-Example

2.4.1 What is Phoenix Tuner?

Phoenix-Tuner is the **2019 replacement for the FRC Web-Based Configuration Utility** that was provided during the previous years.

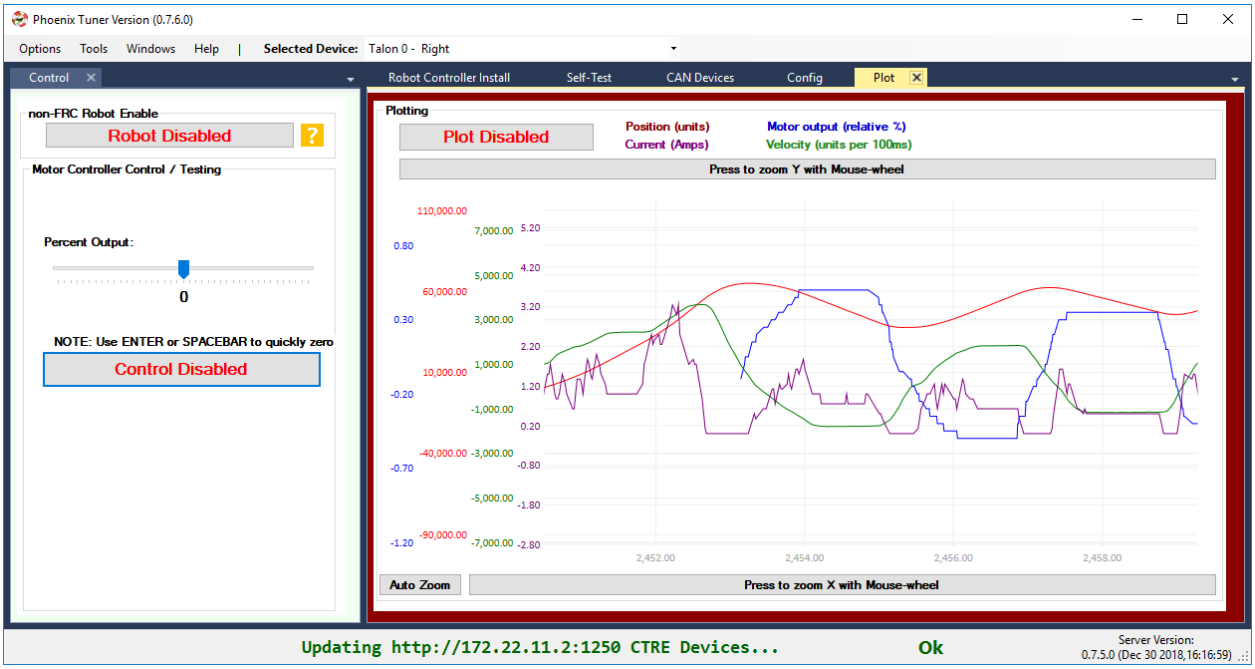


Warning: The RoboRIO web-based config is no longer available or supported by NI.

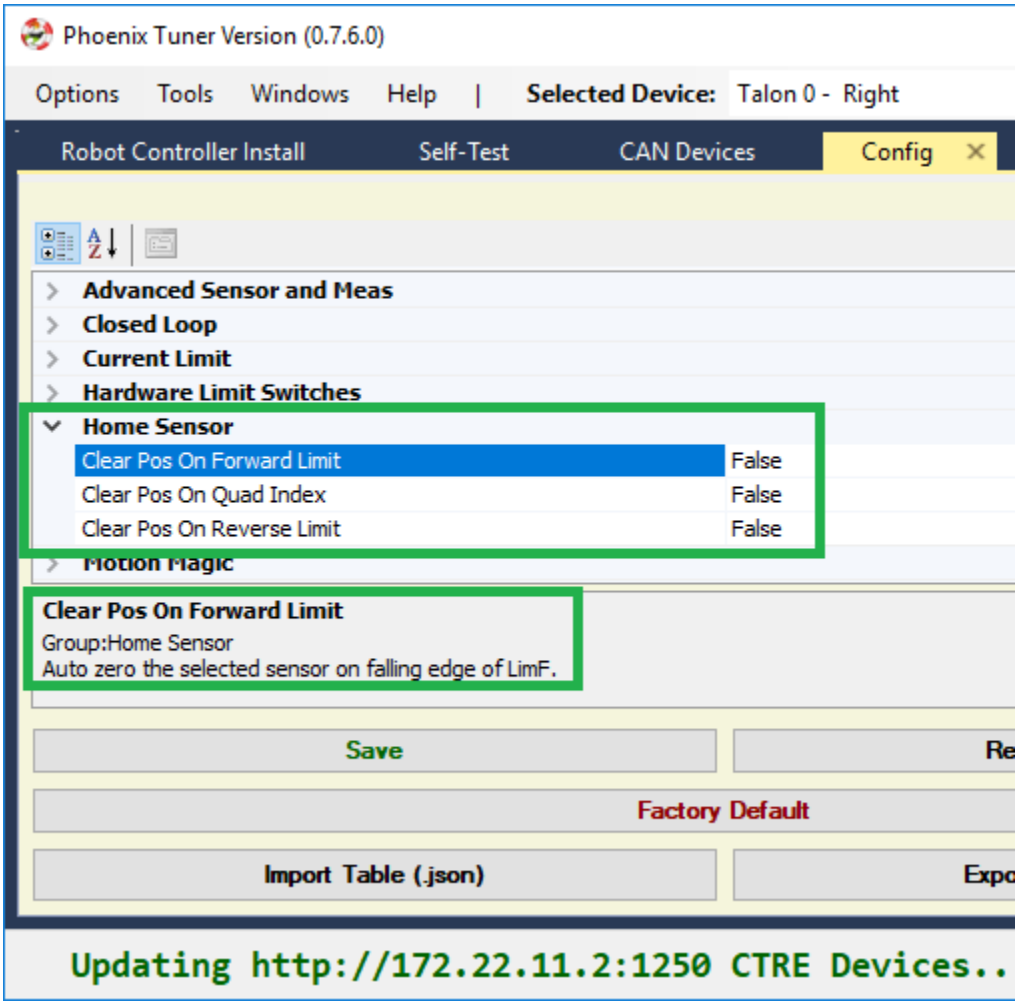
It provides the *same functionality* as the *previous season's web-based interface*, plus a few more features:

- Update device firmware (including PDP/PCM)
- Change CAN IDs
- Self-test devices
- Change configuration settings
- Factory default configuration settings
- Test motors
- Check plots
- Temperature Calibrate Pigeon-IMU

Now you can drive your motors and collect data *without writing any softare*.



Configuration values can be **checked, modified, and defaulted** with the new config view. Config values can also be **imported/exported** as an easy-to-follow JSON formatted file.



The following sections of documentation will cover how to use Phoenix Tuner and the other components of Phoenix.

Tip: Have a feature request? Send to us at support@ctr-electronics.com or report it on GitHub.

2.5 Do I need to install any of this?

Yes, if any of the following:

- You need library support for Talon SRX, Victor SPX, CANifier, Pigeon IMU
- You need to field upgrade Talon SRX, Victor SPX, CANifier, Pigeon IMU, PDP, or PCM
- You want to use Phoenix Tuner for CAN diagnostics (highly recommended)

Note: PCM and PDP objects are already supported in the base FRC installation. However, Phoenix Tuner is required for setting the device ID, field-upgrade, and self-test.

2.6 Prepare your workstation computer

2.6.1 Before Installing Phoenix...

It is strongly recommended to complete the base installation of FRC tools. https://wpilib.screenstepslive.com/s/currentCS/m/getting_started/l/144981-frc-software-component-overview

Warning: You will need to image the roboRIO to 2019 software before continuing. The **roboRIO** kickoff versions are **image 2019_v12 and firmware 6.0**.

Test base FRC Installation - FRC LabVIEW

If a team intends to use LabVIEW to develop robot software, be sure to complete the full NI installer. At which point, opening LabVIEW should reveal the FRC-styled graphical start menu.

At this point it is recommended to create a simple template project and test deploy to the roboRIO. Be sure the DriverStation can communicate with the robot controller, and that DS message log is functional.

Note: LabVIEW is versioned 2018 due to its release schedule. Therefore, LV2018 is used for the 2019 season.

Test base FRC Installation - FRC C++ / Java

It is recommended to install the FRC Driver Station Utilities. This will install the Driver Station software, which is necessary for:

1. Basic comms checks
2. Reading joystick data
3. Generally required for enabling motor actuation (Phoenix Tuner provides an alternative).

Important note regarding FRC 2019 C++ / Java.

The FRC C++/Java standard distribution for 2019 is **quite different** than previous seasons. WPILIB has replaced the Eclipse-based development environment with Microsoft Visual Studio Code and GradleRIO.

This is a considerable change in both user experience and in implementation. If you are developing C++/Java FRC programs, we strongly recommend testing full deployment to your robot controller before installing Phoenix and porting previous season software. A recommended test is to:

1. Create a project from scratch
2. Make a simple change such as add a print statement with a counter.
3. Deploy (or debug) your application.
4. Confirm robot can be teleop-enabled in DS.
5. Open FRC message Console and read through all the messages.
6. Repeat 2 - 5 ten times. This will train students to become familiar with and build general confidence in the tools.

Note: Third-party vendor libraries are installed into the C++/Java project, not the environment. For each C++/Java project you create, you must use the WPI provided tools to select Phoenix to bring the libraries into your project.

2.6.2 What to Download (and why)

Option 1: Windows installer (strongly recommended)

Environments: Windows-LabVIEW, Windows-C++/Java, HERO C#

Phoenix Installer zip can be downloaded at:

http://www.ctr-electronics.com/hro.html#product_tabs_technical_resources.

It is typically named Phoenix Framework_Windows_vW.X.Y.Z.zip

This will install:

- The LabVIEW Phoenix API (if LabVIEW is detected and selected in installer)
- The C++/Java Phoenix API (if selected in installer)
- Device Firmware Files (that were tested with the release)
- CTRE Support of RobotBuilder
- Phoenix Tuner
  - Installs Phoenix API libraries into the roboRIO (required for LabVIEW)
  - Installs Phoenix Diagnostics Server into the RoboRIO (needed for CAN diagnostics).
  - Plotter/Control features
  - Self-test
  - Device ID and field-upgrade

Option 2: Phoenix API via Non-Windows Zip

Environments: Linux/MacOS - C++/Java

The Phoenix API can be manually installed on non-Windows platforms by downloading the "non-Windows" zip and following the instructions found inside.

This essentially contains a maven-style repository that holds the API binaries and headers, as well as a "vendordeps" JSON file that instructs VS how to incorporate the Phoenix C++/Java API libraries.

Note: This is auto installed when using the Windows full installer (Option 1).

Phoenix Tuner

Environments: Windows

If you are using option 2, you will need to download Phoenix Tuner separately. Phoenix Tuner is available here... https://github.com/CrossTheRoadElec/Phoenix-Tuner-Release/releases

This can be convenient for workstations that aren't used for software development, but are used for field-upgrade or testing motor controllers.

Note: LabVIEW teams may need to use Phoenix Tuner to install Phoenix libraries into the roboRIO. More information on this can be found under Prepare Robot Controller.

Note: This is auto installed when using the Windows full installer.

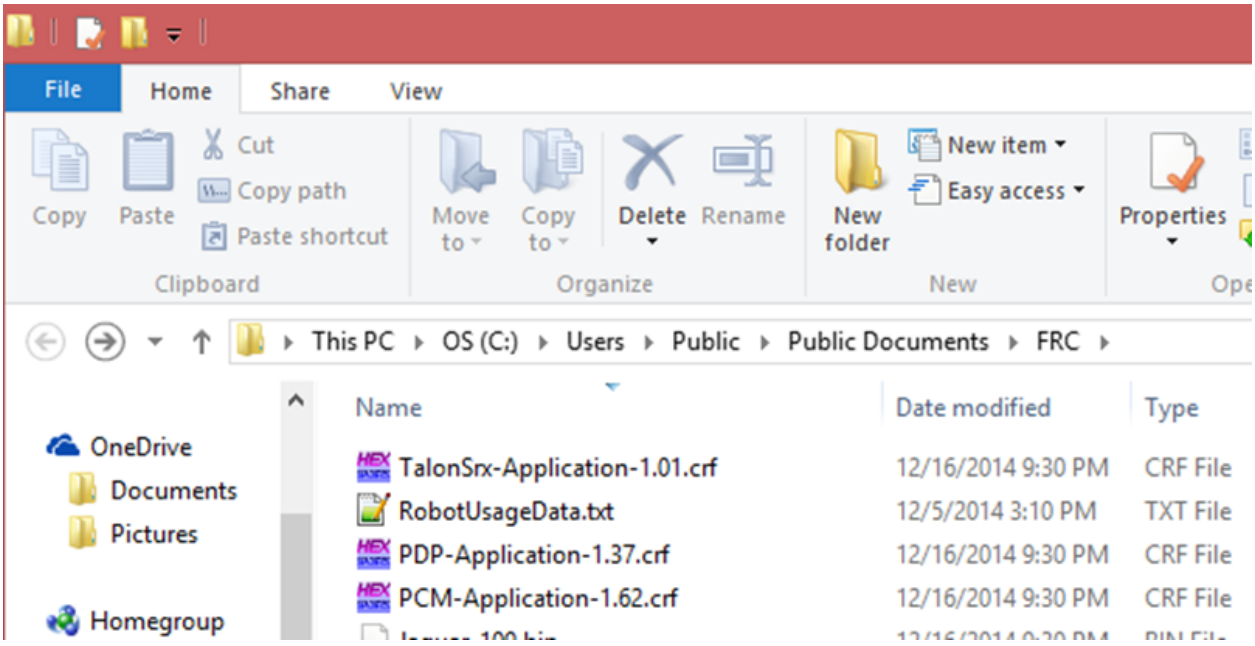
Note: Developers may be interested to know that all Phoenix Tuner features leverage an HTTP API provided by the Phoenix Diagnostics Server. As such, custom tooling can be developed to field-upgrade, test-control, or diagnostics CTRE devices without Tuner.

Device Firmware Files (crf)

The test firmware files for all CTRE devices are packaged with the Windows Installer (and has been for years). However, many FRC teams don't notice, or prefer to download them directly from the product pages on the ctr-electronics.com website. If Internet access is available, they can be downloaded as such.

The FRC Software installer will create a directory with various firmware files/tools for many control system components. Typically, the path is:

```
C:\Users\Public\Documents\FRC
```



When the path is entered into a browser, the browser may fix-up the path:

```
C:\Users\Public\Public Documents\FRC
```

In this directory are the initial release firmware CRF files for all CTRE CAN bus devices, including the Talon SRX.

The latest firmware to be used at time of writing is version 4.11 for Talon SRX, Victor SPX, and 4.00 for CANifier, and Pigeon IMU.

Note: Additionally, newer updates may be provided online at http://www.ctr-electronics.com.

Note: Be sure to watch for team updates for what is legal and required!

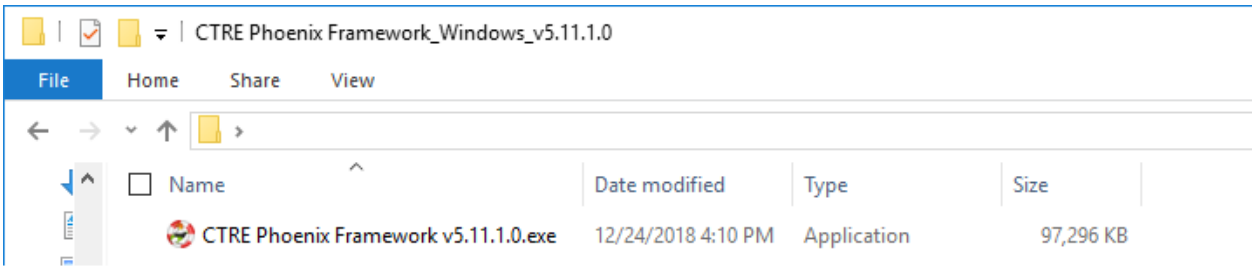
Note: There is no longer FRC versus non-FRC firmware for motor controllers. Instead the latest firmware detects if the use case is FRC. If so, the device will FRC-Lock, and will require the Driver Station for actuation.

2.6.3 Workstation Installation

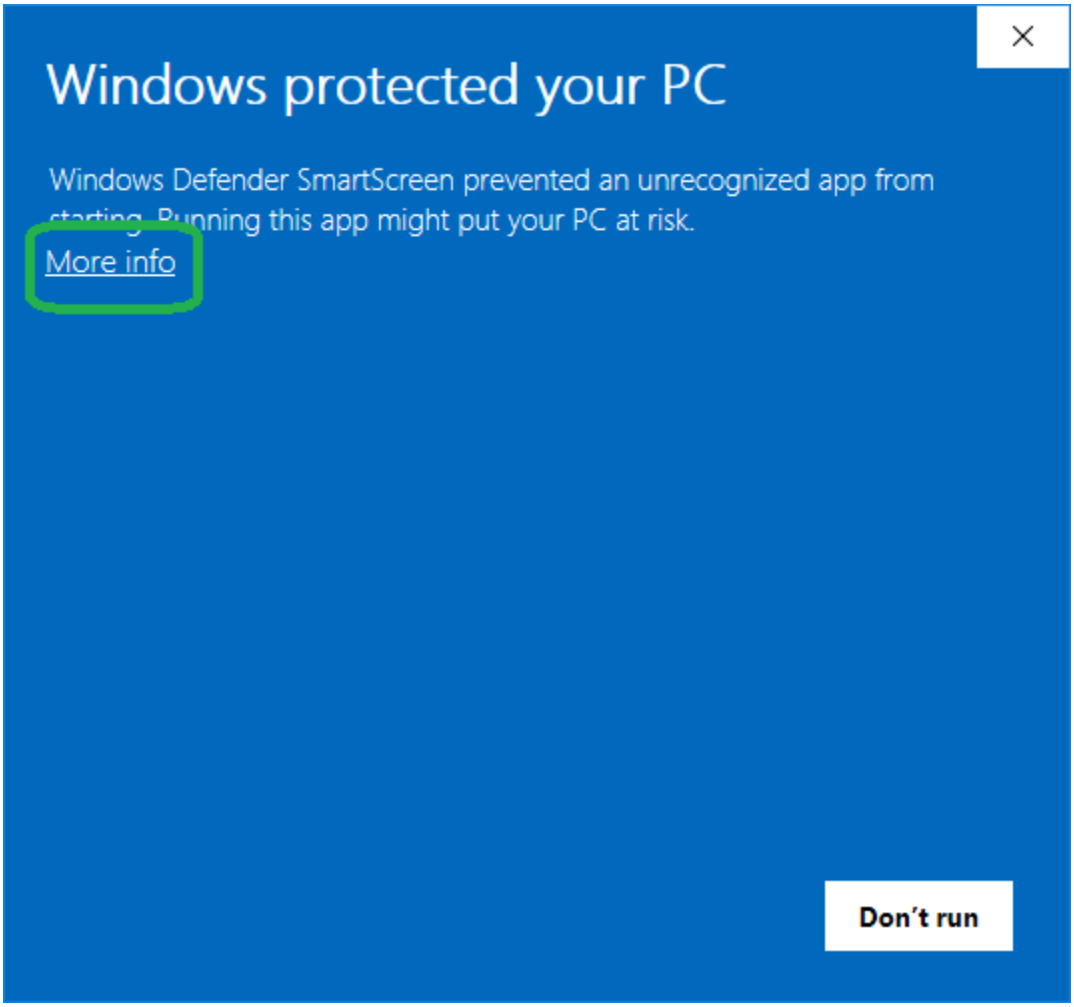
There are three installation methods listed below. The simplest and recommended approach is to run the Windows Installer (Option 1).

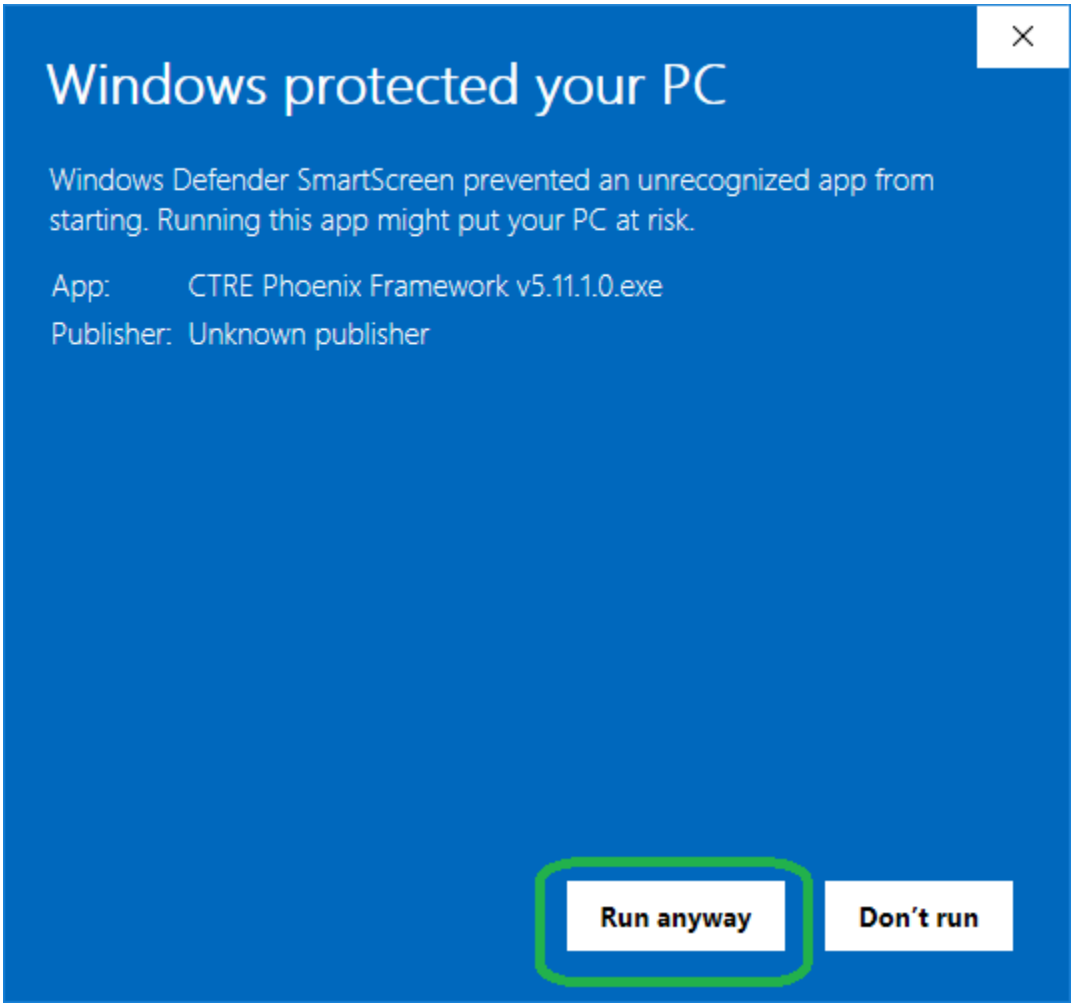
Option 1: Windows Offline Installer (C++/Java/LabVIEW, HERO C#)

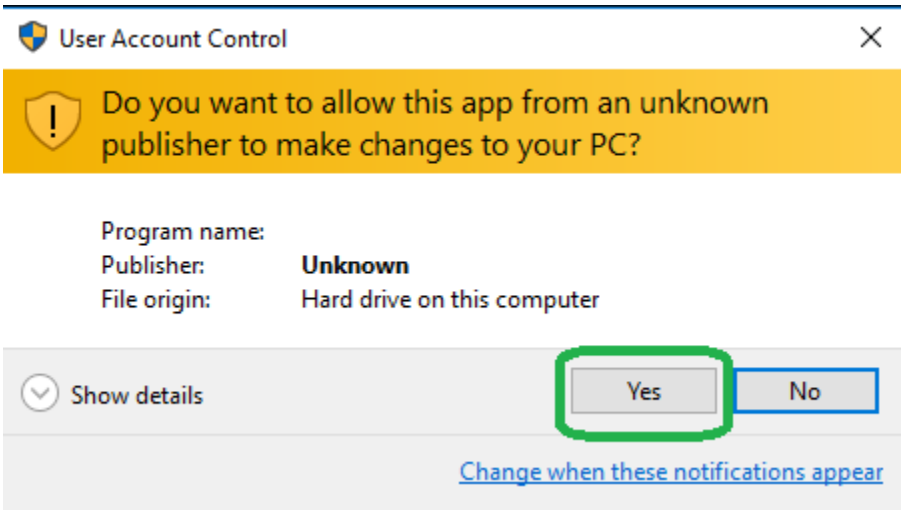
Un-compress the downloaded zip.



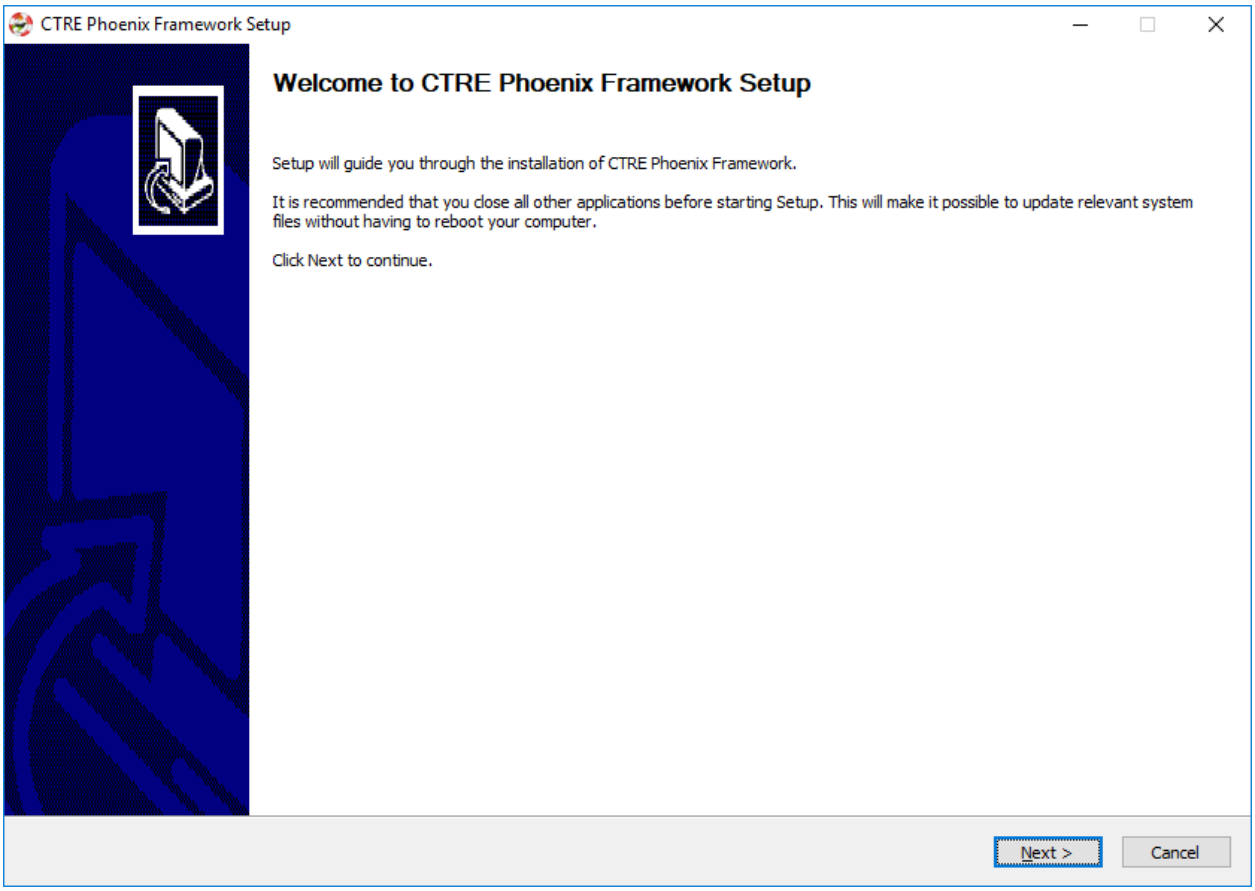
Double click on the installer. If the Windows protection popup appears press More Info, then Run anyway.


Windows protected your PC
Windows Defender SmartScreen prevented an unrecognized app from
starting. Running this app might put your PC at risk.
More info
Don't run





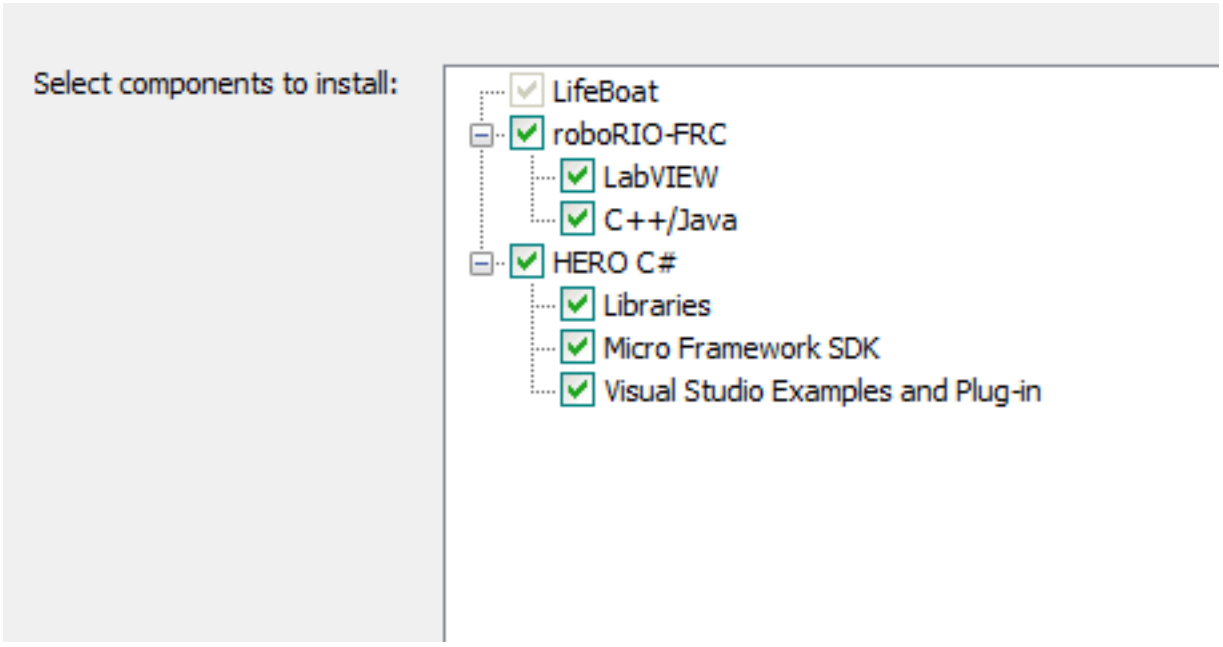
This will look very similar to previous installers - make sure you have the relevant component selected for your programming language.



LV Teams: Make sure LabVIEW is selected. If it is grayed out, then LabVIEW was not installed on the PC.

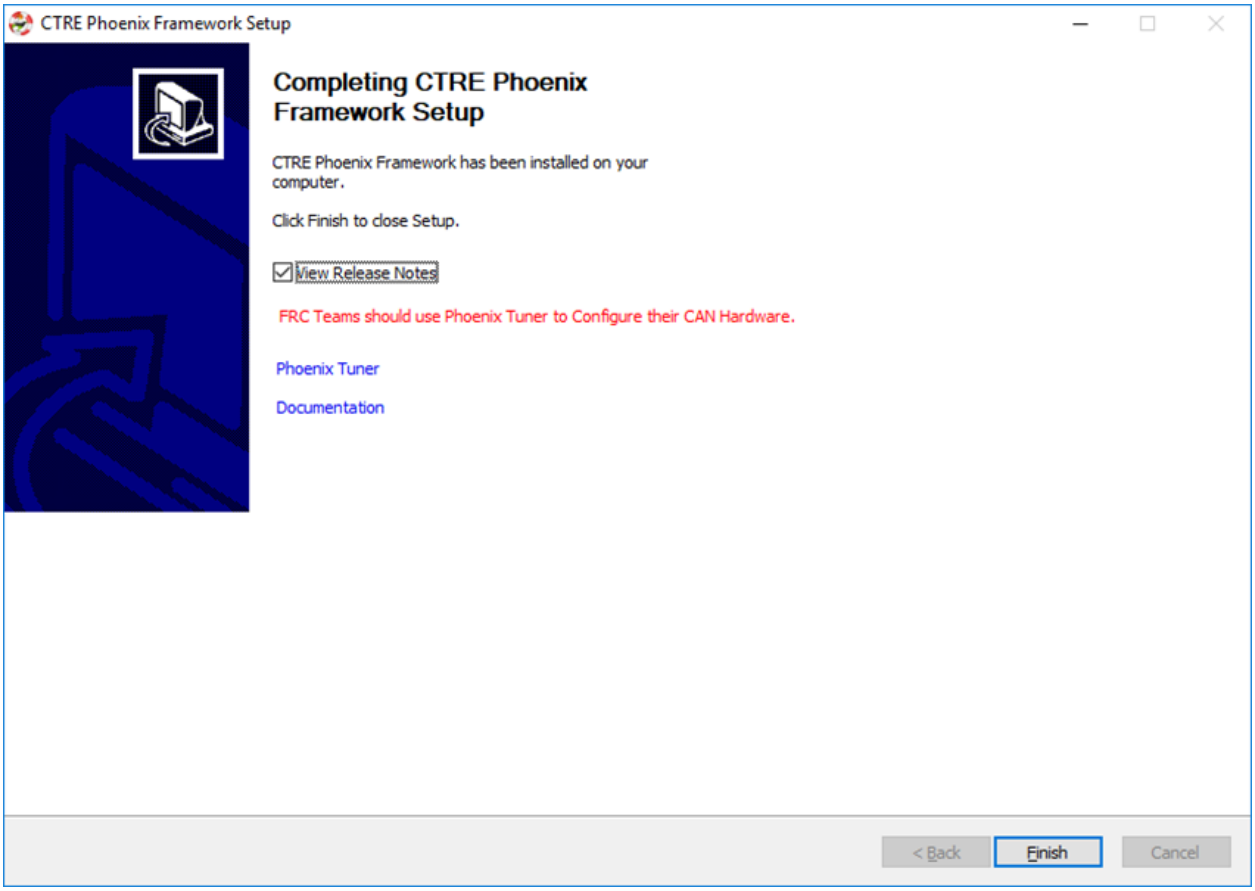
C++/Java Teams: Make sure C++/Java is selected.

If Visual Studio 2017 (Community/Professional) is detected, HERO C# will be selected. This can be turned off to speed up the installer.



Installer can take anywhere from 30 seconds to 5 minutes depending on which Microsoft runtimes need to be installed.

Final page will look like this. The Phoenix Tuner link can be used to open Phoenix Tuner. Alternatively, you can use the Windows Start Menu.



Option 2: Non-Windows Zip (C++/Java)

The zip will contain **two folders, "maven" and "vendordeps"**. These folders are meant to be **inserted into your frc2019 install folder**.

See WPI documentation for typical location. https://wpilib.screenstepslive.com/s/currentCS/m/cpp/l/1027500-installing-c-and-java-development-tools-for-frc

Copy/paste the maven and vendordeps folder into frc2019 folder. This will override a pre-existing Phoenix installation if present.

Note: This will not install Phoenix Tuner or firmware files. If these are necessary (and they typically are) these can be downloaded separately or consider using the complete Phoenix Installer.

2.6.4 Post Installation Steps

After all workstation installs, the following checks should be followed to confirm proper installation.

FRC C++/Java - Verify Installation

The offline files for vscode are typically installed in:

```
C:\Users\Public\frc2019\vendordeps\Phoenix.json (File used by vscode to include␣
↪Phoenix in your project)
C:\Users\Public\frc2019\maven\com\ctre\phoenix (multiple maven-style library files)
```

Your drive letter may be different than "C:". After running the Phoenix Installer, the instructions to add or update Phoenix in your robot project must be followed.

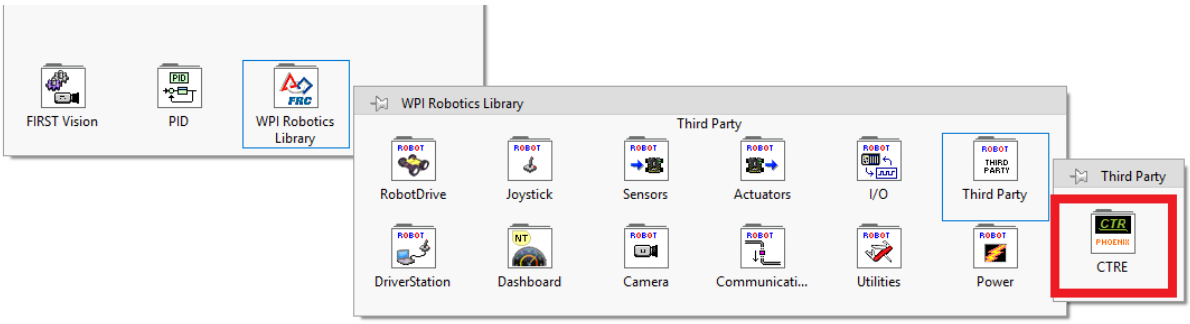
FRC LabVIEW – Verify Installation

After running the installer, open a pristine copy of FRC LabVIEW.

Testing the install can be done by opening LabVIEW and confirming the VIs are installed. This can be done by opening an existing project or creating a new project, or opening a single VI in LabVIEW. Whatever the simplest method to getting to the LabVIEW palette.

The CTRE Palette is located in:

- WPI Robotics Library -> Third Party.

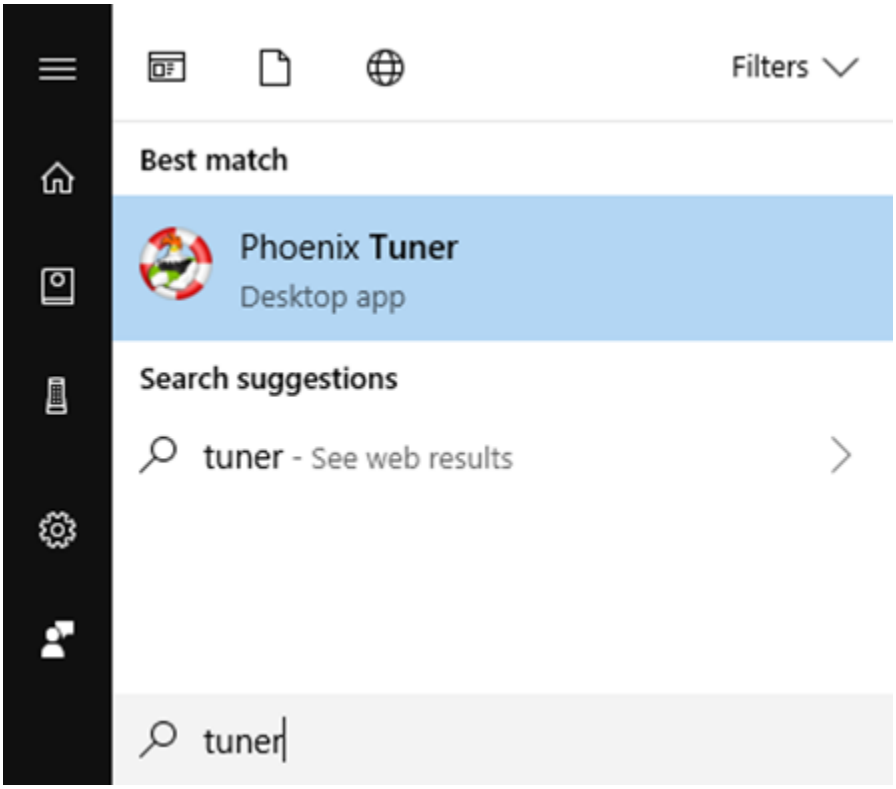


This palette can also be found in:

- WPI Robotics Library -> RobotDrive -> MotorControl -> CanMotor
- WPI Robotics Library -> Sensors -> Third Party
- WPI Robotics Library -> Actuators -> Third Party

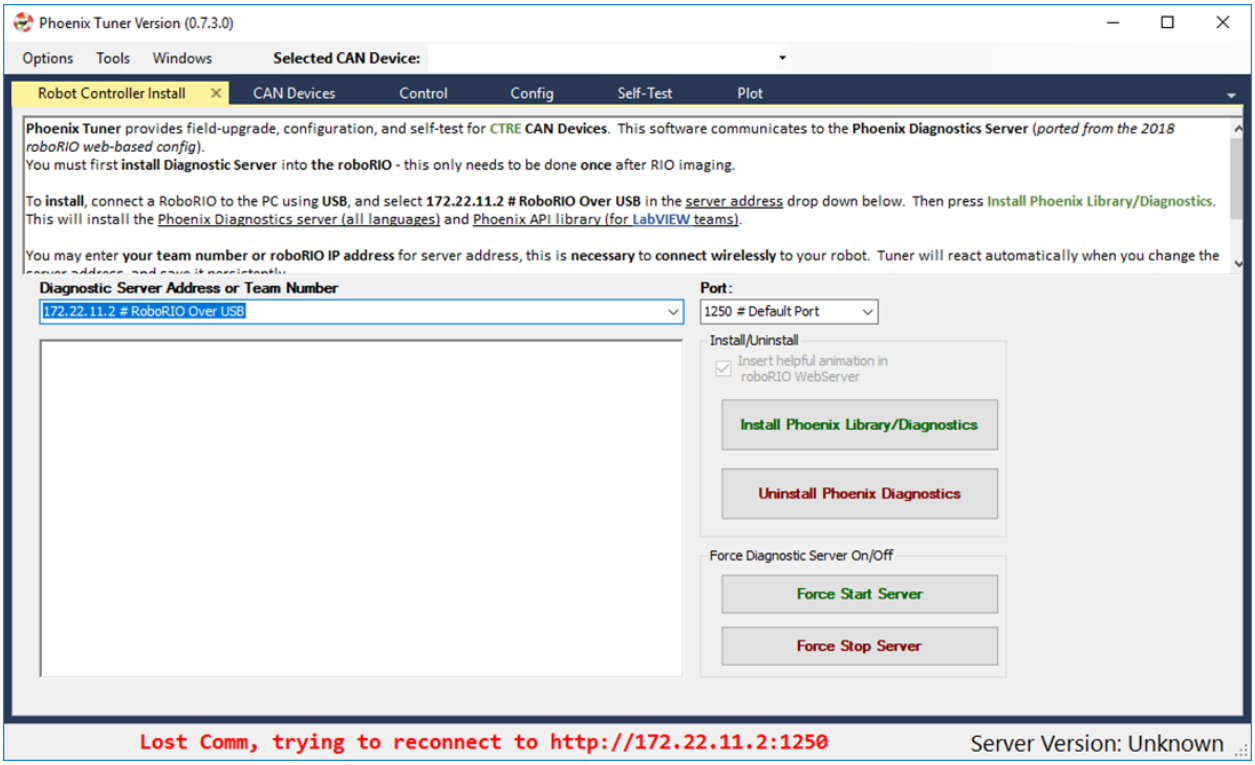
FRC Windows – Open Phoenix Tuner

Open Phoenix Tuner



If this is the first time opening application, confirm the following:

- the status bar should read “Lost Comm”.
- No CAN devices will appear.
- The Server version will be unknown.



2.7 Prepare Robot Controller

2.7.1 Why prepare Robot Controller?

Preparing the Robot Controller typically means:

1. Installing the Phoenix Diagnostic Server
2. Installing the Phoenix API into roboRIO (if using LabVIEW).

Phoenix Diagnostic Server is necessary for Phoenix Tuner to interact with CTRE CAN Devices. Tuner communicates with "Phoenix Diagnostic Server", a roboRIO Linux application that provides an HTTP API.

LabVIEW

NI LabVIEW has a new feature in 2019 that will automatically deploy the Phoenix API libraries to the roboRIO. To enable this feature, the AutoLibDeploy checkbox must be checked in the Phoenix Installer. Because this is a new feature, the check box defaults off. If you choose to use this feature, all 2019 LabVIEW robot projects should automatically install Phoenix into the roboRIO when the program is permanently deployed via "Run As Startup"

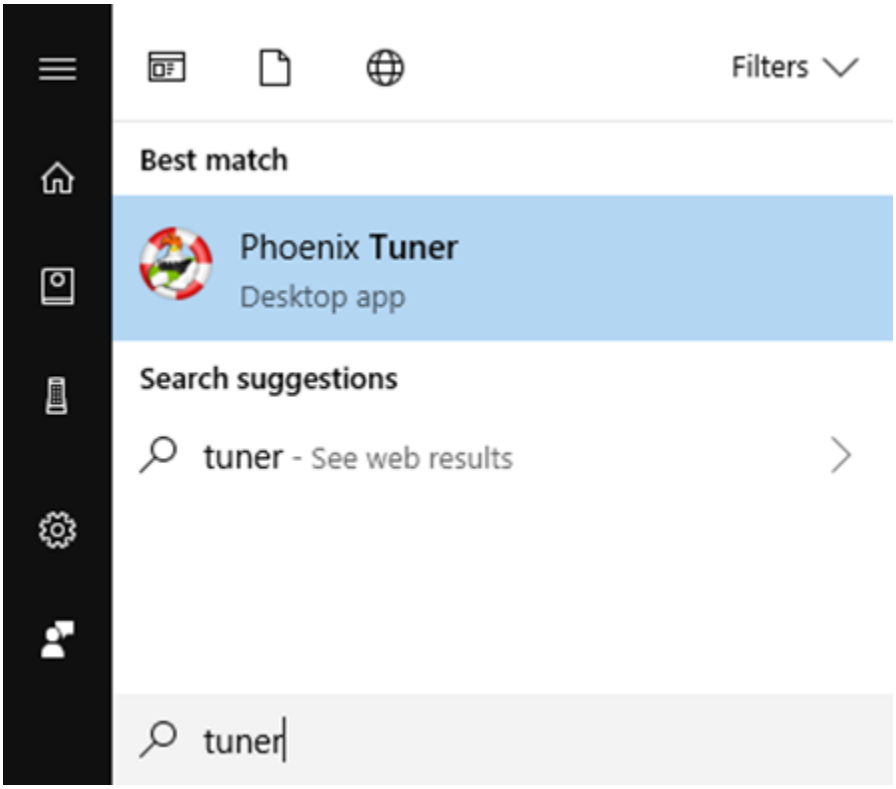
Alternatively you can use Phoenix Tuner to install the Phoenix API libraries into the roboRIO. We recommend this until the AutoLibDeploy features has seen more testing.

This is because the programming language solutions (WPI C++/Java) automatically delivers these libraries whenever the application is deployed in VS Code.

> **Warning:** If the roboRIO is re-imaged, these steps must be followed again for Tuner and LabVIEW to function.

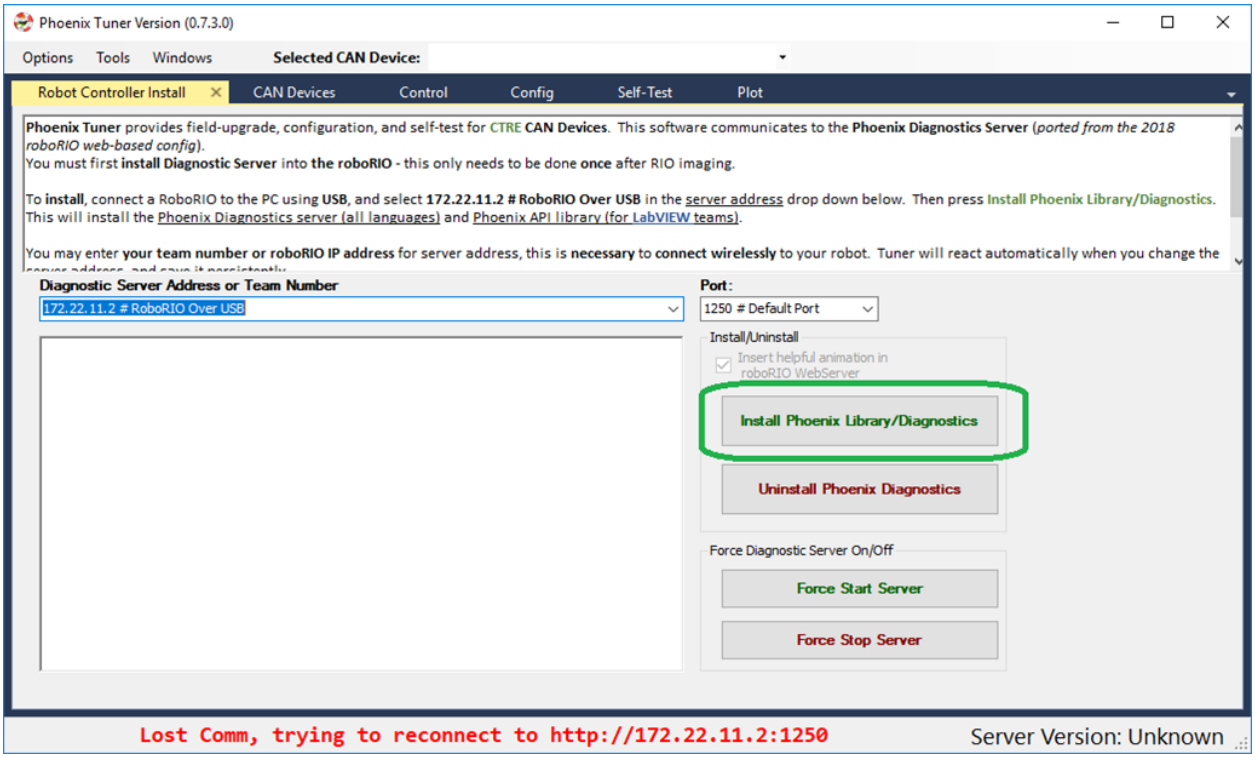
2.7.2 How to prepare Robot Controller

Open Tuner and connect USB between the workstation and the roboRIO.



Select **172.22.11.2 # RoboRIO Over USB** and **1250** for the **address** and **port**. These are generally selected by default, and typically do not require modification.

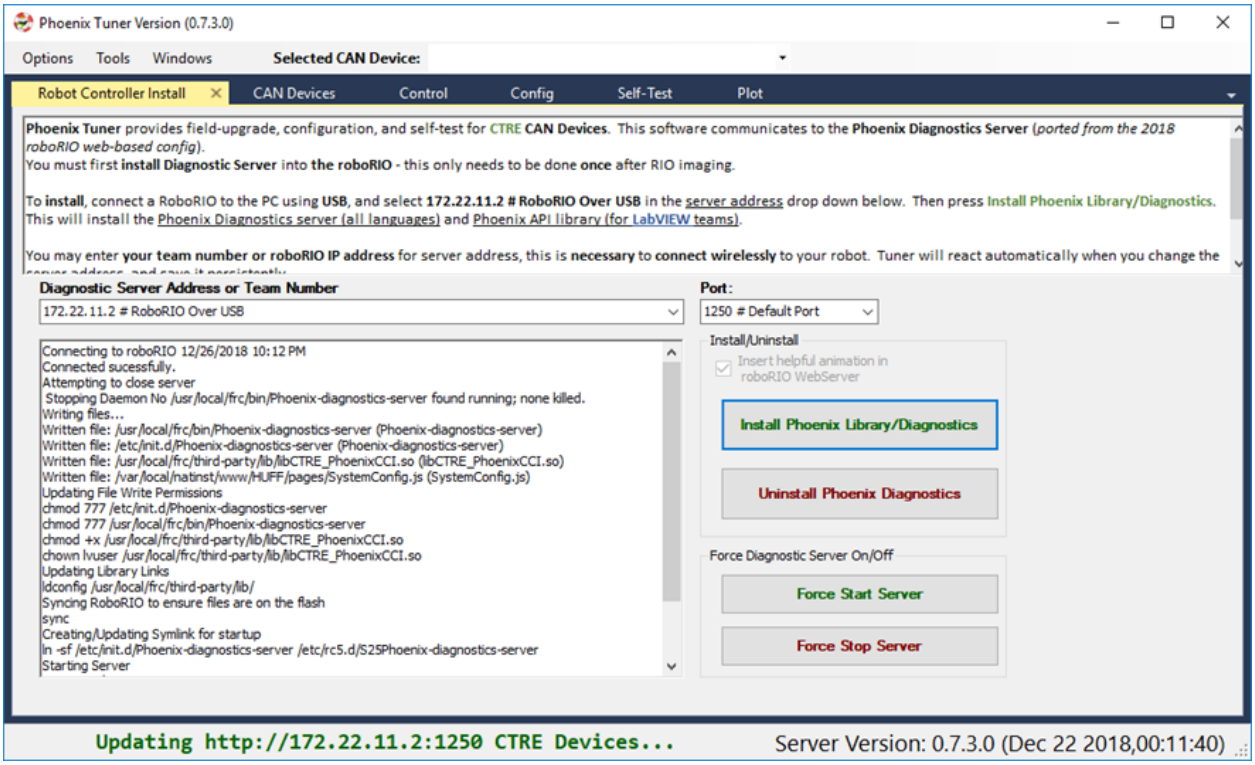
Press the Install button.



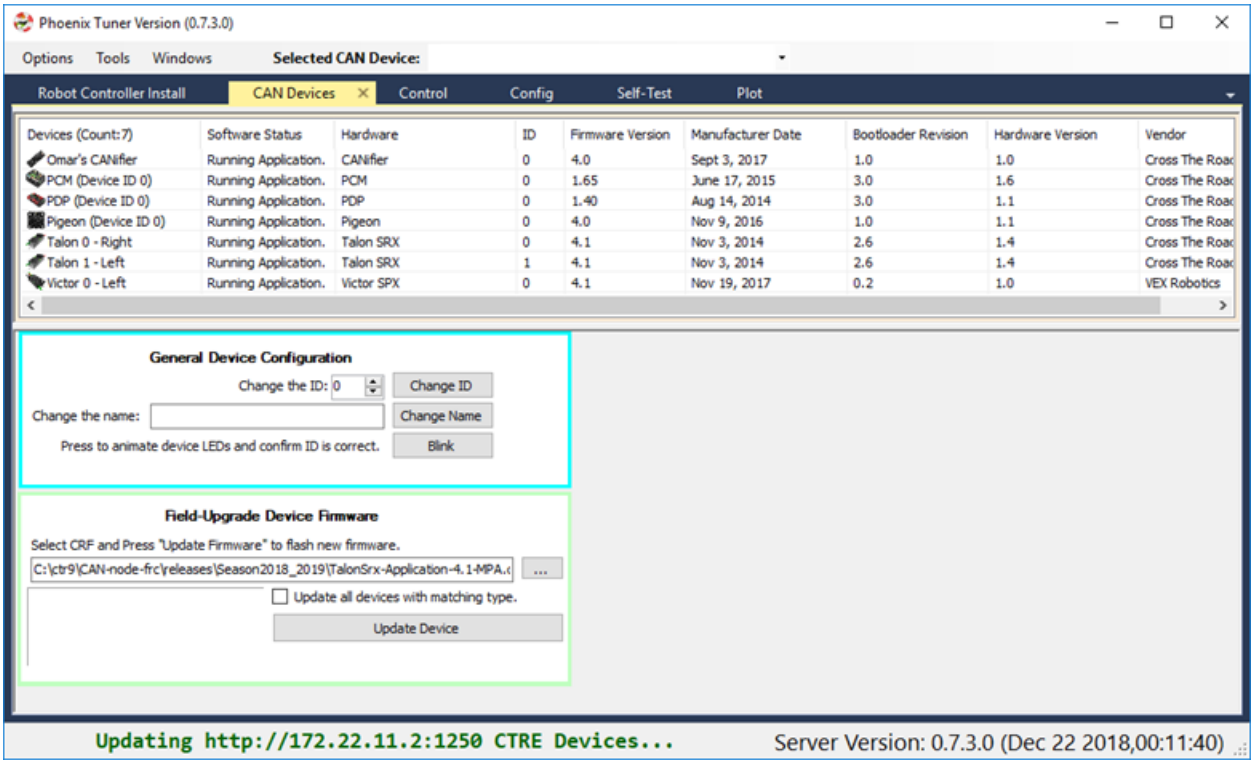
2.7.3 Verify the robot controller - Tuner

After installation is complete, Tuner will immediately connect to the roboRIO.

Confirm the bottom status bar is green and healthy, and server version is present.



If there are CAN device present, they will appear. However, it is possible that devices are missing, this will be resolved in the next major section (CAN Bus bring up).



roboRIO Connection (WiFi/Ethernet)

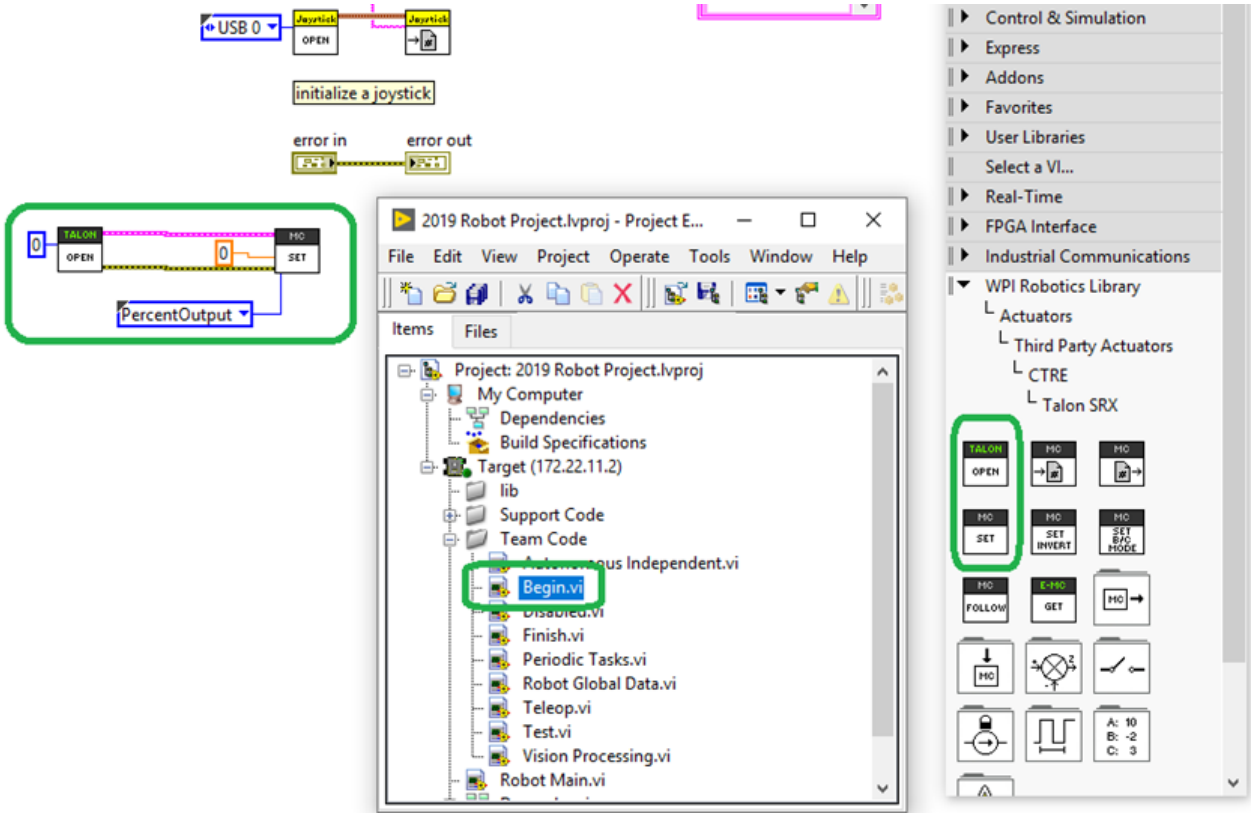
The recommended connection method for control/plotter features is over **USB or using static IP (Ethernet/WiFi)**. The mDNS strategy used by the roboRIO can *sometimes* fail intermittently which can cause hiccups when submitting HTTP requests to the roboRIO.

Testing has shown that using USB (172.22.11.2) or using static IP address has yielded a greater user experience than the roborio-team-frc.local host name has.

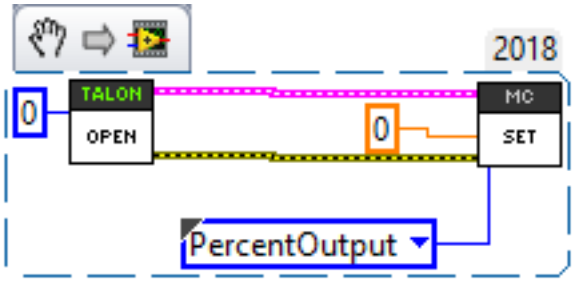
Note: Future releases may have improvements to circumvent the limitations of mDNS.

2.7.4 Verify the robot controller - LabVIEW

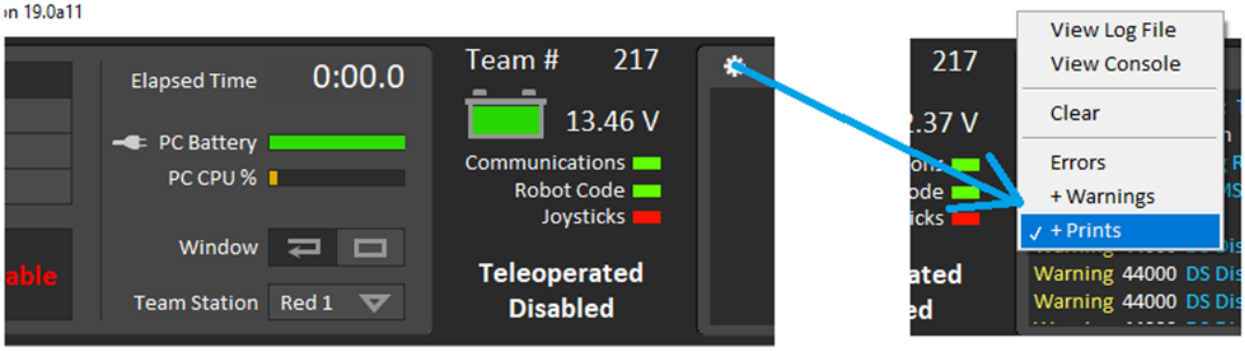
Create a pristine LabVIEW application. Add a CTRE device to Begin.Vi. For example, create a Talon SRX object, even if the device is not physically present.



Tip: Drag drop the following into your Begin.vi

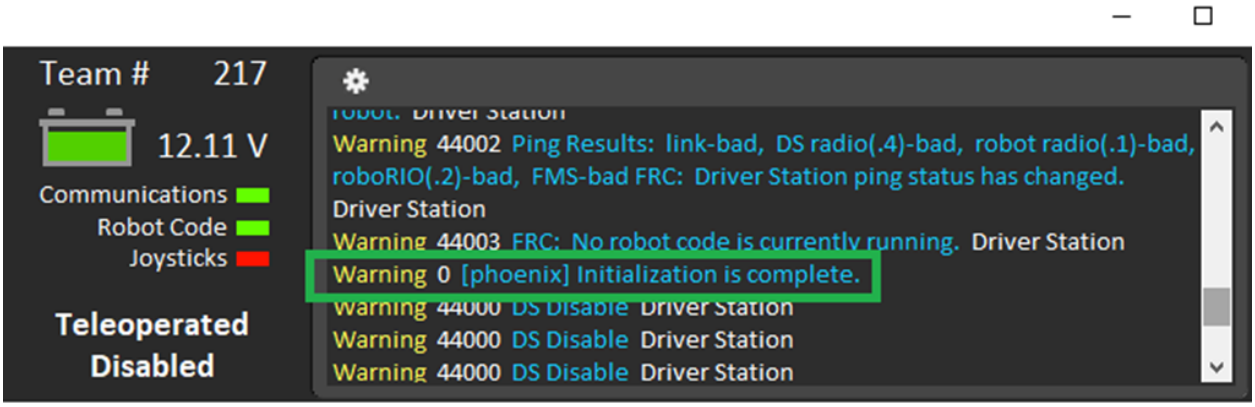


Connect DS and turn on Warnings and Prints by selecting the bottom most option.

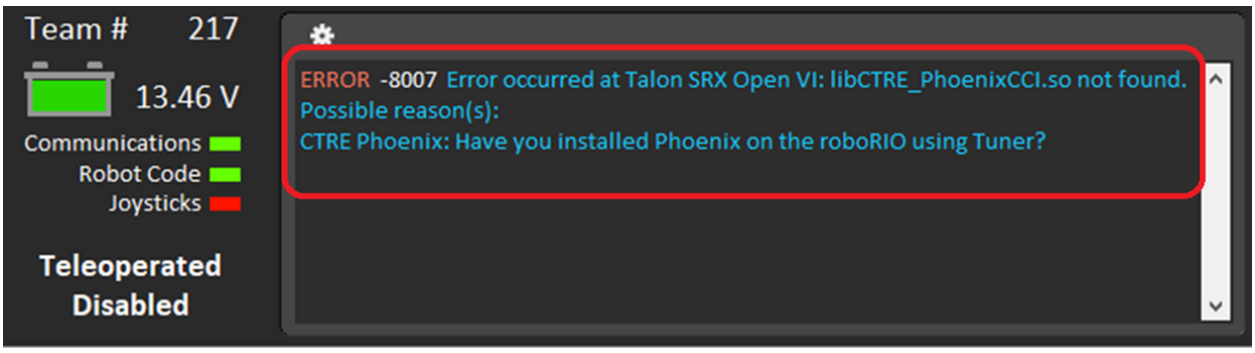


Upload the application to the robot controller and check the driver station message log.

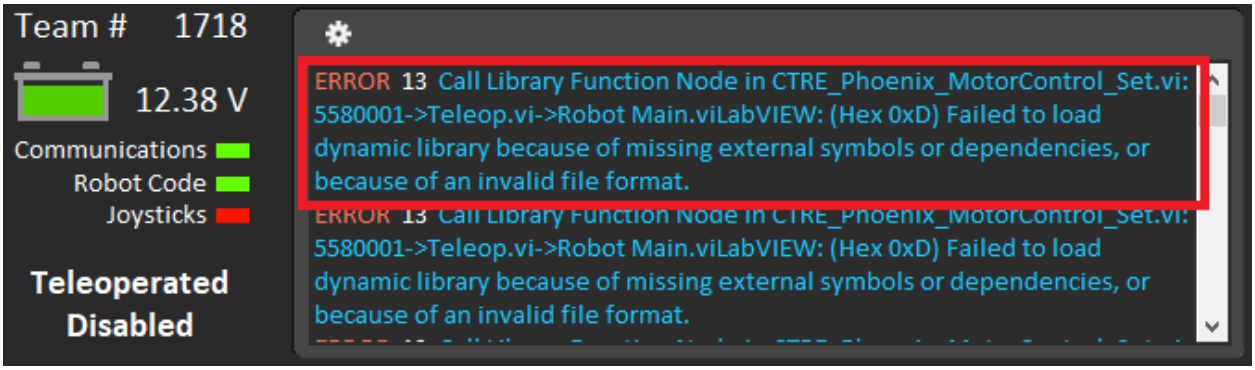
If everything is working, the Phoenix Initialization message can be found. .. note:: This message will not appear after subsequent “soft” deploy (LabVIEW RAM-only temporary deploys).



If Phoenix API has not been installed into the robot controller, this message will appear.



If you have used Phoenix LifeBoat (which should NOT be used), this message will appear. If this occurs you will need to re-image your roboRIO and then re-follow the instructions in this section exactly, without using LifeBoat.

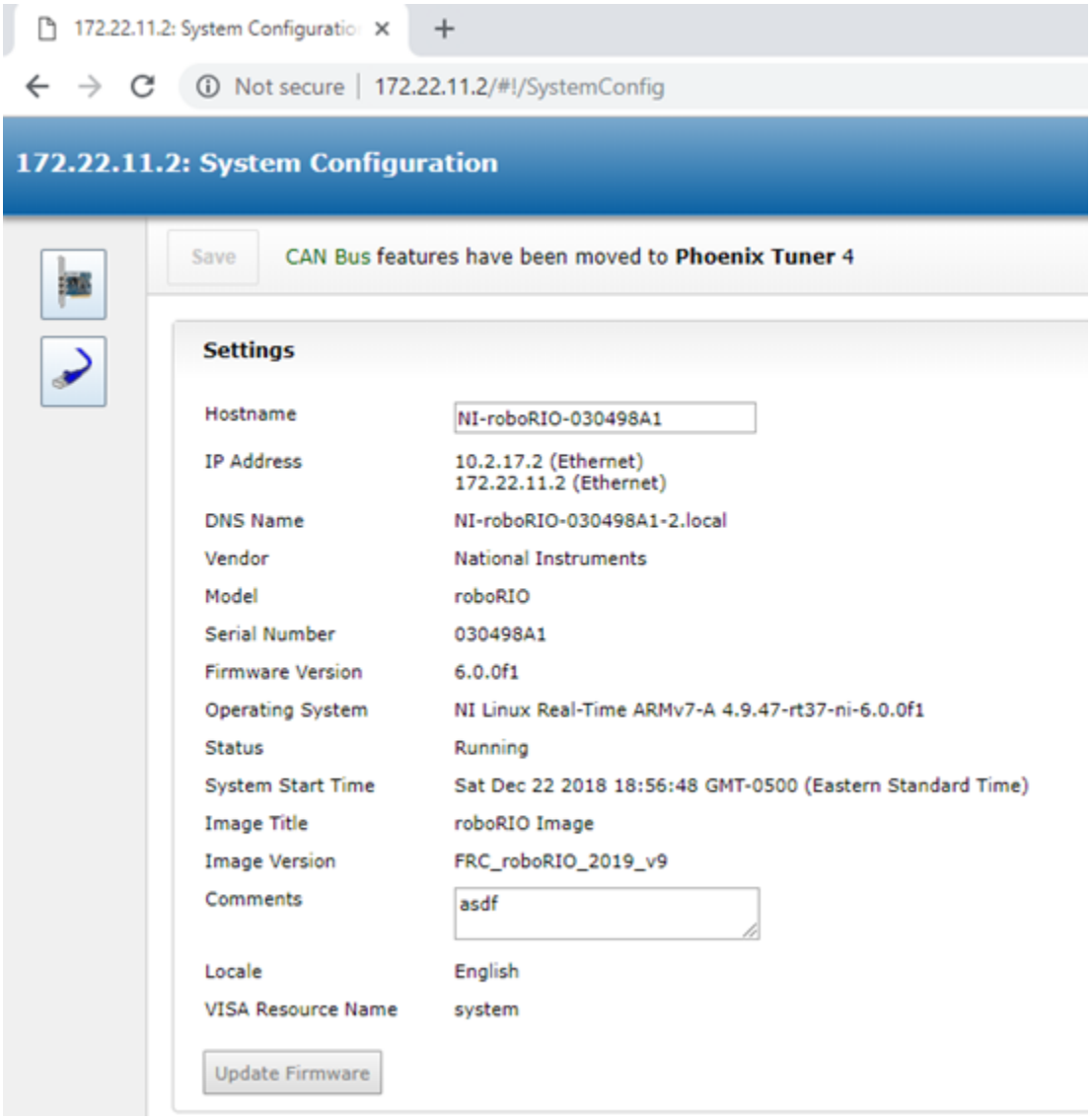


2.7.5 Verify the robot controller – Web page

The Silverlight web interface provided in previous seasons is **no longer available**. Moving forward, the NI web interface will likely be much simpler.

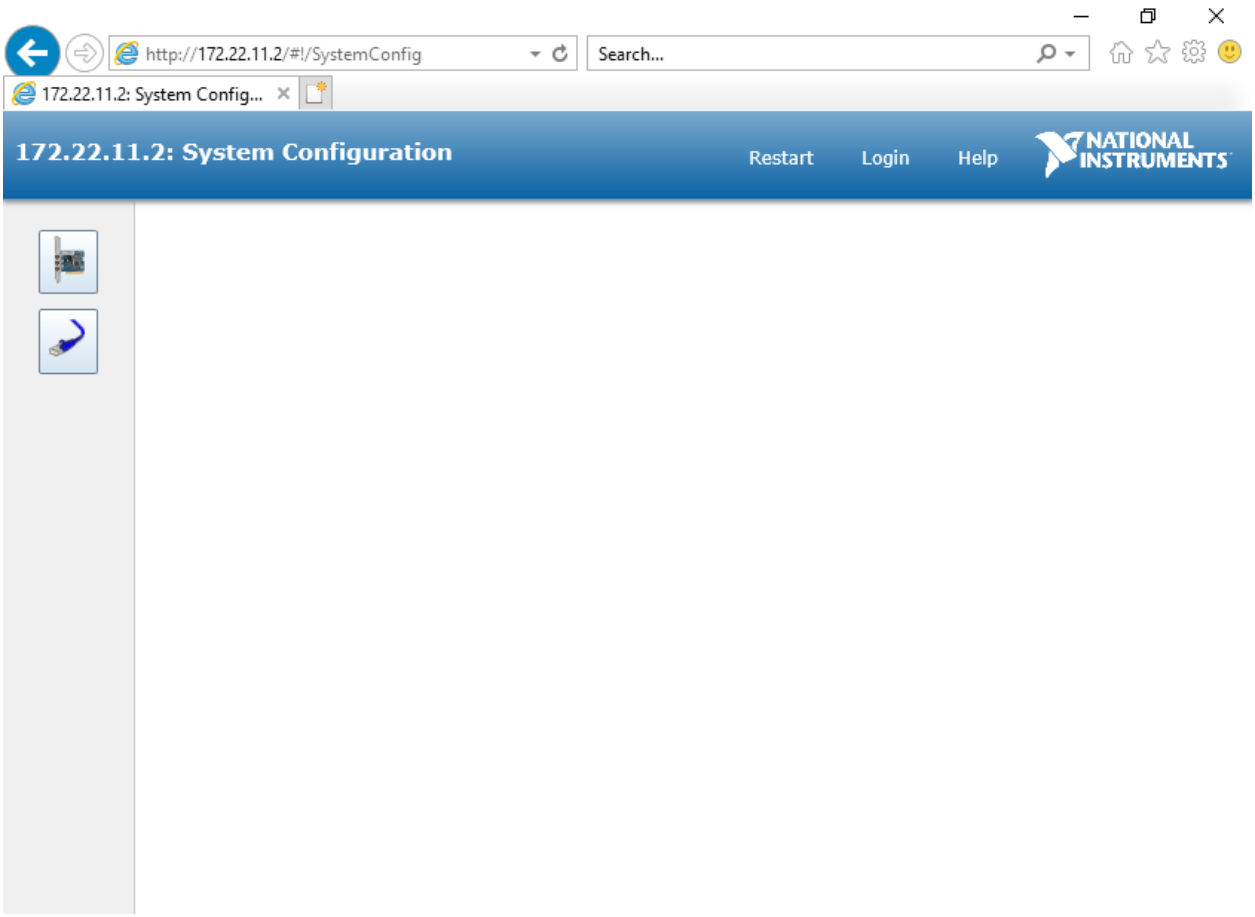
As a result, **Phoenix Tuner** will embed a *small message reminder* **indicating that CAN features have been moved to Tuner**.

Typically, the message will disappear after 5 seconds. This will not interfere with normal web page features (IP Config, etc. . .).



Warning: The roboRIO Web-page does not provide CAN bus support any more as this has been removed by NI. Use Phoenix Tuner instead.

Warning: The roboRIO Web-page does not render correctly if using Internet Explorer (see below). Recommended browsers are Chrome or FireFox.



2.7.6 Verify the robot controller – HTTP API

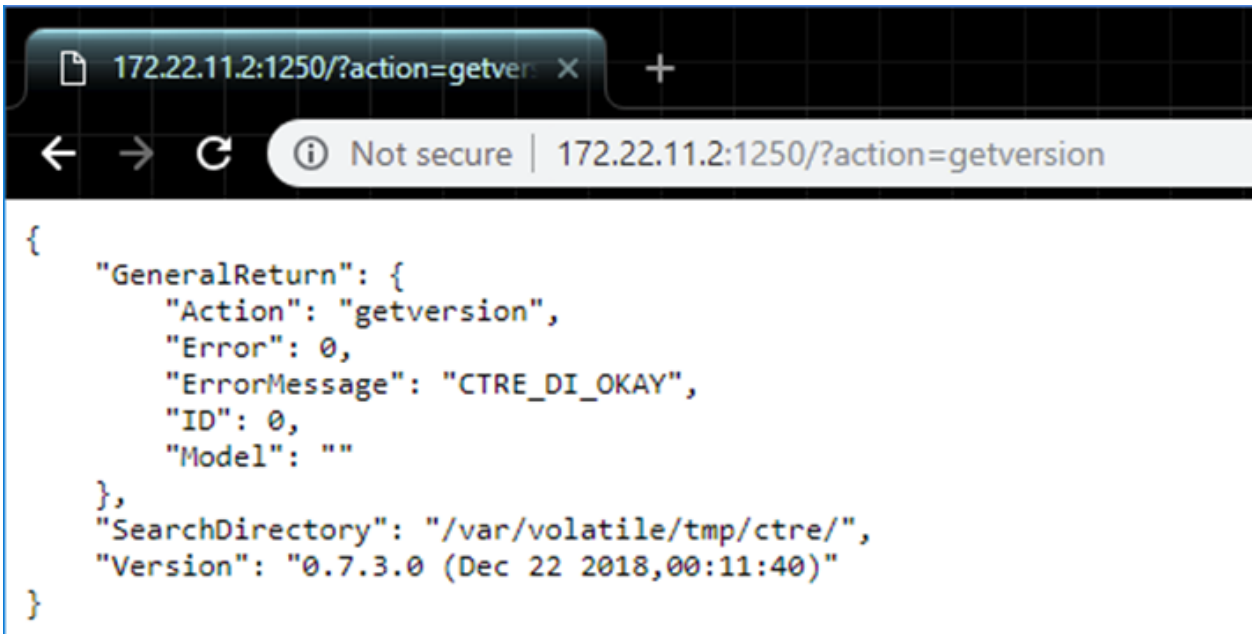
Tuner leverages the HTTP API provided by Phoenix Diagnostics Server.

So technically you have already confirmed this is working.

But, it is worth noting that this HTTP API can potentially be used by third-party software, or even the robot application itself.

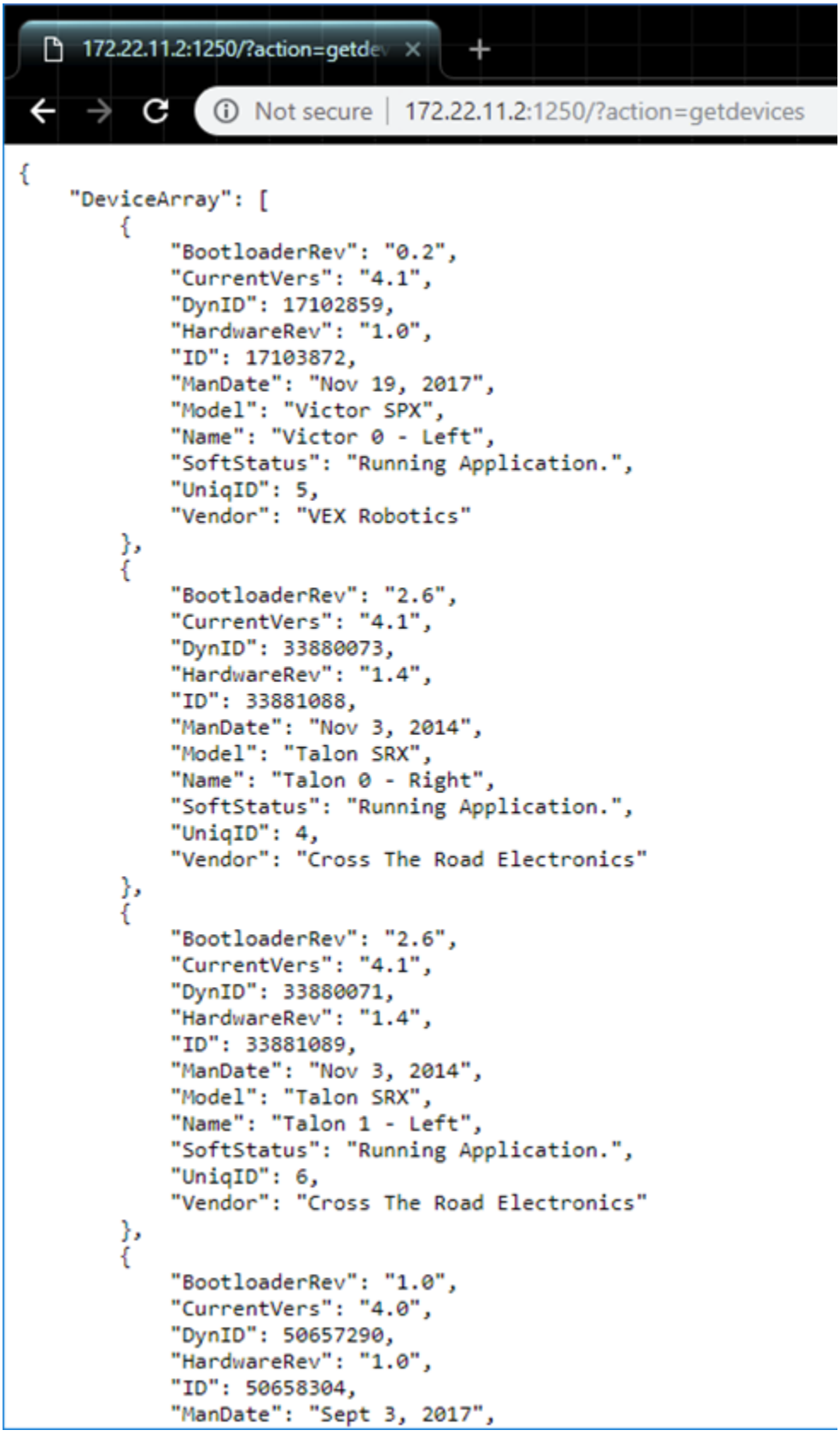
Here is a simple get version command and response.

```
http://172.22.11.2:1250/?action=getversion
```



Here is a simple getdevices command and response.

```
http://172.22.11.2:1250/?action=getdevices
```


172.22.11.2:1250/?action=getdev
Not secure | 172.22.11.2:1250/?action=getdevices
{
"DeviceArray": [
{
"BootloaderRev": "0.2",
"CurrentVers": "4.1",
"DynID": 17102859,
"HardwareRev": "1.0",
"ID": 17103872,
"ManDate": "Nov 19, 2017",
"Model": "Victor SPX",
"Name": "Victor 0 - Left",
"SoftStatus": "Running Application.",
"UniqID": 5,
"Vendor": "VEX Robotics"
},
{
"BootloaderRev": "2.6",
"CurrentVers": "4.1",
"DynID": 33880073,
"HardwareRev": "1.4",
"ID": 33881088,
"ManDate": "Nov 3, 2014",
"Model": "Talon SRX",
"Name": "Talon 0 - Right",
"SoftStatus": "Running Application.",
"UniqID": 4,
"Vendor": "Cross The Road Electronics"
},
{
"BootloaderRev": "2.6",
"CurrentVers": "4.1",
"DynID": 33880071,
"HardwareRev": "1.4",
"ID": 33881089,
"ManDate": "Nov 3, 2014",
"Model": "Talon SRX",
"Name": "Talon 1 - Left",
"SoftStatus": "Running Application.",
"UniqID": 6,
"Vendor": "Cross The Road Electronics"
},
{
"BootloaderRev": "1.0",
"CurrentVers": "4.0",
"DynID": 50657290,
"HardwareRev": "1.0",
"ID": 50658304,
"ManDate": "Sept 3, 2017",

2.8 VS Code C++/Java

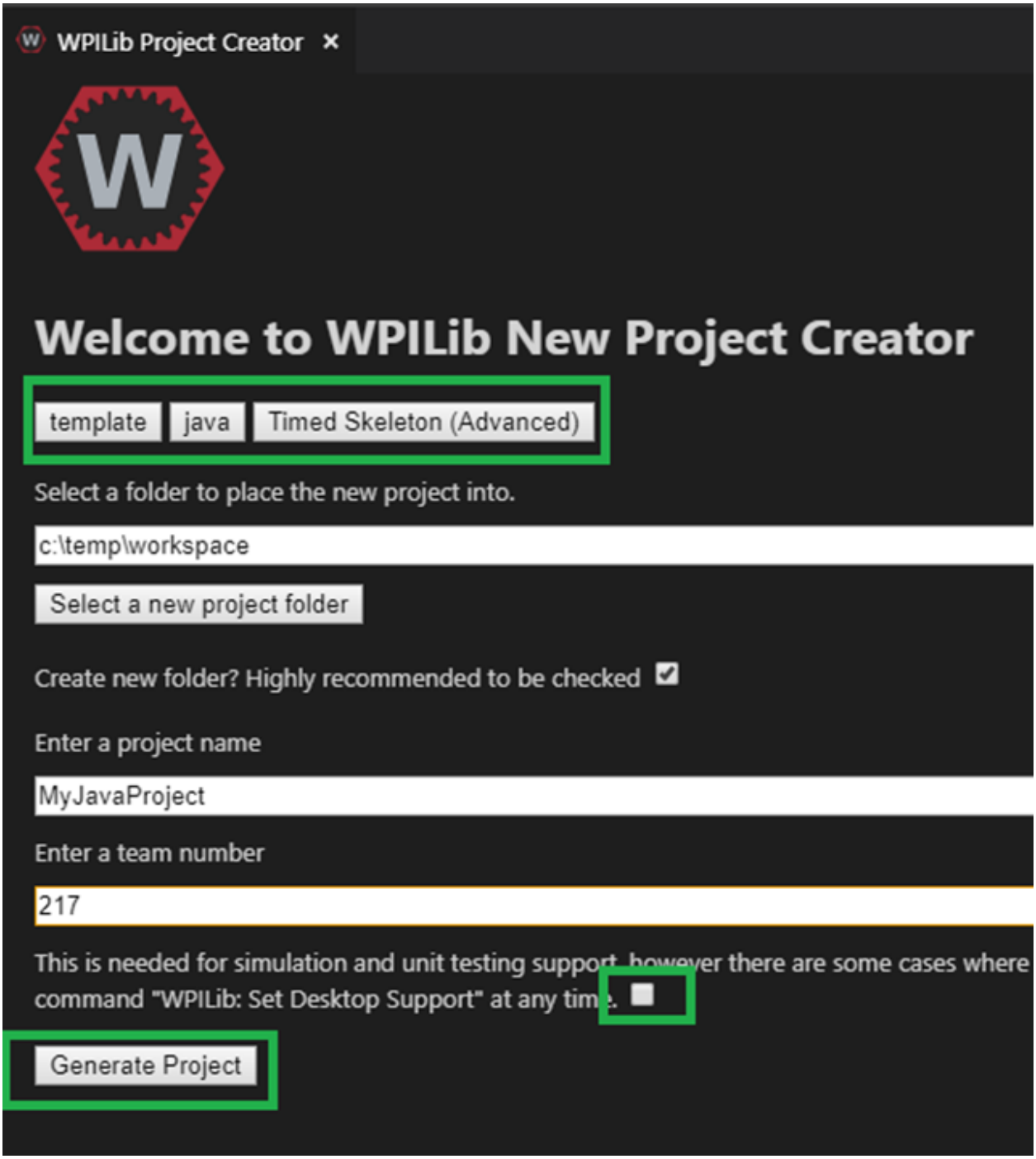
2.8.1 FRC C++/Java – Create a Project

Next we will create a new robot project in vscode and create a Talon SRX. The goal is compile the project only, so hardware is not needed.

Follow the WPI Screensteps instructions on reaching the create new project. Typically, you can use CNTRL+SHIFT+P to open the VS text bar, and type create to reach the WPI command.

Make sure the desktop checkbox is cleared, Phoenix does not currently support desktop simulation. "Timed Skeleton" is used in this example for sake of simplicity.


WPILib Project Creator
Welcome to WPILib New Project Creator
template
cpp
Timed Skeleton (Advanced)
Select a folder to place the new project into.
c:\temp\workspace
Select a new project folder
Create new folder? Highly recommended to be checked
Enter a project name
MyCppProject
Enter a team number
217
Generate Project



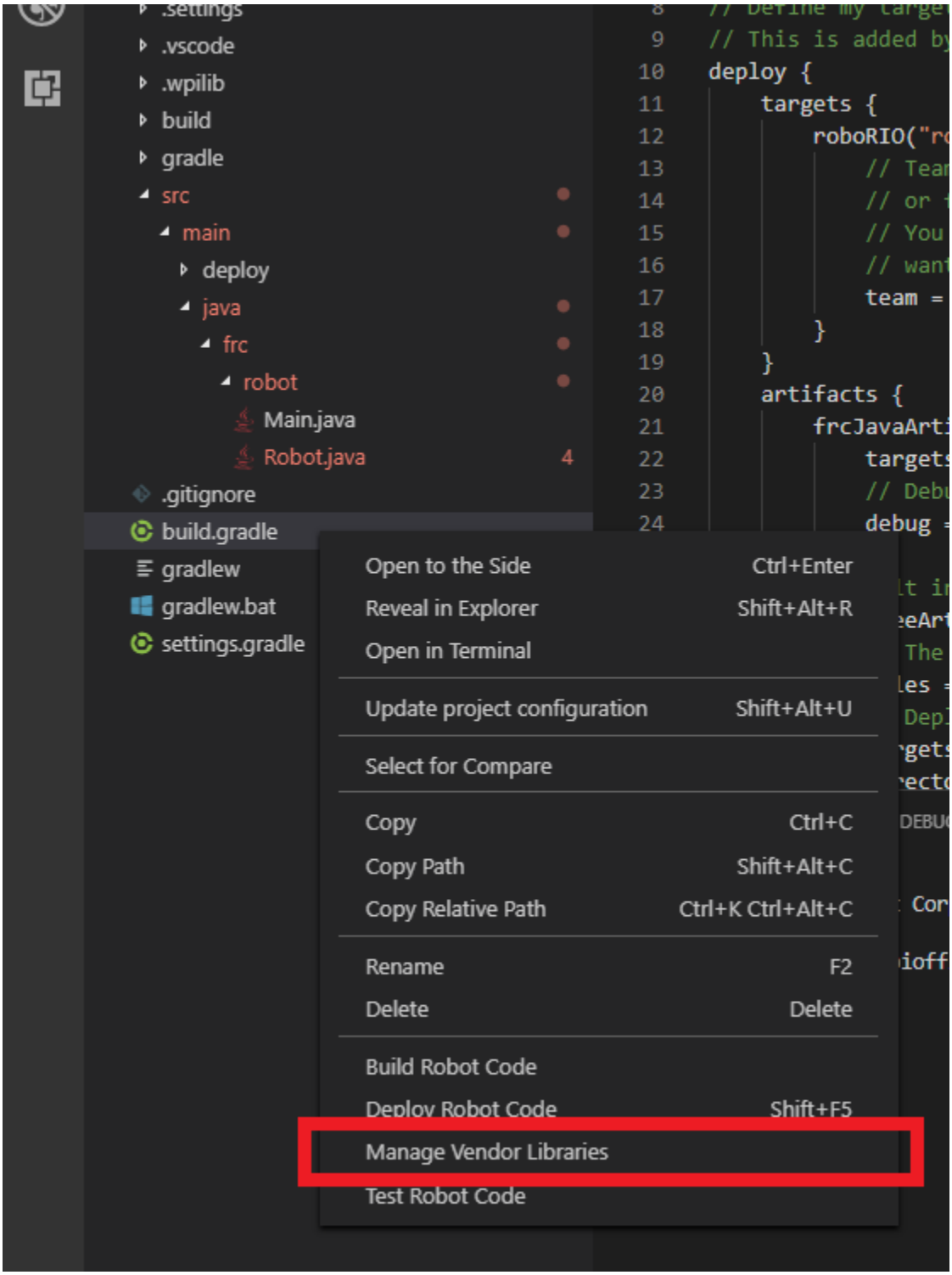
Once the project is created, ensure project builds. Testing robot deploy is also useful if robot controller is available.

2.8.2 FRC C++/Java – Add Phoenix

Right-Click on "build.gradle" in the project tree, then select "Manage Vendor Libraries".

Note: if "Manage Vendor Libraries" is **missing** then you likely are using 2018 Alpha VS. Ensure you are using **2019**

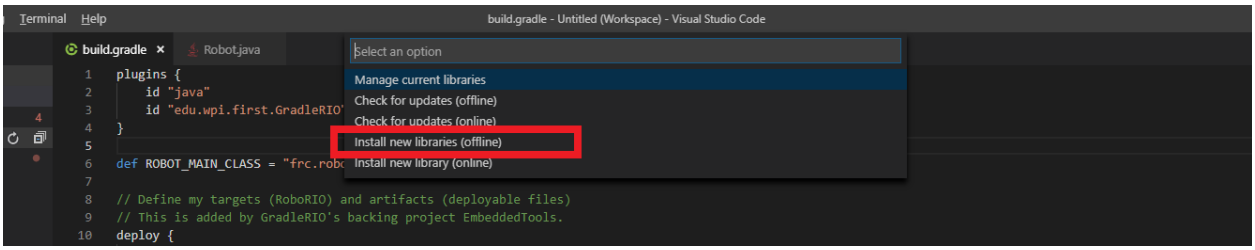
Release VSCode from WPI.



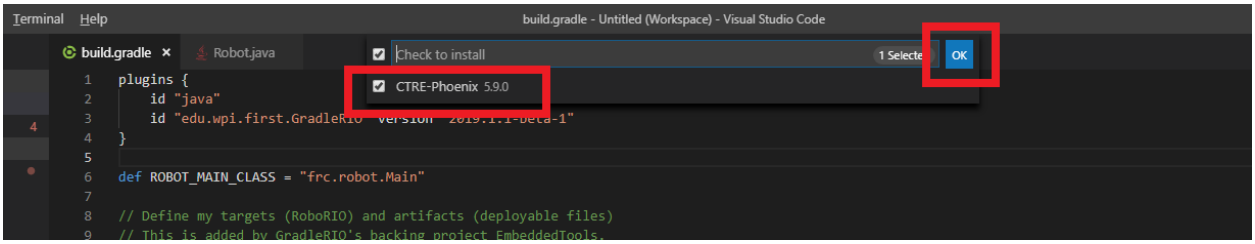
At the top of your screen, a menu will appear. Select "Install new libraries (offline)".

Tip: Alternatively you can use "Install new libraries (online)" option with http://devsite.ctr-electronics.com/maven/

release/com/ctre/phoenix/Phoenix-latest.json. However this is **not recommended** as this requires a live Internet connection to use your FRC project.



The menu will now display a list of vendor libraries you can install. Check "CTRE Phoenix", then click "OK"



Note: This will bring the library into the project references, however the library will not be loaded if the source code does not create a Phoenix object or call any Phoenix routines. Therefore, you must create a Phoenix object to properly test the install.

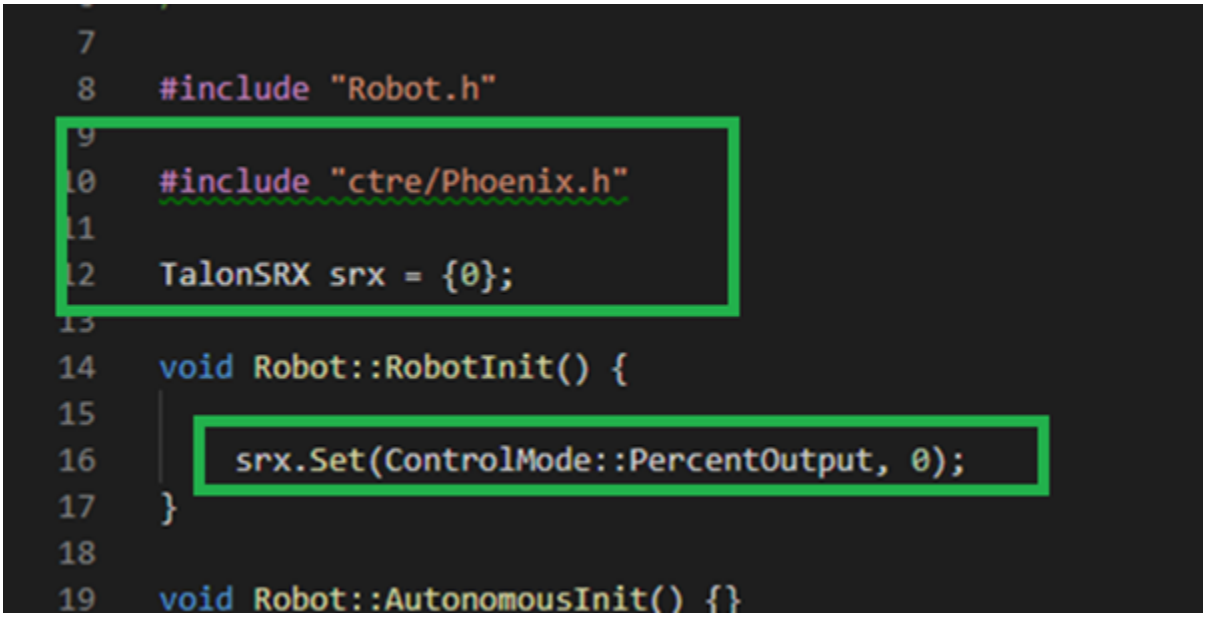
Tip: Teams can verify Phoenix is in their robot project by checking for the existence of vendordeps/Phoenix.json in the project directory.

2.8.3 FRC C++ Build Test: Single Talon

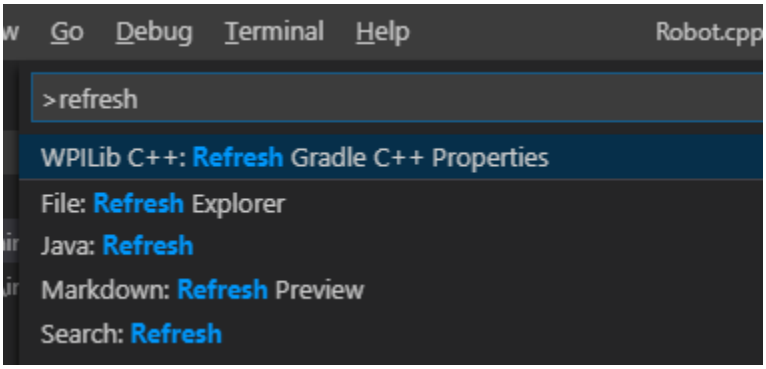
Create a TalonSRX object. The number specified is the Talon's device ID, however for this test, the device ID is irrelevant.

Be sure to include "ctre/Phoenix.h", otherwise TalonSRX will not be recognized as a valid class type.

Add an example call, take your time to ensure to spell it correctly.

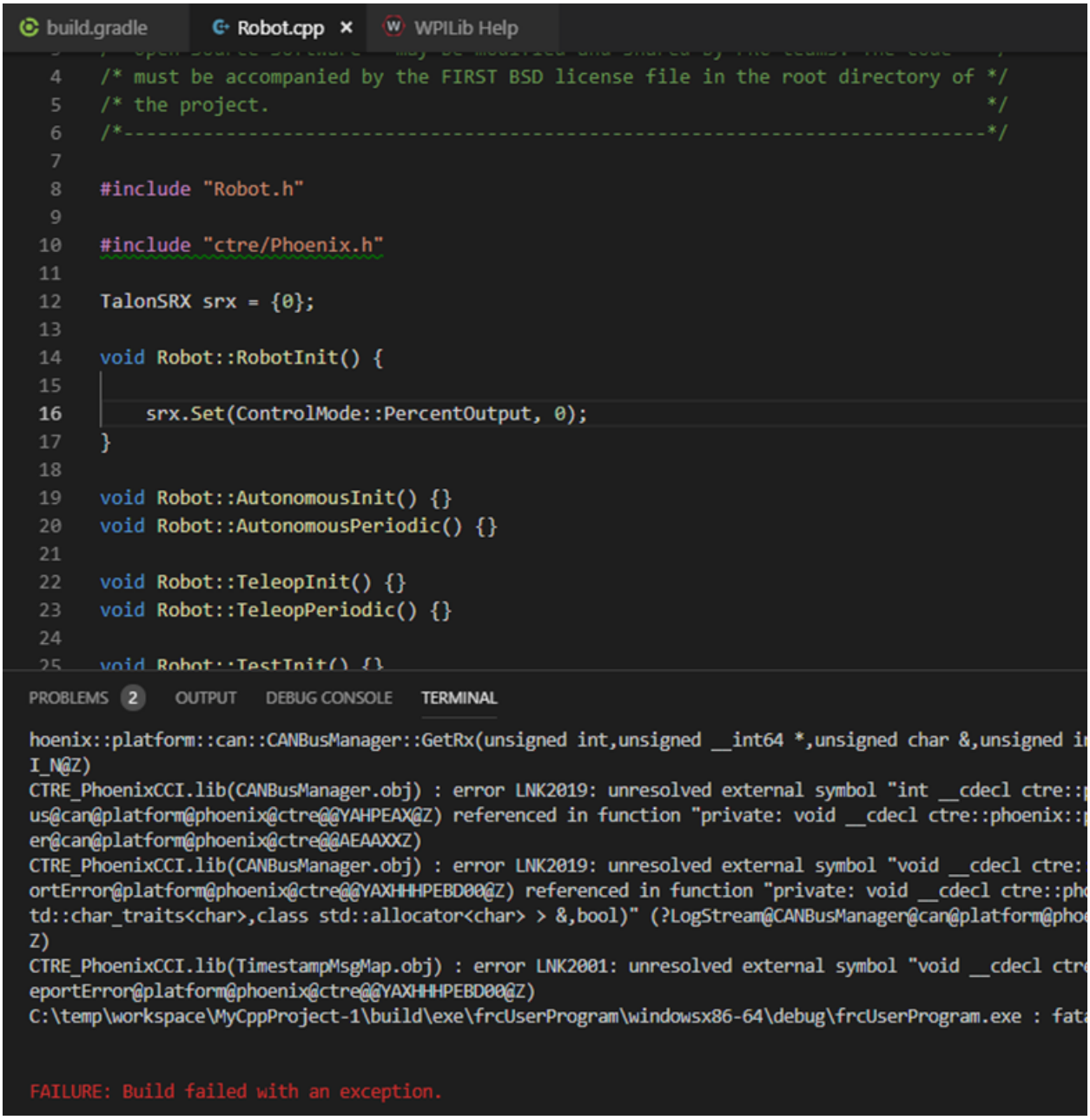


Intellisense may not be functional at this point in time (note the green underline indicating VS did not parse the header).

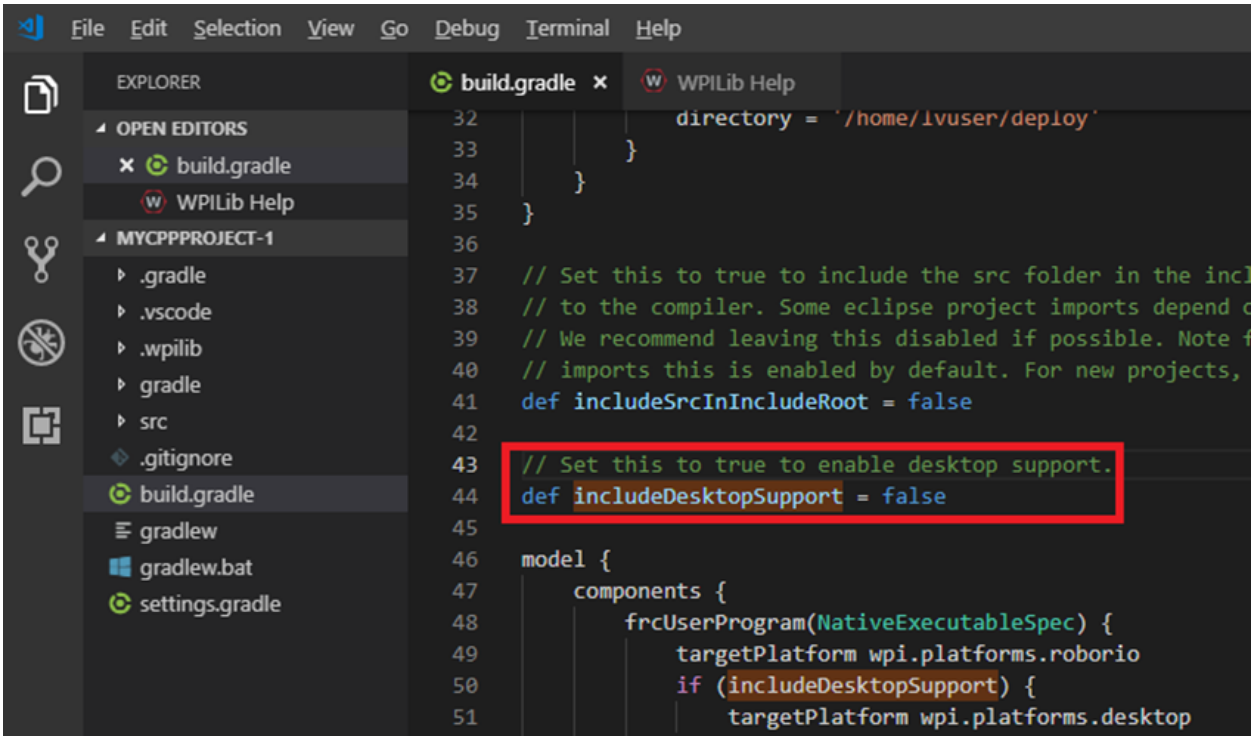


Tip: To correct this - Close all files in the project - Restart VS Code - Wait ~40s - Reopen source files in VS Code

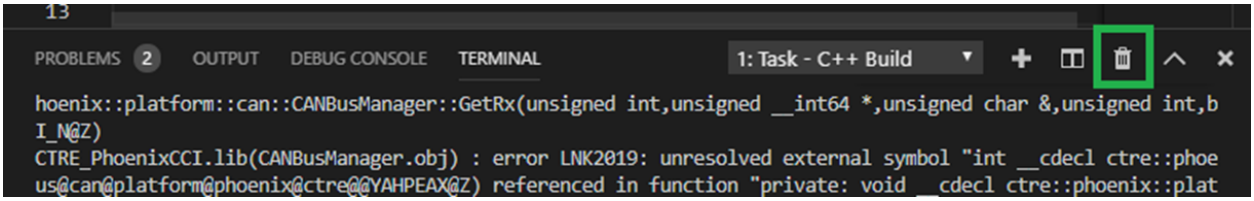
If you see linker errors, then the desktop simulation checkbox was likely checked.



This can be resolved by manually turning off the feature. Set flag to false.

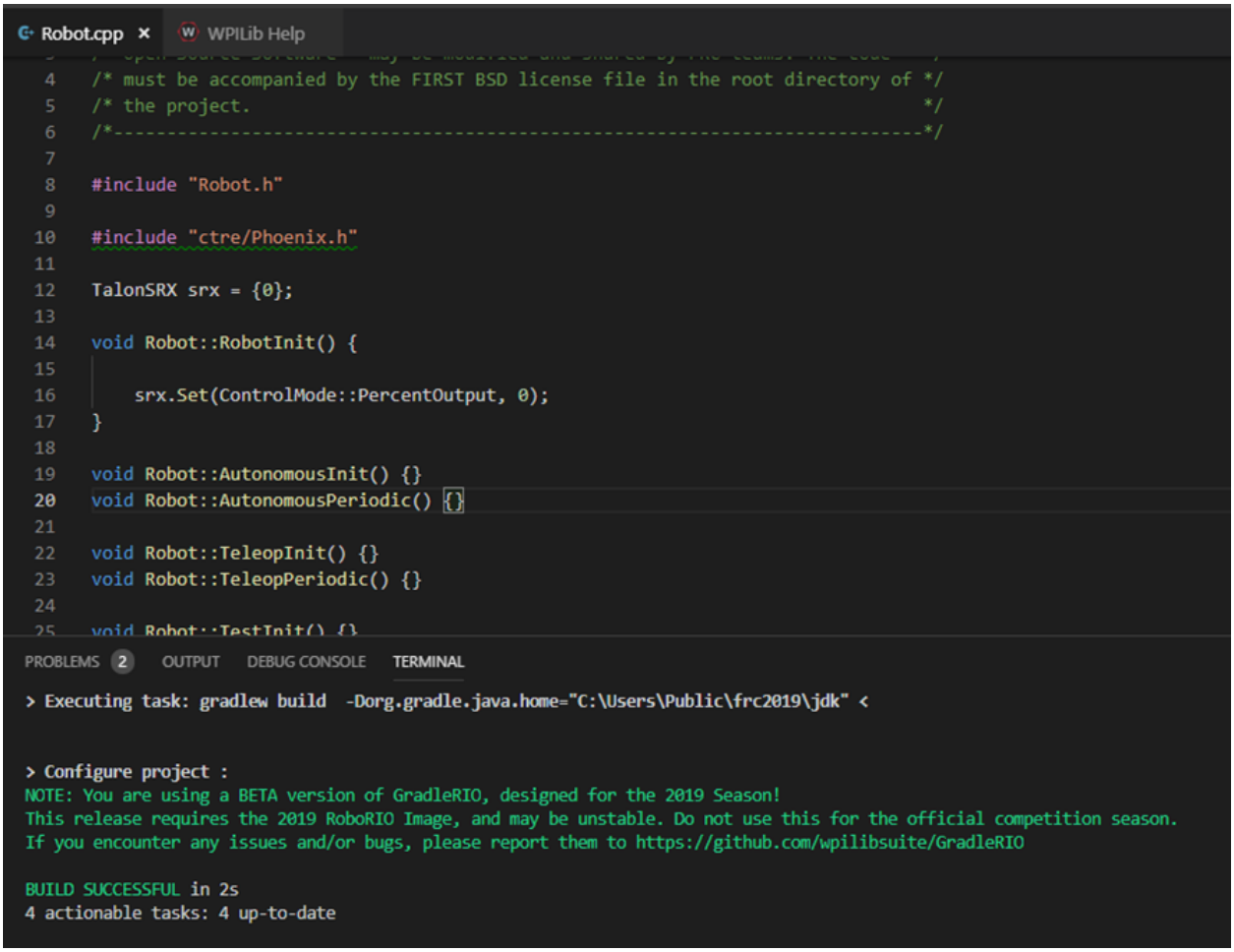


Tip: When resolving compiler/linker errors, press the trash icon first to cleanly erase the previous error lines in the terminal. Or manually scroll the bottom to ensure you are not looking at stale error lines from previously failed builds.

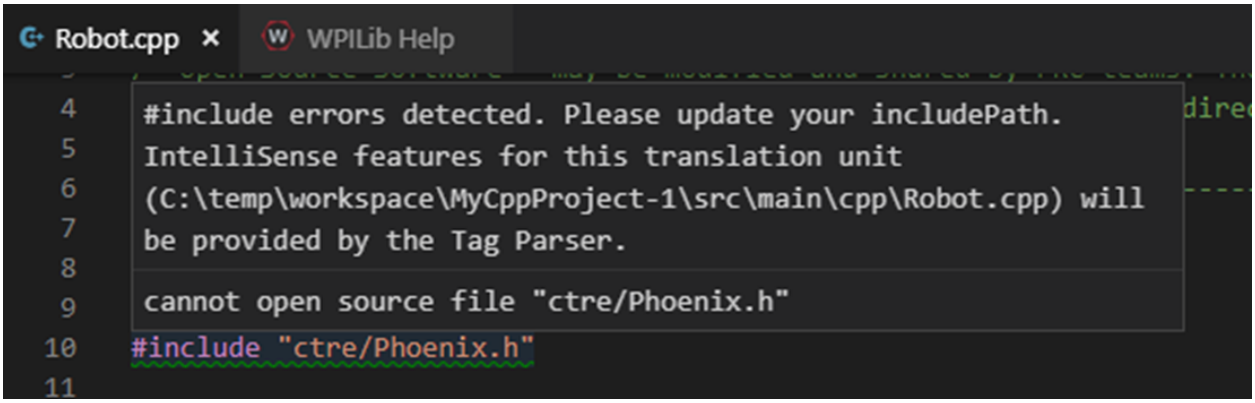


The only reliable way to confirm build was successful is to confirm the BUILD SUCCESSFUL line at the bottom of the TERMINAL.

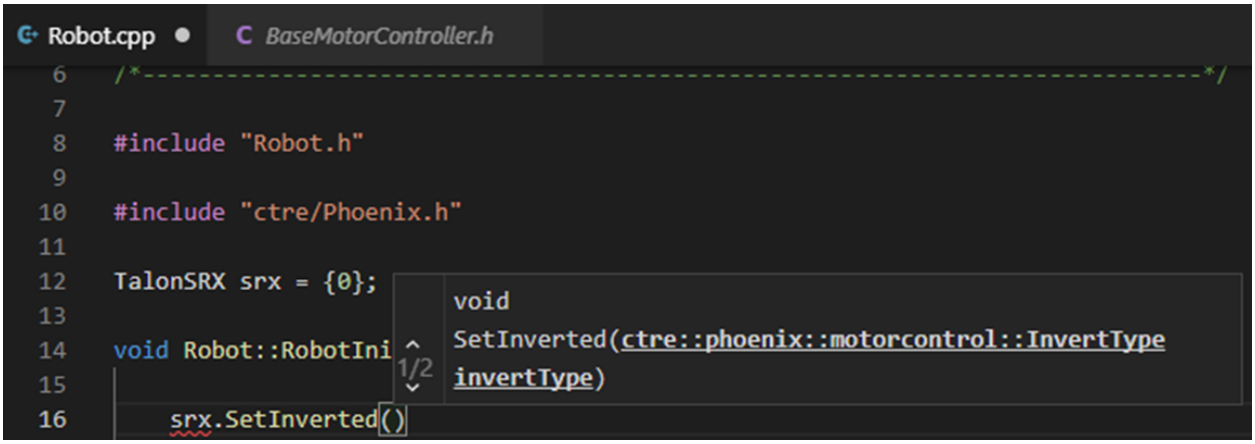
Note: The problems tab may or may not be clear of errors. Our testing with VSCode has shown that it can report stale or incorrect information while making code changes. Always use the TERMINAL output to determine the health of your compilation and build system.



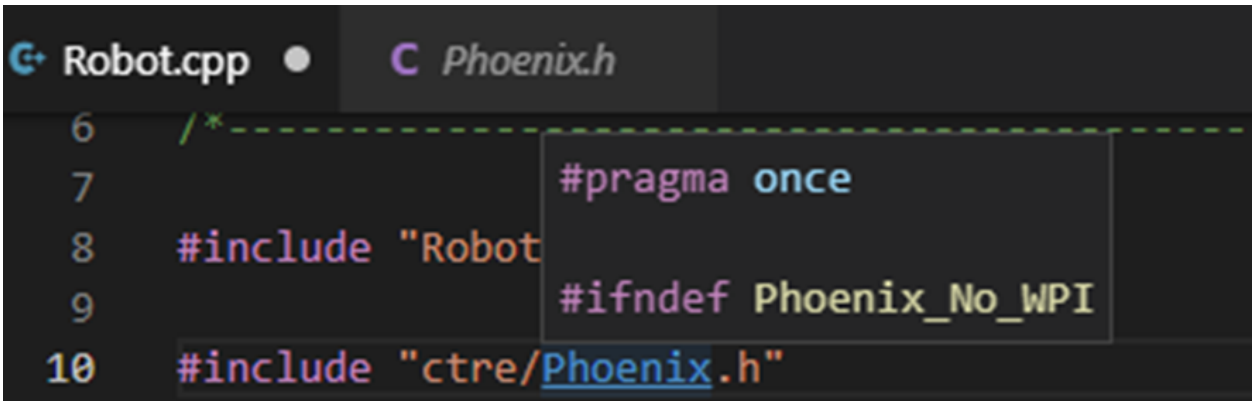
In the event that the intellisense is not resolving symbols (for IDE auto-complete features), restart VSCode.



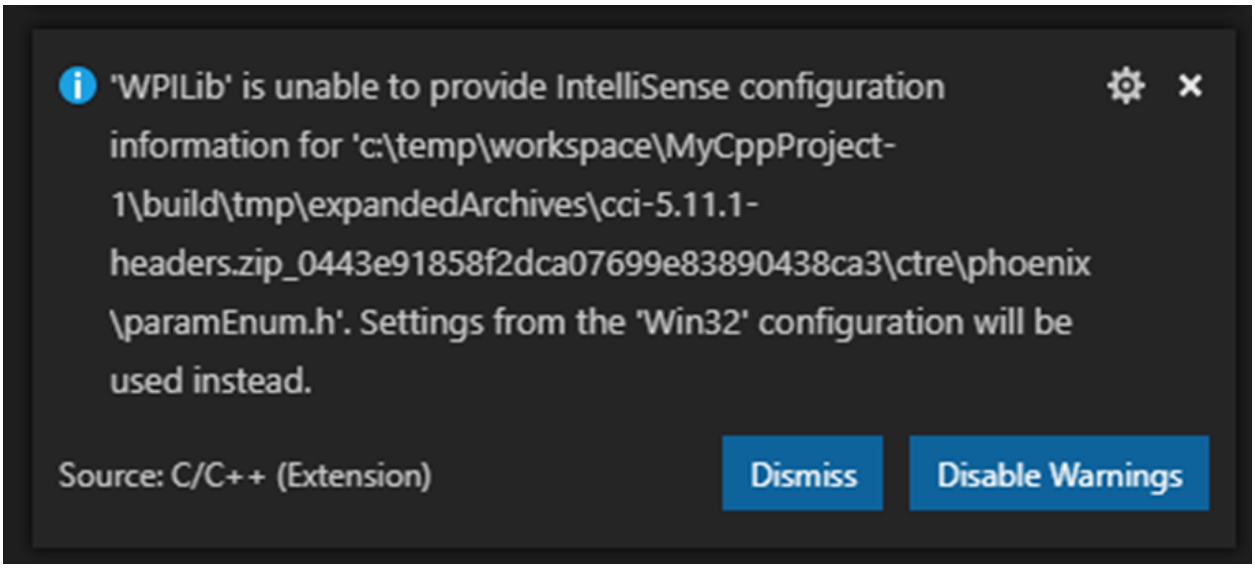
After restart, routines should be found correctly.



Tip: Headers can be auto-opened by CNTRL+CLICK the include line.



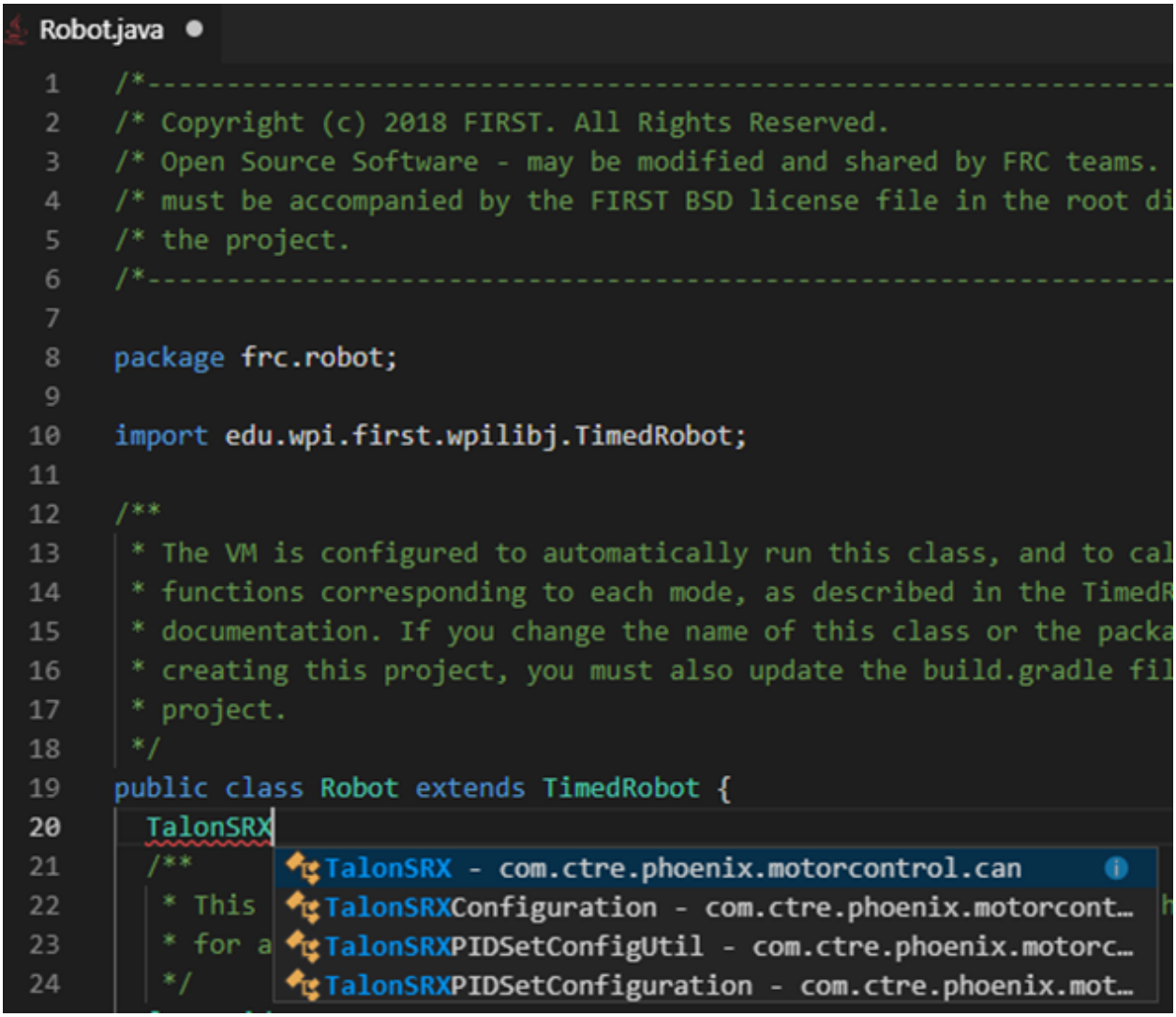
Depending on the version of VS Code used, you may encounter an IntelliSense warning. These can be ignored.



2.8.4 FRC Java Build Test: Single Talon

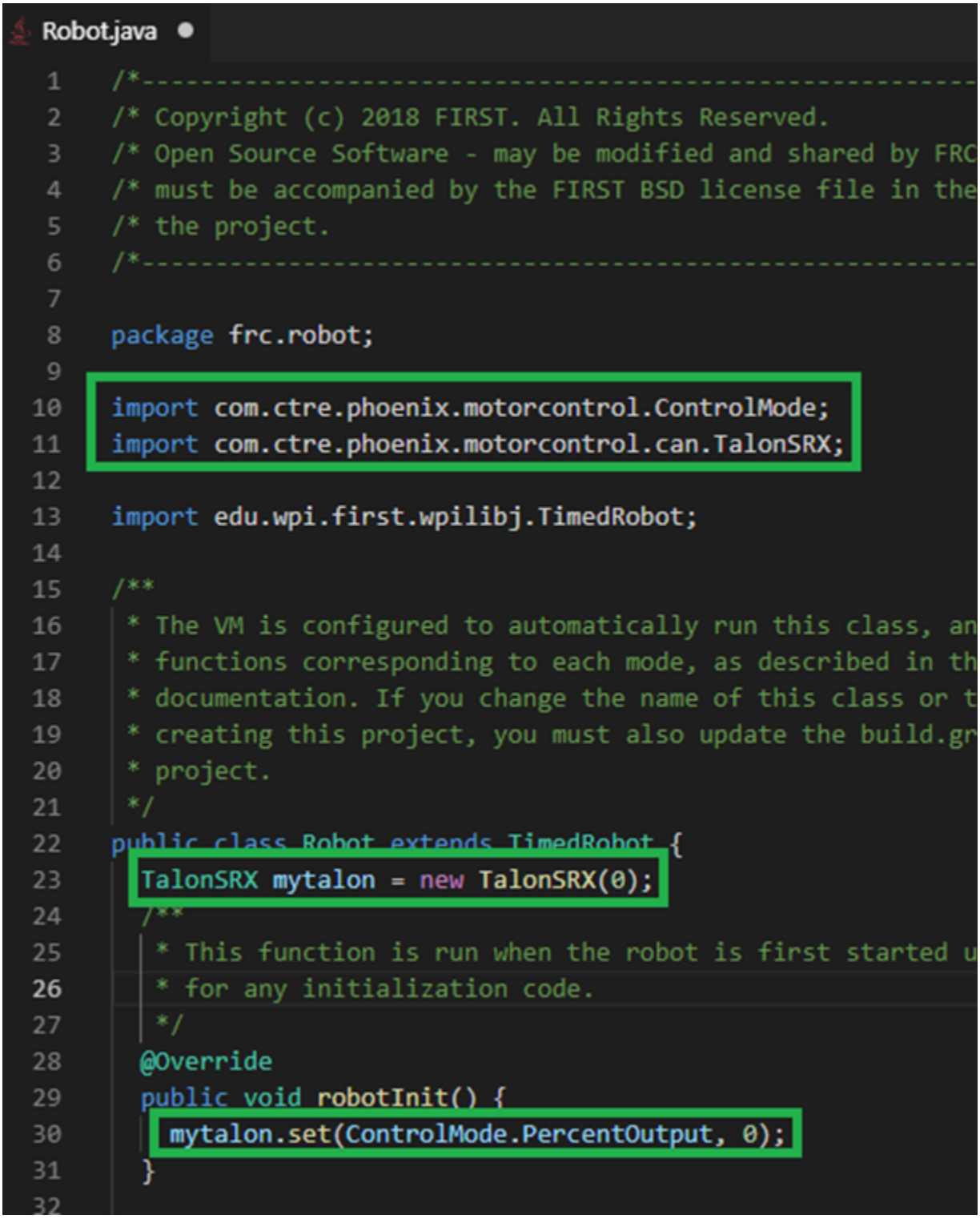
Create a TalonSRX object. The number specified is the Talon's device ID, however for this test, the device ID is irrelevant.

Typically, you can type "TalonSRX" and watch the intellisense auto pop up. If you press ENTER to select the entry, the IDE may auto insert the import line for you.



Add an example call, take your time to ensure to spell it correctly. Use the intellisense features if available.

Here is the final result.



If you see build errors, carefully find the first erroneous line in the TERMINAL output. Typically, you can CNTRL + click the error line and auto-navigate to the source.

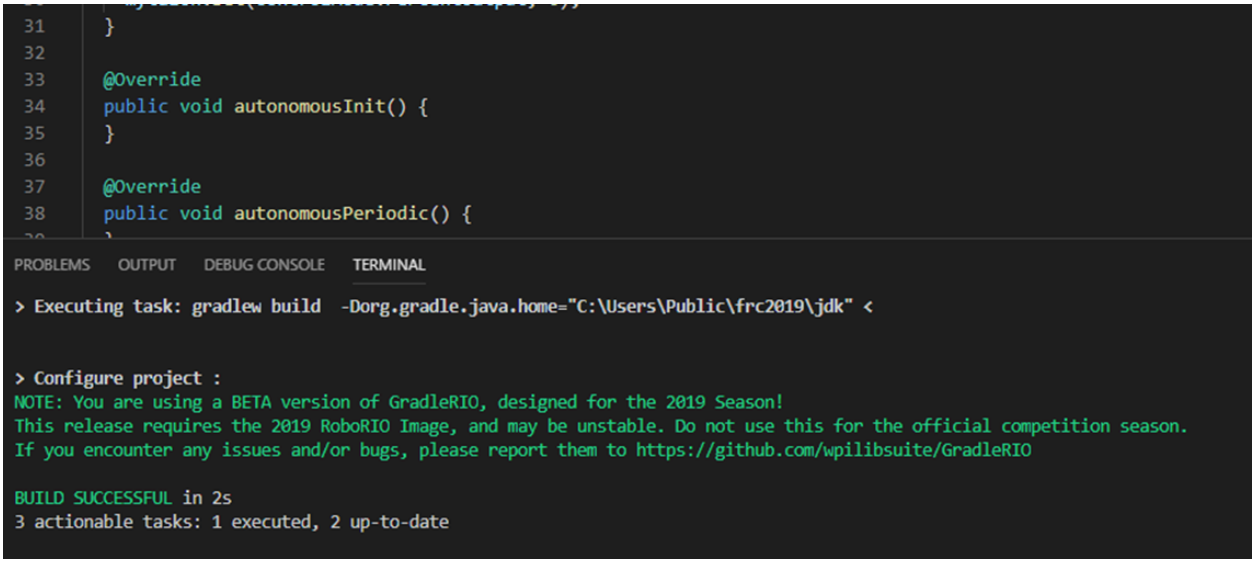
```
22   public class Robot extends TimedRobot {
23     TalonSRX mytalon = new TalonSSRX(0);
24     /**
25      * This function is run when the robot is first started up and should be used
26      * for any initialization code.
27      */
28     @Override

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

> Configure project :
NOTE: You are using a BETA version of GradleRIO, designed for the 2019 Season!
This release requires the 2019 RoboRIO Image, and may be unstable. Do not use this for the official comp
If you encounter any issues and/or bugs, please report them to https://github.com/wpilibsuite/GradleRIO

> Task :compileJava FAILED                                   Ctrl + click to follow link
C:\temp\workspace\MyJavaProject-1\src\main\java\frc\robot\Robot.java:23: error: cannot find symbol
  TalonSRX mytalon = new TalonSSRX(0);
                         ^
  symbol:   class TalonSSRX
  location: class Robot
1 error

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':compileJava'.
> Compilation failed; see the compiler error output for details.
```

When resolving compiler errors, press the trash icon first to cleanly erase the previous error lines in the **terminal**. Or manually scroll the bottom to ensure you are not looking at stale error lines from previously failed builds.

```
13

PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   1: Task - C++ Build

hoenix::platform::can::CANBusManager::GetRx(unsigned int,unsigned __int64 *,unsigned char &,unsigned int,b
I_N@Z)
CTRE_PhoenixCCI.lib(CANBusManager.obj) : error LNK2019: unresolved external symbol "int __cdecl ctre::phoe
us@can@platform@phoenix@ctre@@YAHPEAX@Z) referenced in function "private: void __cdecl ctre::phoenix::plat
```

The only reliable way to confirm build was successful is to confirm the BUILD SUCCESSFUL line at the bottom of the TERMINAL.

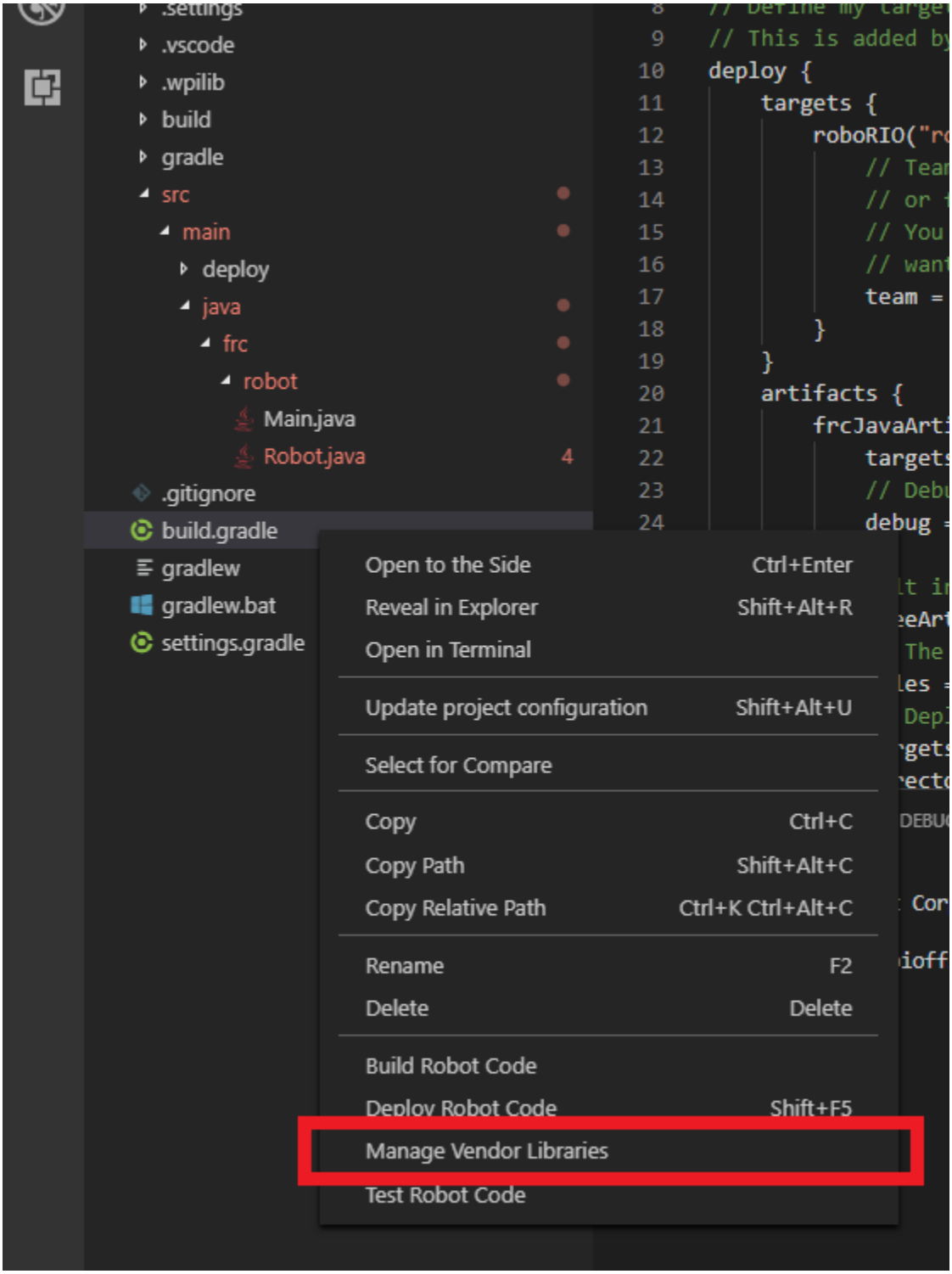
Note: The problems tab may or may not be clear of errors. Our testing with VSCode has shown that it can report stale or incorrect information while making code changes. Always use the TERMINAL output to determine the health of your compilation and build system.



2.8.5 FRC C++/Java - Updating Phoenix

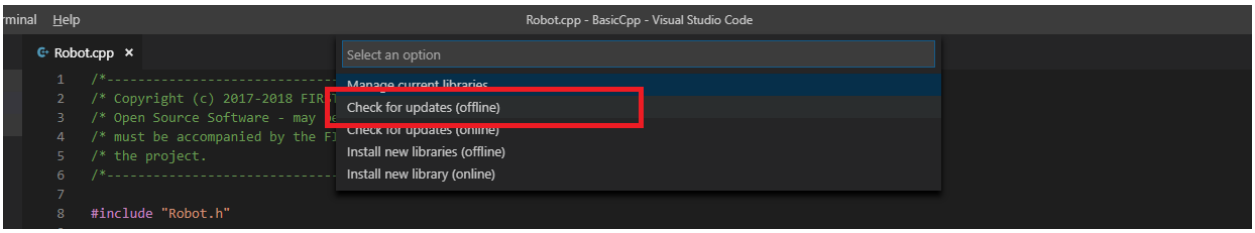
If you already have a 2019 version of Phoenix installed and you want to update to a newer version, follow these steps. Install the latest version of Phoenix on your PC. Basically, rerun the latest installer (same as section above).

Open your robot program in VS Code.

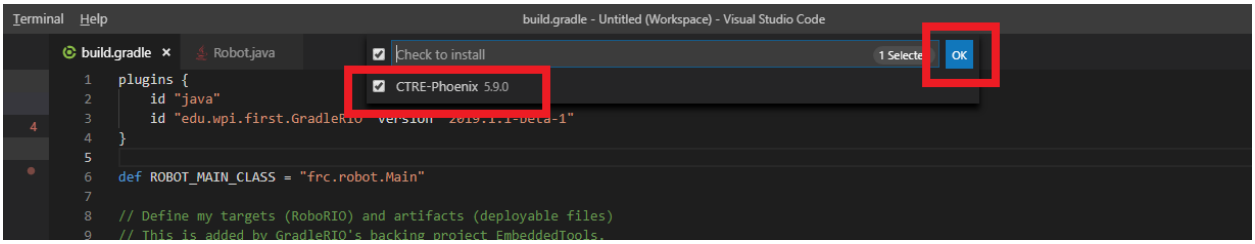


At the top of your screen, a menu will appear. Select “Check for updates (offline)”.

Tip: Alternatively you can use “Check for updates (online)”. However this is **not recommended** as this requires a live Internet connection to use your FRC project.



The menu will now display a list of vendor libraries you can update. Check "CTRE Phoenix", then click "OK"



2.8.6 FRC C++/Java – Test Deploy

Create a Talon SRX (or Pigeon, CANifier, Victor SPX) and attempt to "deploy". Adding a print statement also helps to confirm you are actually deploying the software displayed in VsCode. Confirm that the software deployed using DriverStation. DS may report firmware-too-old / not-retrieved errors since the hardware has not been setup yet.

2.9 Initial Hardware Testing

For your competition team to have the best chance of success, hardware components should be tested as soon as they are received. This is generally done by:

- Powering up the device and confirming LED states.
- Ensuring hardware shows up in Tuner if wired to CAN Bus.
- Drive outputs / drive motor in both directions (if motor controller).

This is explained in the sections below, but it is worth pointing out how important this step is. It is in your team's best interest to test ALL purchased robot components immediately and in isolation. Here are the reasons why:

- Robot *replacement* components should be in a **state of readiness**. Otherwise a replacement during competition can yield erroneous behavior.
- Many robot components (in general) have **fixed warranty periods**, and replacements must be done within them.
- Confirming devices are functional **before handing them to students** ensures best chance of success. If a student later damages hardware, they need to understand how they did it to ensure it does not happen again. Without initial validation, you can't determine root-cause.

Much of this is done during the "bring-up" phase of the robot. However, there is much validation a team can do long before the robot takes form.

Unfortunately, there are **many** teams that do not perform this step, and end up isolating devices and re-implementing their cable solutions at competition, because this was not done during robot bring up.

Note: "Bring up / Board bring up / Hardware bring up" is an engineering colloquial phrase. It is the initial setup and validation phase of your bench or robot setup.

2.10 Bring Up: CAN Bus

Now that all of the software is installed and verified, the next major step is to setup hardware and firmware.

2.10.1 Understand the goal

At this point we want to have reliable communication with CAN devices. There are typically two failure modes that must be resolved:

- There are same-model devices on the bus with the same device ID (devices have a default device ID of '0').
- CAN bus is not wired correctly / robustly.

This is why during hardware validation, you will likely have to isolate each device to assign a unique device ID.

Note: CTRE software has the ability to resolve device ID conflicts without device isolation, and CAN bus is capable of reporting the health of the CAN bus (see Driver Station lightening tab). However, the problem is when **both** root-causes are occurring at the same time, this can confuse students who have no experience with CAN bus systems.

Note: Many teams will preassign and update devices (Talon SRXs for example) long before the robot takes form. This is also a great task for new students who need to start learning the control system (with the appropriate mentor oversight to ensure hardware does not get damaged).

Note: Label the devices appropriately so there is no guessing which device ID is what. Don't have a label maker? Use tape and/or Sharpie (sharpie marks can be removed with alcohol).

2.10.2 Check your wiring

Specific wiring instructions can be found in the user manual of each product, but there are common steps that must be followed for all devices:

- If connectors are used for CANBus, **tug-test each individual crimped wire** one at a time. Bad crimps/connection points are the most common cause of intermittent connection issues.
- Confirm red and black are not flipped. **Motor Controllers typically are not reverse power protected**.
- Confirm battery voltage is adequate (through Driver Station or through voltmeter).
- Manually inspect and confirm that green-connects-to-green and yellow-connects-to-yellow at every connection point. **Flipping/mixing green and yellow is a common failure point during hardware bring up**.
- Confirm breakers are installed in the PDP where appropriate.
- Measure resistance between CANH and CANL when system is not powered (should measure ~60Ω). If the measurement is 120Ω, then confirm both RIO and PDP are in circuit, and PDP jumper is in the correct location.

2.10.3 Power up and check LEDs

If you haven't already, power up the platform (robot, bench setup, etc) and confirm LEDs are illuminated (at all) on all devices.

You may find many of them are blinking or "blipping" red (no communication).

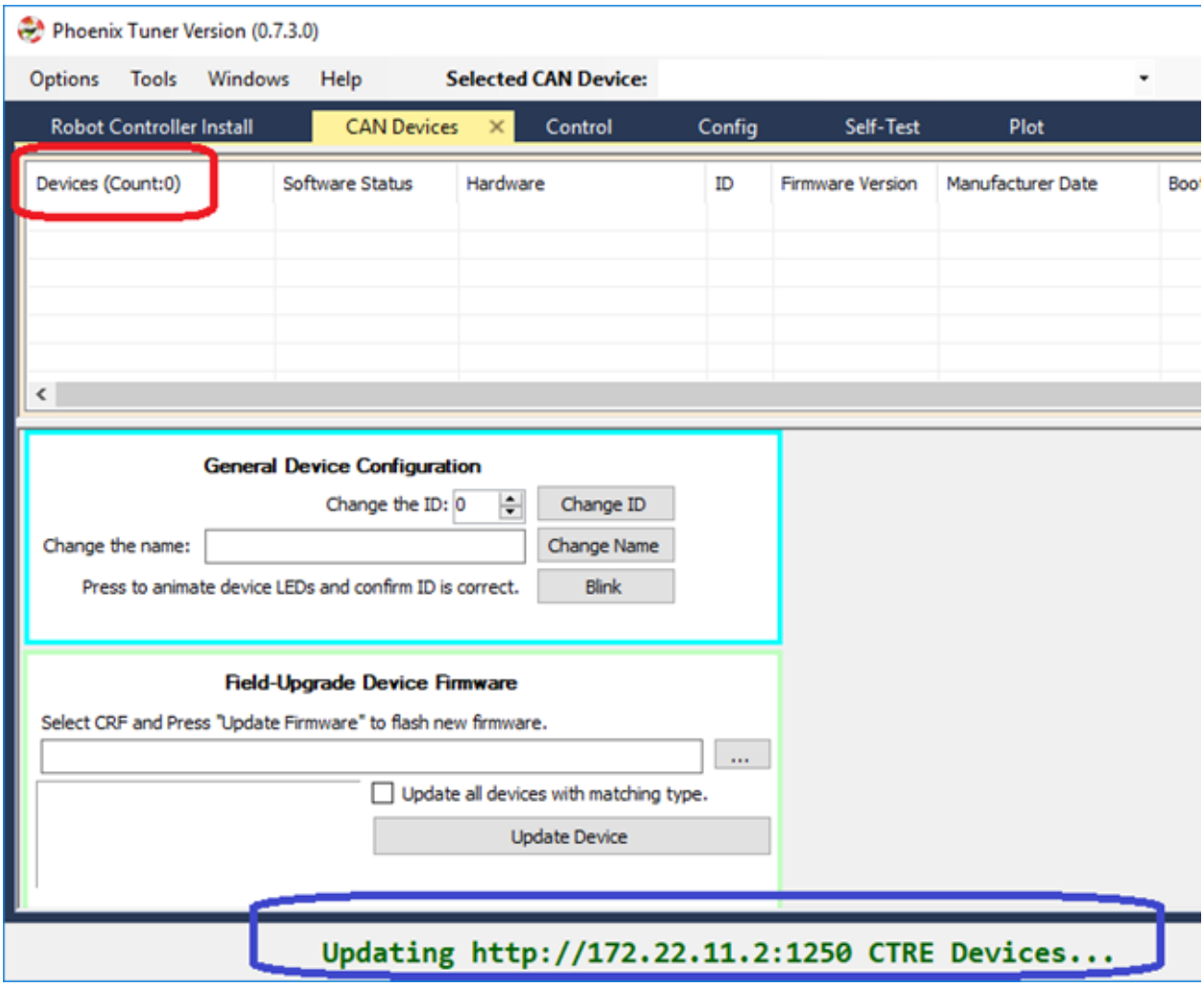
Tip: If you are color-blind or unable to determine color-state, grab another team member to assist you.

Note: If using Ribbon cabled Pigeon IMUs, Pigeon LEDs will reflect the ribbon cable, not the CAN bus. At which point any comm issue with Pigeon will be resolved under section Bring Up: Pigeon IMU.

2.10.4 Open Phoenix Tuner

Navigate to the CAN devices page.

This capture is taken with no devices connected to the roboRIO. roboRIO will take around 30 seconds to boot.



2.10.5 LEDs are red – now what?

We need to rule out same-id versus bad-bus-wiring. There are two approaches. Approach 1 will help troubleshoot bad wiring and common IDs. Approach 2 will only be effective in troubleshooting common IDs. But this method is

noteworthy because it is simple/quick (no wiring changes, just pull breakers).

The specific instructions for changing device ID are in the next section. Review this if needed.

Approach 1 (best)

Procedure:

- **Physically connect CAN bus from roboRIO to one device only. Circumvent your wiring if need be.**
- Power boot robot/bench setup.
- Open Phoenix Tuner and wait for connection (roboRIO may take ~30 seconds to boot)
- Open CAN devices tab
- Confirm if CAN device appears.
- Use Tuner to change the device ID
- Label the new ID on the physical device
- Repeat this procedure for every device, one at a time.

If you find a particular device where communication is not possible, scrutinize device's power and CAN connection to roboRIO. Make the test setup so simple that the only failure mode possible is within the device itself.

Note: Typically, there must be two termination resistors at each end of the bus. One is in the RIO and one is in the PDP. But during bring-up, if you keep your harness short (such as the CAN pigtail leads from a single Talon) then the internal resistor in the RIO is adequate.

Approach 2 (easier)

Procedure:

- **Leave CAN bus wiring as is.**
- **Pull breakers and PCM fuse from PDP.**
- **Disconnect CAN bus pigtail from PDP.**
- **Pick the first device to power up and restore breaker/fuse/pigtail so that only this CAN device is powered.**
- Power boot robot/bench setup.
- Open Phoenix Tuner and wait for connection (roboRIO may take ~30 seconds to boot)
- Open CAN devices tab
- Confirm if CAN device appears. If device does not appear, scrutinize device's power and CAN connection to roboRIO.
- Use Tuner to change the device ID
- Label the new ID on the physical device
- Repeat this procedure for every device.

If you find a particular device or section of devices where communication is not possible, then the CAN bus wiring needs to be re-inspected. Remember to "flick" / "shake" / "jostle" the CAN wiring in various sections to attempt to reproduce red LED blips. This is a sure sign of loose contact points.

If you are able to detect and change device ID on your devices individually, begin piecing your CAN bus together. Start with roboRIO <—-> device <—> PDP, to ensure termination exists at both ends. Then introduce the remaining devices until a failure is observed or until all devices are in-circuit.

If introducing a new device creates a failure symptom, scrutinize that device by replacing it, inspecting common wires, and inspecting power.

Note: If 2014 PDP is the only device that does not appear or has red LEDs, see PDP boot up section for specific failure mode.

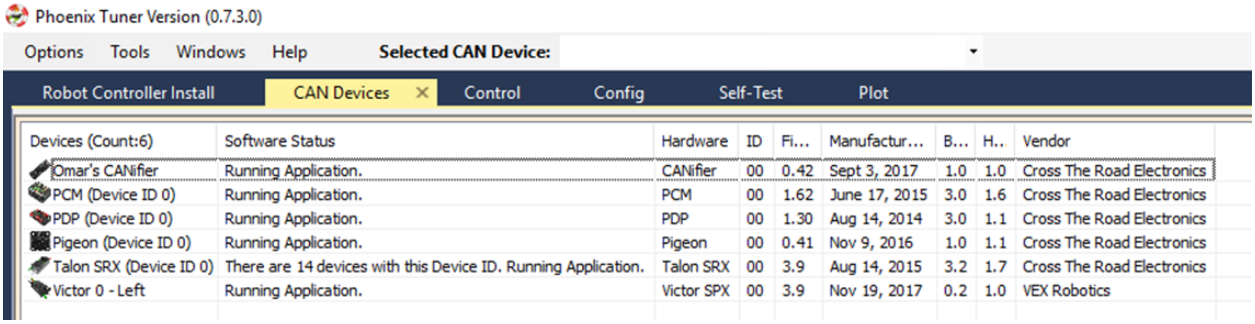
Note: If ribbon cable Pigeon does not appear, it likely is because Talon has old firmware.

At the end of this section, all devices should appear (notwithstanding the above notes) and device LEDs should not be red. PCM, Talon, Victor, Pigeon, and CANifier typically blink orange when they are healthy and not controlled. PDP may be orange or green depending on its sticky faults.

2.10.6 Set Device IDs

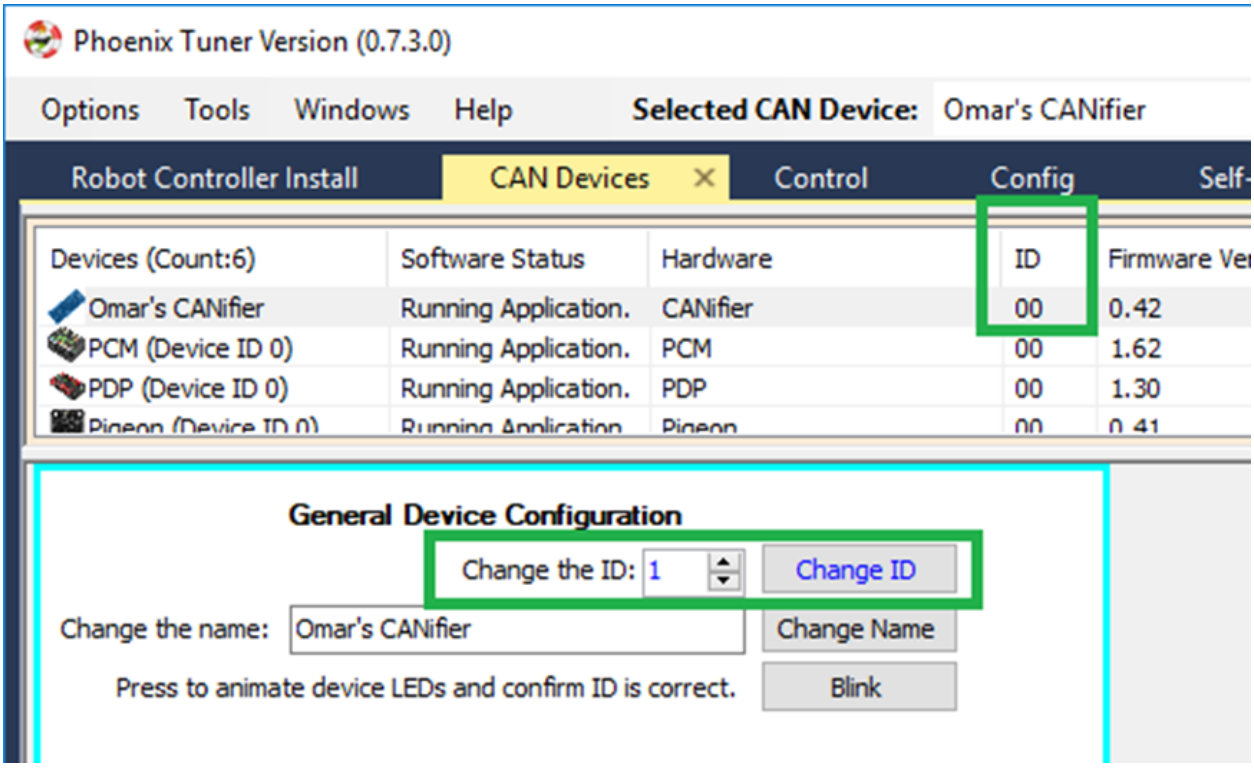
Note: A CTRE device can have an ID from 0 to 62. If you select an invalid ID, you will generally get an immediate prompt.

Below we see several devices, however the physical robot has 19 actual devices. Because all the Talons have a device ID of '0', the do not show up as unique hardware. This must be resolved before you can attempt utilizing them.

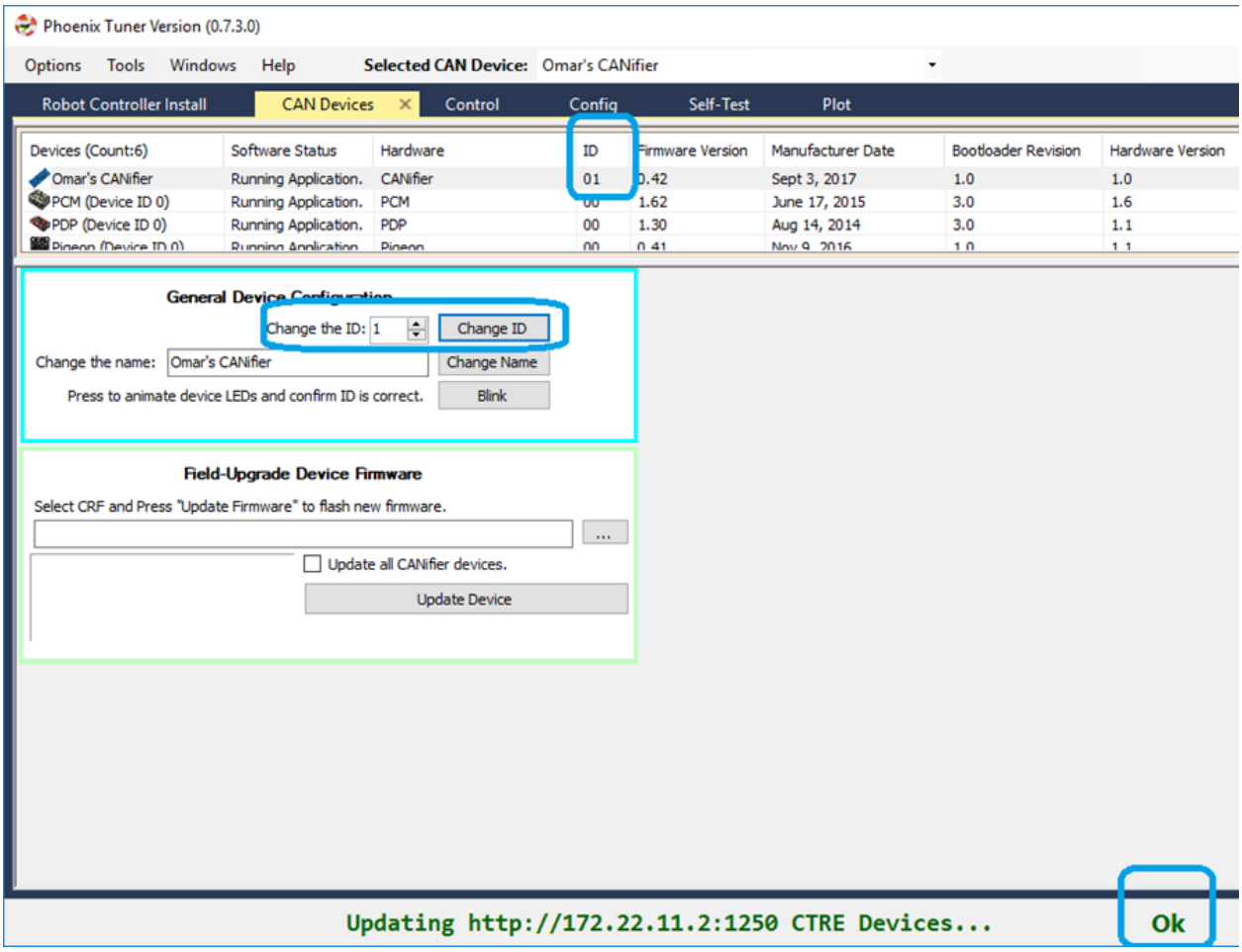


Note: We recommend isolating each device and assigning a unique ID first. But in the event there is a conflict, expect an entry mentioning multiple devices. When selecting a device, the actually physical device selected will be the conflict-id device that booted last. You can use this information to control which Talon you are resolving by power cycling the conflict device, then changing its ID in Tuner.

Select the device and use the numeric entry to change the ID. Note the text will change blue when you do this. Then press "Change ID" to apply the changes.



If operation completes, and OK will appear in the bottom status bar (this is true of all operations). Also note the ID has updated in the device list, and the ID text is now black again.

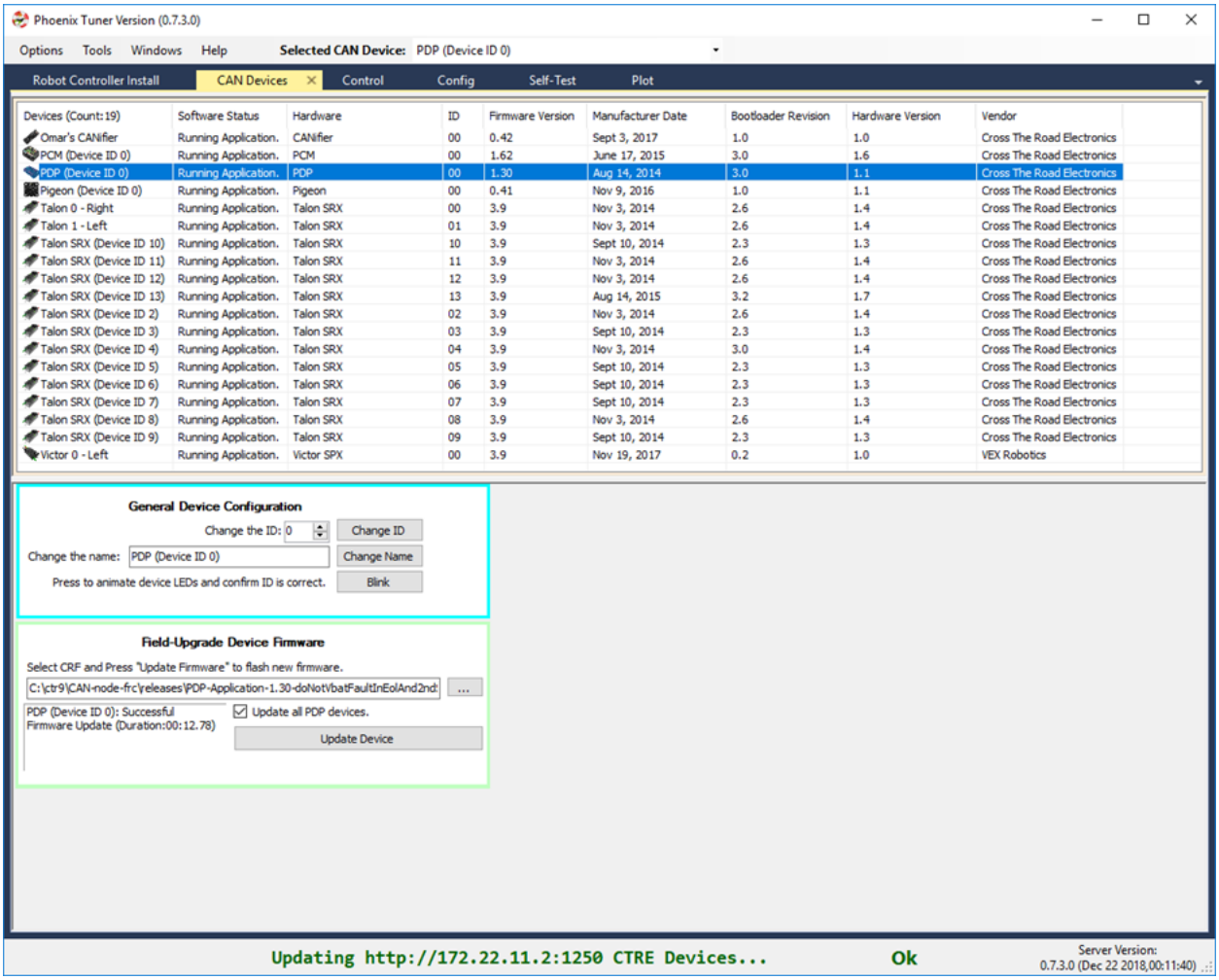


Tip: All production CTRE hardware ships with a default ID of '0'. As a result, it is useful to start your devices at device ID '1', so you can cleanly add another out-of-box device without introducing a conflict.

When complete, make sure every device is visible in the view. Use the Blink button on each device to confirm the ID matches the expected physical device.

Note: The device count is present in the top left corner of the device list. Use this to quickly confirm all devices are present.

Note: If ribbon-cabled pigeon is not present, then the host talon likely has old firmware.



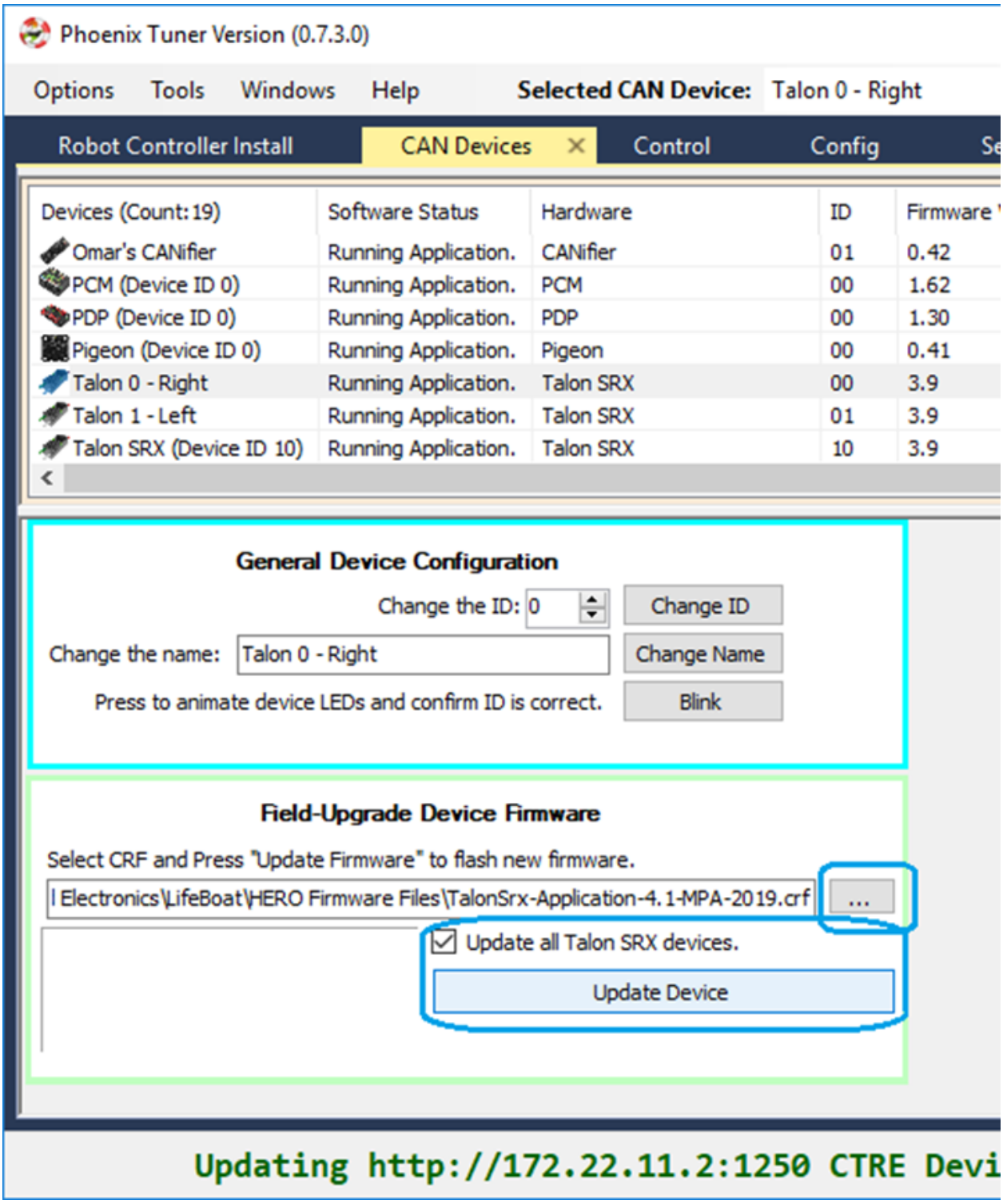
2.10.7 Field upgrade devices

At this point all devices are present, but the firmware is likely old.

The 2019 seasons has 4.X firmware for Talon SRX, Victor SPX, CANifier, and Pigeon IMU. 4.X firmware is required for 2019 targeted Phoenix API and Tuner releases.

Note: Latest PDP is 1.40. PDP typically ship with 1.30. 1.40 has a math fix for the energy measurement, and will tare the current measures so current will read 0 instead of ~1-2 amps when there is no current-draw .

Note: Latest PCM is 1.65. PCM typically ship with 1.62. Firmware 1.65 has an improvement where hardware-revision 1.6 PCMs will not-interrupt compressor when blacklisting a shorted solenoid channel. Older revisions will pause the compressor in order to safely sticky-fault, new revisions have no need to do this (if firmware is up to date).



Select the CRF under the Field-upgrade section then press Update Device. The CRFs are available in multiple places, and likely are already on your PC/ See section "Device Firmware Files (crf)".

If there are multiple devices of same type (multiple Talon SRXs for example), you may check Update all devices. This will automatically iterate through all the devices of the same type, and update them. If a device field-upgrade fails, then the operation will complete. Confirm Firmware Version column in the device list after field-upgrade.

Note: Each device takes approximately 15 seconds to field-upgrade.

When complete every device should have latest firmware.

2.10.8 Pick device names (optional)

The device name can also be changed for certain device types: - CANifier - Pigeon IMU (on CAN bus only) - Talon SRX and Victor SPX

Note: PDP and PCM do not support this.

Note: Ribbon cabled Pigeon IMUs do not support this.

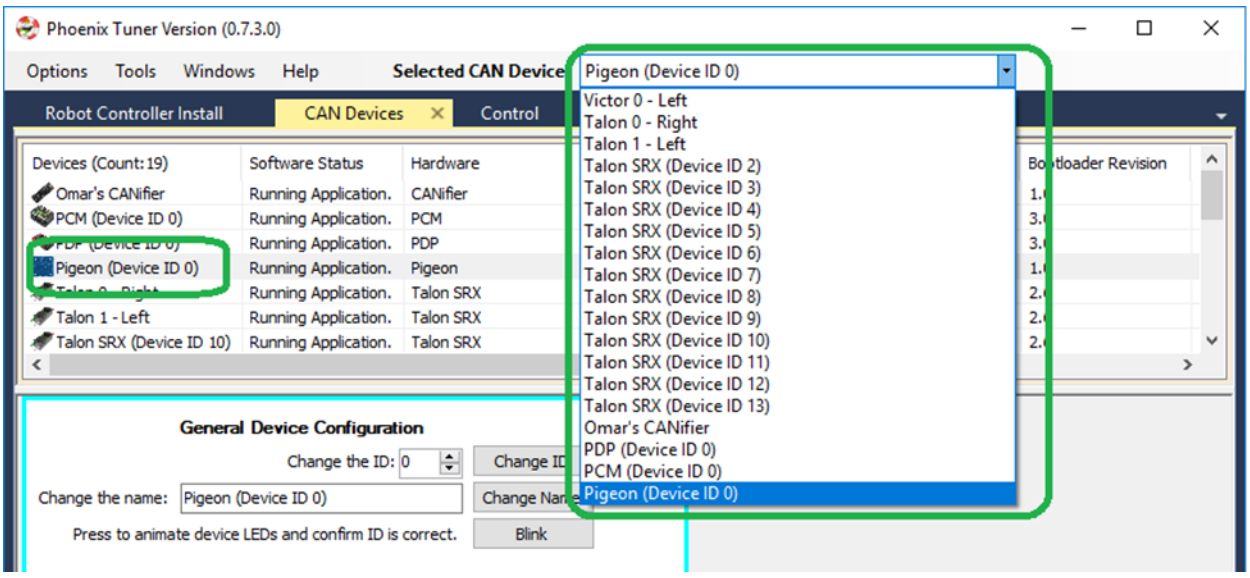
Note: To re-default the custom name, clear the "Name" text entry so it is blank and press "Save".

2.10.9 Self-test

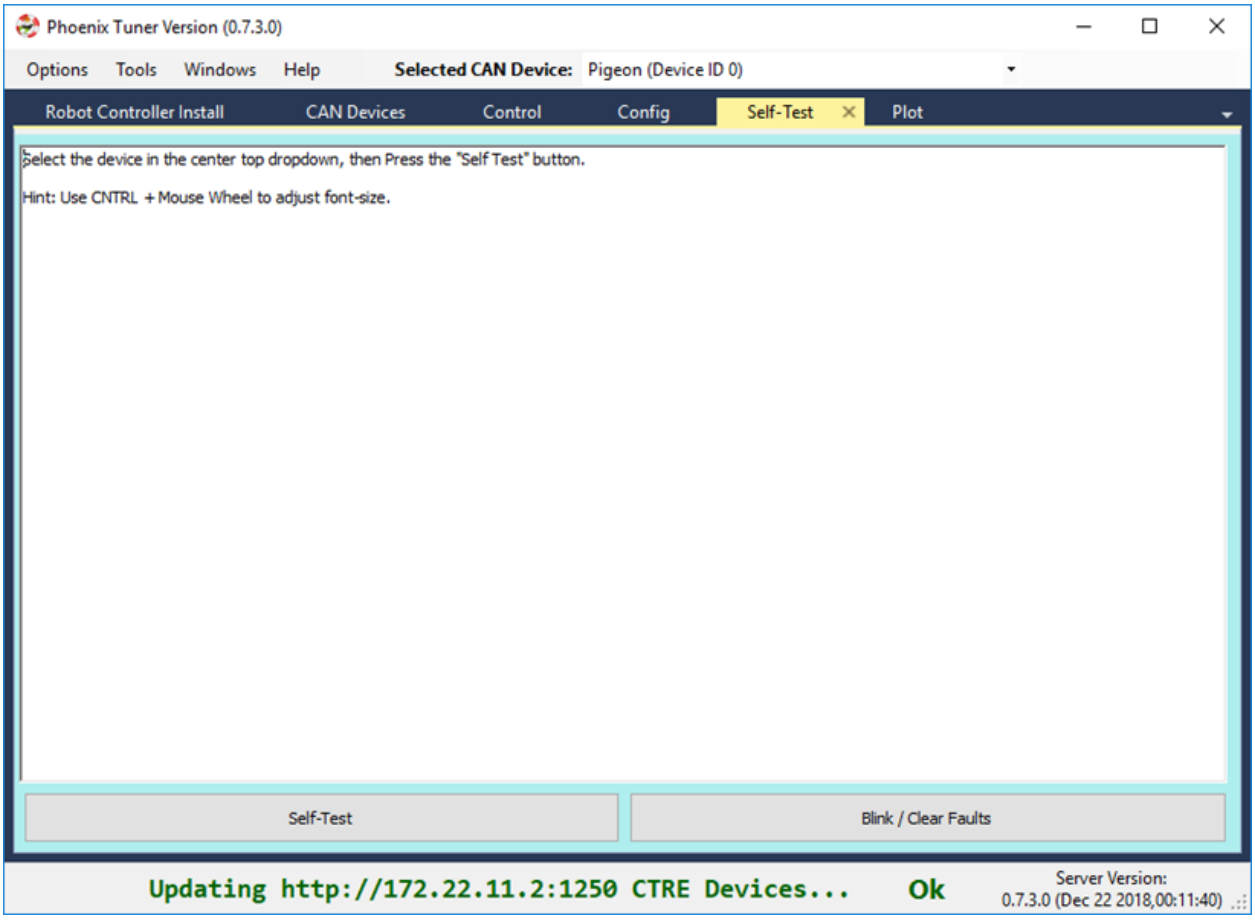
At this point every device should be present on the bus, and updated to latest. This is an opportune time to test the self-test feature of each device.

Select each device either in the device list, or using the dropdown at the center-top. This dropdown is convenient as it is accessible regardless of how the tabs are docked with Tuner.

Note: If you press the "Selected CAN device" text next to dropdown, you will be taken back to the CAN Devices tab.



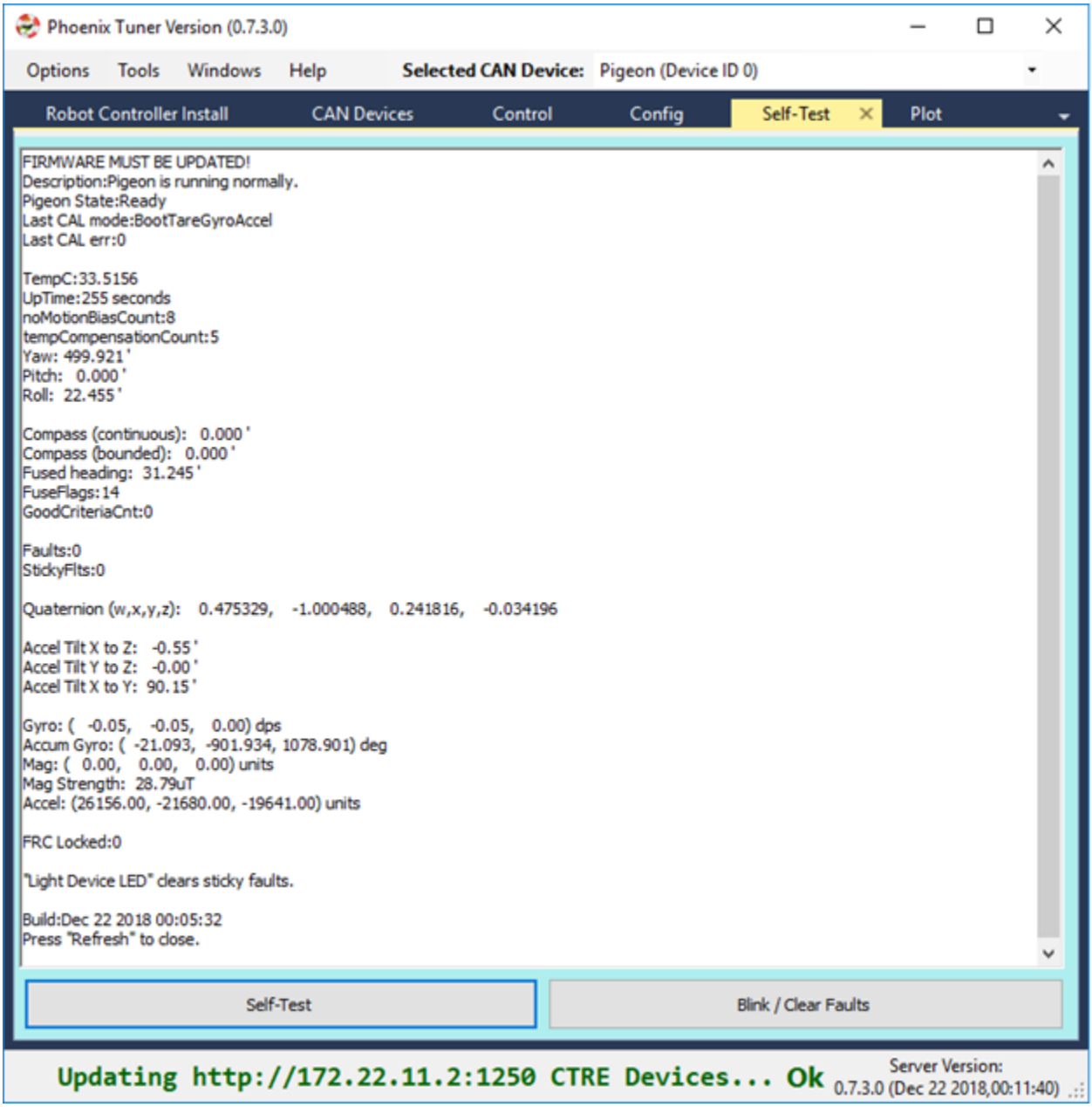
Navigate to the self-test tab. If self-test tab is not present, use the Windows menu bar to reopen it.



Press self-test button and confirm the results.

Note: This Pigeon has not had its firmware updated, this is reported at the top.

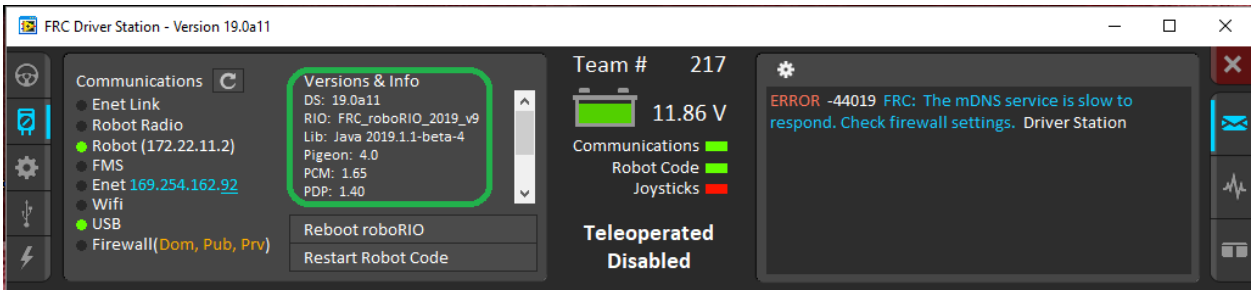
You can also use the Blink/Clear faults button to blink the selected device and clear any previously logged sticky faults.



2.10.10 Driver Station Versions Page

It is worth mentioning there is basic support of reporting the CAN devices and their versions in the diagnostics tab of the Driver Station.

If there is a mixed collection of firmware versions for a given product type, the version will report “Inconsistent”.



Note: The recommended method for confirming firmware versions is to use Phoenix Tuner.

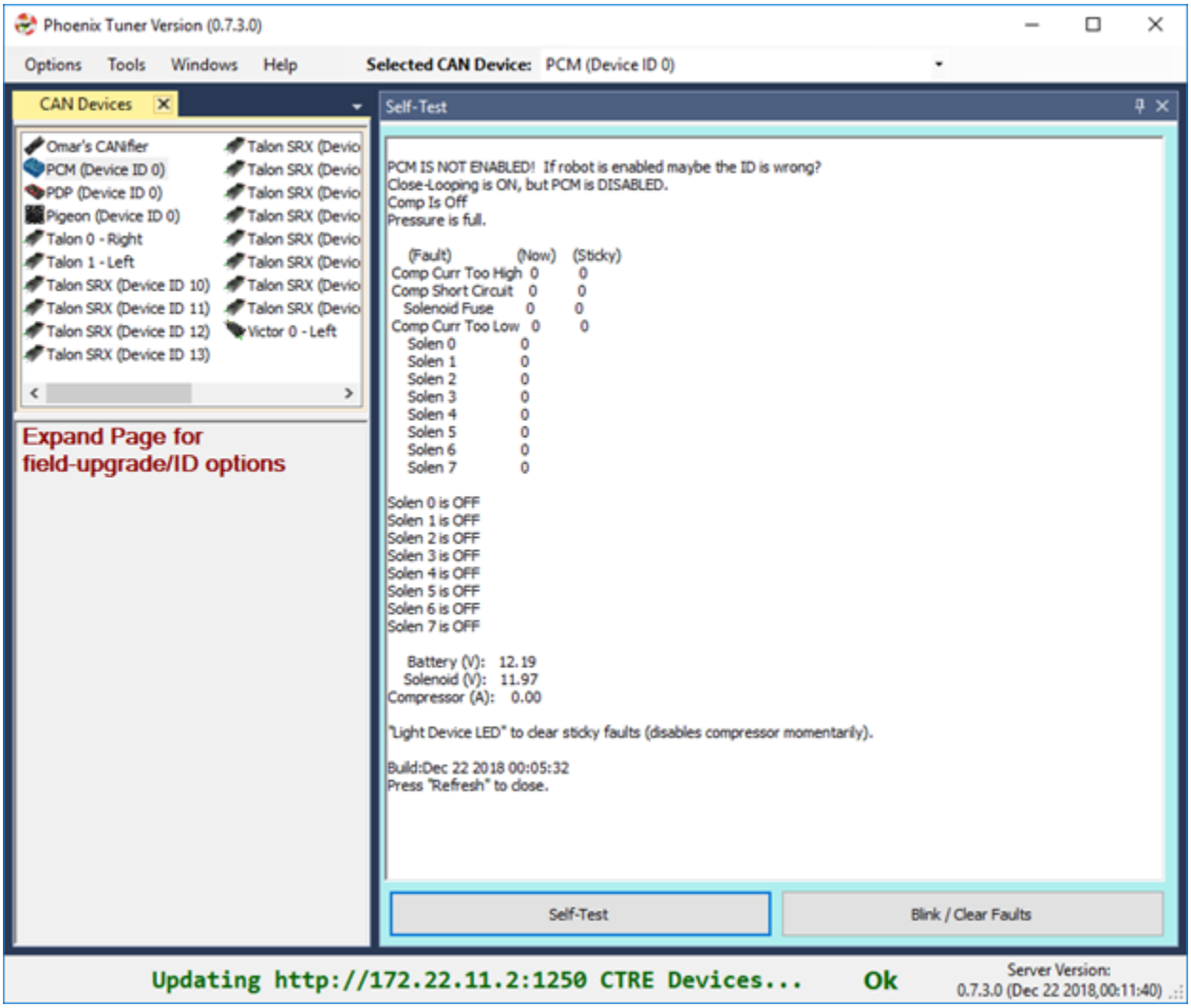
Note: There is a known issue where ribbon-cabled Pigeons may erroneously report as a Talon. Since this is not a critical feature of the Driver Station, this should not be problematic for FRC teams.

2.11 Bring Up: PCM

At this point PCM will have firmware 1.62 or 1.65 (latest). Open Phoenix Tuner to confirm.

2.11.1 Phoenix Tuner Self-Test

Press self-test to confirm solenoid states, compressor state ,and battery/current measurements. Since device is not enabled, no outputs should assert.



Note: In this view, the Self-Test was docked to the right. If CAN Devices width is shrunk small enough, the field-upgrade and Device ID options are hidden and the list view becomes collapsed. This way you can still use the device list as an alternative to the center-top dropdown.

The next step is to get the compressor and solenoids operational.

The PCM User's guide will have the details on how to accomplish this. http://www.ctr-electronics.com/pcm.html#product_tabs_technical_resources

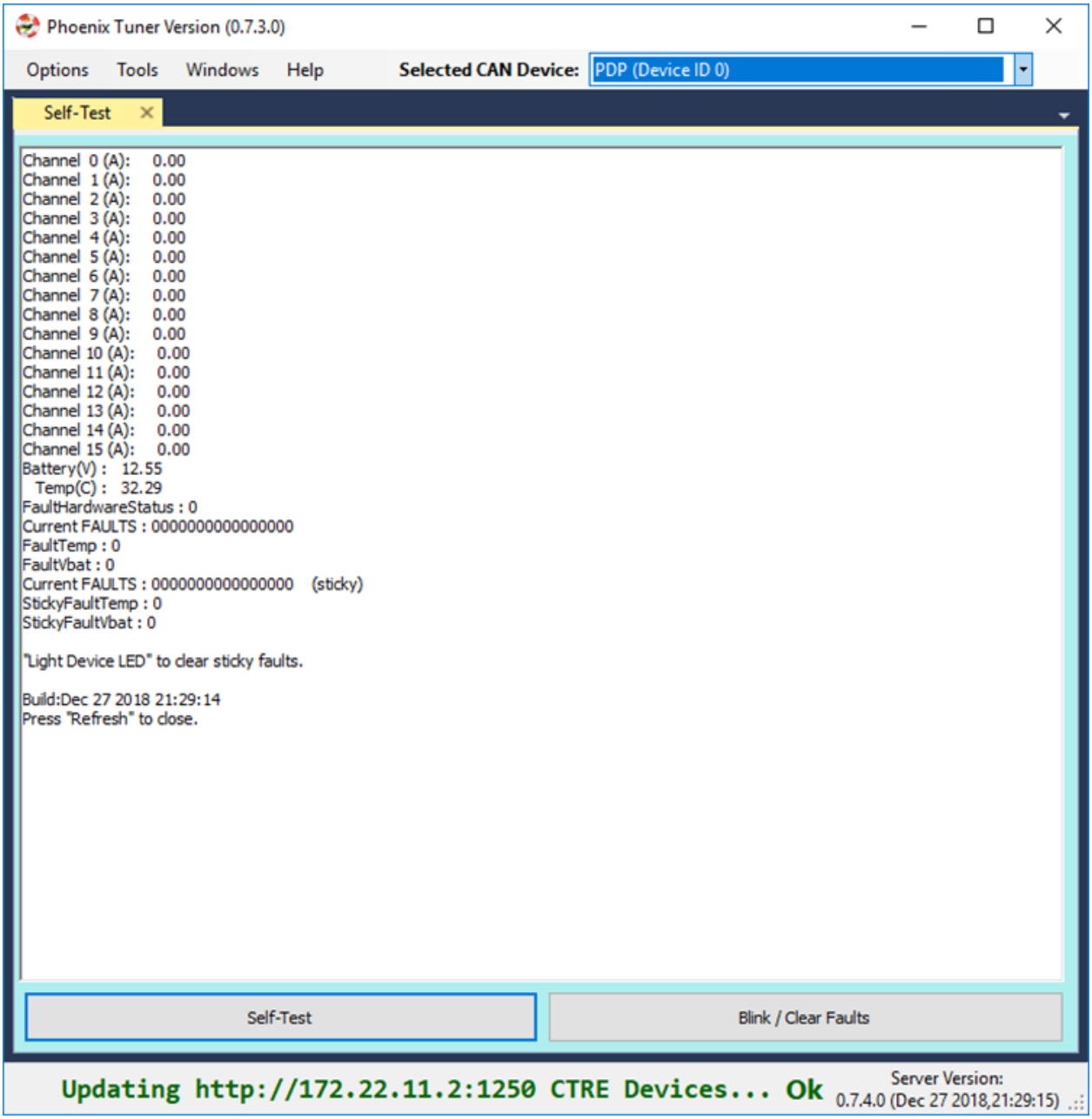
Create a Solenoid object in LabVIEW/C++/Java and set channel 0 to true. Then confirm using the Solenoid LED on the PCM and self-test in Tuner.

After creating a solenoid object, the compressor will automatically engage if the robot is enabled via the Driver Station and if pressure-switch reads too-low (also confirm in self-test).

2.12 Bring Up: PDP

At this point PDP will have firmware 1.30 or 1.40 (latest). Open Phoenix Tuner to confirm.

Use Self-Test to confirm reasonable values for current and voltage.



2.12.1 Getting sensor data

Sensor data can also be retrieved using the base FRC API available in LabVIEW/C++/Java. See WPI/NI/FRC documentation for how.

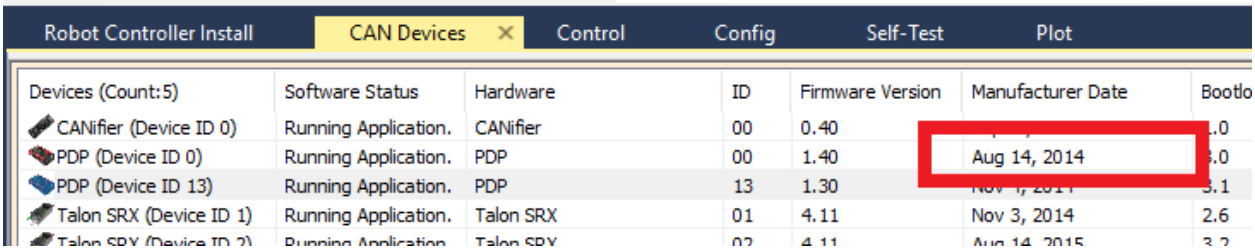
2.12.2 DriverStation Logs

Driver Station logs are automatically generated during normal FRC use. This includes current logging on all PDP Wago channels. Review WPI/NI/FRC documentation for how leverage this.

2.12.3 2015 Kick off Kit PDPs

There is a known issue with 2015 Kickoff-Kit PDPs where the PDP will not appear on CAN bus and/or LEDs will be red, despite all other devices on the CAN bus functioning properly. This is due to an ESD vulnerability that only exists in the initial manufacture run in 2014. Every season PDP afterwards does not have this issue.

Manufacture date of the PDP can be checked in Tuner. Any PDP with a manufacture date of August 14, 2014 may have this issue. No other PDPs (even those with other 2014 manufacture dates) are known to be affected.



These PDPs do correctly provide power and terminate the CAN bus with no compromises. However, the current measurement features may not be correct or available on this version of PDP. If such a PDP is re-used or re-purposed, we recommend using it on your practice robot or for bench setups, and not for competition.

2.13 Bring Up: Pigeon IMU

2.13.1 Power Boot

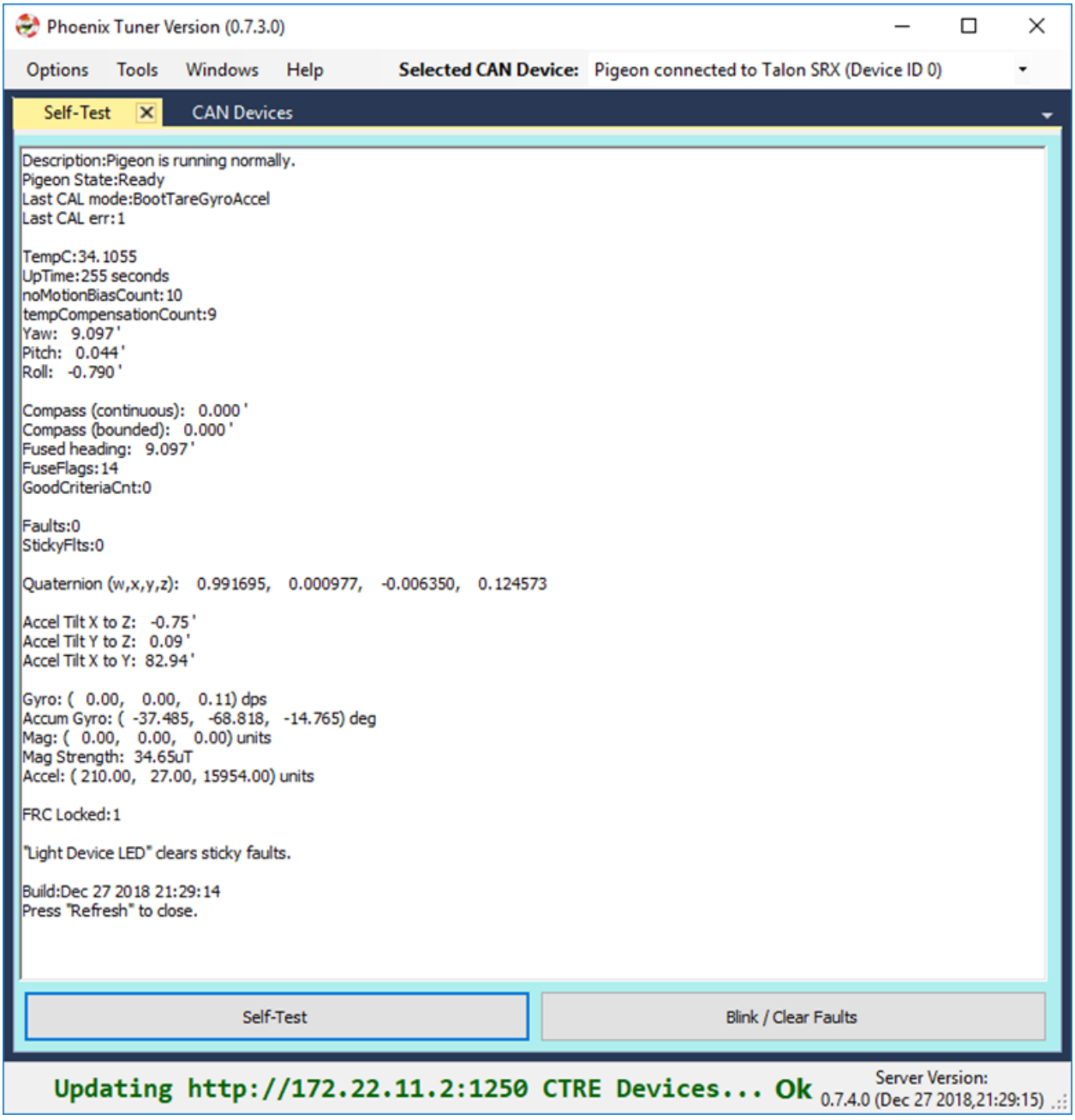
Power boot the robot and wait for Pigeon IMU LED pattern indicating device has settled. This will appear as a symmetric blink pattern (equal time on each side's LED). If the LED strobe is weighted to one side (more time on one side than the other) then IMU is still settling. Typical settle time is four seconds.

Warning: Ribbon cabled Pigeon may not appear in CAN devices if Talon SRX firmware is too old.

Warning: Ribbon cabled Pigeon may not work as a remote sensor unless Pigeon Firmware is at least 4.13.

2.13.2 Phoenix Tuner

Open Phoenix tuner and use the self-test feature to confirm values. Rotate IMU and confirm Yaw moves as expected.



Note: Moving counter-clockwise is interpreted as a positive change.

2.13.3 Pigeon API

Create a Pigeon IMU object in your robot application and poll the Yaw value.

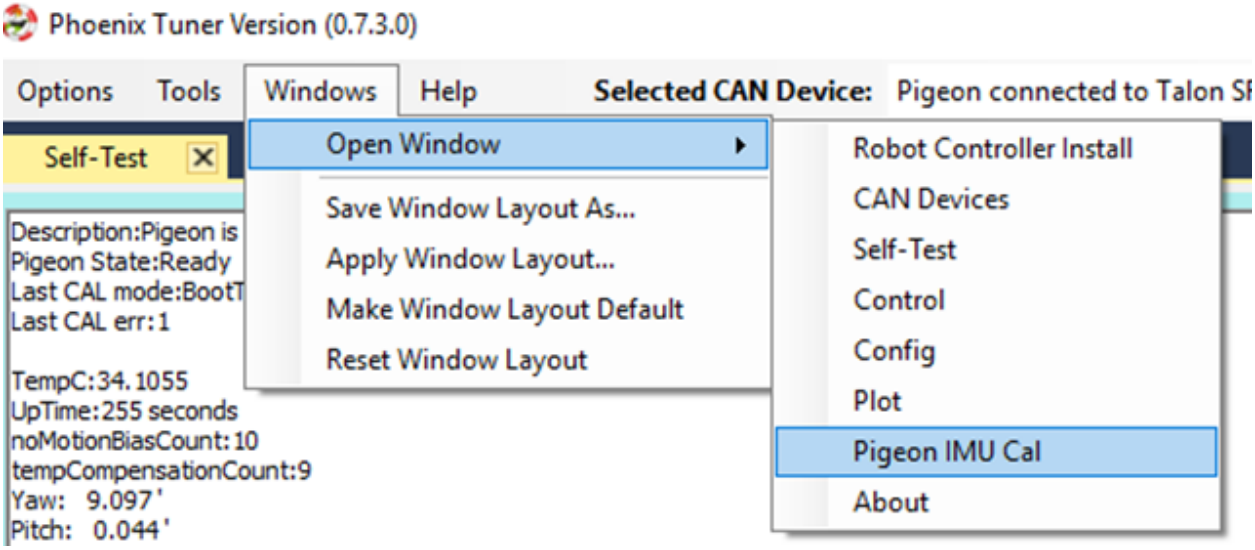
Confirm that the output matches the self-test results.

If using LabVIEW plotter or SmartDash plotting, send the Yaw value into the plotted channel. Then confirm Yaw value provides a smooth curve while robot is rotated by hand.

2.13.4 Temperature Calibration

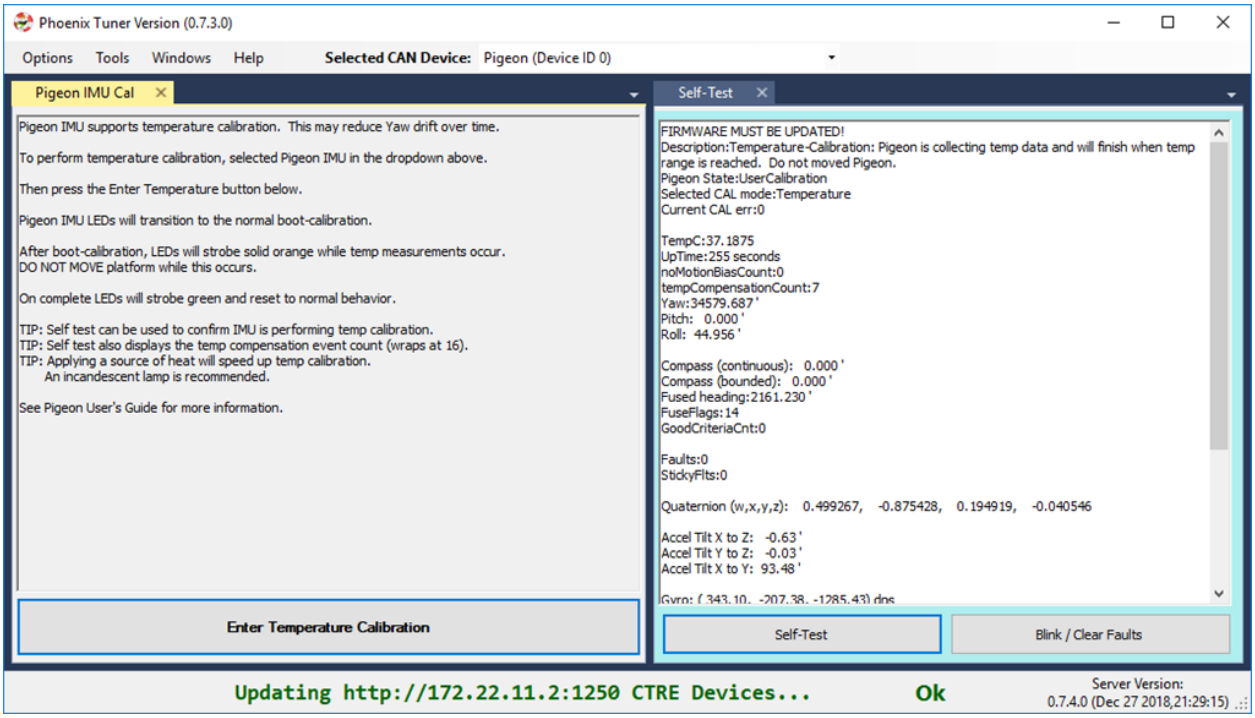
The greatest source of yaw drift in the FRC use case is drift due to changes in temperature. This can be compensated by running the temperature self-calibration once.

In previous seasons this can be invoked via Phoenix API.



However, starting in 2019, you can manually enter temperature compensation mode by opening the Pigeon IMU Cal tab (go to Windows in the top menu bar).

Select the specific Pigeon in the top drop down, and press the Enter Temperature Calibration button. Self-test can be used to monitor the progress.



Note: There is no harm in starting a temp calibration, and aborting by power cycling. Previous temp calibration (if

present) is overridden at the very end of the procedure. See Self-Test for current state of Temperature Calibration and Compensation.

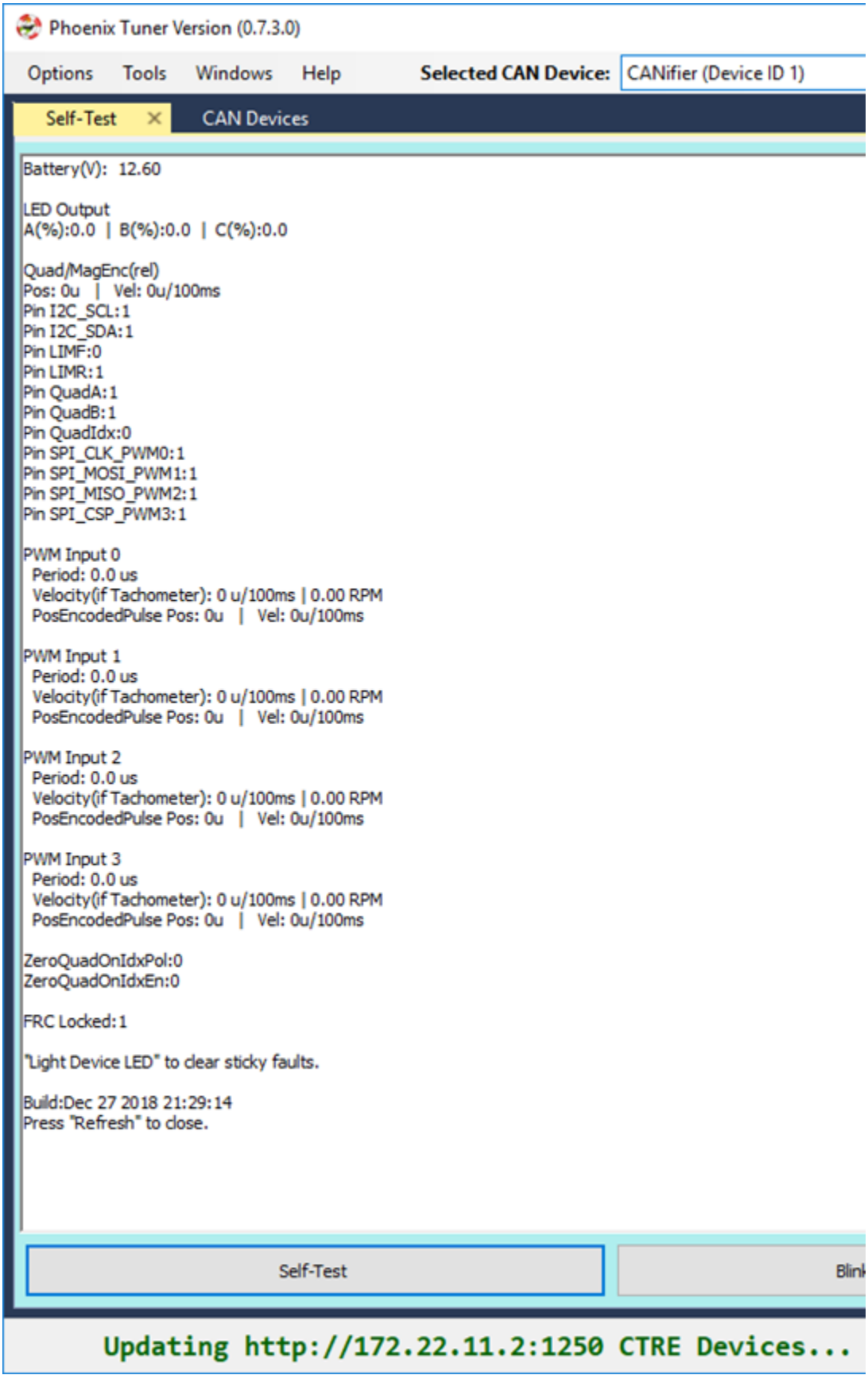
2.14 Bring Up: CANifier

2.14.1 Phoenix Tuner

Using self-test, confirm all sensor inputs required by the robot application.

If using Limit switches, assert each switch one at time. Self-test after each transition to confirm wiring.

If using Quadrature or Pulse width sensor, rotate sensor while performing self-test to confirm sensor values.


Phoenix Tuner Version (0.7.3.0)
Options
Tools
Windows
Help
Selected CAN Device:
CANifier (Device ID 1)
Self-Test
CAN Devices
Battery(V): 12.60
LED Output
A(%):0.0 | B(%):0.0 | C(%):0.0
Quad/MagEnc(rel)
Pos: 0u | Vel: 0u/100ms
Pin I2C_SCL:1
Pin I2C_SDA:1
Pin LIMF:0
Pin LIMR:1
Pin QuadA:1
Pin QuadB:1
Pin QuadIdx:0
Pin SPI_CLK_PWM0:1
Pin SPI_MOSI_PWM1:1
Pin SPI_MISO_PWM2:1
Pin SPI_CSP_PWM3:1
PWM Input 0
Period: 0.0 us
Velocity(if Tachometer): 0 u/100ms | 0.00 RPM
PosEncodedPulse Pos: 0u | Vel: 0u/100ms
PWM Input 1
Period: 0.0 us
Velocity(if Tachometer): 0 u/100ms | 0.00 RPM
PosEncodedPulse Pos: 0u | Vel: 0u/100ms
PWM Input 2
Period: 0.0 us
Velocity(if Tachometer): 0 u/100ms | 0.00 RPM
PosEncodedPulse Pos: 0u | Vel: 0u/100ms
PWM Input 3
Period: 0.0 us
Velocity(if Tachometer): 0 u/100ms | 0.00 RPM
PosEncodedPulse Pos: 0u | Vel: 0u/100ms
ZeroQuadOnIdxPol:0
ZeroQuadOnIdxEn:0
FRC Locked:1
"Light Device LED" to clear sticky faults.
Build:Dec 27 2018 21:29:14
Press "Refresh" to close.
Self-Test
Updating http://172.22.11.2:1250 CTRE Devices...

2.14.2 LED Strip Control

See CANifier user's guide wiring and controlling LED Strip.

2.15 Bring Up: Talon SRX / Victor SPX

At this point all Talon and Victors should appear in Tuner with up to date firmware. The next goal is to drive the motor controller manually. This is done to confirm/test:

- Motor and motor wiring
- Transmission/Linkage
- Mechanism design
- Motor Controller drive (both directions)
- Motor Controller sensor during motion

Note: Talon SRX and Victor SPX can be used with PWM or CAN bus. This document covers the CAN bus use-case.

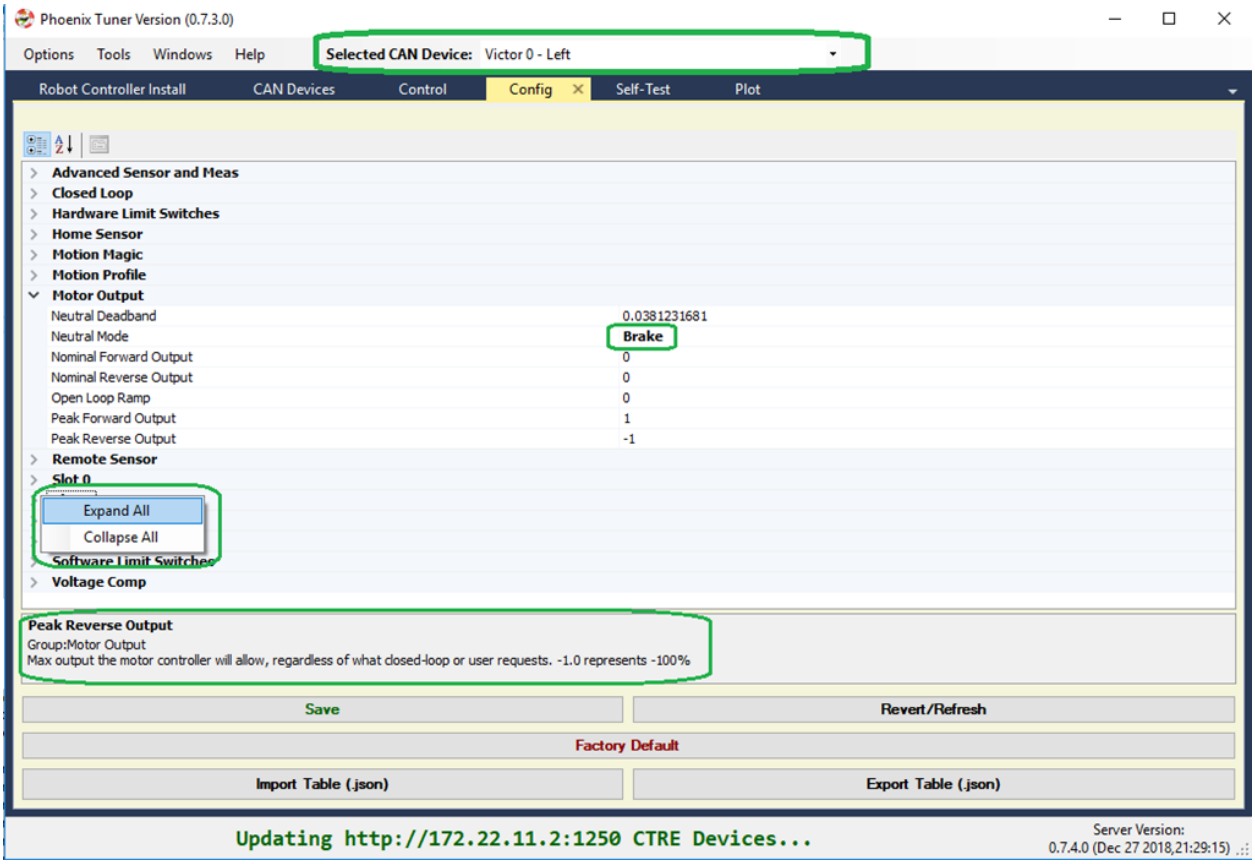
Before we enable the motor controller, first check or reset the configs in the next section.

2.15.1 Factory Default Motor Controller

Open the config view to see all persistent settings in the motor controller. This can be done in the config tab (Windows => Config).

Select the Victor or Talon in the center-top dropdown. This will reveal all persistent config settings.

Press Factory Default to default the motor controller settings so that it has predicable behavior.



Tip: Right-click anywhere in the property inspector and select Collapse-all to collapse each config group.

Tip: Other configs can be set in this view for testing purposes. For example, you may want to restrict the maximum output for testing via the Peak output settings under "Motor Output".

Tip: When a setting is modified, it is set to bold to indicate that it is pending. The bold state will clear after you Save.

Tip: If changing a config live in the robot controller, use the Refresh/Revert button to confirm setting in Tuner.

Note: CTRE devices can be factory defaulted via the API, and thru the B/C mechanical button.

Note: Neutral Mode will not change during factory default as it is stored separately from the other persistent configs.

2.15.2 Configuration

Configurable settings are persistent settings that can be modified via the Phoenix API (from robot code) or via Tuner (Config tab). They can also be factory defaulted using either method.

Configs are modified via the config* routines and LabVIEW Vis. There are two general methods for robust operation of a robot. Additionally you can modify the configs via Tuner.

Method 1 – Use the configAll API

Starting with 2019, there is a single routine/VI for setting all of the configs in a motor controller. This ensures that your application does not need to be aware of every single config in order to reliably configure a fresh or unknown motor controller.

This is the recommend API for new robot projects.

Tip: Config structure/object defaults all values to their factory defaults. This means generally you only need to change the settings you care about.

Tip: When using C++/Java, leverage the IntelliSense (Auto-complete) features of the IDE to quickly discover the config settings you need.

Method 2 – Factory Default and config* routines

Phoenix provides individual config* routines for each config setting. Although this is adequate when the number of configs was small, this can be difficult to manage due to the many features/configs in the CTRE motor controllers.

If using individual config routines, we recommend first calling the configFactoryDefault routine/VI to ensure motor controller is restored to a known state, thus allowing you to only config the settings that you intend to change.

This is recommend for legacy applications to avoid porting effort.

Method 3 – Use Tuner

Tuner can be used to get/set/export/import the configs.

However, it is **highly recommended to ultimately set them via the software API**. This way, in the event a device is replaced, you can rely on your software to properly configured the new device, without having to remember to use Tuner to apply the correct values.

A general recommendation is to:

- Configure all devices during robot-bootup using the API,
- Use Tuner to dial values quickly during testing/calibration.
- Export the settings so they are not lost.
- Update your software config values so that Tuner is no longer necessary.

Control Signals

The majority of the behavior in the Talon/Victor is controlled via configs, however there is a small number of control signals that are controlled via the API.

This list includes:

- Current Limit **Enable** (though the thresholds are configs)
- Voltage Compensation **Enable** (though the nominal voltage is a config)
- Control Mode and Target/Output demand (percent, position, velocity, etc.)
- Invert direction and sensor phase
- Closed-loop slot selection [0,3] for primary and aux PID loops.
- Neutral mode override (convenient to temporarily override configs)
- Limit switch override (convenient to temporarily override configs)
- Soft Limit override (convenient to temporarily override configs)
- Status Frame Periods

These control signals do not require periodic calls to ensure they "stick". All of the above signals are automatically restored even after motor controller is power cycled during use except for Status Frame Periods, which can be manually restore by polling for device resets via hasResetOccurred().

Note: WPI motor safety features may require periodic calls to Set() if team software has chosen to enable it.

Note: The override control signals are useful for applications that require temporarily disabling or changing behavior. For example, overriding-disable the soft limits while performing a self-calibration routine to tare sensors, then restoring soft limits for robot operation.

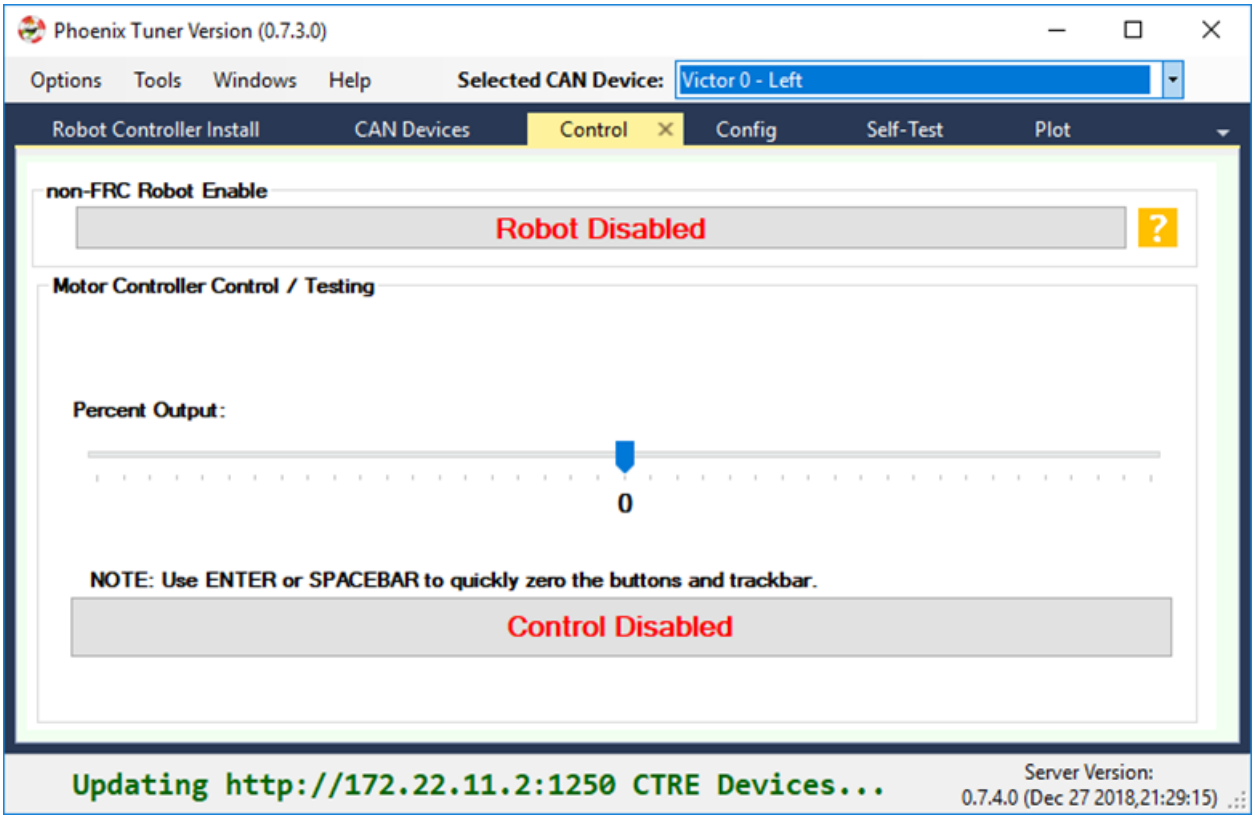
Note: The routines to manipulate control signals are not prefixed with config* to highlight that they are not configs

---

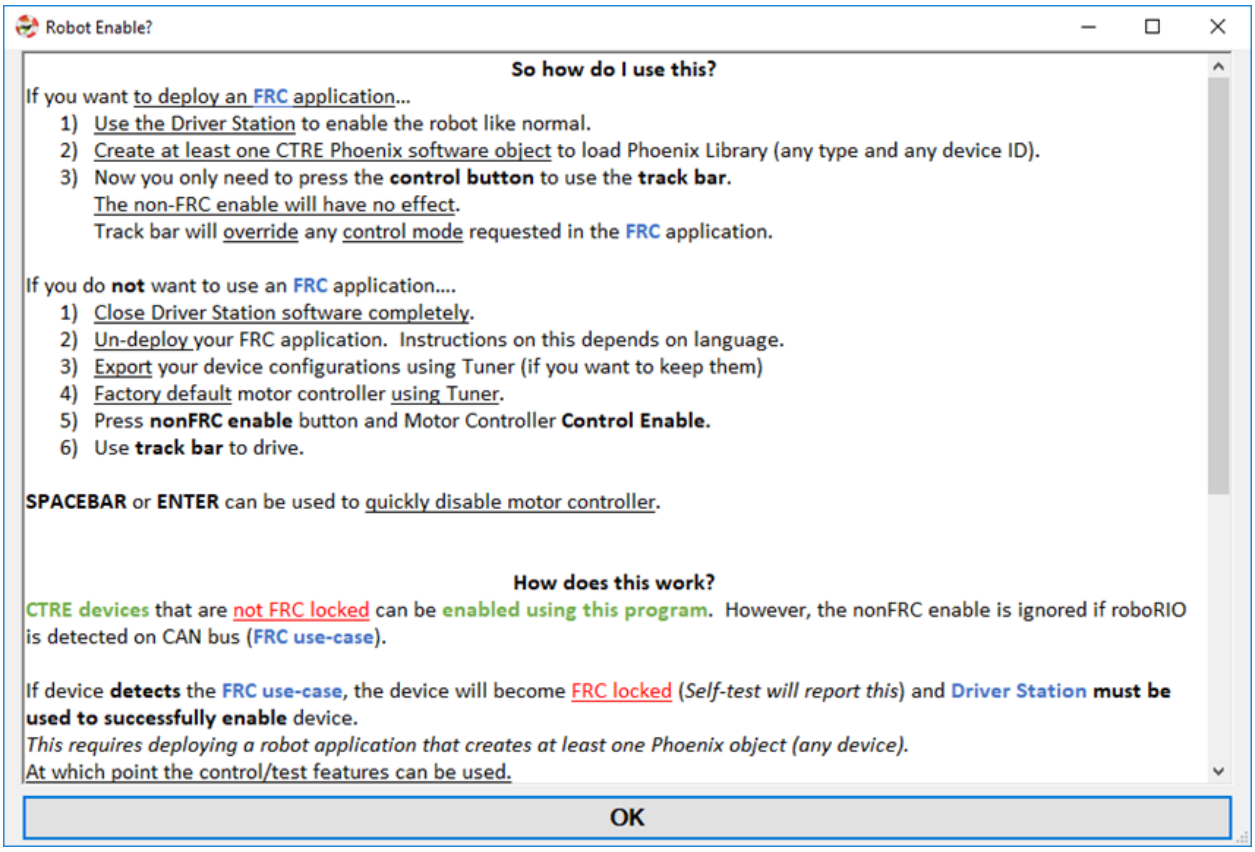
2.15.3 Test Drive with Tuner

Navigate to the control tab to view the control interface. Notice there are two enable/disable buttons. One is for non-FRC style robot-enable (alternative to the Driver Station enable), and one is for Motor Controller Control-Enable.

Press on the question mark next to the robot disabled/enabled button.



This will reveal the full explanation of how to safely enable your motor controller. Follow the appropriate instructions depending on if you want to use Driver Station for your robot-enable.



Setting up non-FRC Control

In order to enable without the Driver Station and without a deployed FRC application, you must first ensure no FRC application is running in the roboRIO.

Or alternatively you can deploy a simple application that does not create any Phoenix objects.

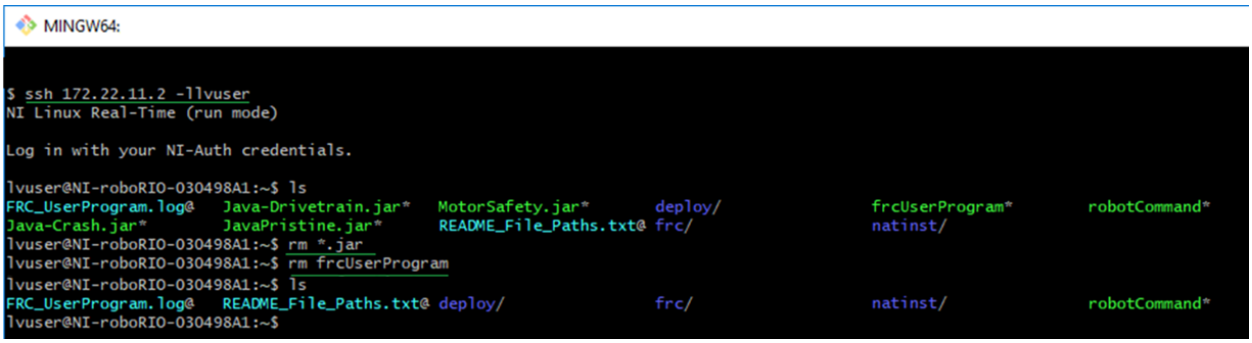
Otherwise CTRE CAN devices will detect the FRC use case and FRC-lock (meaning they will again require the Driver Station).

Option 1 (easiest): deploy a "dummy" FRC application

This simply means create a small project from one of the available templates. Do not create any Phoenix CAN objects.

Option 2: Un-deploy FRC application from your RIO

For C++/Java teams familiar with ssh, you can quickly un-deploy your roboRIO application by removing or naming your program jar file (Java) or frcUserProgram (C++). Power cycle after executing commands. Then confirm Driver Station reads "No Code".



Option 3 (slowest): Reimage the roboRIO

Re-imaging the RIO also will effectively remove the application, however this is a "sledge hammer" approach will take several minutes to perform.

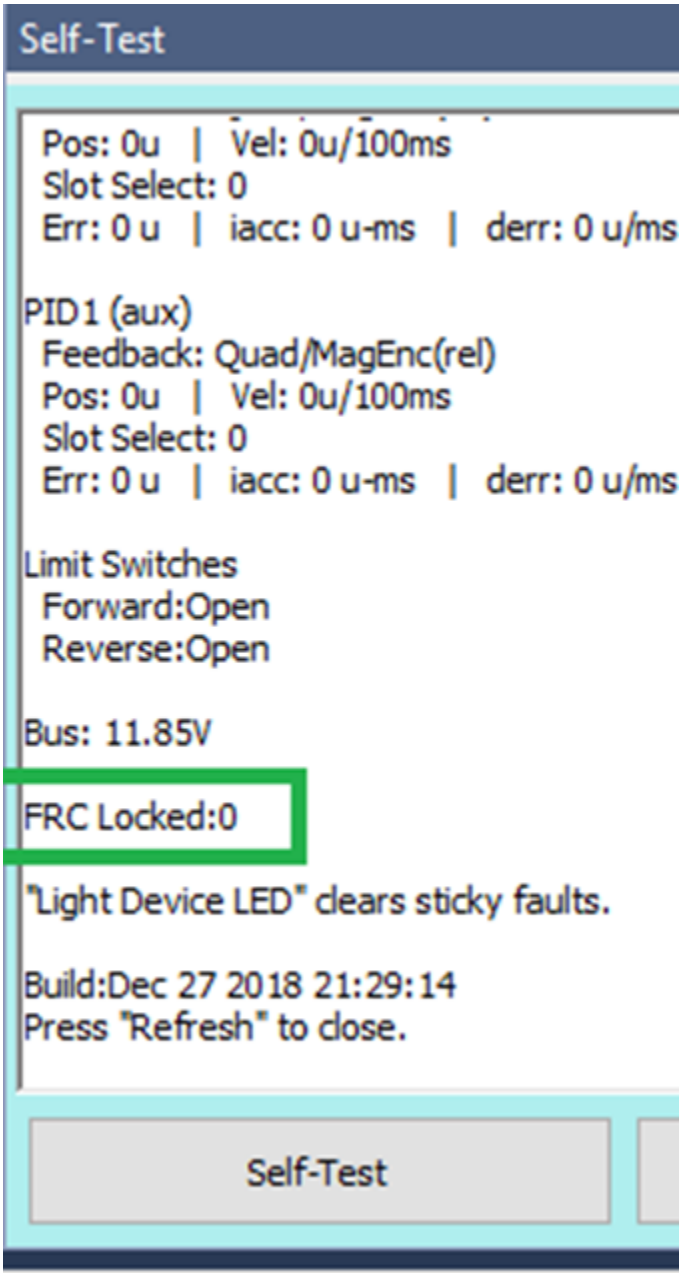
Confirm FRC Unlock

Close Driver Station software if it is running. Do not allow DS to communicate with roboRIO, or CTRE devices will detect the FRC use case.

Self-Test Motor Controller to confirm device FRCLocked = 0.

If device is FRC Locked (=1), use factory default in the config tab to clear the state.

Note: Use the config export tool if you need to keep your config settings.

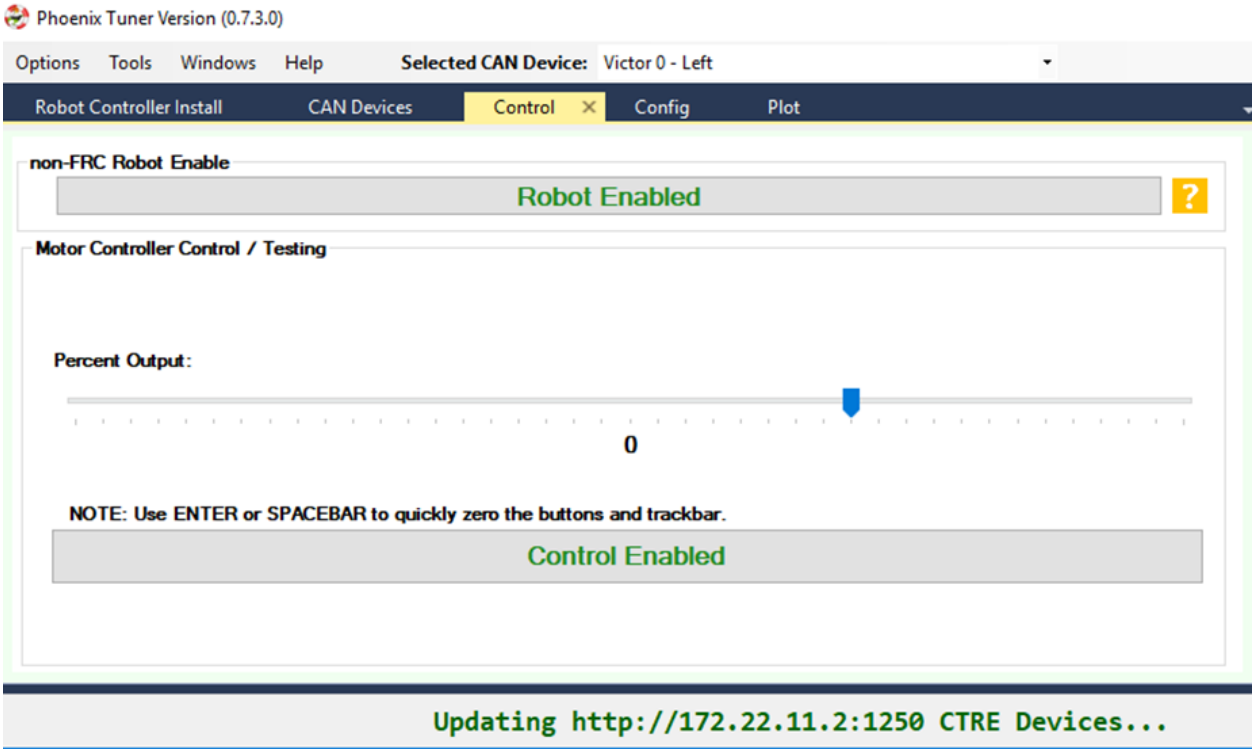


Control tab

Press both Robot Enabled and Control Enabled. At this point you can use the track bar to drive the Victor/Talon.

Note: If you do connect the driver station, the Talon/Victor will FRC Lock again. At which point you can use the driver station to enable, and you no longer need to use the non-FRC Robot enable in Tuner.

Note: Spacebar or enter can be used to clear the control tab and neutral the selected motor controller.



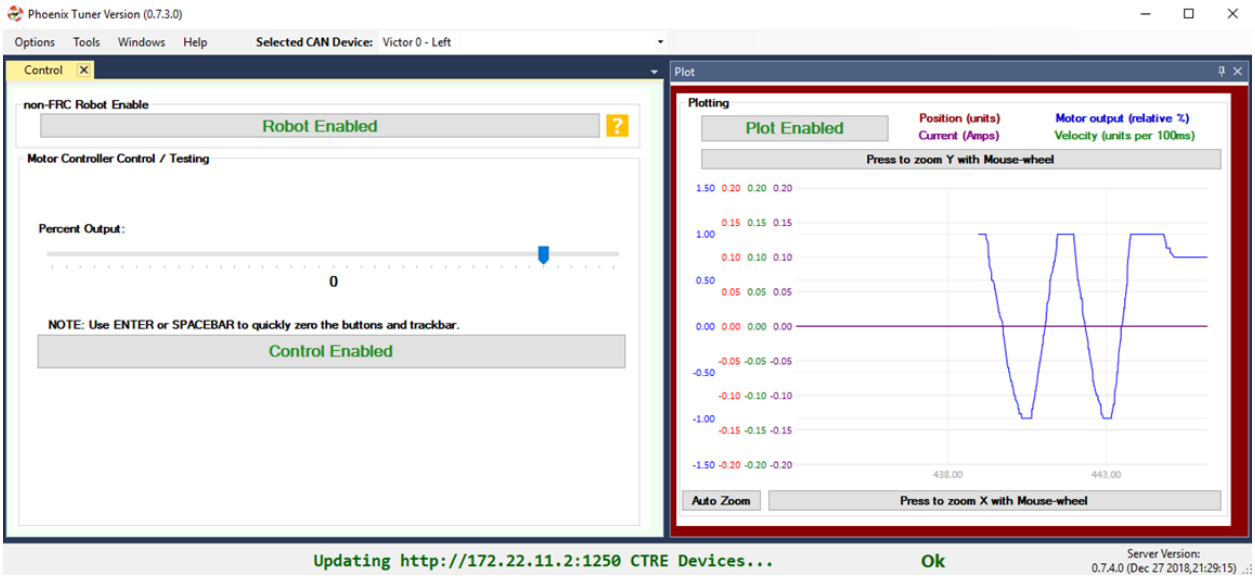
Plot tab

Now open the Plot window. Drive the motor controller while observing the plot. Confirm the blue motor output curve matches LED behavior and trackbar. Confirm motor movement follows expectations.

Note: Press the Plot enable button to effectively pause the plot for review

Note: Use the Zoom buttons to select whether the mouse adjust the Y or X axis.

Note: If using a Victor SPX, current-draw will always read zero (SPX does not have current-measurement features).



Tip: Plot can be used anytime, regardless of what is commanding the motor controller (FRC or non-FRC).

2.15.4 Test Drive with Robot Controller

Next we will create control software in the roboRIO. Currently this is necessary for more advanced control. This is also required for controlling your robot during competition.

Note: Future features of Tuner will likely provide greater granularity of control – closed-loops, invert/follower, etc.. Initial release is meant to help teams get started and learn the subtleties of testing without developing robot software first.

Below is a simple example that reads the Joystick and drives the Talon

```
package frc.robot;

import com.ctre.phoenix.motorcontrol.ControlMode;
import com.ctre.phoenix.motorcontrol.can.TalonSRX;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
  TalonSRX _talon0 = new TalonSRX(0);
  Joystick _joystick = new Joystick(0);

  @Override
  public void teleopPeriodic() {
    double stick = _joystick.getRawAxis(1);
    _talon0.set(ControlMode.PercentOutput, stick);
  }
}
```

Deploy the project, and confirm success.

Note: WPI's terminal output may read "Build" successful despite the project was deployed.

```
 8  package frc.robot;
 9
10  import com.ctre.phoenix.motorcontrol.ControlMode;
11  import com.ctre.phoenix.motorcontrol.can.TalonSRX;
12
13  import edu.wpi.first.wpilibj.Joystick;
14  import edu.wpi.first.wpilibj.TimedRobot;
15
16  public class Robot extends TimedRobot {
17    TalonSRX _talon0 = new TalonSRX(0);
18    Joystick _joystick = new Joystick(0);
19
20    @Override
21    public void teleopPeriodic() {
22      double stick = _joystick.getRawAxis(0);
23      _talon0.set(ControlMode.PercentOutput, stick);
24    }
25  }
26
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
    -C-> chmod +x "/home/lvuser/MyJavaProject-1.jar"; chown lvuser "/home/lvuser/MyJavaProject-1.jar" @ /
    -C-> sync @ /home/lvuser
      -[-1]->
    -C-> . /etc/profile.d/natinst-path.sh; /usr/local/frc/bin/frcKillRobot.sh -t -r 2> /dev/null @ /home/

> Task :deployNativeZipRoborio
    42 file(s) are up-to-date and were not deployed
    -C-> chmod -R 777 "/usr/local/frc/third-party/lib" || true; chown -R lvuser:ni "/usr/local/frc/third-
    -C-> ldconfig @ /usr/local/frc/third-party/lib

Deprecated Gradle features were used in this build, making it incompatible with Gradle 6.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/5.0/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 5s
10 actionable tasks: 8 executed, 2 up-to-date
```

Note: Before you enable the DS, spin the Joystick axis so it reaches the X and Y extremities are reached. USB Gamepads calibrate on-the-fly so if the Gamepad was just inserted into the DS, it likely has not auto detected the max mechanical range of the sticks.

Note: Make sure joystick is detected by the DS before enabling.

Note: getRawAxis may not return a positive value on forward-stick. Confirm this by watching Talon/Victor LED. Green suggests a positive output.

Enable the Driver Station and confirm:

- motor drive in both directions using gamepad stick.
- motor controller LEDs show green for forward and red for reverse

Disable Driver Station after finished testing.

Note: If the LED is solid orange than use Tuner to determine the cause. Self-Test will report the current state of the motor controller (do this while troubleshooting). Confirm firmware is up to date.

2.15.5 Open-Loop Features

After some rudimentary testing, you will likely need to configure several open-loop features of the Talon SRX and Victor SPX.

Note: We recommend configuring Inverts and Followers first.

Inverts

To determine the desired invert of our motor controller, we will add two more lines of call. SetInverted is added to decide if motor should spin clockwise or counter clockwise when told to move positive/forward (green LEDs).

We also multiply the joystick so that forward is positive (intuitive). This can be verified by watching the console print in the Driver Station.

```
package frc.robot;
import com.ctre.phoenix.motorcontrol.*;
import com.ctre.phoenix.motorcontrol.can.*;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
  TalonSRX _talon0 = new TalonSRX(0);
  Joystick _joystick = new Joystick(0);

  @Override
  public void teleopInit() {
    _talon0.setInverted(false); // pick CW versus CCW when motor controller is␣
↪positive/green
  }

  @Override
  public void teleopPeriodic() {
    double stick = _joystick.getRawAxis(1) * -1; // make forward stick positive
    System.out.println("stick:" + stick);

    _talon0.set(ControlMode.PercentOutput, stick);
  }
}
```

Follower

If a mechanism requires multiple motors, than there are likely multiple motor controllers. The Follower feature of the Talon SRX and Victor SPX is a convenient method to keep two or more motor controller outputs consistent. If you have a sensor for closed-looping, connect that to the "master" Talon SRX (unless it is a remote sensor such as CANifier/Pigeon).

Below we've added a new Victor to follow Talon 0.

Generally, a follower is intended to match the direction of the master, or drive in the opposite direction depending on mechanical orientation. In previous seasons teams would have to update the bool true/false of the follower to match or oppose the master manually.

Starting in 2019, C++/Java users can set the setInverted(InvertType) to instruct the motor controller to either match or oppose the direction of the master instead.

```
package frc.robot;

import com.ctre.phoenix.motorcontrol.*;
import com.ctre.phoenix.motorcontrol.can.*;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
  TalonSRX _talon0 = new TalonSRX(0);
  VictorSPX _victor0 = new VictorSPX(0);
  Joystick _joystick = new Joystick(0);

  @Override
  public void teleopInit() {
    _victor0.follow(_talon0);

    _talon0.setInverted(false); // pick CW versus CCW when motor controller is␣
↪positive/green
    _victor0.setInverted(InvertType.FollowMaster); // match whatever talon0 is
    //_victor0.setInverted(InvertType.OpposeMaster); // opposite whatever talon0 is
  }

  @Override
  public void teleopPeriodic() {
    double stick = _joystick.getRawAxis(1) * -1; // make forward stick positive
    System.out.println("stick:" + stick);

    _talon0.set(ControlMode.PercentOutput, stick);
  }
}
```

Enable the Driver Station and slowly drive both MCs from neutral. Confirm both LEDs are blinking the same color.

Disable Driver Station when complete.

To confirm motor controllers are truly driving in the same direction, disconnect the master motor controller from its motor.

Enable the Driver Station and confirm follower motor direction matches previously measured master motor direction.

Disable Driver Station when complete.

Open Tuner and select the master motor controller.

Open plot tab and enable plotter while driving motor controller

Confirm current plot is appropriate. If motors are free-spinning, then current should be near 0 if motor output is constant. When testing drive train, the robot should be rested on a crate/tote to ensure all wheels spin freely.

Select follower motor in Tuner, and confirm current via plot.

Note: Follower mode can be canceled by simple calling set()

Note: Calling follow() in the periodic loop is not required, but also does not affect anything in a negative way.

Neutral Mode

You may note that when the motor output transitions to neutral, the motors free spin (coast) in the last direction they were driven. If the Talon/Victor is set to "coast" neutral mode, then this is expected. The neutral mode can also be set to "brake" to electrically common the motor leads during neutral, causing a deceleration that combats the spinning motor motion.

Note: SetNeutralMode() can be used change the neutral mode on the fly.

Follower motor controllers have separate neutral modes than their masters, so you must choose both. Additionally, you may want to mix your neutral modes to achieve a partial electric brake when using multiple motors.

Ramping

The Talon SRX can be set to honor a ramp rate to prevent instantaneous changes in throttle. This ramp rate is in effect regardless of which mode is selected (throttle, slave, or closed-loop).

Ramp can be set in time from neutral to full using configOpenLoopRampRate().

Note: configClosedLoopRampRate() can be used to select the ramp during closed-loop (sensor) operations.

Note: The slowest ramp possible is ten seconds (from neutral to full), though this is quite excessive.

Peak/Nominal Outputs Often a mechanism may not require full motor output. The application can cap the output via the peak forward and reverse config setting (through Tuner or API).

Additionally, the nominal outputs can be selected to ensure that any non-zero requested motor output gets promoted to a minimum output. For example, if the nominal forward is set to +0.10 (+10%), then any motor request within (0%, +10%) will be promoted to +10% assuming request is beyond the neutral dead band. This is useful for mechanisms that require a minimum output for movement, and can be used as a simpler alternative to the kI (integral) component of closed-looping in some circumstances.

Voltage Compensation

Talon SRX and Victor SPX can be configured to adjust their outputs in response to the battery voltage measurement (in all control modes). Use the voltage compensation saturation config to determine what voltage represents 100%

output.

Then enable the voltage compensation using enableVoltageCompensation().

Advanced users can adjust the Voltage Measurement Filter to make the compensation more or less responsive by increasing or decreasing the filter. This is available via API and via Tuner

Current Limit

Talon SRX supports current limiting in all control modes.

The limiting is characterized by three configs:

- Peak Current (Amperes), threshold that must be exceeded before limiting occurs.
- Peak Time (milliseconds), thresholds that must be exceed before limiting occurs
- Continuous Current (Amperes), maximum allowable current after limiting occurs.

If enabled, Talon SRX will monitor the supply-current looking for a conditions where current has exceeded the Peak Current for at least Peak Time. If detected, output is reduced until current measurement is at or under Continuous Current.

Once limiting is active, current limiting will deactivate if motor controller can apply the requested motor output and still measure current-draw under the Continuous Current Limit.

After setting the three configurations, current limiting must be enabled via enableCurrentLimit() or LabVIEW VI.

Note: Use Self-Test to confirm if Current Limiting is occurring

---

2.15.6 Reading status signals

The Talon SRX transmits most of its status signals periodically, i.e. in an unsolicited fashion. This improves bus efficiency by removing the need for "request" frames, and guarantees the signals necessary for the wide range of use cases Talon supports, are available.

These signals are available in API regardless of what control mode the Talon SRX is in. Additionally the signals can be polled in the roboRIO Web-based Configuration (see Section 2.4. Self-Test).

Included in the list of signals are:

- Quadrature Encoder Position, Velocity, Index Rise Count, Pin States (A, B, Index)
- Analog-In Position, Analog-In Velocity, 10bit ADC Value,
- Battery Voltage, Current, Temperature
- Fault states, sticky fault states,
- Limit switch pin states
- Applied Throttle (duty cycle) regardless of control mode.
- Applied Control mode: Voltage % (duty-cycle), Position/Velocity closed-loop, or slave follower.
- Brake State (coast vs brake)
- Closed-Loop Error, the difference between closed-loop set point and actual position/velocity.
- Sensor Position and Velocity, the signed output of the selected Feedback device (robot must select a Feedback device, or rely on default setting of Quadrature Encoder).

2.15.7 Limit Switches

Talon SRX and Victor SPX have limit features that will auto-neutral the motor output if a limit switch activates.

An "out of the box" Talon will default with the limit switch setting of "Normally Open" for both forward and reverse. This means that motor drive is allowed when a limit switch input is not closed (i.e. not connected to ground). When a limit switch input is closed (is connected to ground) the Talon SRX will disable motor drive and individually blink both LEDs red in the direction of the fault (red blink pattern will move towards the M+/white wire for positive limit fault, and towards M-/green wire for negative limit fault).

Since an "out of the box" Talon will likely not be connected to limit switches (at least not initially) and because limit switch inputs are internally pulled high (i.e. the switch is open), the limit switch feature is default to "normally open". This ensures an "out of the box" Talon will drive even if no limit switches are connected.

For more information on Limit Switch wiring/setup, see the Talon SRX User's Guide.

Forward Limit Switch Mode	Limit Switch NO pin	Limit Switch NC pin	Limit Switch COM pin	Motor Drive Switch open Fwd. throttle	Motor Drive Switch closed Fwd. throttle	*Voltage (Switch Open)	*Voltage (Switch Closed)
Normally Open	pin4	N.A.	pin10	Y	N	~2.5V	0 V
Normally Closed	N.A.	pin4	pin10	N	Y	0 V	~2.5V
Disabled	N.A.	N.A.	N.A.	Y	Y	N.A.	N.A.
Reverse Limit Switch Mode	Limit Switch NO pin	Limit Switch NC pin	Limit Switch COM pin	Motor Drive Switch open Rev. throttle	Motor Drive Switch closed Rev. throttle	*Voltage (Switch Open)	*Voltage (Switch Closed)
Normally Open	pin8	N.A.	pin10	Y	N	~2.5V	0 V
Normally Closed	N.A.	pin8	pin10	N	Y	0 V	~2.5V
Disabled	N.A.	N.A.	N.A.	Y	Y	N.A.	N.A.
	*Measured voltage at the Talon SRX Limit Switch Input pin.						
	Limit Switch Input Forward Input - pin4 on Talon SRX						
	Limit Switch Input Reverse Input - pin8 on Talon SRX						
	Limit Switch Ground - pin10 on Talon SRX						

Limit switch features can be disabled or changed to "Normally Closed" in Tuner and in API.

Remote Limit Switches

A Talon SRX or Victor SPX can use a remote sensor as the limit switch (such as another Talon SRX or CANifier).

Config the Limit Forward/Reverse Source from Gadgeteer Pins, to Remote Talon or Remote CANifier. Then config the Limit Forward/Reverse Device ID for the remote Talon or CANifier.

Use self-test on the motor-driving motor controller to confirm limit switches are interpreted correctly. If they are not correct, then self-test the remote device to determine the issue.

2.15.8 Soft Limits

Soft limits can be used to disable motor drive when the "Sensor Position" is outside of a specified range. Forward throttle will be disabled if the "Sensor Position" is greater than the Forward Soft Limit. Reverse throttle will be disabled if the "Sensor Position" is less than the Reverse Soft Limit. The respective Soft Limit Enable must be enabled for this feature to take effect.

The settings can be set and confirmed in the roboRIO Web-based Configuration.

2.16 Bring Up: Talon SRX Sensors

This section is dedicated to validating any rotary sensor attached to the Talon SRX. Generally attaching a sensor is necessary for:

- Close-Loop control modes (Position, MotionMagic, Velocity, MotionProfile)
- Soft limits (auto neutral motor if out of range)

For these features to function correctly, it is imperative to validate the sensor first.

2.16.1 Checking your wiring

Inspect the entire wiring harness between the Talon SRX and the sensor.

If connectors/crimps are used, **tug-test each individual wire one at a time**.

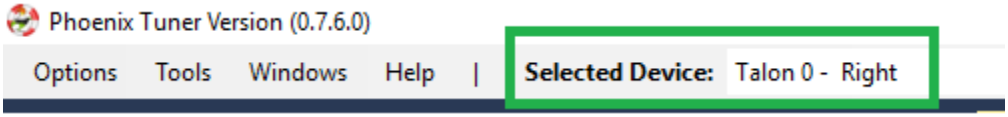
Tip: If sensor is a CTRE Mag Encoder, inspect the ribbon cable for any frayed or damaged sections.

Tip: If sensor is a CTRE Mag Encoder, confirm green LED. When using Versa-Planetary Sensor Slice, orange LED is acceptable.

Warning: When using a contactless sensor on a rolling mechanism (shooter / roller / intake), care should be taken to ensure spinning mechanism is electrically common to the remainder of the robot chassis. Otherwise ESD strikes can occur between mechanism and the contactless sensor (due to its proximity to the mechanism).

2.16.2 Sensor Check – No Motor Drive

Open Phoenix Tuner, and select device in the top dropdown.



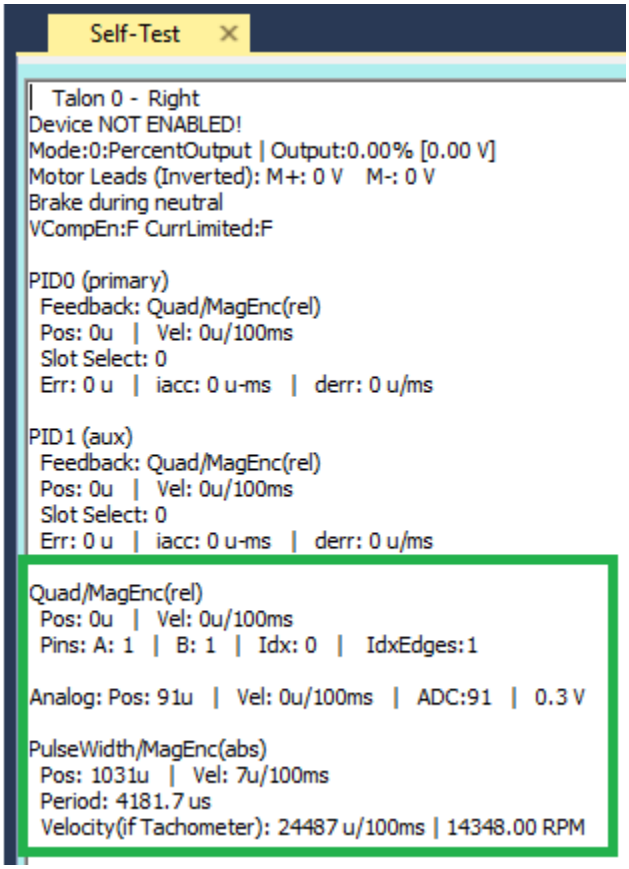
Take an initial capture to first check the settled values (no motion).

Tip: Analog should read ~100 units if floating. If analog sensor is meant to be in-circuit, recheck sensor signal/power/harness/etc.

Tip: Quadrature resets to 0 on power boot.

Tip: Pulse Width should report a period of ~4 milliseconds when using a CTRE Mag encoder.

Tip: If using Talon Tach to measure velocity, focus on "Velocity (if Tach)" under Pulse Width section.



Sensor Options

Many feedback interfaces are supported. The complete list is below.

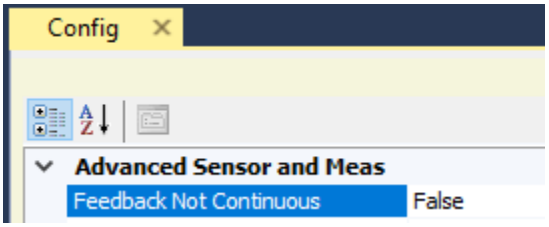
Quadrature

The Talon directly supports quadrature sensors. The decoding is done in 4x mode (every edge on A and B are counted). This is available via the Gadgeteer feedback port.

Analog (Potentiometer or Encoder)

Analog feedback sensors are sensors that provide a variable voltage to represent position. Some devices (such as the MA3 US Digital encoder) are continuous and wrap around from 3.3V back to 0V. In such cases the overwrap is tracked, and Talon continues counting 1023 => 1024.

This feature can be disabled by setting the config via API or Tuner.



Pulse Width Decoder

For sensors that encode position as a pulse width this mode can be used to decode the position. The pulse width decoder is <1us accurate and the maximum time between edges is 120 ms.

Cross The Road Electronics Magnetic Encoder (Absolute and Relative)

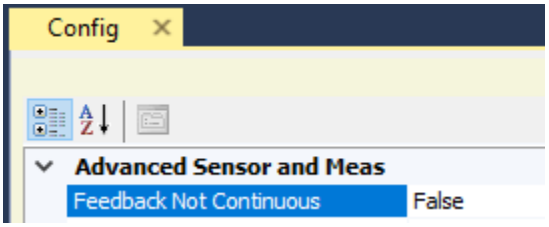
The CTRE Magnetic Encoder is actually two sensor interfaces packaged into one (pulse width and quadrature encoder). Therefore the sensor provides two modes of use: absolute and relative. Each mode provides 4096 units per rotation.

The advantage of absolute mode is having a solid reference to where a mechanism is without re-tare-ing or re-zero-ing the robot. The advantage of the relative mode is the faster update rate. However both values can be read/written at the same time. So a combined strategy of seeding the relative position based on the absolute position can be used to benefit from the higher sampling rate of the relative mode and still have an absolute sensor position.

Parameter	Absolute Mode	Relative Mode
Update rate (period)	4 ms	100 us
Max RPM	7,500 RPM	15,000 RPM
Accuracy	12 bits (4096 units per rotation)	12 bits (4096 units per rotation)
Software API	Select Pulse Width	Select Quadrature

In circumstances where the absolute pulse width wraps from one extremity to the other (due to overflow), the Talon continues counting 4095 => 4096.

This feature can be disabled by setting the config via API or Tuner.



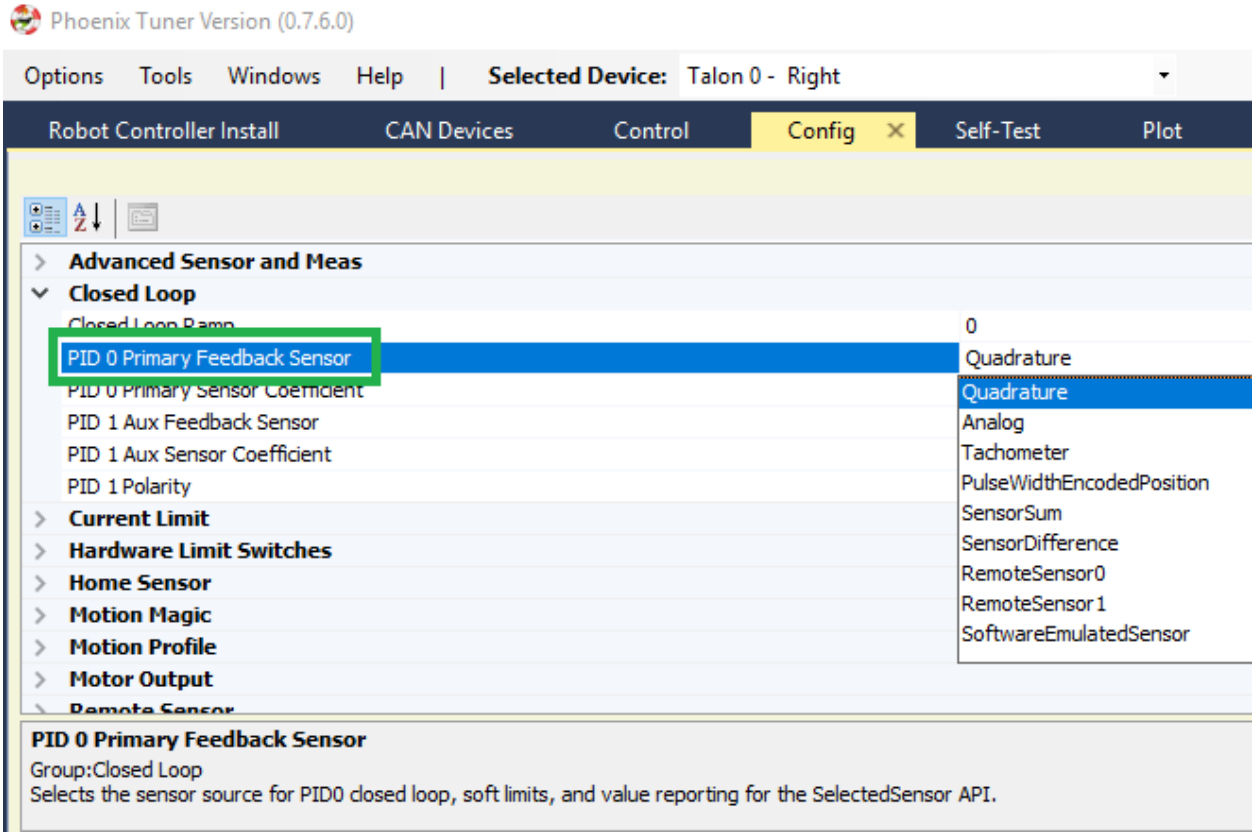
2.16.3 Software Select Sensor Type

Although all sensor values are available all the time, the Talon SRX generally requires the robot application to "pick" a particular "Feedback Device" for soft limit, closed-loop features, and plotter.

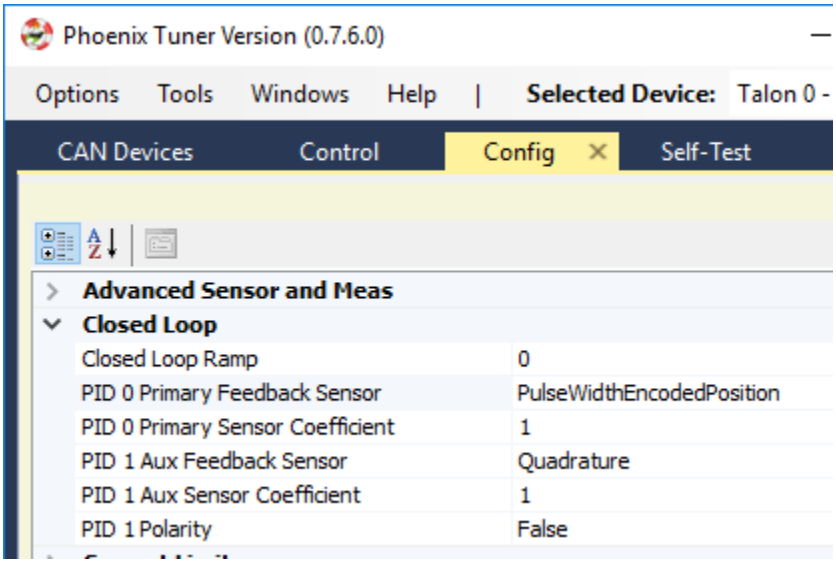
Note: The selected "Feedback Device" defaults to Quadrature Encoder.

Once a "Feedback Device" is selected, the "Sensor Position" and "Sensor Velocity" signals will update with the output of the selected feedback device.

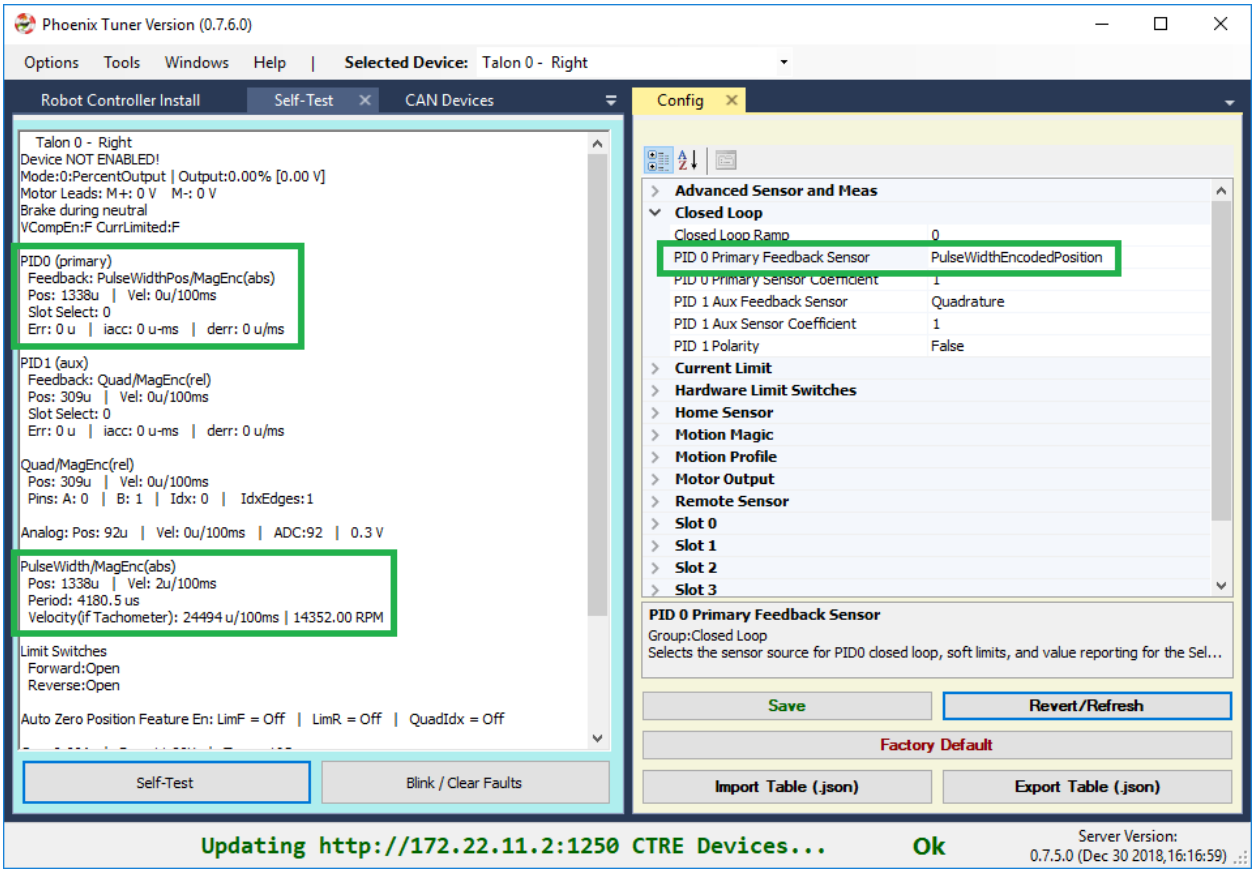
Select the sensor (under PID0 Primary) with either Phoenix API or using Tuner.



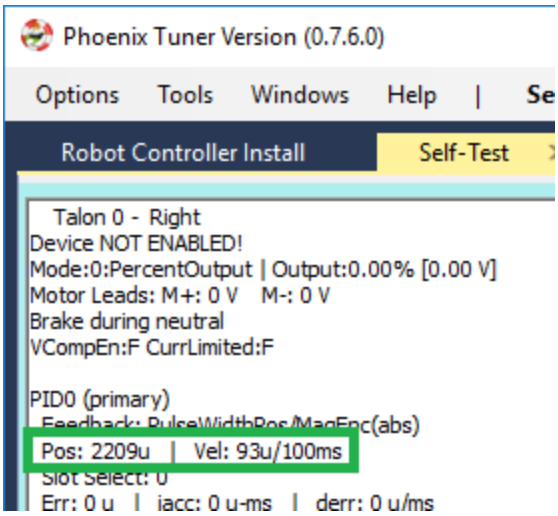
In this example, pulse width position is selected (absolute position within rotation when using CTRE Mag Encoder).



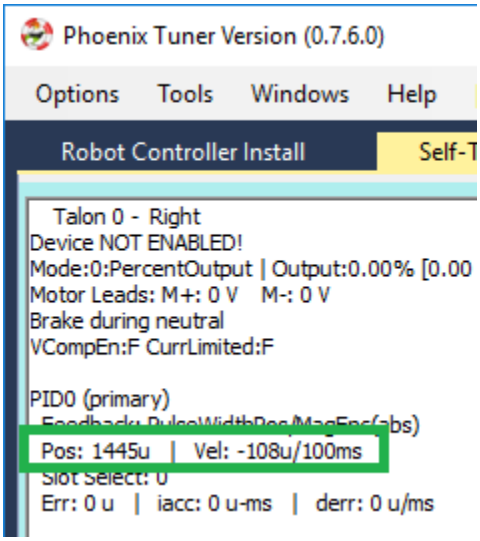
Take another self-test and notice the Selected Sensor (PID0) matches the selection.



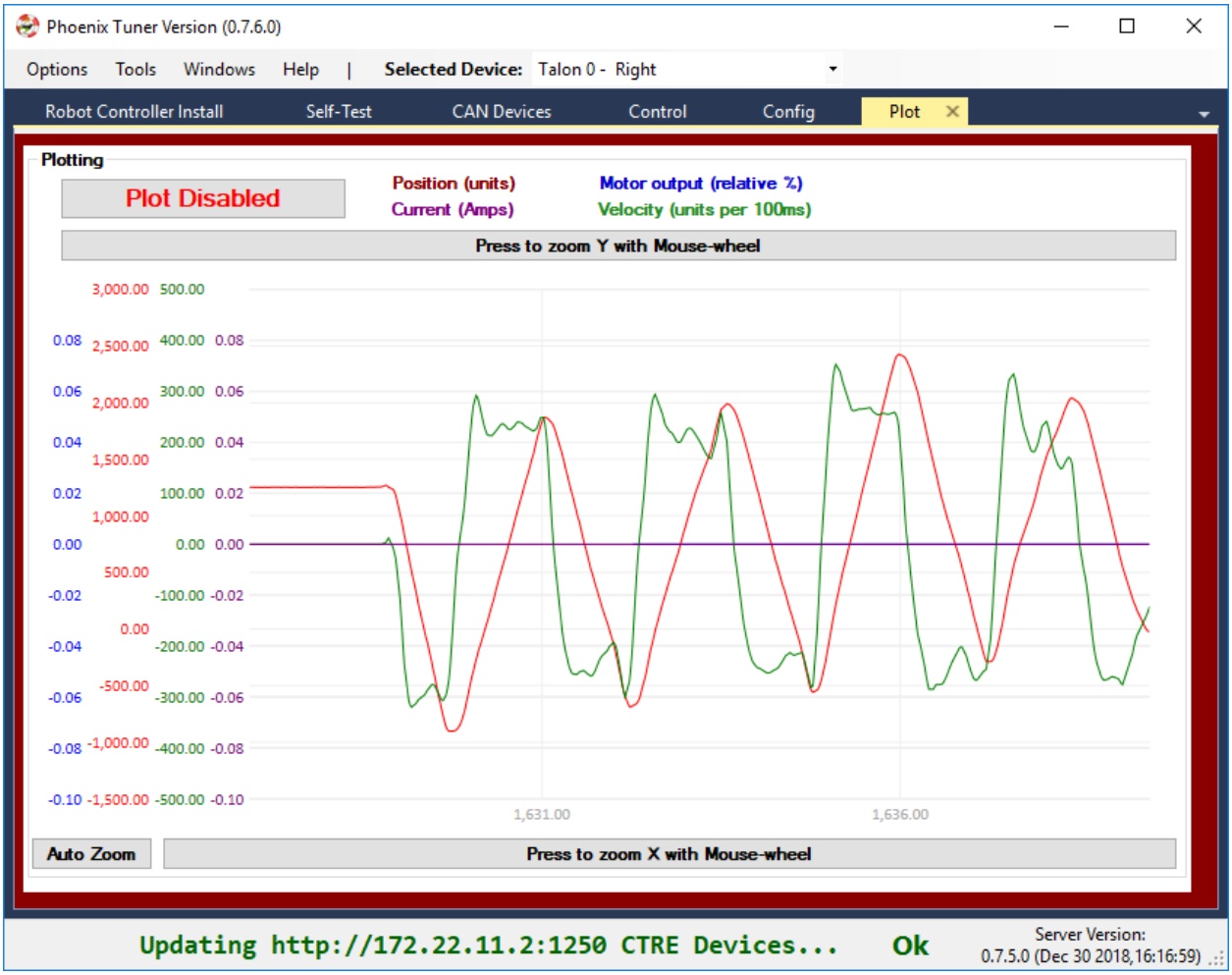
Manually rotate in one direction and take another self-test. Confirm velocity is nonzero.



Now spin the other way and confirm opposite polarity.



Because the sensor is now "selected", turn on the plot and hand rotate sensor back and forth. Disable plot to pause after capturing several seconds.



Checks:

- Focus the velocity and position curves and look for any discontinuities in the plot.
- Shake the sensor harness while hand-turning mechanism.
- This is also a good opportunity to confirm the resolution of the sensor.

Selecting the CTRE Magnetic Encoder

Selecting the Magnetic Encoder for closed-loop / soft-limit features is no different than selecting other sensor feedback devices. Select Quadrature for the faster incremental/relative signal. Select Pulse Width for the slower absolute (within one rotation) signal.

2.16.4 Sensor Check – With Motor Drive

In this step we will attempt to drive motor while monitoring sensor value. Motor controller can be controlled using Control-tab (see previous relevant section) or controlled from robot application via Phoenix API (see previous relevant section).

Sensor Phase

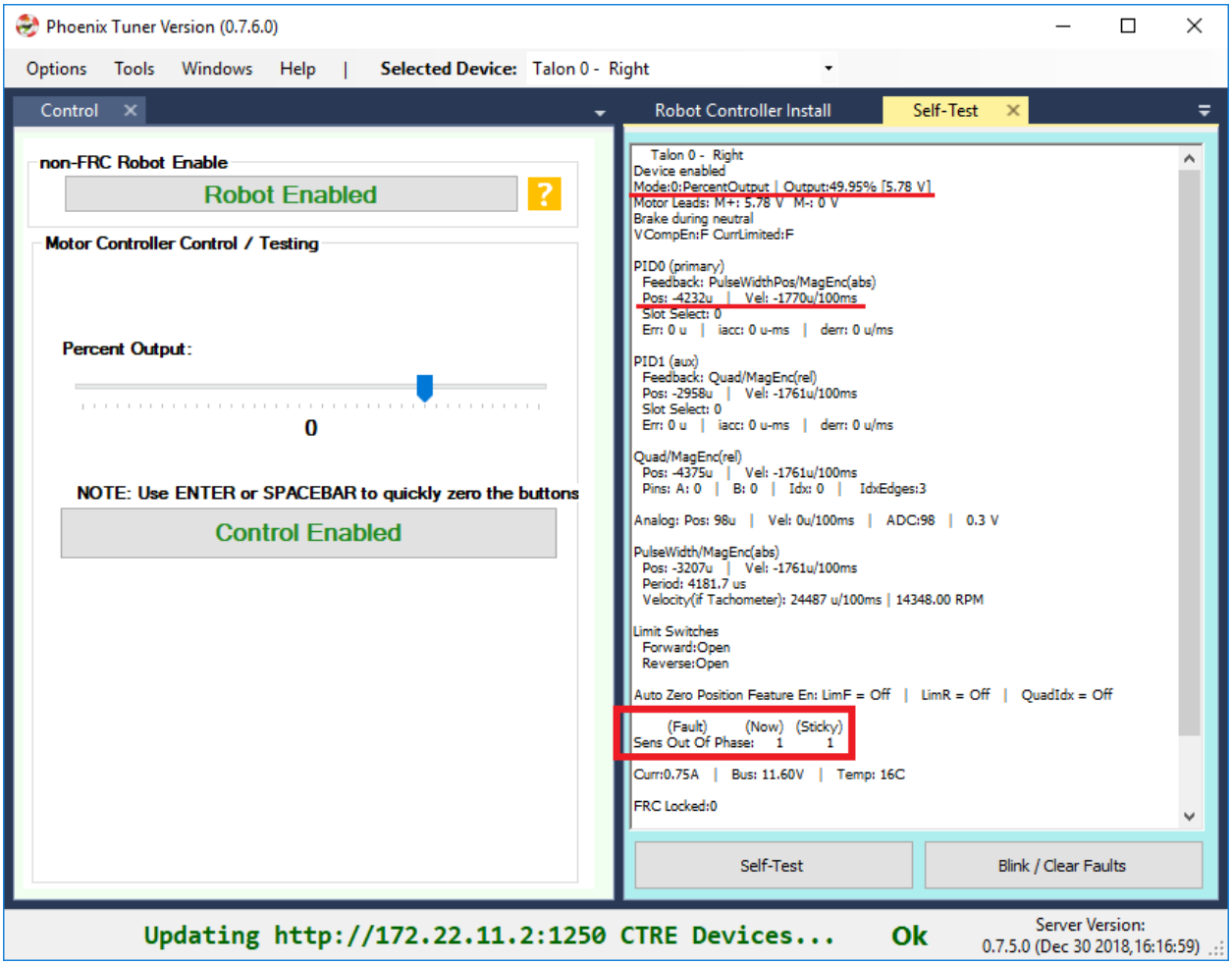
Sensor phase describes the relationship between the motor output direction (positive vs negative) and sensor velocity (positive vs negative). For soft-limits and closed-loop features to function correctly, the sensor measurement and motor output must be "in-phase".

Measure Sensor Phase

Take another measurement using your preferred control method and check the sensor phase using any of the following methods.

Here we sweep the motor output forward and reverse. Notice that sensor velocity (green) and motor output (blue) are out of phase.

In this capture we use the self-test to observe the motor output and selected (PID0) sensor velocity are signed in opposite directions. Additionally the Talon SRX noticed this and reported a live fault of "Sensor Out of Phase".



Note: Talon SRX will check sensor direction versus output direction once motor output and velocity exceeds a minimum threshold.

Adjust Sensor Phase using API

If the sensor is out of phase with the motor drive, you can use any method below to align them:

- **Recommended**: Use setSensorPhase routine/VI to adjust the sensor phase. If already called, toggle the input so that the sensor phase becomes aligned with motor output.
- Exchange/flip the green/white motor leads. **This is generally not recommended** as this makes maintaining motor controller orientation across multiple robots difficult (practice versus competition).

Warning: Do not use setInverted to correct sensor orientation with respect to motor output. setInverted synchronously inverts both signals, ensuring that sensor phase is maintained. **This is a feature** that allows you to choose what direction is considered positive without breaking closed-looping features.

Confirm Sensor Phase using API

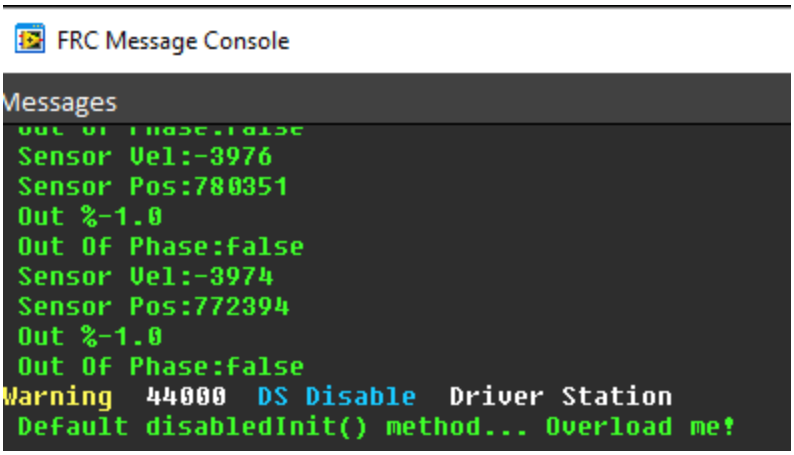
The next test is to control the motor controller using Phoenix API on the robot controller.

This is ultimately how you will leverage the motor controller in competition.

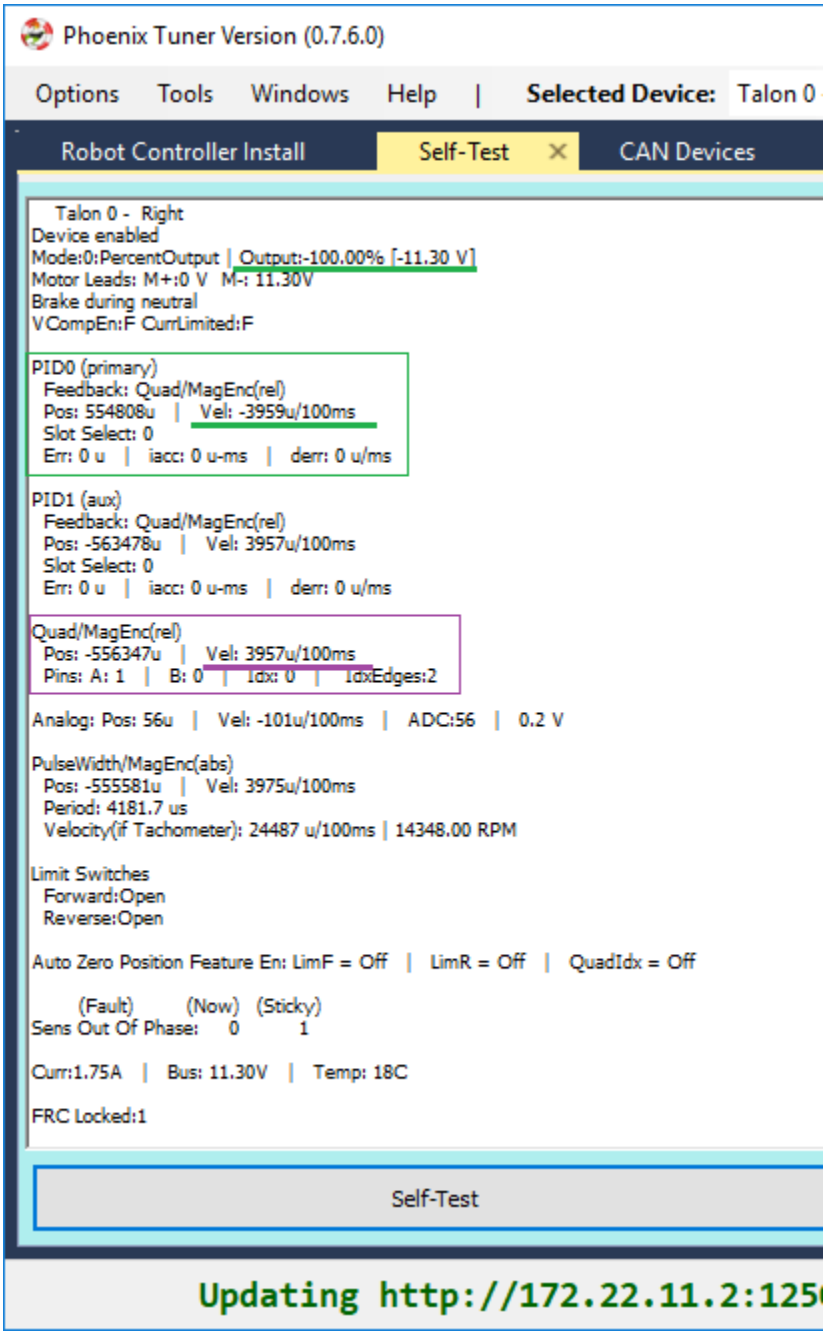
```
package frc.robot;

import com.ctre.phoenix.motorcontrol.*;
import com.ctre.phoenix.motorcontrol.can.*;
import edu.wpi.first.wpilibj.*;

public class Robot extends TimedRobot {
  TalonSRX _talon = new TalonSRX(0); /* make a Talon */
  Joystick _joystick = new Joystick(0); /* make a joystick */
  Faults _faults = new Faults(); /* temp to fill with latest faults */

  @Override
  public void teleopInit() {
    /* factory default values */
    _talon.configFactoryDefault();

    /*
     * choose whatever you want so "positive" values moves mechanism forward,
     * upwards, outward, etc...
     *
     * Note that you can set this to whatever you want, but this will not fix motor
     * output direction vs sensor direction.
     */
    _talon.setInverted(false);

    /*
     * flip value so that motor output and sensor velocity are the same polarity. Do
     * this before closed-looping
     */
    _talon.setSensorPhase(false); // <<<<<< Adjust this
  }

  @Override
  public void teleopPeriodic() {
    double xSpeed = _joystick.getRawAxis(1) * -1; // make forward stick positive

    /* update motor controller */
    _talon.set(ControlMode.PercentOutput, xSpeed);
    /* check our live faults */
    _talon.getFaults(_faults);
    /* hold down btn1 to print stick values */
    if (_joystick.getRawButton(1)) {
      System.out.println("Sensor Vel:" + _talon.getSelectedSensorVelocity());
      System.out.println("Sensor Pos:" + _talon.getSelectedSensorPosition());
      System.out.println("Out %" + _talon.getMotorOutputPercent());
      System.out.println("Out Of Phase:" + _faults.SensorOutOfPhase);
    }
  }
}
```

Confirm sensor velocity is in phase with motor output using any of the methods documented above.

Below is an example screenshot of a successfully phased sensor and motor output. Both are negative (good).



Below is an example screenshot of a successfully phased sensor and motor output. Both are negative (in green).



Note: The natural sensor measurement (purple) under Quad is opposite of the Selected sensor value. This is proof-positive that setSensorPhase(true) was used to adjust the sensor phase to better match the motor voltage direction.

What if the sensor Phase is already correct?

The recommendation is to always call setSensorPhase routine/VI. If the phase is naturally correct, then pass false. The reasons to do this are:

- During competition, you may find the pit-crew / repair-team wired a replacement motor/harness incorrectly and

must resolve this with a "quick software fix".

- During competition, you may find the pit-crew / repair-team wired a replacement sensor/harness incorrectly and must resolve this with a "quick software fix".
- This provides the means of changing the sensor phase to the "wrong value" during hardware-bring up, so you can demonstrate to other team members what an out of phase sensor looks like in your telemetry.

2.16.5 Confirm Sensor Resolution/Velocity

After correcting the sensor phase, the next step is to confirm sensor resolution matches your expectations. This is an important step in sensor validation.

Listed below are the typical sensor resolutions for common sensors. Lookup your sensor type and note the expected resolution. Call this kSensorUnitsPerRotation.

Sensor Resolution

Sensor Type	Units per rotation
Quadrature Encoder : US Digital 1024 CPR	4096 (Talon SRX / CANifer counts every edge)
CTRE Magnetic Encoder (relative/quadrature)	4096
CTRE Magnetic Encoder (absolute/pulse width)	4096
Any pulse width encoded position	4096 represents 100% duty cycle
AndyMark CIMcoder	80 (because 20 pulses => 80 edges)
Analog	1024

Note: Sensor are typically reported in the raw sensor units to ensure all of the available sensor resolution is utilized. However starting in 2020 season, there likely will be scaling options to universally adjust how sensor position is interpreted (for example, in fractional rotations).

Lookup the kMaxRPM of your motor. This will be advertised as the free-speed or max-velocity of your motor.

Determine if your mechanism has a gear-ratio between the motor and your sensor. Typically this is a reduction, meaning that there are several motor rotations per single sensor rotation. Call this kGearRatio.

Calculate the expect peak sensor velocity (sensor units per 100ms) as:

```
(kMaxRPM  / 600) * (kSensorUnitsPerRotation / kGearRatio)
```

Knowing the maximum possible sensor velocity, compare this against the sensor velocity report in any of the following:

- self-test under selected sensor (PID0).
- getSelectedSensorVelocity() API
- Tuner plotter sensor velocity

You will likely find your ideal value is greater than your measured value due to load. In the case of testing a drive train, it is recommend to place robot on a tote/crate so that wheels can spin free.

If your mechanism does not allow for full motor output due to its design, choose a slower duty cycle and scale by the expected velocity.

2.16.6 Setting Sensor Position

Depending on the sensor selected, the user can modify the "Sensor Position". This is particularly useful when using a Quadrature Encoder (or any relative sensor) which needs to be "zeroed" or "home-ed" when the robot is in a known position.

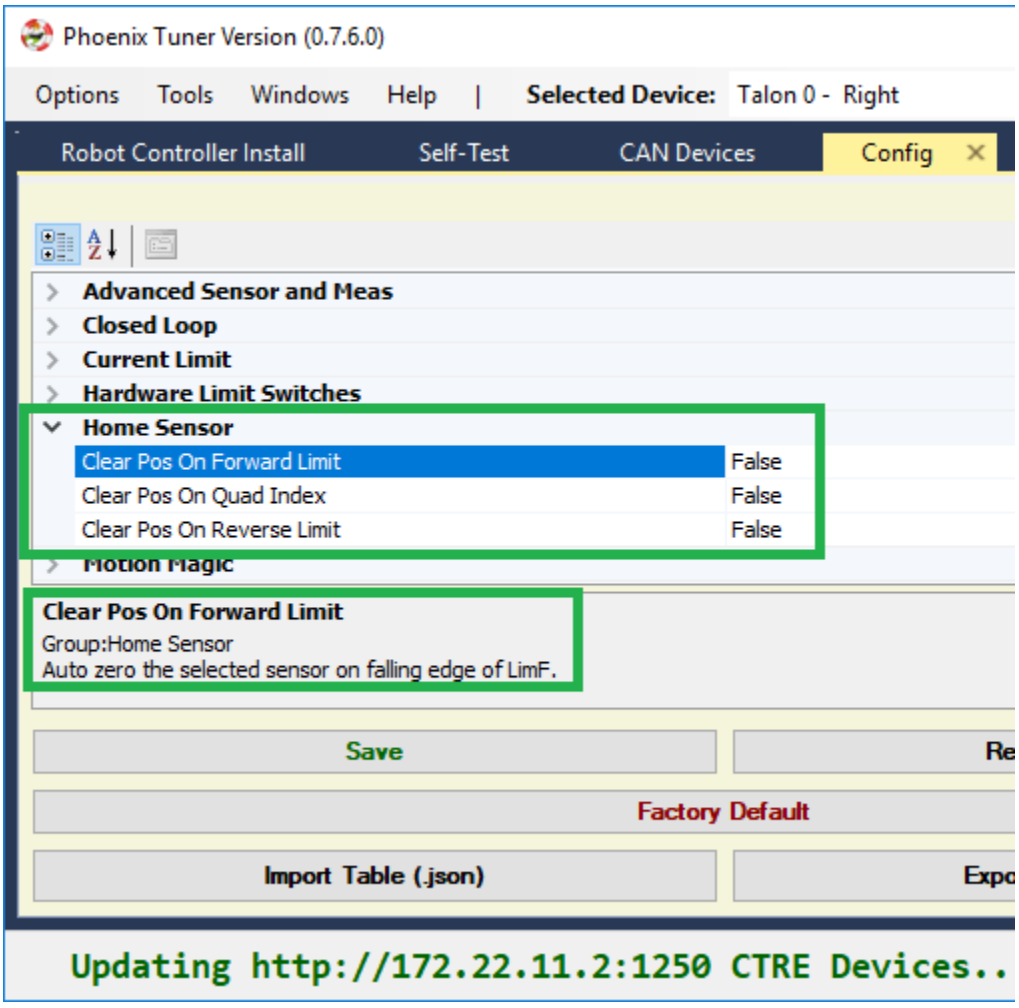
Auto Clear Position using Index Pin Or Limit Switches

In addition to manually changing the sensor position, the Talon SRX supports automatically resetting the Selected Sensor Position to zero whenever a digital edge is detected.

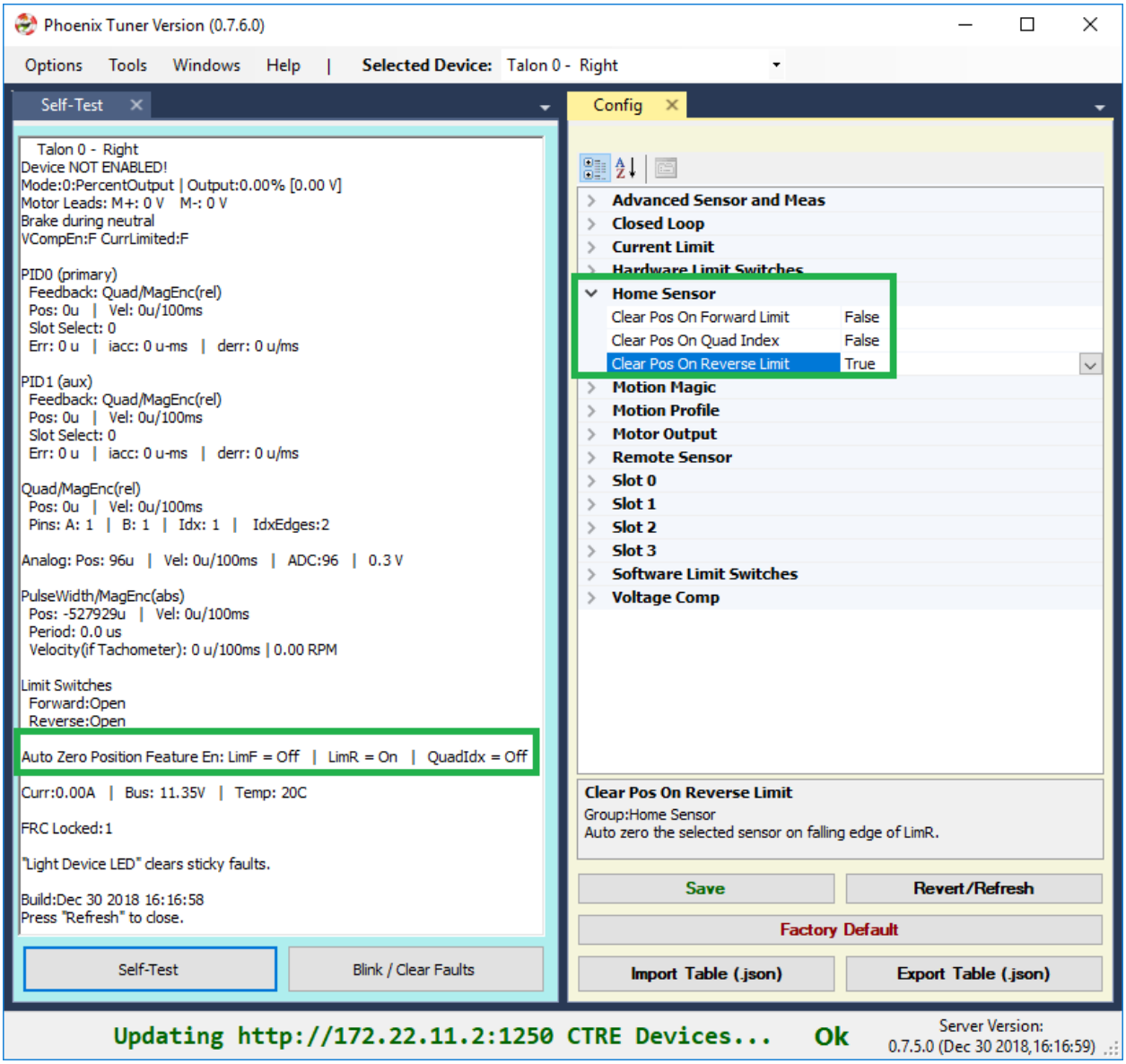
This can be activated via config API or config tab in Tuner.

Clear Pos event can be triggered by:

- Falling edge on Forward Limit (pin 4)
- Falling edge on Reverse Limit (pin 8)
- Rising edge on Quadrature Index (pin 9)



Self-test can also be used to confirm the enabling of auto zero features.



2.16.7 Velocity Measurement Filter

The Talon SRX measures the velocity of all supported sensor types as well as the current position. Every 1ms a velocity sample is measured and inserted into a rolling average.

The velocity sample is measured as the change in position at the time-of-sample versus the position sampled 100ms-prior-to-time-of-sample. The rolling average is sized for 64 samples. Though these settings can be modified, the (100ms, 64 samples) parameters are default.

Changing Velocity Measurement Parameters.

The two configs for the Talon Velocity Measurement are:

- Sample Period (Default 100ms)
- Rolling Average Window Size (Default 64 samples).

Each can be modified through programming API, and through Tuner.

Note: When the sample period is reduced, the units of the native velocity measurement is still change-in-position-per-100ms. In other words, the measurement is up-scaled to normalize the units. Additionally, a velocity sample is always inserted every 1ms regardless of setting selection.

Note: The Velocity Measurement Sample Period is selected from a fixed list of pre-supported sampling periods [1, 5, 10, 20, 25, 50, 100(default)] milliseconds.

Note: The Velocity Measurement Rolling Average Window is selected from a fixed list of pre-supported sample counts: [1, 2, 4, 8, 16, 32, 64(default)]. If an alternative value is passed into the API, the firmware will truncate to the nearest supported value.

Recommended Procedure

The general recommended procedure is to first set these two parameters to the minimal value of '1' (Measure change in position per 1ms, and no rolling average). Then plot the measured velocity while manually driving the Talon SRX(s) with a joystick/gamepad. Sweep the motor output to cover the expected range that the sensor will be expected to cover.

Unless the sensor velocity is considerably fast (hundreds of sensor units per sampling period) the measurement will be very coarse (visual stair-stepping as the motor output is increased). Increase the sampling period until the measured velocity is sufficiently granular.

At this point the sensor velocity will have minimal stair-stepping (good) but will be quite noisy. Increase the rolling average window until the velocity plot is sufficiently smooth, but still responsive enough to meet the timing requirements of the mechanism.

2.17 WPI/NI Software Integration

The stock software frameworks in the FRC control system has several features used by teams. To leverage these features, the C++ /Java Phoenix API has two additional classes:

- WPI_TalonSRX
- WPI_VictorSPX

These are wrappers for the Talon SRX and Victor SPX, that provide:

- LiveWindow support
- Motor Safety features
- Compatibility with DriveTrain classes

2.17.1 C++ / Java Drive Train classes

To leverage the Drive Train classes in WPILib: - Create your motor controller objects like normal. - Create the Drive object like normal. - Call setRightSideInverted(false) so that when moving forward, positive output is applied to left and right. - Adjust setInverted() so that motors cause robot to drive straight forward when stick is forward.

```
package frc.robot;

import com.ctre.phoenix.motorcontrol.can.*;
import edu.wpi.first.wpilibj.*;
import edu.wpi.first.wpilibj.drive.*;

public class Robot extends TimedRobot {
  WPI_TalonSRX _talonL = new WPI_TalonSRX(1);
  WPI_TalonSRX _talonR = new WPI_TalonSRX(0);
  DifferentialDrive _drive = new DifferentialDrive(_talonL, _talonR);
  Joystick _joystick = new Joystick(0);

  @Override
  public void teleopInit() {
    /* factory default values */
    _talonL.configFactoryDefault();
    _talonR.configFactoryDefault();

    /* flip values so robot moves forward when stick-forward/LEDs-green */
    _talonL.setInverted(false); // <<<<<< Adjust this
    _talonR.setInverted(true); // <<<<<< Adjust this

    /*
     * WPI drivetrain classes defaultly assume left and right are opposite. call
     * this so we can apply + to both sides when moving forward. DO NOT CHANGE
     */
    _drive.setRightSideInverted(false);
  }

  @Override
  public void teleopPeriodic() {
    double xSpeed = _joystick.getRawAxis(1) * -1; // make forward stick positive
    double zRotation = _joystick.getRawAxis(2); // WPI Drivetrain uses positive=>␣
↪right

    _drive.arcadeDrive(xSpeed, zRotation);

    /* hold down btn1 to print stick values */
    if (_joystick.getRawButton(1)) {
      System.out.println("xSpeed:" + xSpeed + "    zRotation:" + zRotation);
    }
  }
}
```

Tip: It is advantageous to setup Talon / Victors in this fashion so that positive (green) represents forward motion. This makes integrating the other control modes into your drive train simpler.

2.17.2 C++ / Java Motor Safety Feature

The Java class WPI_TalonSRX and WPI_VictorSPX both implement the motor safety interface.

The C++ class WPI_TalonSRX and WPI_VictorSPX does not inherent the motor safety abstract class, but they do implement the exact same routines. This means the same routines can be called on the Phoenix WPI objects.

2.18 Motor Controller Closed Loop

2.18.1 Primer on Closed-loop

Talon SRX and Victor SPX supports are variety of closed-loop modes including position closed-loop, velocity closed-loop, Motion Profiling, and Motion Magic. Talon SRX additionally supports current closed-loop.

Note: All closed-loop modes update every 1ms (1000Hz) unless configured otherwise.

Tip: While tuning the closed-loop, use the Tuner configuration tab to quickly change the gains "on the fly". Once the PID is stable, set the gain values in code so that Talons can be swapped/replaced easily.

Regardless of which closed loop control mode, the following statements apply:

- Current limit and voltage compensation selections are honored (just like in open-loop PercentOutput mode)
- "Ramping" can be configured using configClosedloopRamp (routine or VI)
- All other open-loop features are honored during closed loop (neutral mode, peaks, nominal outputs, etc).
- Closed Loop controller will pull closed-loop gain/setting information from a selected slot. There are four slots to choose from (for gain-scheduling).
- PIDF controller takes in target and sensor position measurements in "raw" sensor units. This means a CTRE Mag Encoder will count 4096 units per rotation.
- PIDF controller takes in target and sensor velocity measurements in "raw" sensor units per 100ms.
- PIDF controller calculates the motor output such that, 1023 is interpreted as "full". This means a closed loop error of 341 (sensor units) X kP of 3.0 will produce full motor output (1023).

Note: A target goal of 2020 is to normalize the PID controller to interpret sensor using normalized units, and adjusting the PID output such that '1' is interpreted as full. This will likely be a "back-breaking" change. This also means 2019 will likely be the last time you see "1023" used anywhere.

Below are descriptions for the various control modes.

Position Closed-Loop Control Mode

The Position Closed-Loop control mode can be used to abruptly servo to and maintain a target position.

A simple strategy for setting up a closed loop is to zero out all Closed-Loop Control Parameters and start with the Proportional Gain.

For example if you want your mechanism to drive 50% throttle when the error is 4096 (one rotation when using CTRE Mag Encoder), then the calculated Proportional Gain would be (0.50 X 1023) / 4096 = ~0.125.

Tune this until the sensed value is close to the target under typical load. Many prefer to simply double the P-gain until oscillations occur, then reduce accordingly.

If the mechanism accelerates too abruptly, Derivative Gain can be used to smooth the motion. Typically start with 10x to 100x of your current Proportional Gain. If application requires a controlled (smooth) deceleration towards the target, we strongly recommend motion-magic.

If the mechanism never quite reaches the target and increasing Integral Gain is viable, start with 1/100th of the Proportional Gain.

Current Closed-Loop Control Mode

The Talon's Closed-Loop logic can be used to approach a target current-draw. Target and sampled current is passed into the PIDF controller in milliamperes. However the robot API expresses the target current in amperes.

Note: Current Control Mode is separate from Current Limit.

Tip: A simple strategy for setting up a current-draw closed loop is to zero out all Closed-Loop Control Parameters and start with the Feed-Forward Gain. Tune this until the current-draw is close to the target under typical load. Then start increasing P gain so that the closed-loop will make up for the remaining error. If necessary, reduce Feed-Forward gain and increase P Gain so that the closed-loop will react more strongly to the ClosedLoopError.

Warning: This feature is not available on Victor SPX.

Velocity Closed-Loop Control Mode

The Talon's Closed-Loop logic can be used to maintain a target velocity. Target and sampled velocity is passed into the equation in native sensor units per 100ms.

Tip: A simple strategy for setting up a closed loop is to zero out all Closed-Loop Control Parameters and start with the Feed-Forward Gain. Tune this until the sensed value is close to the target under typical load. Then start increasing P gain so that the closed-loop will make up for the remaining error. If necessary, reduce Feed-Forward gain and increase P Gain so that the closed-loop will react more strongly to the ClosedLoopError.

Tip: Velocity Closed-Loop tuning is similar to Current Closed-Loop tuning in their use of feed-forward. Begin by measuring the sensor velocity while driving the Talon at a large throttle.

Motion Magic Control Mode

Motion Magic is a control mode for Talon SRX that provides the benefits of Motion Profiling without needing to generate motion profile trajectory points. When using Motion Magic, Talon SRX / Victor SPX will move to a set target position using a Trapezoidal Motion Profile, while honoring the user specified acceleration and maximum velocity (cruise velocity).

The benefits of this control mode over "simple" PID position closed-looping are:

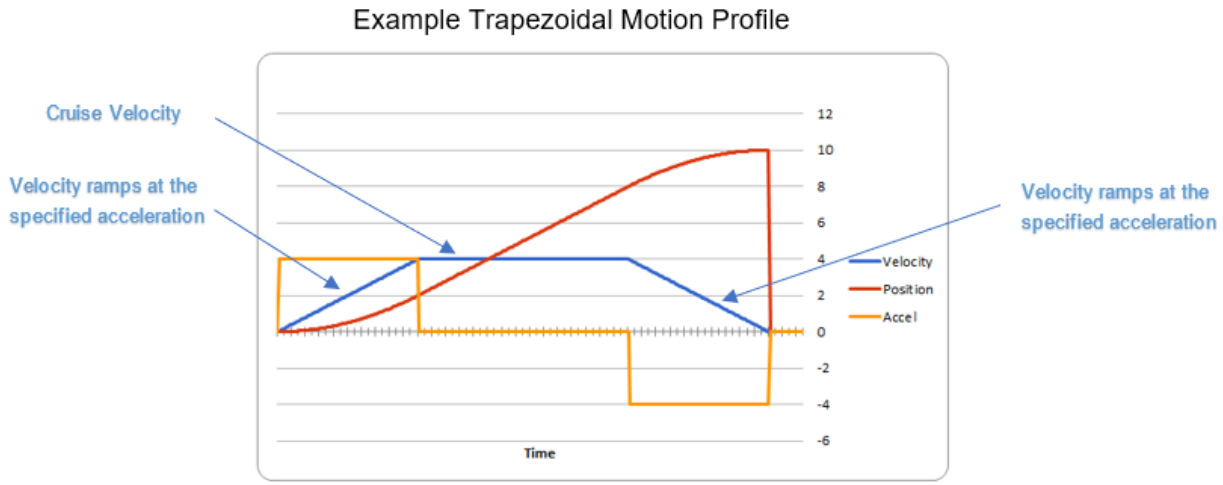
- Control of the mechanism throughout the entire motion (as opposed to racing to the end target position).
- Control of the mechanism's inertia to ensure smooth transitions between set points.
- Improved repeatability despite changes in battery voltage.
- Improved repeatability despite changes in motor load.

After gain/settings are determined, the robot-application only needs to periodically set the target position.

There is no general requirement to "wait for the profile to finish", however the robot application can poll the sensor position and determine when the motion is finished if need be.

A Trapezoidal Motion Profile generally ramps the output velocity at a specified acceleration until cruise velocity is reached. This cruise velocity is then maintained until the system needs to deaccelerate to reach the target position and stop motion. Talon determines when these critical points occur on-the-fly.

Note: If the remaining sensor distance to travel is small, the velocity may not reach cruise velocity as this would overshoot the target position. This is often referred to as a "triangle profile".



Motion Magic utilizes the same PIDF parameters as Motion Profiling.

Two additional parameters need to be set in the Talon SRX– Acceleration and Cruise Velocity.

The Acceleration parameter controls acceleration and deacceleration rates during the beginning and end of the trapezoidal motion. The Cruise Velocity parameter controls the cruising velocity of the motion.

Motion Profile Control Mode

Talon SRX and Victor SPX support other closed-loop modes that allow a "Robot Controller" to specify/select a target value to meet. The target can simply be the percent output motor drive, or a target current-draw. When used with a feedback sensor, the robot controller may also simply set the target position, or velocity to servo/maintain.

However, for advanced motion profiling, the Talon SRX / Victor SPX additionally supports a mode whereby the robot controller can *stream* a sequence of trajectory points to express an *entire motion profile*.

Each trajectory point holds the desired velocity, position, arbitrary feedforward, and time duration to honor said point until moving on to the next point. The point also holds targets for both the primary and auxiliary PID controller, allowing for differential control (drivetrain, differential mechanisms).

Alternatively, the trajectory points can be streamed into the motor controller *as the motor controller is executing the profile*, so long as the robot controller sends the trajectory points faster than the Talon consumes them. This also means that there is no practical limit to how long a profile can be.

Tip: Starting in 2019, the Talon and Victor will linearly interpolate targets between two buffer points every 1ms. This means you can send points with larger time durations, but still have a smooth continuous motion. This features default

on.

What is the benefit? Leveraging the Motion Profile Control Mode in the Talon SRX has the following benefits:

- Direct control of the mechanism throughout the entire motion (as opposed to a single PID closed-loop which directly servos to the end target position).
- Accurate scheduling of the trajectory points that is not affected by the performance of the primary robot controller.
- Improved repeatability despite changes in battery voltage.
- Improved repeatability despite changes in motor load.
- Provides a method to synchronously gain-schedule.

Additionally, this mode could be used to schedule several position servos in advance with precise time outs. For example, one could map out a collection of positions and timeouts, then stream the array to the Talon SRX to execute them.

2.18.2 Sensor Preparation

Before invoking any of the closed loop modes, the following must be done:

- Complete the sensor bring up procedure to ensure sensor phase and general health.
- Record the maximum sensor velocity (position units per 100ms) at 100% motor output.
- Calculating kF gain if applicable (Velocity Closed Loop, Motion Profile, Motion Magic).

The first two are covered in section "Confirm Sensor Resolution/Velocity". Calculating feed forward is done in the next section.

2.18.3 Calculating Feed Forward gain(kF)

A typical strategy for estimating the necessary motor output is to take the target velocity and multiplying by a tuned/calculated scalar. More advanced feed forward methods (gravity compensation, velocity and acceleration feed forwards, static offsets, etc) can be done with the arbitrary feed forward features.

Do I need to calculate kF?

If using any of the control modes, we recommend calculating the kF:

- Velocity Closed Loop: kF is multiplied by target velocity and added to output.
- Current (Draw) Closed Loop: kF is multiplied by the target current-draw and added to output.
- MotionMagic/ MotionProfile / MotionProfileArc: kF is multiplied by the runtime-calculated target and added to output.

Note: Most control modes also provide an "arbitrary feed forward" term that user can provide during the runtime. This allows for complete custom implementation of feedforward beyond the simple kF X target. Implementing kS, kV,kA terms can be done this way.

Note: When using position closed loop, it is generally desired to use a kF of '0'. During this mode target position is multiplied by kF and added to motor output. If providing a feedforward is necessary, we recommend using the arbitrary feed forward term (4 param Set) to better implement this.

How to calculate kF

Using Tuner (Self-Test or Plotter), we've measured a peak velocity of **9326** native units per 100ms at 100% output. This can also be retrieved using getSelectedSensorVelocity (routine or VI).

Now let's calculate a Feed-forward gain so that 100% motor output is calculated when the requested speed is **9326** native units per 100ms.

F-gain = (100% X 1023) / **9326** F-gain = 0.1097

Let's check our math, if the target speed is **9326** native units per 100ms, Closed-loop output will be (0.1097 X **9326**) => 1023 (full forward).

..note the output of the PIDF engine in Talon/Victor uses 1023 as the "full output. However the 2020 software release will likely normalize this so that a value of '1' yields "full output." This is

2.18.4 Motion Magic / Position / Velocity / Current Closed Loop Closed Loop

Closed-looping the position/velocity value of a sensor is explained in this section. This section also applies to the current (draw) closed loop mode.

Relevant source examples can be found at:

- https://github.com/CrossTheRoadElec/Phoenix-Examples-Languages
- https://github.com/CrossTheRoadElec/Phoenix-Examples-LabVIEW

The general steps are:

- Selecting the sensor type (see previous Bring-Up sections)
- Confirm motor and sensor health (see previous Bring-Up section on sensor)
- Confirm sensor phase (see previous Bring-Up sections)
- Collect max sensor velocity information (see calculating kF section)
- Bring up plotting interface so you can visually see sensor position and motor output. This can be done via Tuner Plotter, or through LabVIEW/SmartDash/API plotting.
- Configure gains and closed-loop centric configs.

Note: if you are using current closed-loop, than a sensor is not necessary

Note: Current closed loop is not available on Victor SPX, it is only available on Talon SRX.

Once these previous checks are done, continue down to the gain instructions.

Note: This assumes all previous steps have been followed correctly.

1. Checkout the relevant example from CTREs GitHub.
2. Set all of your gains to zero. Use either API or Phoenix Tuner.
3. If not using Position-Closed loop mode, set the kF to your calculated value.
4. If using Motion Magic, set your initial cruise velocity and acceleration.

The recommended way to do this is to take your max sensor velocity (previous section).

Suppose your kMaxSensorVelocity is **9326** units per 100ms. A reasonable initial cruise velocity may be half of this velocity, which is **4663**.

Config **4663** to be the cruiseVelocity via configMotionCruiseVelocity routine/VI.

Next lets set the acceleration, which is in velocity units per second (where velocity units = change in sensor per 100ms). This means that if we choose the same value of **4663** for our acceleration, than Motion Magic will ensure it takes one full second to reach peak cruise velocity.

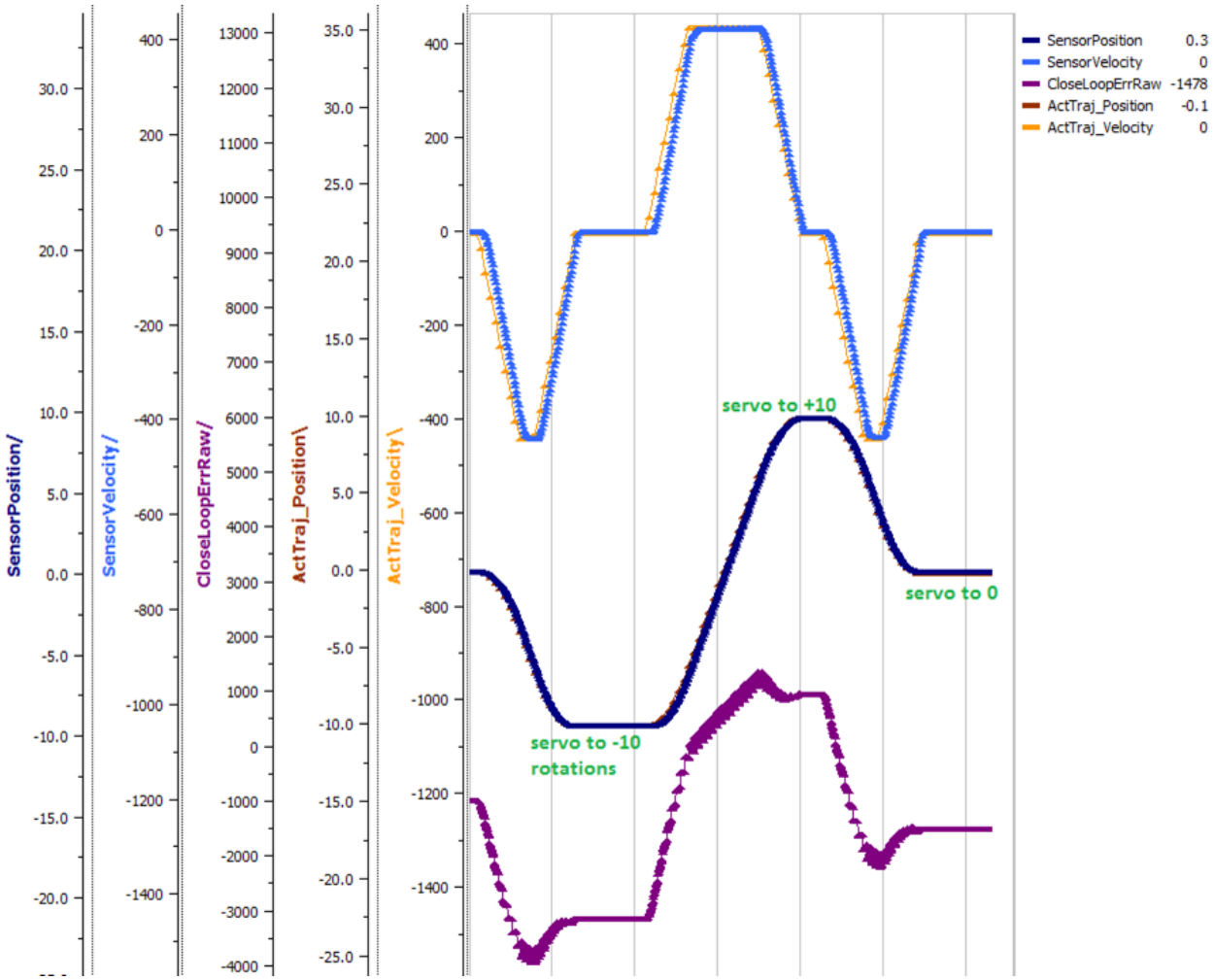
In short set the acceleration to be the same **4663** value via configMotionAcceleration routine/VI.

Later you can increase these values based on the application requirements.

5. Deploy the application and use the joystick to adjust your target. Normally this requires holding down a button on the gamepad (to enter closed loop mode).

Plot the sensor-position to access how well it is tracking.

In this example the mechanism is the left-side of a robot's drivetrain. The robot is elevated such that the wheels spin free. In the capture below we see the sensor position/velocity (blue) and the Active Trajectory position/velocity (brown/orange). At the end of the movement the closed-loop error (which is in raw units) is sitting at ~1400.units. Given the resolution of the sensor this is approximately 0.34 rotations (4096 units per rotation). Another note is that when the movement is finished, you can freely back-drive the mechanism without motor-response (because PID gains are zero).



Dialing kP

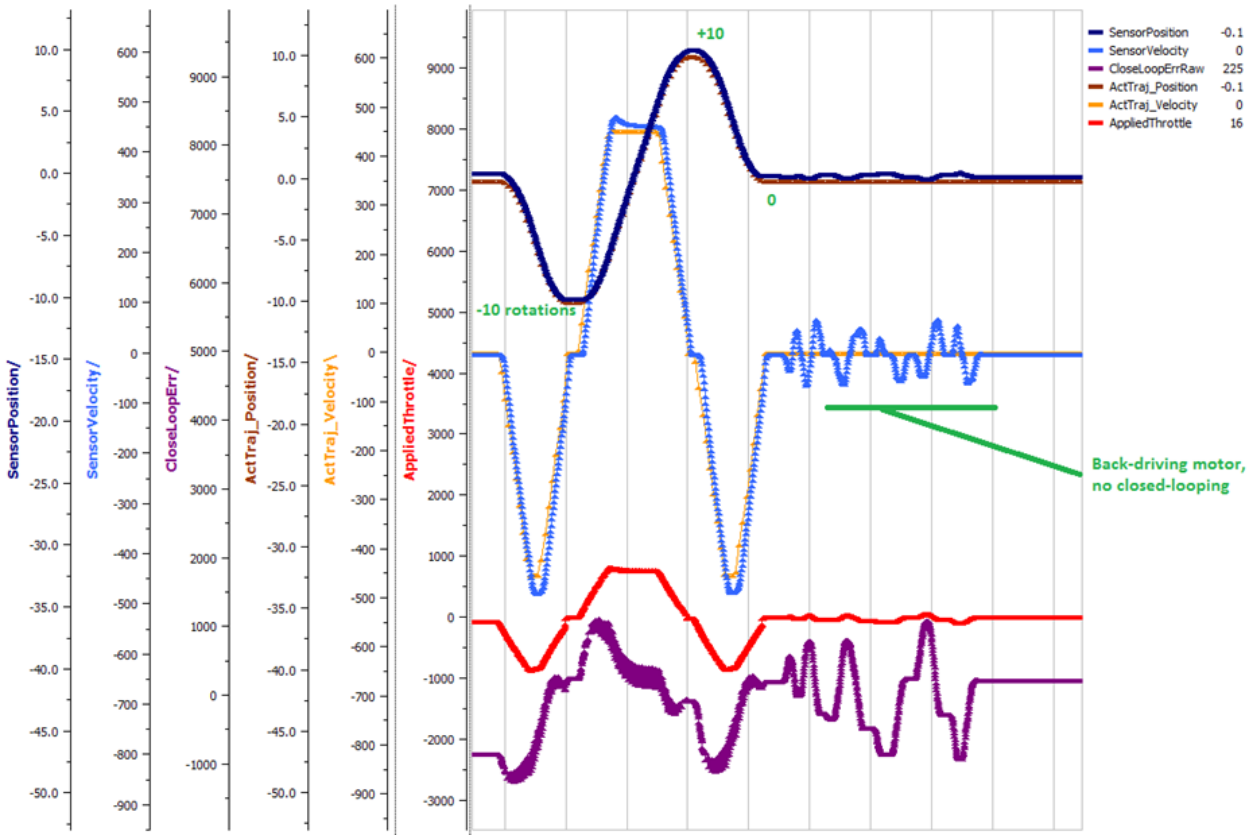
Next we will add in P-gain so that the closed-loop can react to error. In the previous section, after running the mechanism with just F-gain, the servo appears to settle with an error or ~1400.

Given an error of (~1400.), suppose we want to respond with another 10% of throttle. Then our starting kP would be. . . .

(10% X 1023) / (1400) = 0.0731 Now let's check our math, if the Talon SRX sees an error of 1400 the P-term will be 1400 X 0.0731= 102 (which is about 10% of 1023) kP = 0.0731

> Apply the P -gain programmatically using your preferred method. Now retest to see how well the closed-loop responds to varying loads.

Retest the maneuver by holding button 1 and sweeping the gamepad stick. At the end of this capture, the wheels were hand-spun to demonstrate how aggressive the position servo responds. Because the wheel still back-drives considerably before motor holds position, the P-gain still needs to be increased.

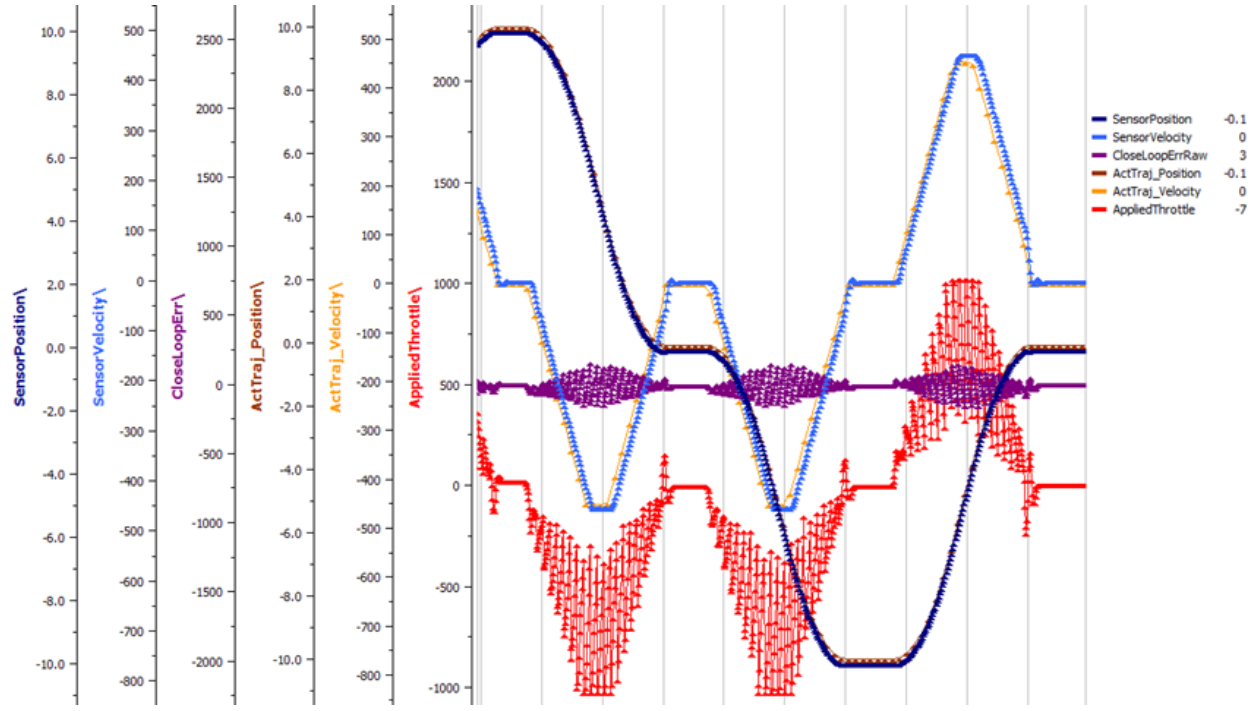


Double the P-gain until the system oscillates (by a small amount) or until the system responds adequately.

After a few rounds the P gain is at 0.6.

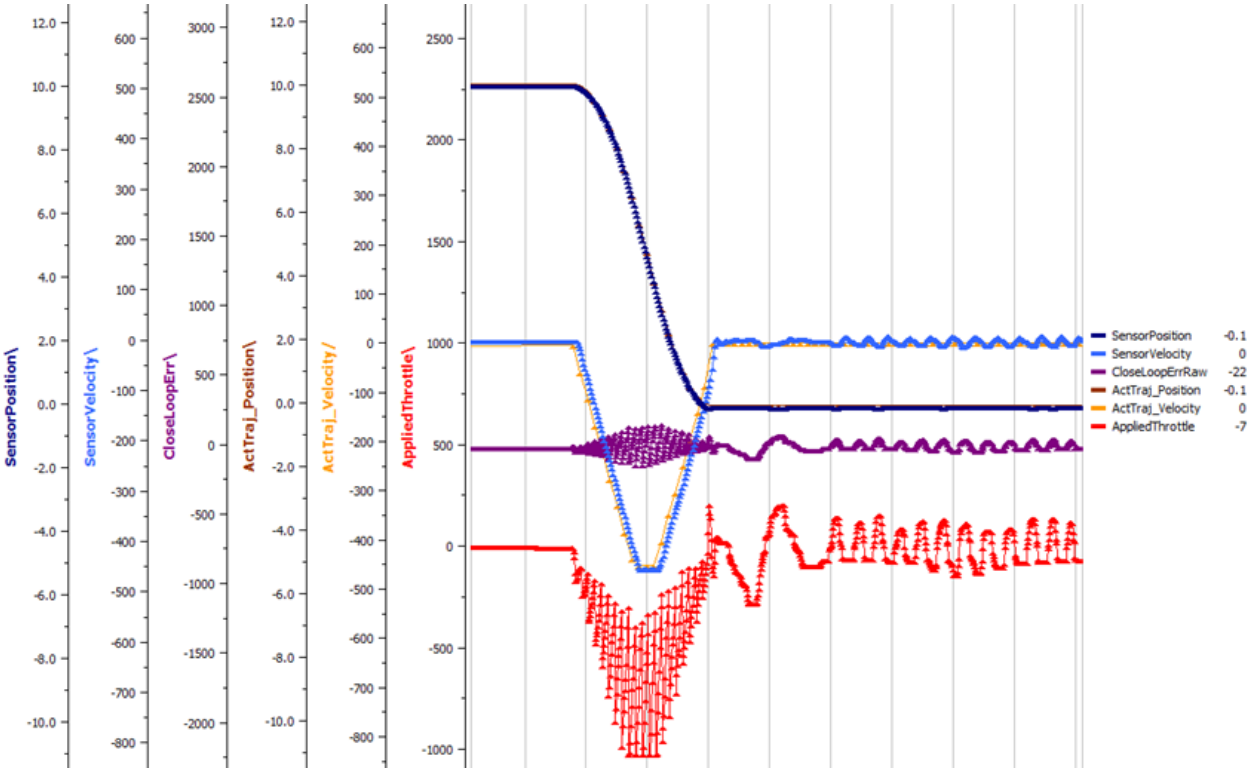
Scope captures below show the sensor position and target position follows visually, but back-driving the motor still shows a minimal motor response.

After several rounds, we've landed on a P gain value of 3. The mechanism overshoots a bit at the end of the maneuver. Additionally, back-driving the wheel is very difficult as the motor-response is immediate (good).



Once settles, the motor is back-driven to assess how firm the motor holds position.

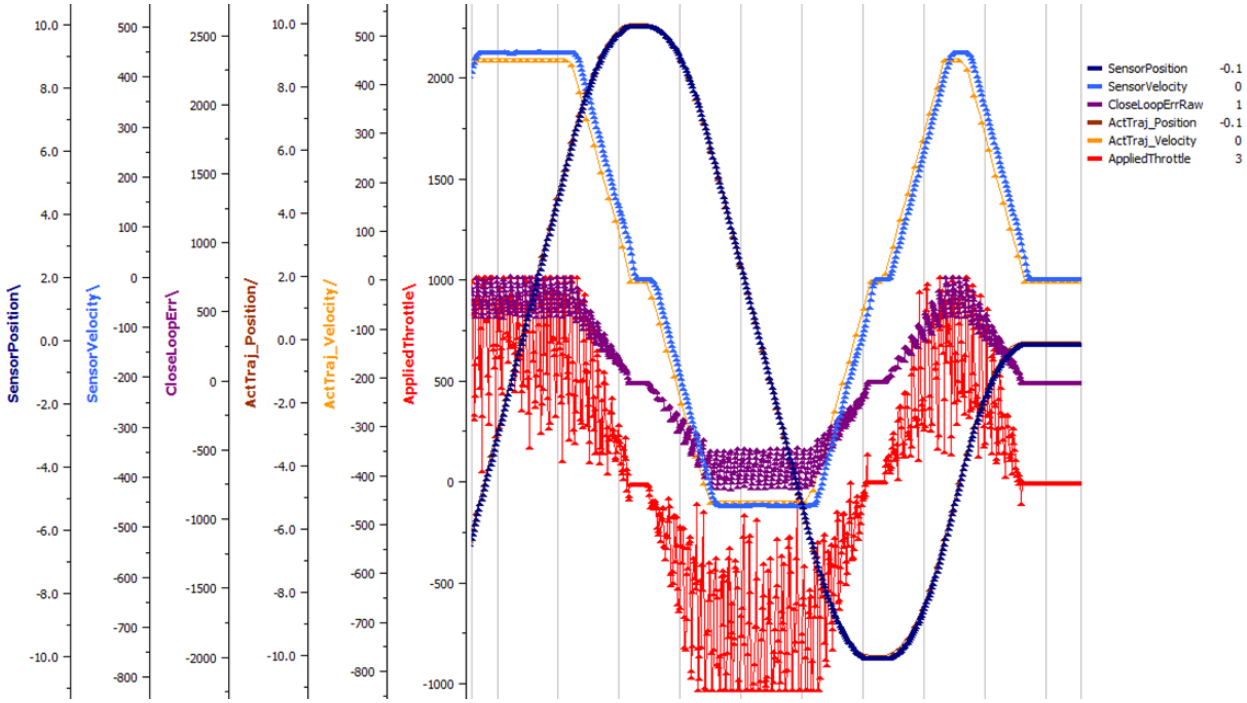
The wheel is held by the motor firmly.



Dialing kD

To resolve the overshoot at the end of the maneuver, D-gain is added. D-gain can start typically at 10 X P-gain.

With this change the visual overshoot of the wheel is gone. The plots also reveal reduced overshoot at the end of the maneuver.



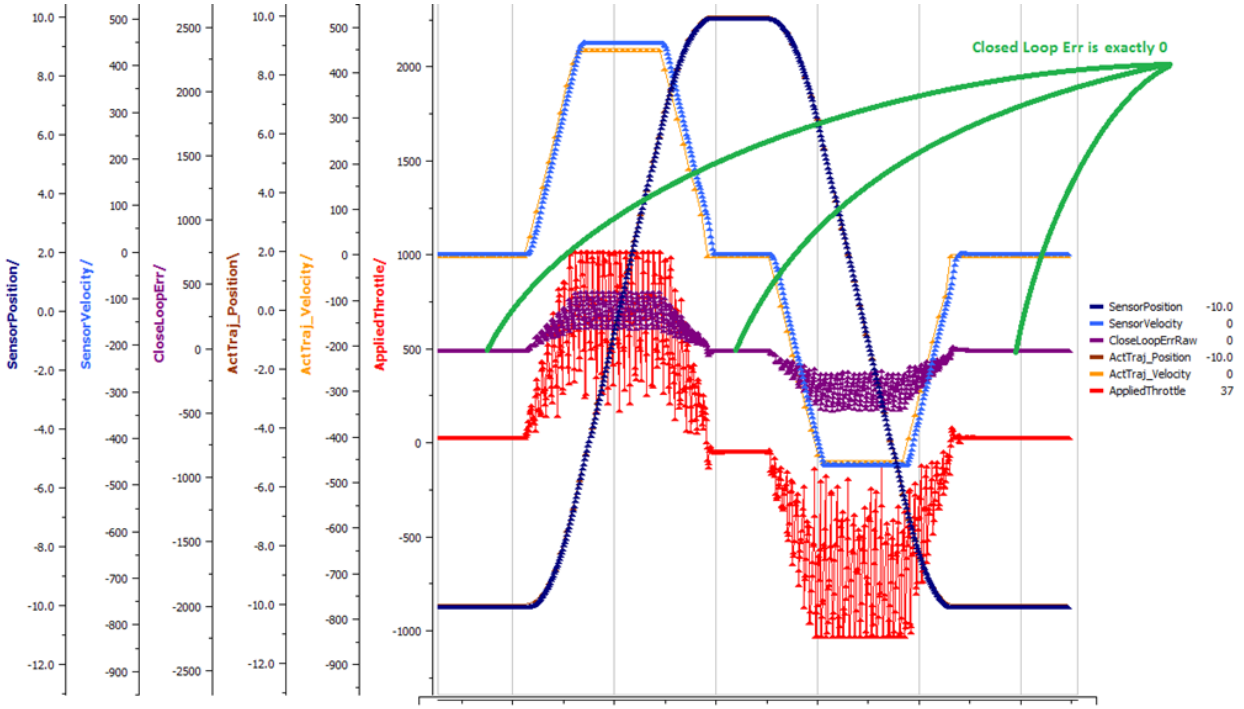
Dialing kI

Typically, the final step is to confirm the sensor settles very close to the target position. If the final closed-loop error is not quite close enough to zero, consider adding I-gain and I-zone to ensure the Closed-Loop Error ultimately lands at zero (or close enough).

In testing the closed-loop error settles around 20 units, so we'll set the Izone to 50 units (large enough to cover the typical error), and start the I-gain at something small (0.001).

Keep doubling I-gain until the error reliably settles to zero.

With some tweaking, we find an I-gain that ensures maneuver settles with an error of 0.



If using Motion Magic, the acceleration and cruise-velocity can be modified to hasten/dampen the maneuver as the application requires.

2.18.5 Closed-Loop Configurations

The remaining closed-loop centric configs are listed below.

General Closed-Loop Configs

Name	Description
PID 0 Primary Feedback Sensor	Selects the sensor source for PID0 closed loop, soft limits, and value reporting for the SelectedSensor API.
PID 0 Primary Sensor Coefficient	Scalar (0,1] to multiply selected sensor value before using. Note this will reduce resolution of the closed-loop.
PID 1 Aux Feedback Sensor	Select the sensor to use for Aux PID[1].
PID 1 Aux Sensor Coefficient	Scalar (0,1] to multiply selected sensor value before using. Note that this will reduce the resolution of the closed-loop.
PID 1 Polarity	False: motor output = PID[0] + PID[1], follower = PID[0] - PID[1]. True : motor output = PID[0] - PID[1], follower = PID[0] + PID[1].
Closed Loop Ramp	This only occurs if follower is an auxiliary type. How much ramping to apply in seconds from neutral-to-full. A value of 0.100 means 100ms from neutral to full output. Set to 0 to disable. Max value is 10 seconds.

Closed-Loop configs per slot (four slots available)

Name	Description
kF	Feed Fwd gain for Closed loop. See documentation for calculation details. If using velocity, motion magic, or motion profile, use (1023 * duty-cycle / sensor-velocity-sensor-units-per-100ms)
kP	Proportional gain for closed loop. This is multiplied by closed loop error in sensor units. Note the closed loop output interprets a final value of 1023 as full output. So use a gain of '0.25' to get full output if err is 4096u (Mag Encoder 1 rotation)
kI	Integral gain for closed loop. This is multiplied by closed loop error in sensor units every PID Loop. Note the closed loop output interprets a final value of 1023 as full output. So use a gain of '0.00025' to get full output if err is 4096u (Mag Encoder 1 rotation) after 1000 loops
kD	Derivative gain for closed loop. This is multiplied by derivative error (sensor units per PID loop). Note the closed loop output interprets a final value of 1023 as full output. So use a gain of '250' to get full output if derr is 4096u per (Mag Encoder 1 rotation) per 1000 loops (typ 1 sec)
Loop Period Ms	Number of milliseconds per PID loop. Typically, this is 1ms.
Allowable Error	If the closed loop error is within this threshold, the motor output will be neutral. Set to 0 to disable. Value is in sensor units.
I Zone	Integral Zone can be used to auto clear the integral accumulator if the sensor pos is too far from the target. This prevent unstable oscillation if the kI is too large. Value is in sensor units.
Max Integral Accum	Cap on the integral accumulator in sensor units. Note accumulator is multiplied by kI AFTER this cap takes effect.
Peak Output	Absolute max motor output during closed-loop control modes only. A value of '1' represents full output in both directions.

Motion Magic Closed-Loop Configs

Name	Description
Acceleration	Motion Magic target acceleration in (sensor units per 100ms) per second.
Cruise Velocity	Motion Magic maximum target velocity in sensor units per 100ms.

Motion Profile Configs

Name	Description
Base Trajectory Period	Base value (ms) ADDED to every buffered trajectory point. Note that each trajectory point has an individual duration (0-127ms). This can be used to uniformly delay every point.
Trajectory Interpolation Enable	Set to true so Motion Profile Executor to linearize the target position and velocity every 1ms. Set to false to match 2018 season behavior (no linearization). This feature allows sending less points over time and still having resolute control Default is set to true.

2.19 Faults

Faults are stored in two fashions. There are "live" faults that are reported in real-time, and "sticky" faults which assert persistently and stay asserted until they are manually cleared (like trouble codes in a vehicle).

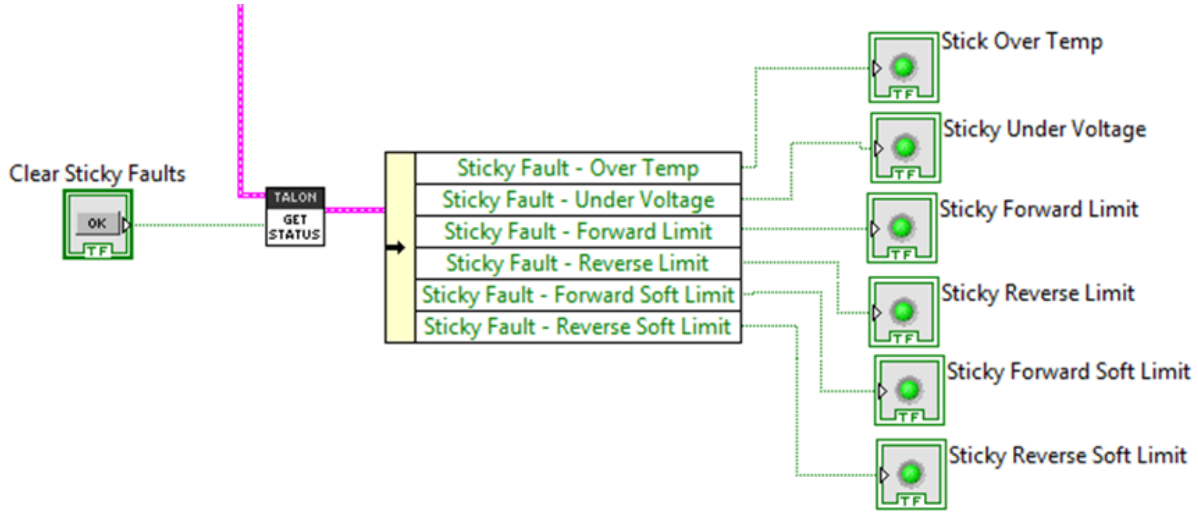
Note: Sticky Faults can be cleared in Tuner and via API.

Note: Faults and Sticky Faults can be polled using Tuner-Self-Test or via API.

Tip: Motor Controllers have a sticky fault to detect if device reset during robot-enable. This is useful for detecting breaker events.

2.19.1 LabVIEW

The GET STATUS VI can be used to retrieve sticky flags, and clear them.



2.19.2 C++/Java

The APIs getFaults() and getStickyFaults() can be used to check the latest received faults. clearStickyFaults() can be used to clear all sticky fault flags.

2.20 Common Device API

2.20.1 Setting Status Frame Periods

All Phoenix devices have a setStatusFramePeriod() routine that allows manually adjusting the frame periods of the status signals.

2.20.2 Detecting device resets

All Phoenix devices have a hasResetOccurred() routine that will return true if device reset has been detected since previous call.

Detecting this is useful for two reasons: - Reapply any custom status frame periods that were set using setStatusFramePeriod(). - Telemetry / general troubleshooting (in addition to sticky fault, see tip below).

Tip: Motor Controllers have a sticky fault to detect if device reset during robot-enable. This is useful for detecting breaker events.

2.21 Support

2.21.1 GitHub Examples

All documentation and examples can be found in the public organization... https://github.com/CrossTheRoadElec

There many examples in all three FRC languages available at... https://github.com/CrossTheRoadElec/Phoenix-Examples-Languages https://github.com/CrossTheRoadElec/Phoenix-Examples-LabVIEW

2.21.2 Contact information

CTR Staff can be reached out with the contact information available at...

http://www.ctr-electronics.com/contacts-us

The best method for contacting support is via our email (support@ctr-electronics.com). This allows for simple sharing of screenshots and supplemental file attachments.

If seeking help troubleshooting hardware issues, please answer the questions under "Warranty" inside the "Support Request form".

To resolve your issue in an expedient fashion, we need the following:

- What behavior are you seeing versus what are you expecting to observe?
- What procedure are you following to reproduce the issue?
- If using the roboRIO, we need a screenshot of the web-based configuration to confirm firmware version, gain values, etc.
- If using the roboRIO, we need a screenshot of a self-test taken during the undesired behavior.
- Part numbers of all devices involved. For example, if using a sensor, what is the sensor part number?
- Firmware versions of all devices involved.
- If using motor controllers, are they on CAN bus or PWM?
- If using Phoenix, screenshot of the About form in Phoenix Tuner / Phoenix LifeBoat.

2.22 Frequently Asked Questions

2.22.1 What do I do when I see errors in Driver Station?

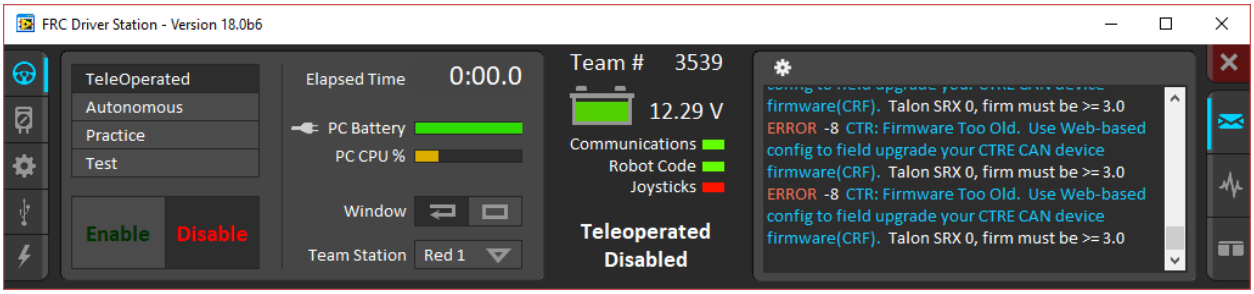
DS Errors should be addressed as soon as they appear. This is because:

- Phoenix API will report if a device is missing, not functioning, has too-old firmware, etc.
- If errors are numerous and typical, then users cannot determine if there is a new problem to address.
- A large stream of errors can bog down the Driverstation/roboRIO. Phoenix Framework has a debouncing strategy to ensure this does not happen, but not all libraries do this.

Phoenix DS errors occur on call. Meaning VIs/API functions must be called in robot code for any errors to occur. When an error does occur, a stack trace will report where in the robot code to look.

The Debouncing Strategy that Phoenix uses is 3 seconds long. Phoenix keys a new error on device ID & function. This is to ensure that all unique errors are logged while making sure the DriverStation/roboRIO does not generate excessive errors.

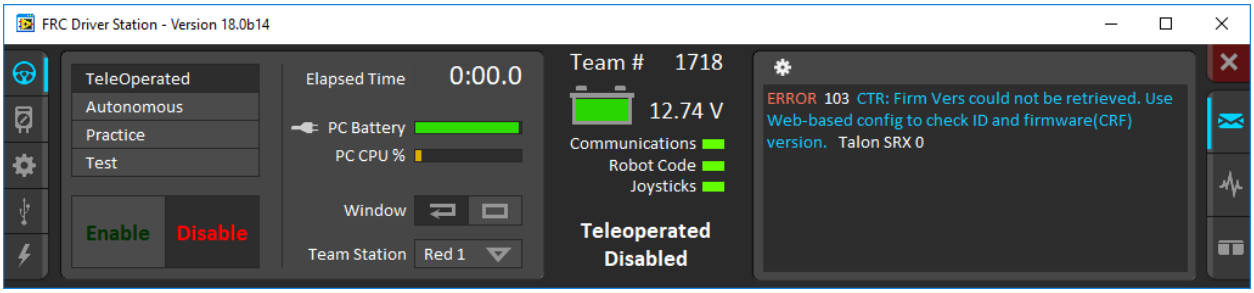
2.22.2 Driver Station says the firmware is too old.



Use Phoenix Tuner to update the firmware of the device.

Note that the robot application must be restarted for the firmware version check to clear. This can be done by redeploying the robot application or simply restarting the robot.

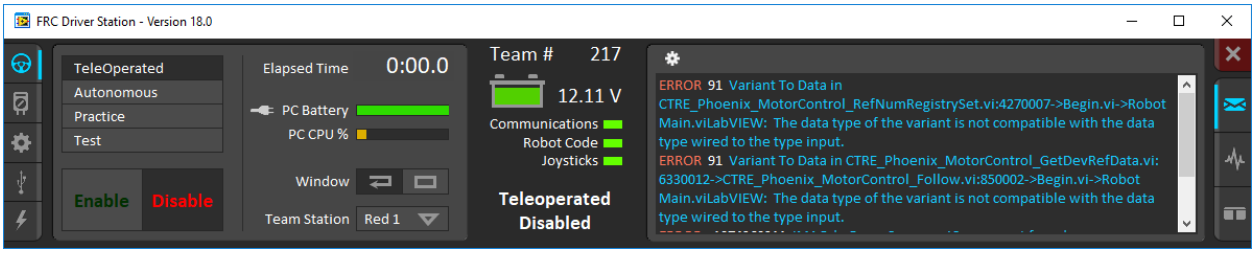
2.22.3 Driver Station says the firmware could not be retrieved and to check the firmware and ID.



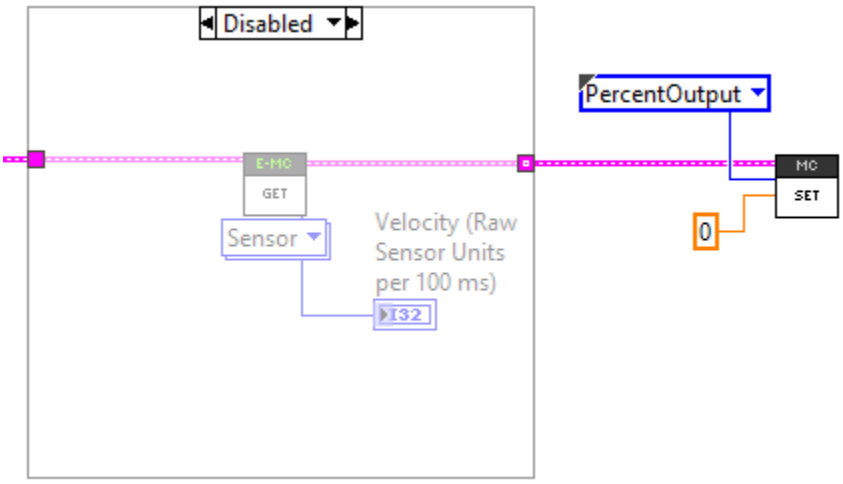
This usually indicates that your **device ID is wrong** in your robot software, or your firmware is **very old**.

Use Phoenix Tuner to check your device IDs and make sure your firmware is up-to-date.

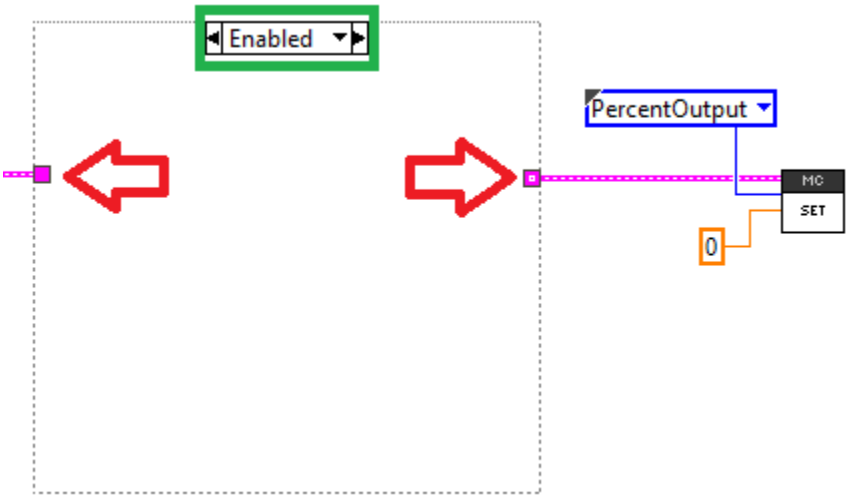
2.22.4 Driver Station Says Variant To Data in . . .

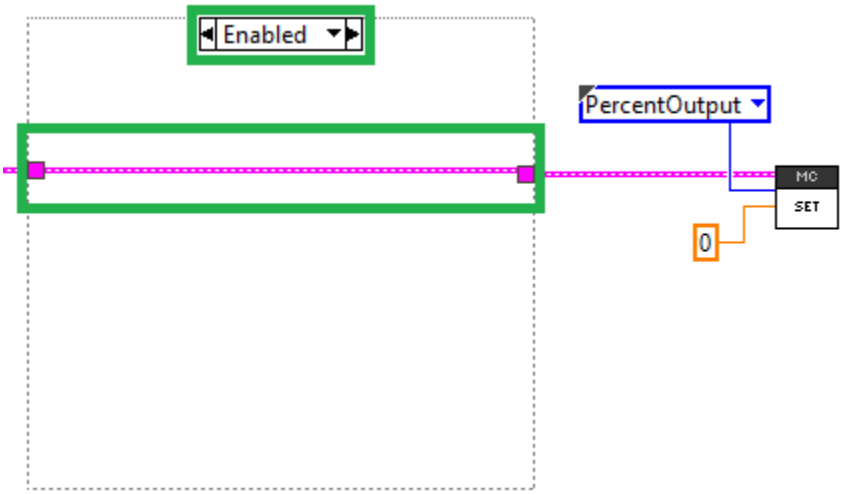


This is usually caused by a diagram disable structure around a MotorController or EnhancedMotorController VI



In order to fix this, you must wire the device reference through the enabled state of the diagram disabled block





2.23 Errata

2.23.1 HERO firmware compatibility with firmware 4.X

The HERO robot controller still requires 10.X firmware in the motor controllers to function correctly. This will addressed in a future release (which updates HERO).

2.23.2 Phoenix Tuner is odd at higher display resolution/DPI

Opening Tuner on a monitor that has a *high* dpi setting (where Windows recommends 250%) can cause the form to not render correctly. We recommend using a monitor resolution near 1920 X 1080 until issue is resolved.

Tip: Phoenix-Tuner 1.2.0 has improvements for this.

2.23.3 Phoenix Tuner specifying team number may work inconsistently

Tuner will connect via hostname/IP or via team number. When specifying team number, the roborio-TEAM-FRC.local host name is used correctly, however mDNS does not appear to reliably resolve.

Using static-IP or using USB works as expected.

Tip: Phoenix-Tuner 1.2.0 has improvements for this.

A tentative fix has been tested where Tuner will manually resolve it, working around this limitation. This improvement will be incorporated in a future update.

2.23.4 Pigeon IMU may not work as Remote Sensor

Pigeon IMU CRF 4.0 has a reported issue where if Pigeon is connected via ribbon cable, the Talon/Victor will not detect it when performing remote sensor features.

Tip: Pigeon Firmware 4.13 has a fix for this.

2.24 Software Release Notes

2.24.1 Talon SRX / Victor SPX

The kickoff/latest firmware is 4.11.

This version is adequate for FRC and nonFRC use (notwithstanding known errata with HERO C#).

2.24.2 CANifier

The kickoff/latest firmware is 4.0.

2.24.3 Pigeon IMU

The kickoff/latest firmware is 4.0.

2.24.4 PCM

PCMs ship with firmware 1.62. Latest firmware is 1.65. Either firmware is functional for FRC use.

2.24.5 PDP

PDPs ship with firmware 1.30. Latest firmware is 1.40 (auto zeros channels on boot). Either firmware is functional for FRC use.

2.25 Additional Resources

2.25.1 FRC Screensteps

Core documentation for the base FRC control system. http://wpilib.screenstepslive.com/s/currentCS

2.25.2 Power Distribution Panel (PDP)

http://wpilib.screenstepslive.com/s/currentCS/m/cpp/l/599692-power-distribution-panel http://wpilib.screenstepslive.com/s/currentCS/m/java/l/599694-using-the-can-subsystem-with-the-roborio

2.25.3 Pneumatics Control Module (PCM)

http://wpilib.screenstepslive.com/s/currentCS/m/java/l/219351-pneumatics-control-module http://wpilib.screenstepslive.com/s/currentCS/m/cpp/l/241865-operating-a-compressor-for-pneumatics