HERO User's Guide

Rev 1.2

Cross The Road Electronics

www.ctr-electronics.com

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your CTRE products. To this end, we will continue to improve our publications, examples, and support to better suit your needs.

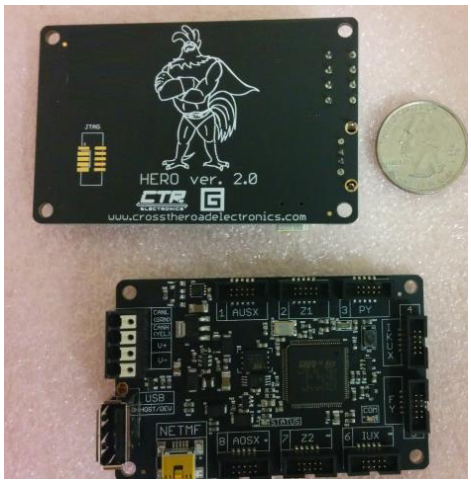
If you have any questions or comments regarding this document, or any CTRE product, please contact support@crosstheroadelectronics.com

To obtain the most recent version of this document, please visit www.ctr-electronics.com.

1. Introduction: What is the HERO development board?

The HERO is an STM32F4-based development board designed to support all CTRE CAN Bus devices.
It is a single hardware platform meant to …

- Support integrators/developers who would like to **use CTRE CAN devices in their custom solution**. HERO can be used in your application to control supported CAN devices, or can be used as an **aid to help integrate CAN framing** into your hardware.
- Allow non-technical users to **directly control a CAN Bus device** by plugging in a USB gamepad/joystick. This provides a quick method for throttling Talon SRXs **without writing any code** or finding a computer (requires non-FRC firmware).
- Serve as a **H**euristic for **E**ducating **RO**botics. With a few lines of C# (Visual Studio Custom/Express) anyone can control supported actuators over CAN Bus. This makes closed-looping, motion profile, and advanced robotics accessible to the next generation of STEM participants (Science, Technology, Engineering, and Math).
- Provide diagnostics for **checking health** of CAN devices and **modifying device IDs**.
- Provide an easy method for **field-upgrading supported CAN devices**. This is particularly helpful as CTRE CAN devices have FRC firmware (**FIRST Robotics Competition**) and non-FRC (firmware meant for **general** use).



In short it is the ideal development kit for learning and integrating the following modules into your custom application…

- Talon SRX / Victor SPX (12V "smart" brushed motor controllers)
- Pneumatics Control Module (control 12V/24V solenoids, automatic compressor control with pressure-switch input)
- Power Distribution Panel (monitoring currents, voltage, energy, power)
- Future CAN Devices

HERO is field-upgradeable over USB using the HERO LifeBoat application. This requires an A-to-A USB cable (see Section 4).

HERO stands for **H**euristic **E**ducating **RO**botics. Although it serves many purposes, its primary focus is to empower users to develop robotic control with minimal effort.

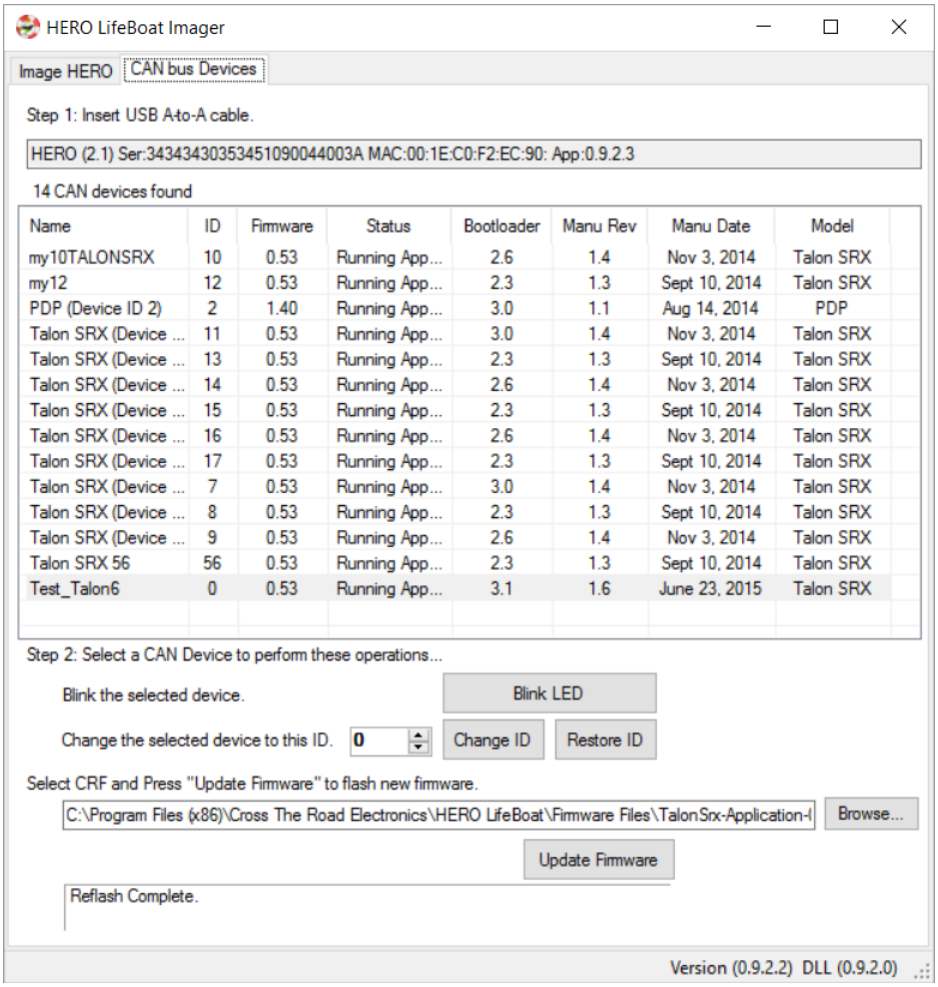
1.1. HERO LifeBoat

In order to field upgrade the HERO, and the supported CAN devices it discovers, CTRE provides a Windows application called "HERO LifeBoat Imager" (or LifeBoat for short). This application communicates with HERO using **a USB A-to-A cable**.

LifeBoat stands for **L**ightweight **I**mage **F**lasher **E**nsuring **B**ulletproof **O**peration **A**nd **T**esting.

With this utility you can…

- Field-upgrade the HERO
- Field-upgrade support CAN Devices: Talon SRXs, PCMs, and PDPs.
- Modify the Device IDs of CAN devices so that they can be addressed uniquely.
- Blink a selected device to sanity check the Device ID.

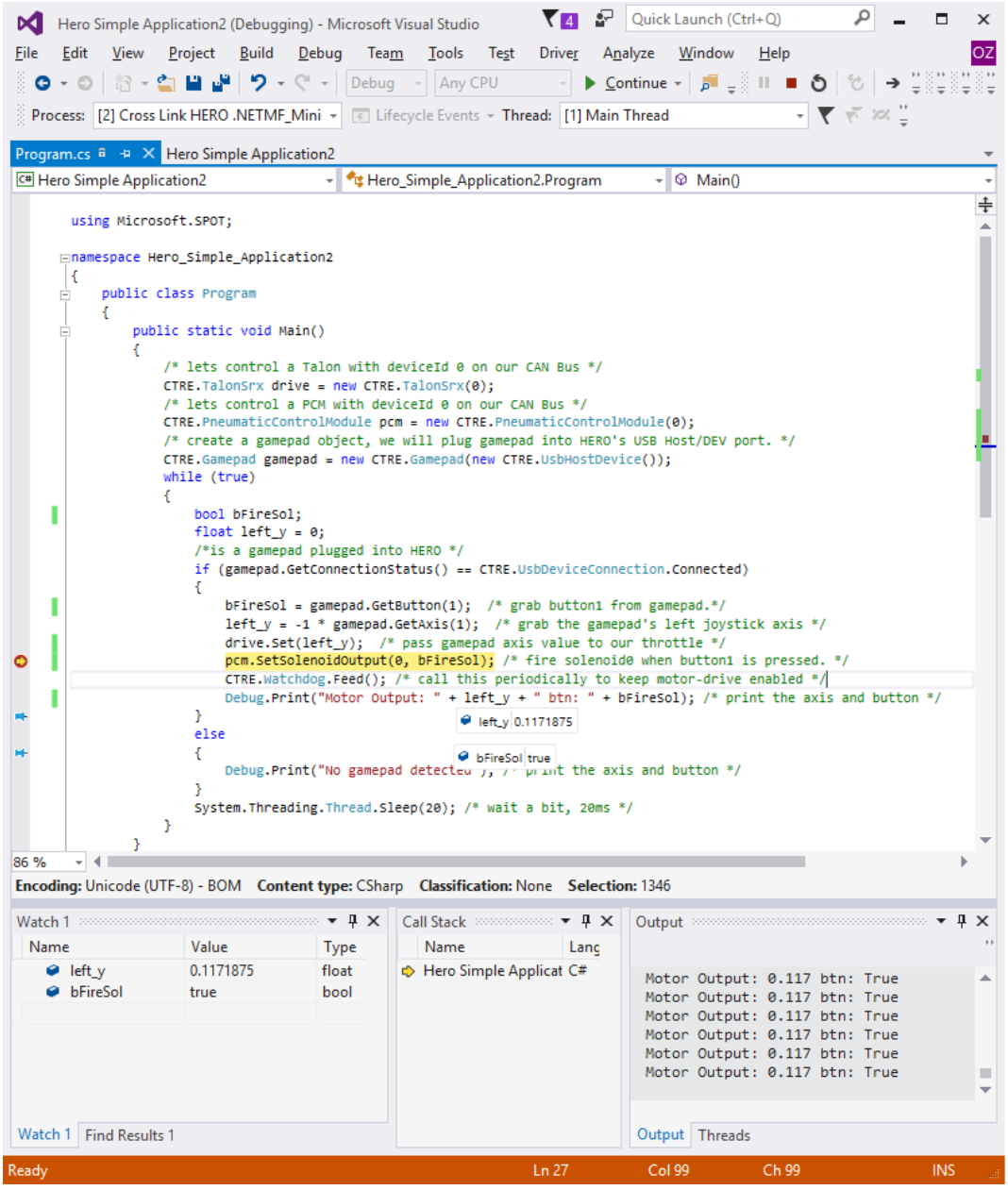


See Section 7 for more information on this utility.

1.2. What is .NET Micro Framework?

.NET Micro framework is an open-source .NET platform for embedded devices. It includes a small version of the .NET CLR and supports development in C#, Visual Basic .NET, and debugging (in an emulator or on hardware) using Microsoft Visual Studio. Basically it's a software stack that can run a "minified" version of a .NET application with the goal of running on an embedded processor.

When NETMF is combined with CTRE's native libraries, the result is a tool suite that allows users to develop robotic applications in Visual Studio, (with full debugger, threads, watch list, breakpoints, etc.).



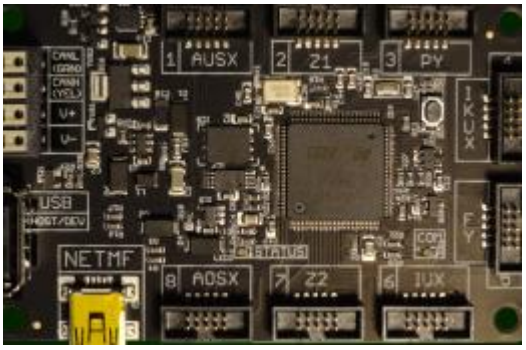
1.3. What are Gadgeteer devices?

Another benefit to NETMF is the common hardware interface for supplemental devices known as "Gadgeteer". These common interfaces use a small footprint ribbon-cable connector for connecting peripherals that use USART, SPI, I2C, PWM, general IO, and more. The Gadgeteer ports are labeled on the HERO itself, and more examples and documentation will be provided as support for these ports are added in future software releases.

This is the same connector/cable used on the Talon SRX for sensor feedback.

Below is the pinout of each Gadgeteer port type. Each port on HERO (numbered 1 through 8) have a collection of letters that indicate the possible uses for that port. Each port has several interfaces available to it, so choose the role of each port based on what combination of features you will need.

For example, HERO port 1 is labeled "AUSX". This means the HERO port 1 can be used for analog input, UART, SPI, or 3-GPIOs. After deciding which role you would like to leverage, consult the pinout table below for wiring.



TYPE	LETTER	PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8	PIN 9	PIN 10
3 GPIO	X	+3.3V	+5V	GPIO!	GPIO	GPIO	[UN]	[UN]	[UN]	[UN]	GND
7 GPIO	Y	+3.3V	+5V	GPIO!	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GND
Analog In	A	+3.3V	+5V	AIN (G!)	AIN (G)	AIN	GPIO	[UN]	[UN]	[UN]	GND
SD Card	F	+3.3V	+5V	GPIO!	DAT0	DAT1	CMD	DAT2	DAT3	CLK	GND
USB Host	H	+3.3V	+5V	GPIO!	D-	D+	[UN]	[UN]	[UN]	[UN]	GND
I²C	I	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	[UN]	SDA	SCL	GND
UART+ Handshaking	K	+3.3V	+5V	GPIO!	TX (G)	RX (G)	RTS	CTS	[UN]	[UN]	GND
Analog Out	O	+3.3V	+5V	GPIO!	GPIO	AOUT	[UN]	[UN]	[UN]	[UN]	GND
PWM	P	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	PWM (G)	PWM (G)	PWM	GND
SPI	S	+3.3V	+5V	GPIO!	GPIO	GPIO	GPIO	MOSI	MISO	SCK	GND
UART	U	+3.3V	+5V	GPIO!	TX (G)	RX (G)	GPIO	[UN]	[UN]	[UN]	GND
Manufacturer Specific	Z	+3.3V	+5V	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	GND

2. Specifications

2.1. Electrical Specifications

Symbol	Parameter	Condition	Min	Typ.	Max	Unit
HERO Power Input (Weidmuller only)						
V_{dd}	Supply voltage		5.2	12.0	28.0	V
I_{supp}	Supply Current	DC supply @12.0V No Gadgeteer ports in use	48	58 (1)	72	mA
HERO Power Input (mini USB only)						
V_{dd}			4.7		5.0	V
I_{supp}	Supply Current	No Gadgeteer ports in use	85	107 (1)	133	mA
USB Host						
V_{USB}	USB Output Voltage			5.0		V
I_{USB}	USB Output Current				500 (2)	mA

NOTE 1: Typical measurement is done when both LEDs are **green** and Logitech USB gamepad present.

2: This assumes the power source for HERO can source enough current to meet this.

2.2. Inputs/Outputs

Input and Outputs	
Gadgeteer Ports Total	8
Supported Communication Protocols	I2C, SPI, USART/UART, PWM, CAN 2.0B DWCAN bus (1Mbps)
Supported CAN Devices	Talon SRX, PDP, PCM
USB Ports	1 USB Host/Device (Gamepad HID) 1 USB Device (mini)
Analog Input	2 Gadgeteer A ports (3 AINs each)
PWM Output	1 Gadgeteer P port (3 PWM outputs)
SPI	2 Gadgeteer SPI ports
USART/UART	3 Gadgeteer U ports (UART) 1 Gadgeteer K port (USART) 3 serial ports total
I2C	2 Gadgeteer I ports
GPIO	4 Gadgeteer X ports (3 GPIO each) 2 Gadgeteer Y ports (7 GPIO each)
3.3V Analog Output	1 Gadgeteer O Port (1 AOUT)
SD Card	1 Gadgeteer F port (4bit SDIO)
Reserved ports	Z1 and Z2 reserved for future use

2.3. Processor/Memory

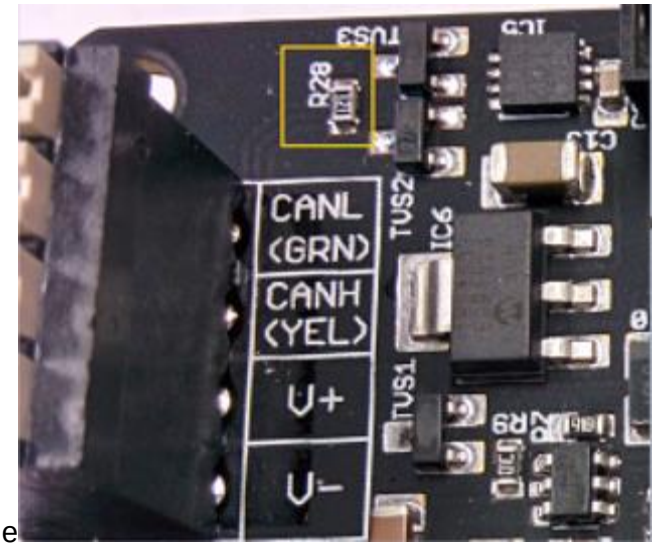
Process and Memory	
Processor	STM32F427 (Cortex M4)
Processor RAM	256KB
Processor Flash	2 MB
Processor Frequency	168 MHz
Usable FLASH for NETMF (will be increased to at least 1MB in future release)	640 KB
Usable RAM for NETMF (can increase in future)	100KB

3. CAN Bus Features

The HERO supports one CAN2.0B DW-CAN channel meant for communicating with Talon SRXs, PCMs, and PDPs. The default bitrate is 1Mbps (75% sample point) and uses 29bit arbitration IDs. However, in the future, API may be available to modify the timing parameters for custom applications.

3.1 CAN Bus Termination

HERO has an integrated CAN Bus Termination Resistor R28 (120Ω). For applications that require the HERO to not include termination, the surface mount resistor can be removed. This would be necessary if HERO was connecting to an existing CAN Bus that already had proper termination at each of the two ends of the bus harness, such as a vehicle or industrial bus.



e

However, if using a **CTRE Power Distribution Panel**, you can simply remove the termination jumper on the PDP instead. See the PDP User's Guide for more information.

4. HERO USB Ports Explained

HERO has two USB ports…

- Mini USB port labeled NETMF.

  This port requires a **mini USB cable** and is used for deploying/debugging your C# applications into HERO.

- Host and device port labeled "Host/Dev".

  This is used to field-upgrade the HERO and discovered CAN devices (when used with a **USB male-A-to-male-A cable**). Additionally HERO will support a limited number USB devices that may be inserted into HERO and used in your NETMF application.

Both cable types will be available for purchase at ctr-electronics.com.

4.1. USB Host features

At the time of writing, the USB Host port supports HID Gamepad/Joystick devices for control of robot applications, as well as X-input for Xbox controller devices.

This includes the **Logitech F310** Gamepad…

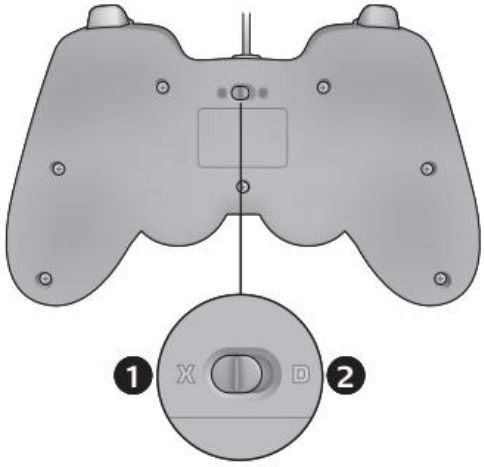
…and for those looking for a wireless solution, the **Logitech F710**…

For developers looking for a reliable and simple wireless solution for control and robot-disable, we recommend the **Logitech F710**. Future releases of HERO will support a variety or wireless control solutions, however in the short term we have found these wireless gamepads to be adequate.

Note: When using the **Logitech F710**, you may have to **press a button to "wake up"** the gamepad. Since this wireless device uses batteries, it will enter a low-power mode **when left without activity** for several minutes. When this happens the HERO will behave as though the Gamepad is still connected, but with all signals zeroed to prevent erroneous motor activity.

4.2. Logitech: XInput versus DirectInput

At the time of writing, HERO supports **'D'** or **'DirectInput'** for control by default. Switching the selector to position 2 or 'D' is necessary to use a Logitech Gamepad by default.



This is done because this allows the 'X' setting to serve as a reliable robot-disable or emergency-stop. In other words, selecting 'X' will be equivalent to unplugging the gamepad, which will zero all gamepad signals.

For details on how to use a Logitech controller with "X" mode, see the example projects on GitHub.

4.3. X-Input (Xbox) Controllers

For devices like Xbox Controllers that require X-input support, you can use the dedicated Xbox controller class. See Section 9.1.

5. Hardware setup

5.1. General Handling

When handling bare electronics, care should be taken to not touch the electronic components directly. Hold the HERO by the board edges to prevent electrostatic discharge (ESD) events from damaging the board. Be sure to discharge by touching a grounded surface before handling the HERO board.

Furthermore, when using the HERO in a robotics application, choose a mount or enclosure strategy to prevent potential shorts caused by conductive debris (fine metal chips or swarf, spare or untucked wires, metal chassis).

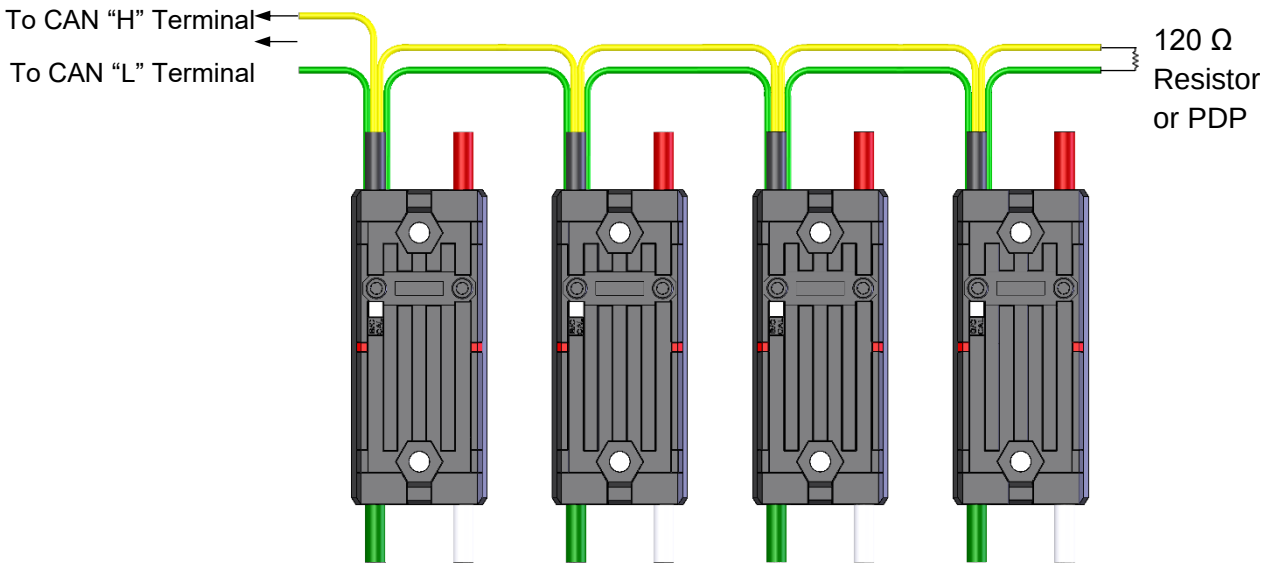
5.2. Connecting CAN devices

Comprehensive instructions for connecting Talon SRXs, PCMs, and PDPs to HERO's Weidmuller connector can be found in the User's Guide of each product respectively. PCMs and PDPs each have two pairs of Weidmuller connectors with CANH/CANL. This allows PCMs and PDPs to be part of the daisy chain. Talon SRXs can also be daisy chained as they each have two wire pigtails than can conveniently be used to connect two devices that use the Weidmuller connectors such as PCM, PDP, and HERO.

5.2.1. Wiring the Talon SRX for use with CAN bus

To wire Talon SRXs, connect a **yellow** signal wire to the CAN terminal marked "**CANH**" on the HERO and connect a **green** signal wire to the CAN terminal marked "**CANL**" on the HERO.
To connect additional Talon SRXs, use tightly crimped connectors to connect the signal wires **green**-to-**green** & **yellow**-to-**yellow** as shown below. For the best connection, it is recommended that each connector is crimped **and** soldered. However CAN connectors are available to provide reliable connections that are not permanent (see Section 5.2.1.1).

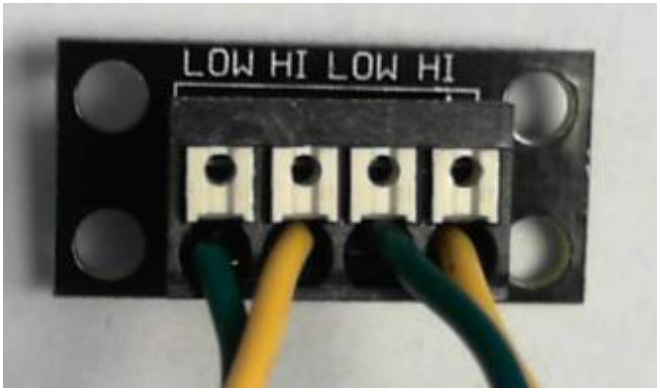
The **yellow** and **green** wires should also be wrapped in a twisted pair fashion (not illustrated) to maximize tolerance to electrical noise.
Note: Signal wires of the same color are electrically identical – it does not matter which wire is used as long as the color is correct.



After all of the Talon SRXs have been wired, there will be 2 remaining signal wires – connect these two wires using a 120 Ω resistor or to the CAN interface on the Power Distribution Panel (PDP) to properly terminate the cable end.

5.2.1.1. CAN Connector

The CAN Connector can be used to chain Talon SRX's together without crimping connectors or soldering. Each CAN Connector contains a four channel Weidmuller terminal block with two CAN pairs (similar to other FRC CAN devices such as the Power Distribution Panel and Pneumatic Control Module). The four holes can be used to provide strain relief to the CAN wires. Additionally the holes can be used for mounting to the robot frame. Spacers or electrical tape may be used to prevent shorting to the robot frame.



These are available for purchase at ctr-electronics.com.

5.2.1.2. CAN Bus Wire Selection

It is recommended to use **yellow** for CANH and **green** for CANL for the following reasons…

- Makes inspection and troubleshooting easier.
- The colors match what is labeled on the HERO, Power Distribution Panel, and Pneumatic Control Module.
- The colors match the Talon SRX cable harness.

AWG 22 or similar gauge wiring can be used. An electric drill can be used to twist the CANH/CANL wire pair. Pre-twisted wire is also available at ctr-electronics.com.

5.3. Powering Options for HERO

HERO can be powered using one of several power paths (or multiple simultaneously). Below are the three different connection methods.

5.3.1. Powering HERO using Weidmuller connector

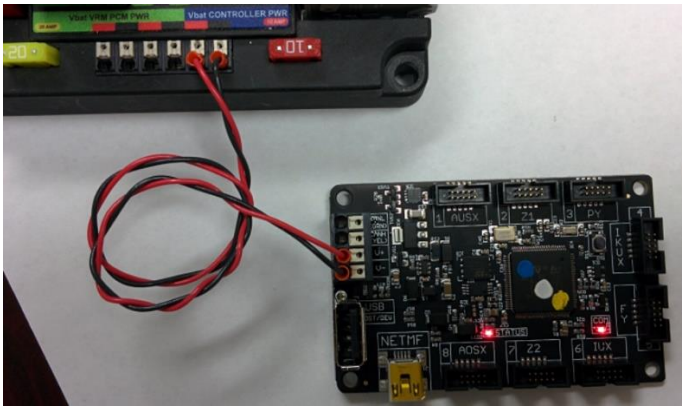
For robot applications that use 12V sources (such as lead-acid batteries or power supply) the Weidmuller press-insert connector can be used to connect V^+ and V^-.

Consult the electrical specifications for power requirements (see Section 2.1).

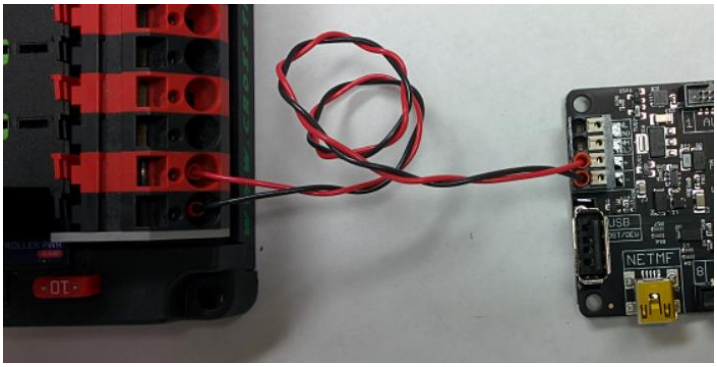
The Weidmuller power input is reverse protected. This means that reversing the V^+ and V^- wires will not damage the HERO.

Although the picture above shows a red/black wire pair with wire ferrules, the connector will also work using regular stripped wires as well. Press down on the white plunger to insert/release the wire.

For applications that use the CTRE Power Distribution Panel (PDP), the source end of the power harness can be connected to the bottom output Weidmuller pair on the PDP.



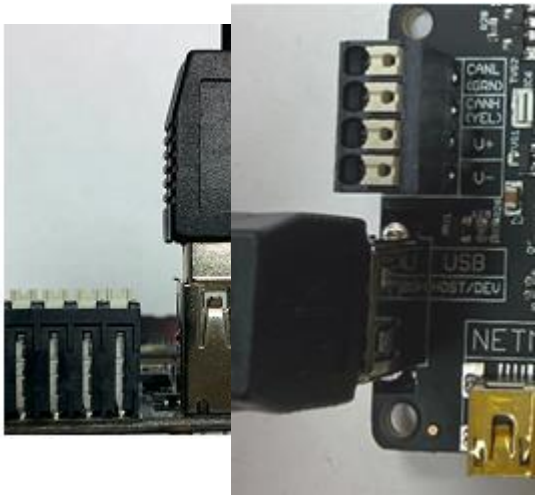
Alternatively the HERO can be powered using one of the PDP WAGO channels.



5.3.2. Powering/Connecting HERO using USB-A-to-A

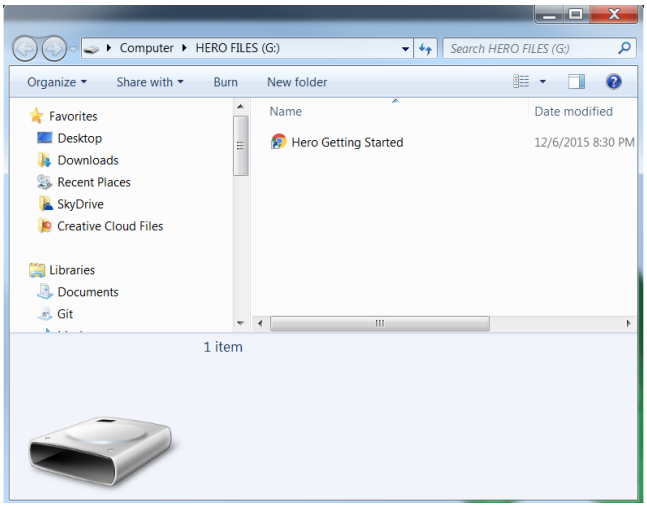
The USB-A-to-A cable can also be used to power the HERO. The USB-A-to-A connection is also used for LifeBoat connectivity (field upgrade and CAN device configuration)

Plug one male end into the HERO's Host/Device USB port.



Plug the other male end into the PC's host port.

5.2.2.1. Initial connection of an out-of-the-box HERO



The first time you connect the USB-A-to-A cable, you will see a mass storage device for HERO Files. This is due to the ship-firmware in the HERO, and will function on any Window PC (even if HERO SDK is not installed).

This is meant to redirect you to the HERO product page, where this document and necessary software can be downloaded.

5.3.3. Powering/Connecting HERO using mini USB

The mini-USB port will also power the HERO development board. The mini USB port is also used for deploying/debugging a NETMF application using Visual Studio.

Plug the mini connector into 

the HERO's NETMF port.

Plug the other end of the mini USB cable into PC Host port.

5.3.3.1. Initial connection of an out-of-the-box HERO

The mini USB connection is disabled in the ship firmware of an out-of-the-box HERO. It will become NETMF enabled after flashing latest firmware using HERO LifeBoat.

6. Software Installation (does not require HERO hardware)

If you want to develop C# applications on your HERO, you will need to…

- **Download** and **install** **Visual Studio 2019 Community** (download from **visualstudio.com**, see Section 6.1 below).
- **Download** and **install** the **Phoenix Framework Installer** from **CTRE**, see Section 6.2).
    - Make sure to **select** all HERO options when installing.

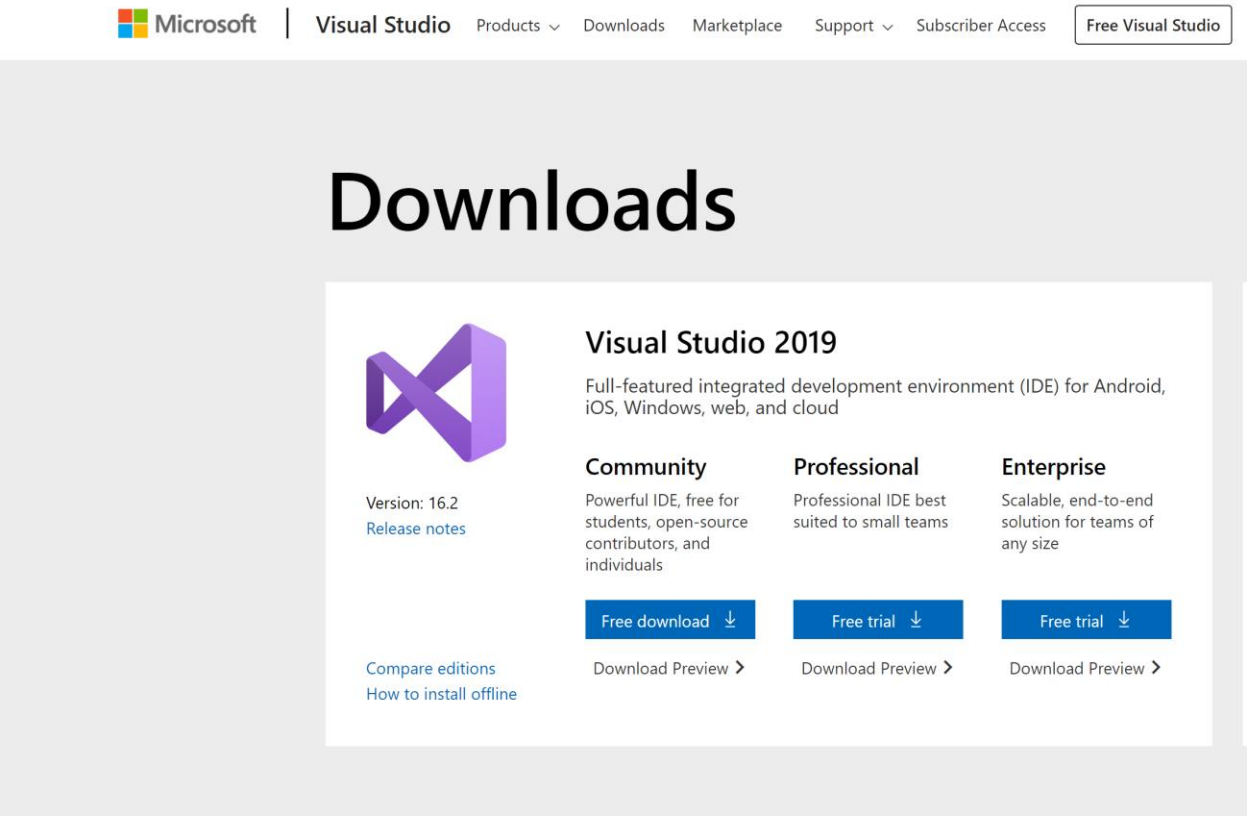
…However, if you simply want to **direct drive** all Talon SRXs that are connected to HERO's CAN Bus, you can use the **CTRE Phoenix Framework Installer** to just install **Lifeboat**.

6.1. Microsoft Visual Studio Community (or Express or Professional) 2019

The HERO netmf SDK requires Visual Studio 2017 version 15.3 or later.

If you already have a compatible version of Visual Studio 2017 or 2019 , you can skip installing it.The download location may change as Microsoft updates their website, but a good general link to start with https://learn.microsoft.com/en-us/visualstudio/releases/2019/history?branch=updates%2Fhistory

Follow the links for "Download Community Free"…



6.2. HERO SDK

The CTRE Phoenix Framework will install all of the HERO-related content. The installer is available at http://www.ctr-electronics.com/links/hero_sdk_installer.html under **Tech Resources**. For more detailed installation instructions, see our GitHub documentation here.

For advanced users who may be interested, here is a breakdown of what is installed for HERO…

- USB drivers for HERO.
- Firmware images (crf) to flash into HERO.
- Firmware images (crf) to flash into supported CAN devices.
- A compiled plugin file for Visual Studio (this is how example templates are installed in Visual Studio).
- A beta build of Microsoft's 4.4 .NET Micro framework SDK. This predates the RTM release from Microsoft so avoid installing the release version that Microsoft provides.
- The compiled class library (Assemblies) for HERO.
- Visual Studio 2015 C++ x86 runtime (if it's not already installed).
- Visual Studio 2013 C++ x86 runtime (if it's not already installed). This is necessary as some of LifeBoat's components were built with VS2013.
- HERO .NETMF Plug-in (if Visual Studio 2017/2019 is installed).

The entire install takes one to four minutes depending on if the runtimes are already installed.

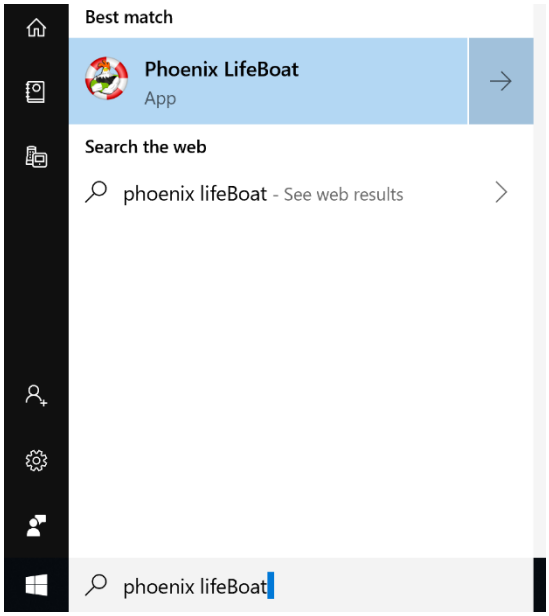
7. HERO LifeBoat – Software Configuration

HERO is field-upgradeable over USB by using a **USB A-to-A cable** and **not the mini USB port**.

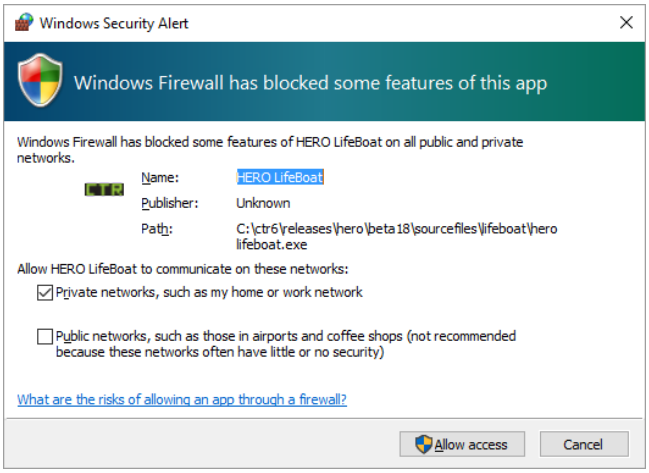
When a HERO is first unboxed, the first step is to update its firmware.

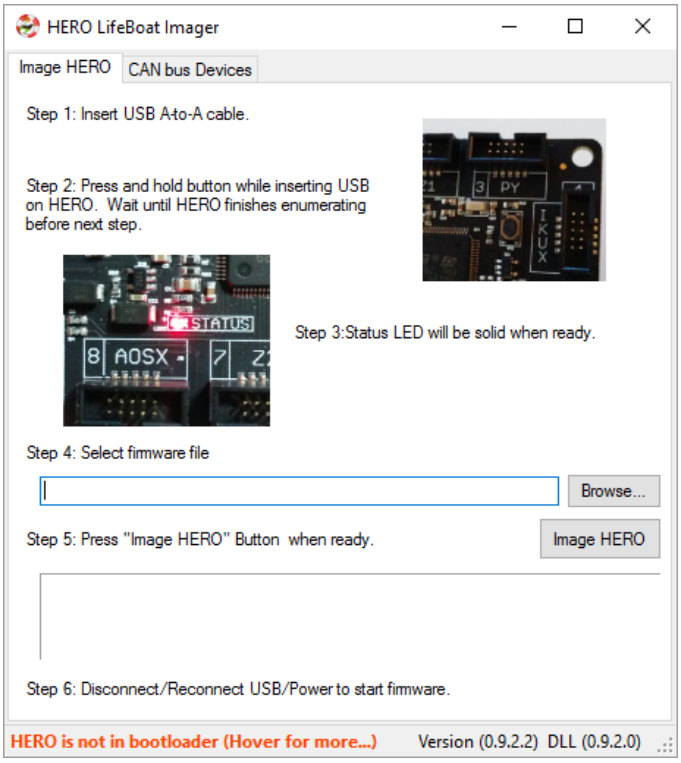
7.1. HERO LifeBoat – Re-image HERO Development Board

The HERO development board will come out of the box with a minimal firmware image. To update the firmware, download and run the HERO-SDK-Installer (see Section 6.2) as this will also install HERO LifeBoat. Once installed, open **HERO Lifeboat** by using the **Desktop shortcut**, or by typing "HERO Lifeboat" in the **Windows start menu**.



When LifeBoat runs the first time, you may get a prompt to allow LifeBoat to communicate over the network. **Check "private networks…" and then "Allow access".**





If not already selected, select the first tab "**Image HERO**".

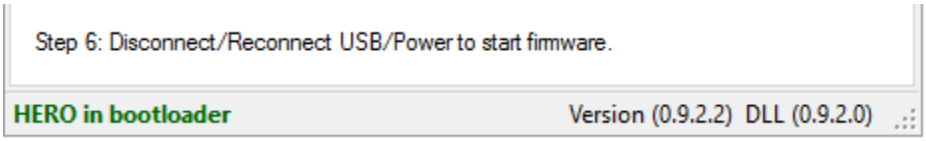
The bottom status bar will likely indicate that HERO is not in bootloader mode.

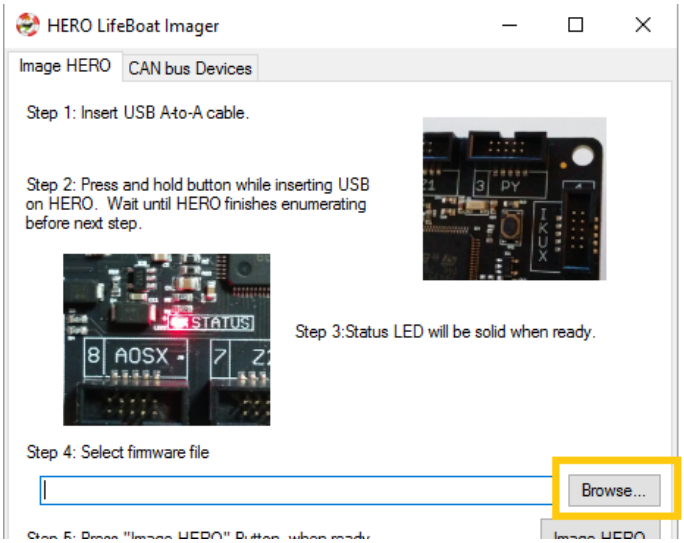
Connect the HERO to the PC by inserting the **USB A-to-A** cable into HERO's USB Host/Dev port. Do this **while holding the surface mount button on HERO down**.

When HERO boots up the STATUS LED will be **solid RED**.

This method of entering bootloader leverages a hardware feature called "DFU", that ensures that no matter what the state of the firmware, we can always reliably enter the bootloader. This makes HERO a brick-proof device.

When this is done, Windows will load the USB driver for HERO and the status bar will transition to say that HERO is in bootloader mode.

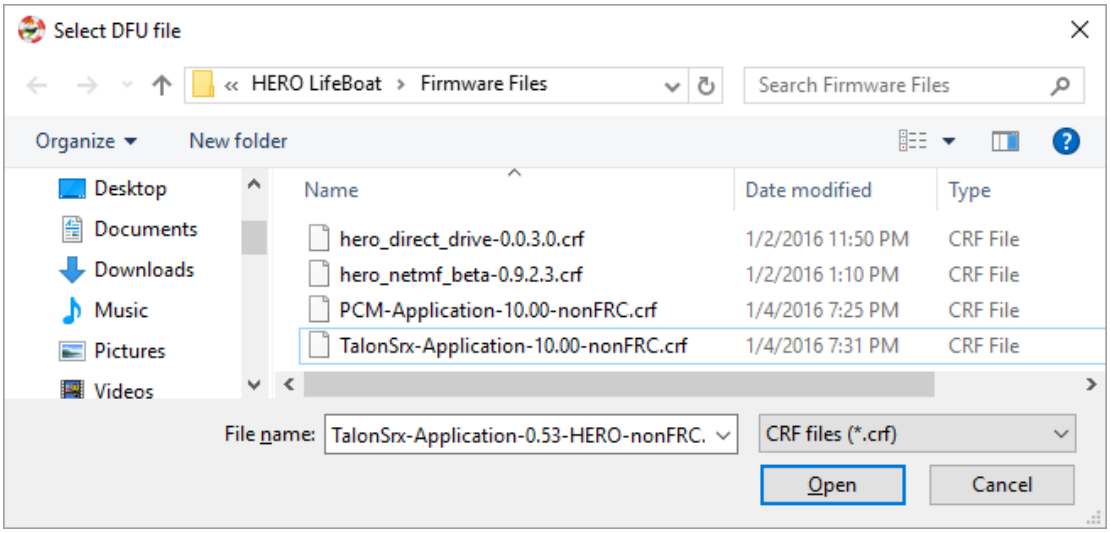




Next press the "**Browse**" button to navigate to the firmware file. Because the file path is blank, LifeBoat will start the dialog box inside the install location of LifeBoat, which is ideal since that is where the firmware files are located.

Typical Firmware File Path:
C:\Users\Public\Documents\Cross The Road Electronics\LifeBoat\HERO Firmware Files

There will be multiple CRF files (**C**ross The **R**oad **F**irmware). The files prefixed with "HERO" are the hero firmware files.

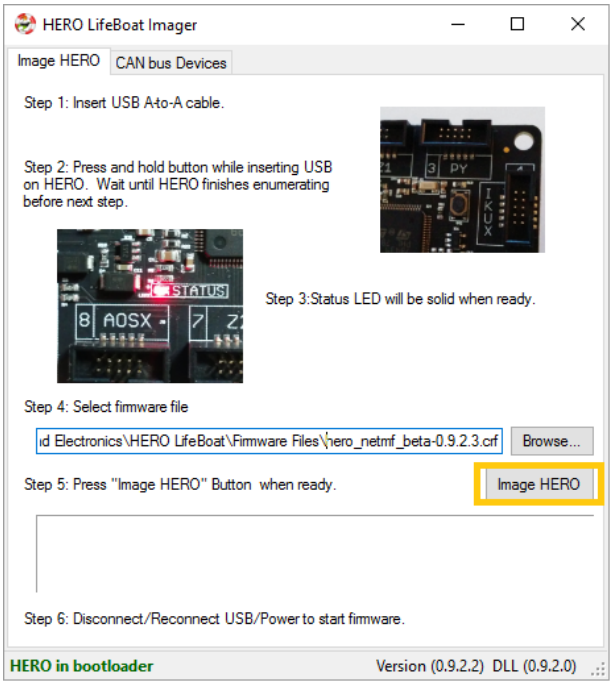


If the NETMF firmware is required (for developing robot applications in Visual Studio) select the latest "hero_netmf_XXXXXX.crf".

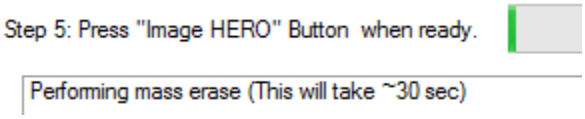
If the direct drive firmware is desired then select the latest "hero_direct_drive-X.X.X.X.crf". This is the firmware that simply takes the first Y-axis of a gamepad and drives all Talon SRXs for no-code simple-testing.

Then press "Open"

Next press the "Image HERO" button.



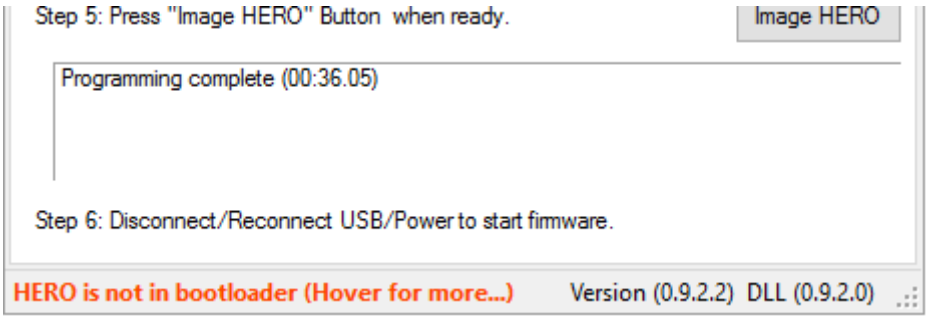
LifeBoat will first erase the contents of HERO…



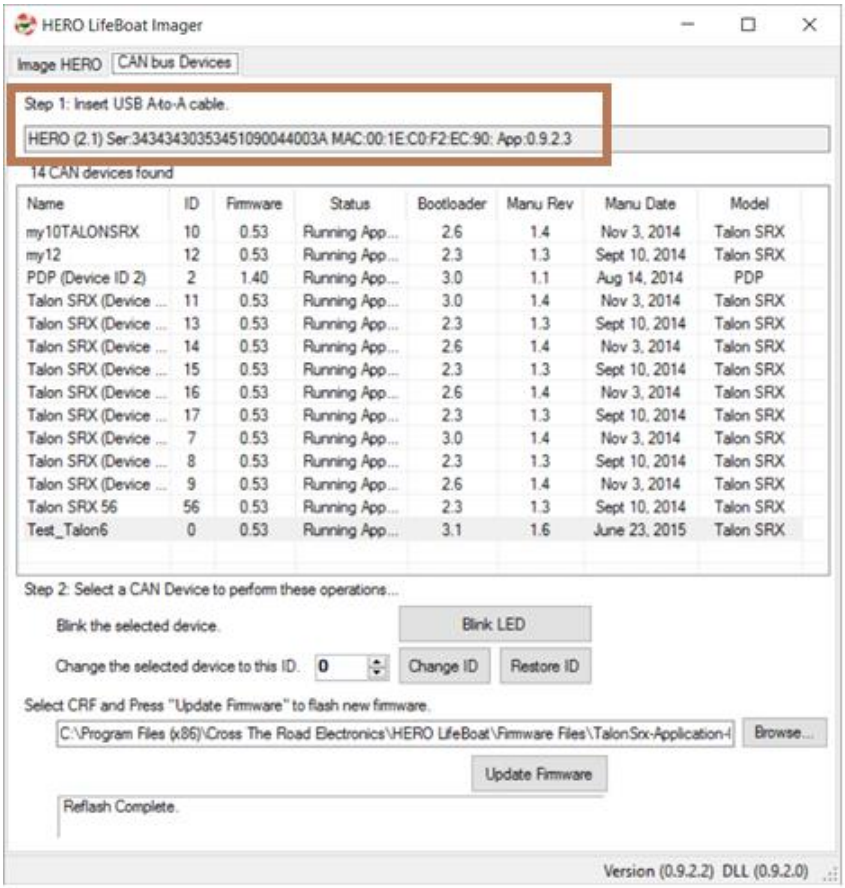
…then will begin writing the contents of the new image. When the imaging is finished you will hear several USB disconnect/connect events and LifeBoat will revert back to the original status bar message ("not in bootloader").

When flashing the **hero_netmf** firmware, the entire flash session can take ~30 seconds to one minute depending on the PC.

When flashing the **hero_direct_drive** firmware, flashing takes ~10 seconds.



At this point you can navigate to the second tab "CAN Bus Devices", and see your HERO's revision, serial number, MAC address and firmware version. Additionally if the CAN Bus is wired, the auto-detected CAN devices will appear in the list view.



7.1.1. Manual Install Driver location

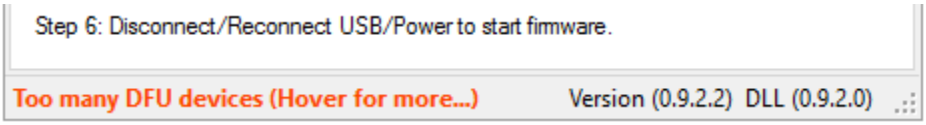
If, for some reason, Windows does not load the driver, you can manually install the drivers by directing Windows toward…

"C:\ Users\Public\Documents\Cross The Road Electronics\HERO\USB Drivers\dfu_Win8"

However this is typically not necessary as the HERO-SDK-Installer will install the drivers automatically.

7.1.2. Avoid multiple HERO connections

If multiple HERO's are connected and in bootloader mode, Lifeboat will not function correctly and you will see this message in the bottom status bar. Resolve this by disconnecting one of the HERO boards from the PC.

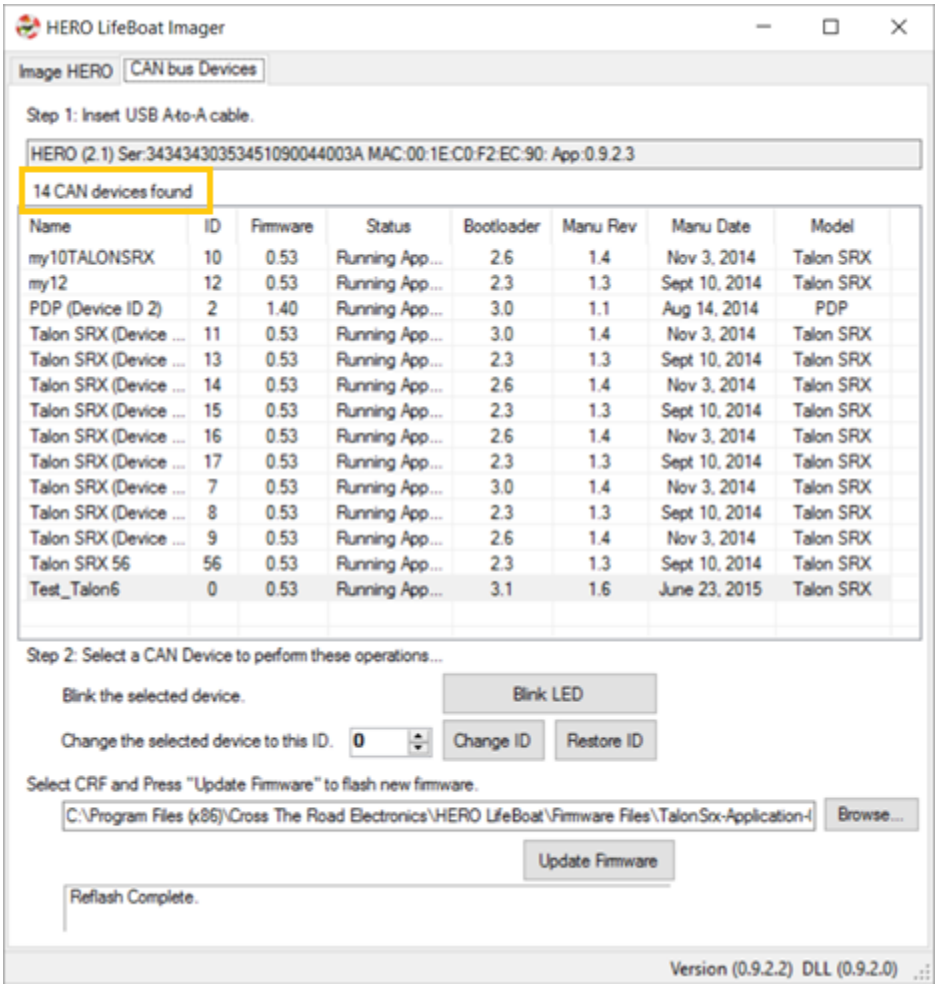


7.2. HERO LifeBoat – Configuring CAN Devices

The second tab in LifeBoat is the "CAN bus Devices" tab. This page can be used to configure the automatically discovered CAN devices. Once this tab is selected, LifeBoat will automatically discover the connected HERO, and begin searching and reporting each device's…

- Name (Which will be configurable in a future release)
- Device ID
- Firmware Version
- Status (Running Application versus Bootloader)
- Bootloader Version
- Manufacture Revision
- Manufacture Date
- Model

TIP: Quickly glance at the total device count and compare against the expected total count of physical devices to quickly confirm the bus is healthy.



7.2.1. CAN Device IDs

The robot API works by addressing each CAN device with a unique device ID. This is how you can chain several Talon SRXs (for example) and control each one individually.

Every device of a given model should have a unique device ID (range 0 through 62). So for example there **cannot** be two Talon SRXs with device ID '0'. However two different models can have the same device ID (you can have a PCM with device ID '0' and a Talon SRX with device ID '0').

All devices ship with a factory default ID of '0', so it is recommended to isolate each device initially and assign unique device IDs. A future update will be capable of detecting and gracefully resolving conflicted device IDs (similar strategy done in FRC).

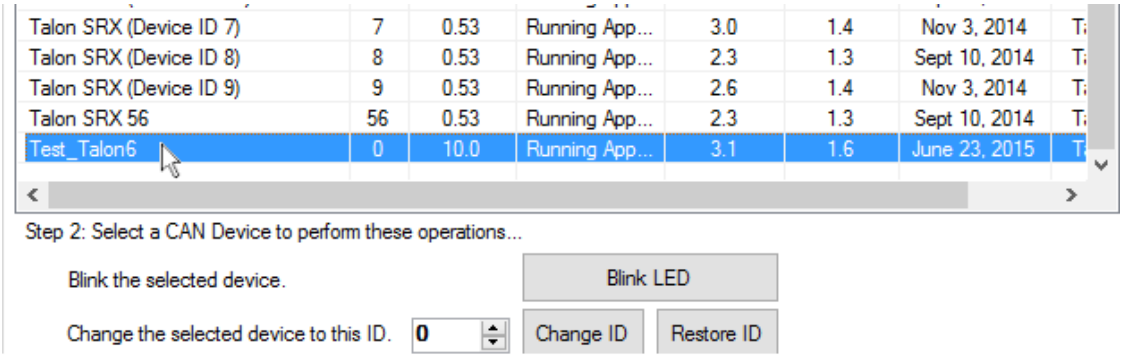
The Device ID makes up the bottom six bits of the arbitration ID of all control CAN frames. '63' is reserved for broadcast framing.

7.2.2. Updating the firmware of a CAN Device.

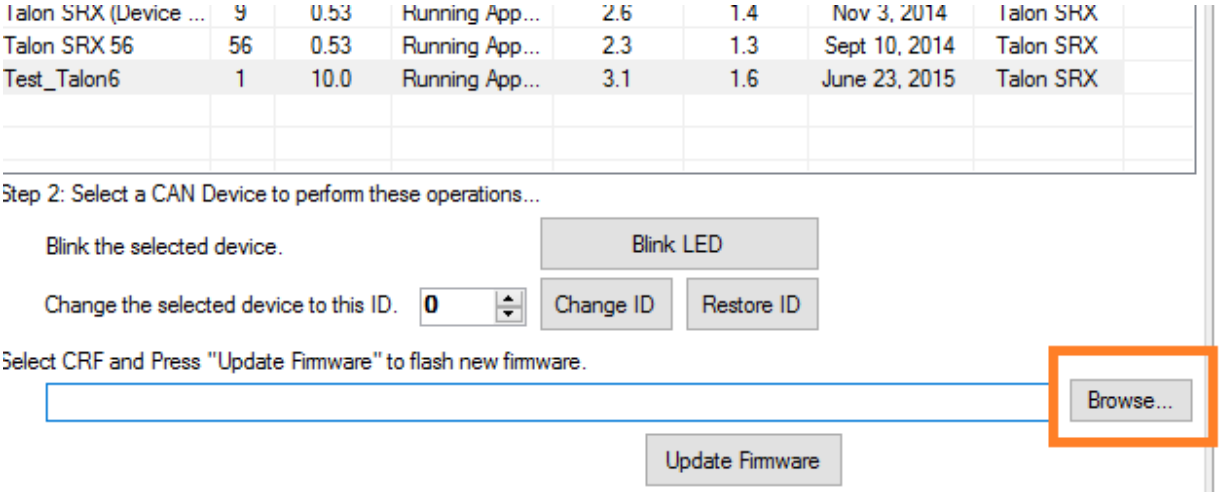
HERO LifeBoat can also be used to field upgrade supported CAN Devices. This is beneficial as most CAN actuators ship with FRC firmware meant to be used in the FIRST Robotics Competition. However for use outside of the competition, it will be necessary to update to the "non-FRC" firmware for general/public use.

These firmware files will be available on CTRE's website under the various product pages, and also are automatically installed in the same location as the HERO firmware on your PC.

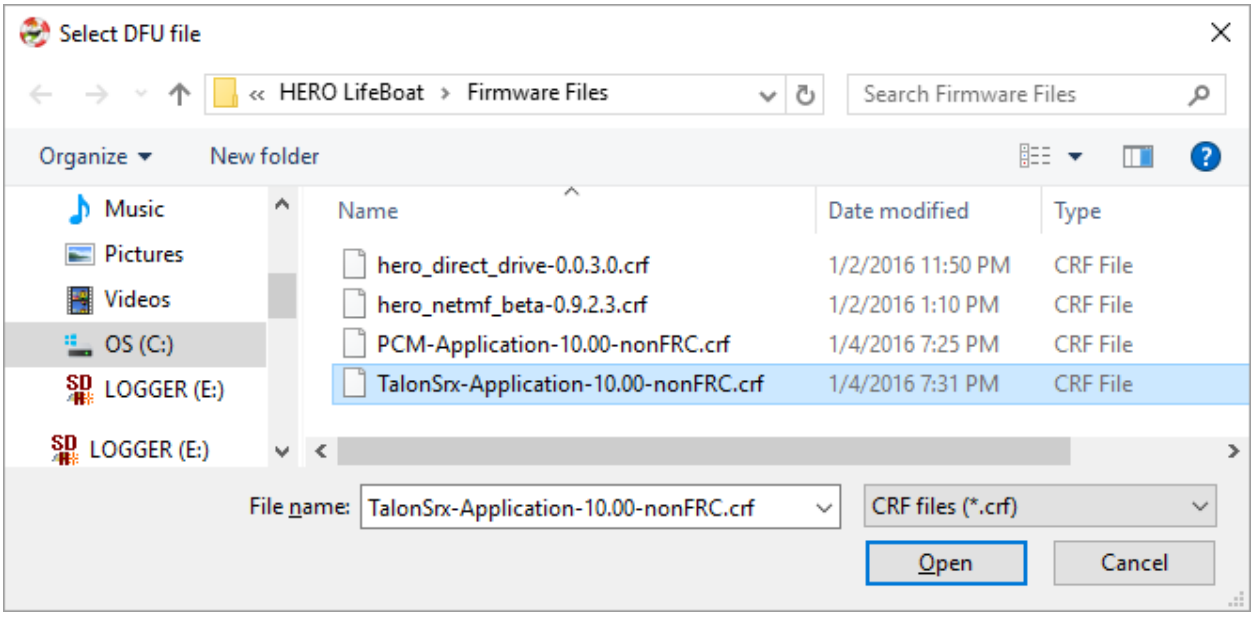
To begin, select the CAN device to update.



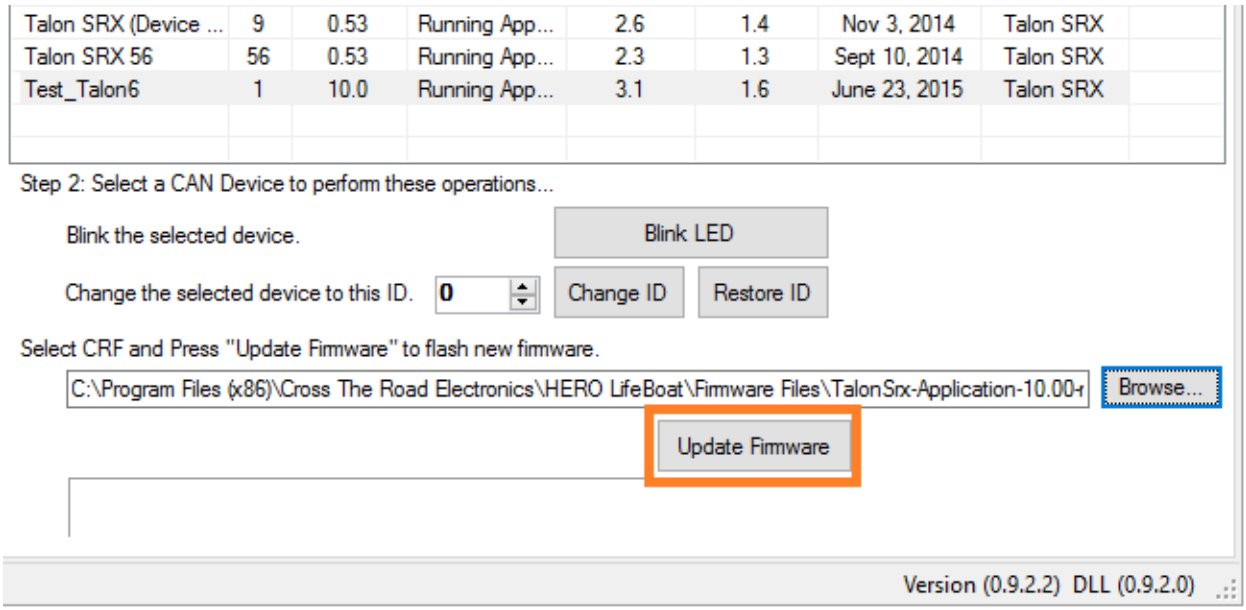
Next press the "Browse" button to select a firmware file. If the file path text entry is blank, LifeBoat will start in the "HERO Firmware Files" folder in the LifeBoat install path.
[Note: The current default install path is C:/Users/Public/Public Documents/Cross The Road Electronics/Lifeboat.]



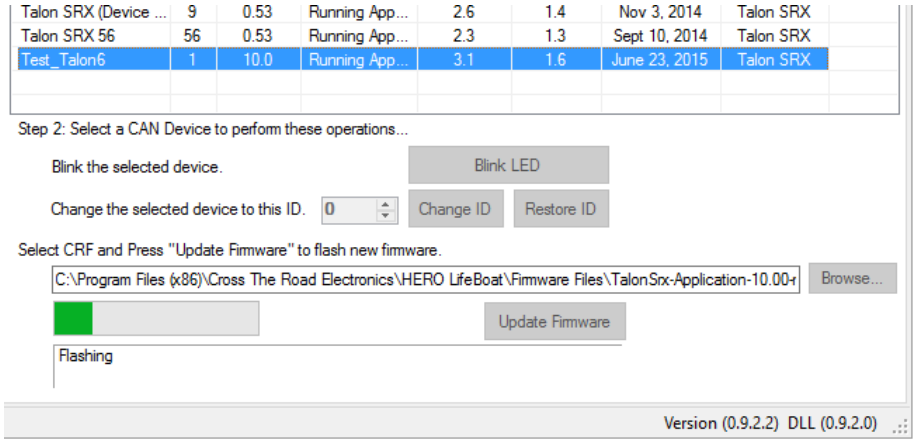
For example, if re-flashing a Talon SRX, choose the Talon SRX crf file. Then press "Open".



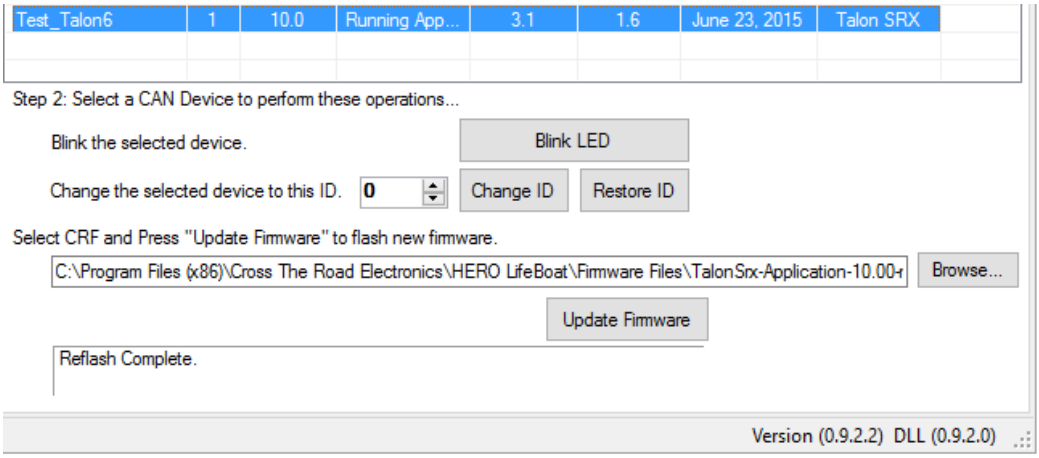
Next press "Update Firmware" to begin flashing the CAN device.



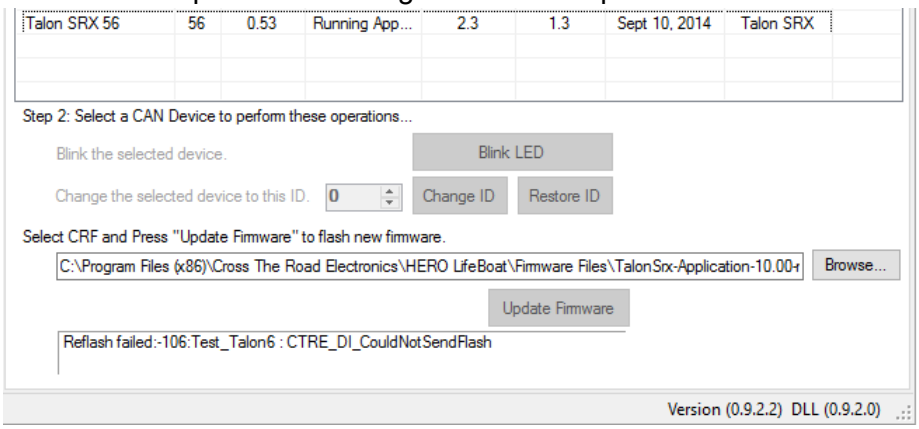
The interface will disable and a progress bar will appear.



When the field-upgrade is complete, the following success message will show. The entire flash session should take ~10 seconds and the version will be updated in the list view.



If power or communications is disrupted during the field-upgrade, LifeBoat will report the error and the CAN device will be left in bootloader mode (which is harmless). At which point you can simply try again after resolving the intermittent power/communication issue. In other words, the CAN devices are not susceptible to "bricking" due to incomplete flashes.

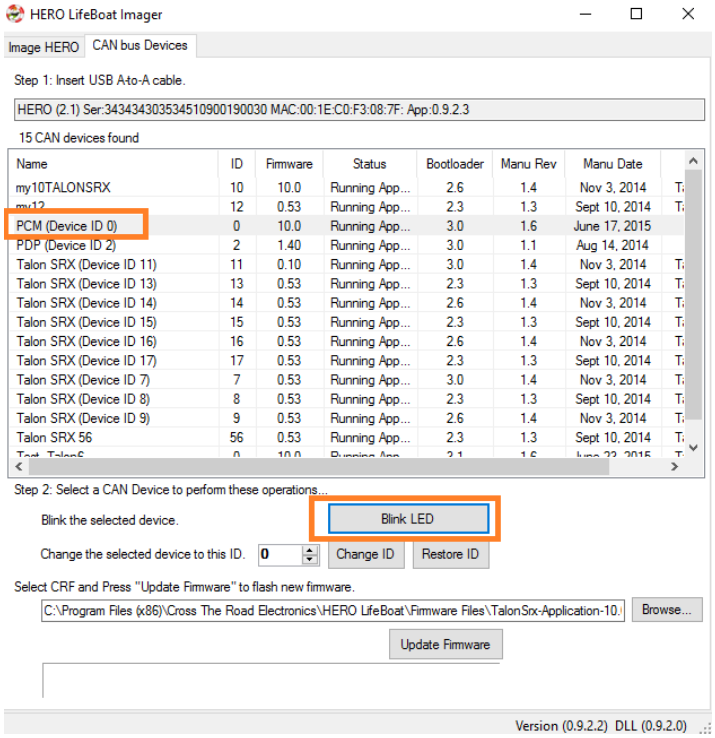


7.2.2.1. What happens if the wrong product's CRF is selected?

In both cases of field-upgrading the HERO, or updating a CAN device, if a CRF is selected that is meant for an entirely different product, LifeBoat will provide you with an error describing this. So there is no possibility of permanently damaging a CTRE product with mismatched firmware.

7.2.3. Blinking the LED(s) on a selected CAN Device.

Another fundamental operation of HERO LifeBoat is to confirm the device ID of a given CAN Device. Select the desired CAN device and then press the "Blink LED" button.

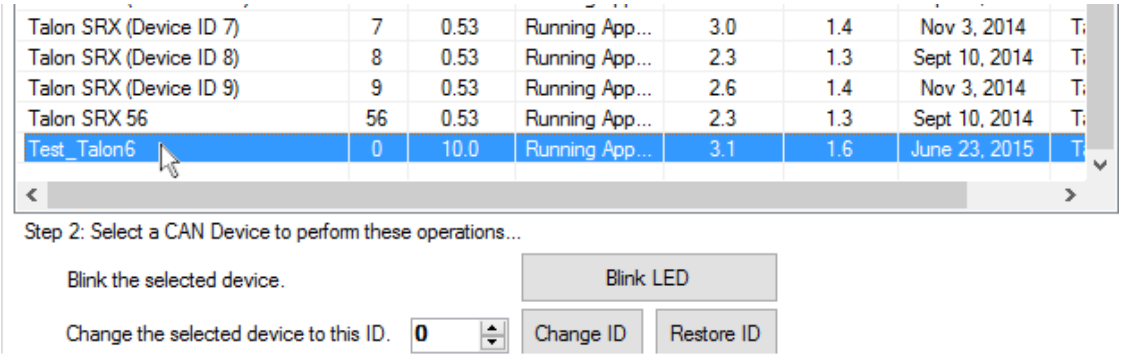


Then, in a moment, the device's status LED will rapidly blink. If the CAN device already has application firmware, it will rapidly blink **orange**. If the CAN device is in bootloader (perhaps the previous field-upgrade attempt was interrupted) then it will rapidly blink **green-orange**. Regardless of the state of the CAN device, it will respond in a unique fashion.

7.2.4. Change the Device ID.

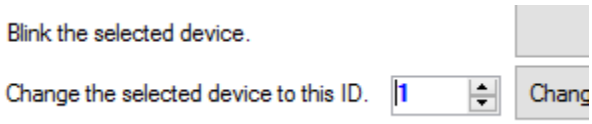
Although the "Blink LED" feature can be used to confirm or double-check the Device IDs, it is additionally important to be able to modify them to ensure that each device has a unique device ID.

First, select the device to modify.

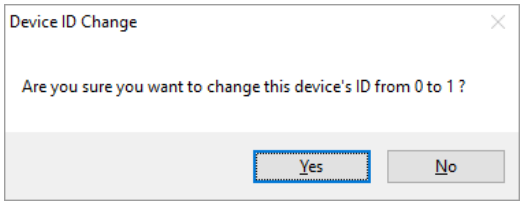


Then enter the new device ID into the number entry. If need be, the "Restore ID" button can be used to modify the number entry back to the device's current value. Beyond this, "Restore ID" has no effect on the actual CAN devices.

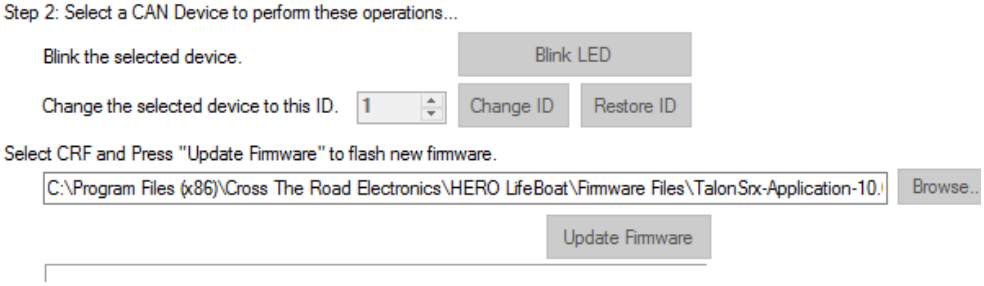
You will notice a blue color change in the number entry when it is modified.



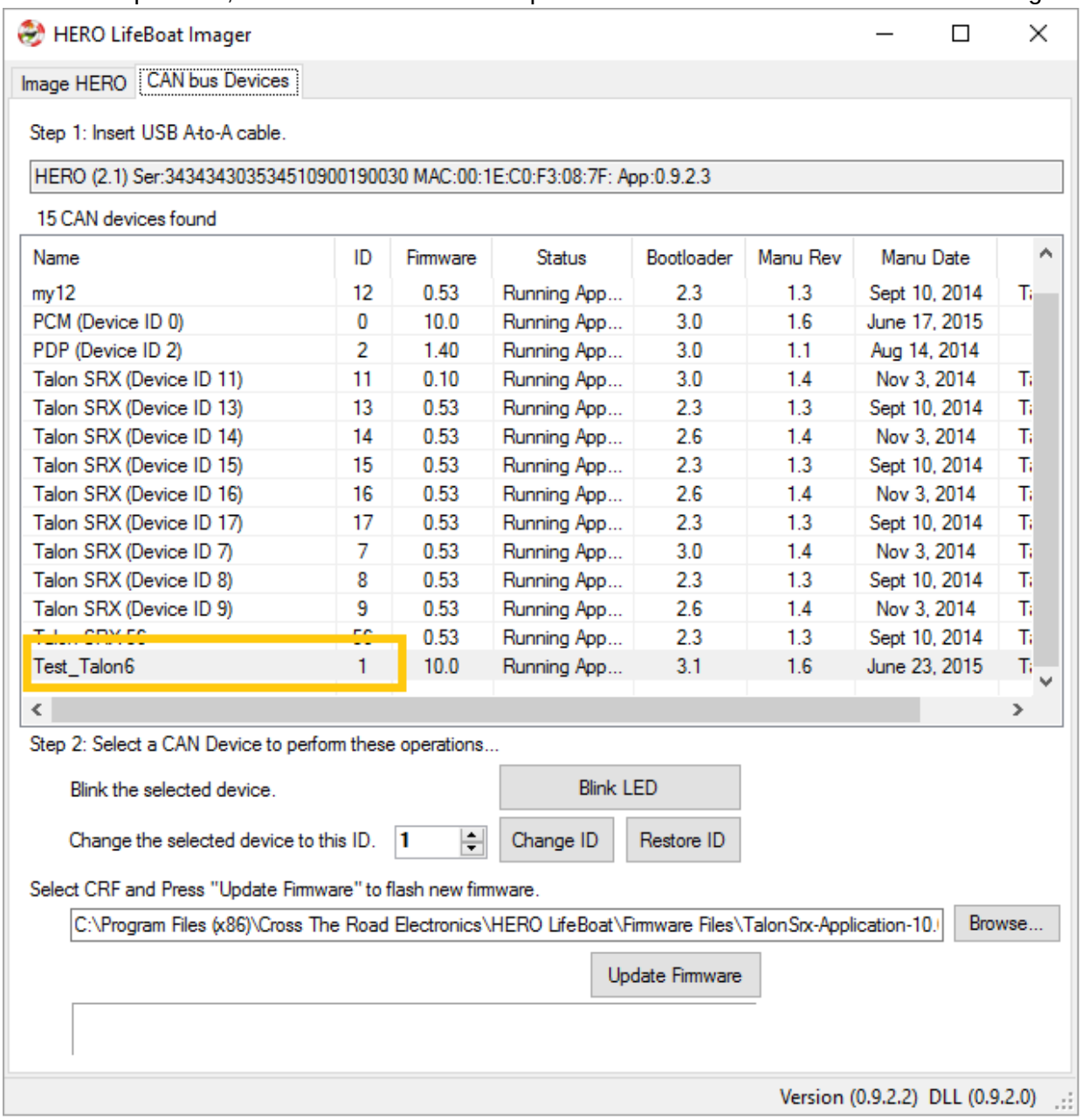
Then finally press the "Change ID" button. The form will prompt with a "Are you sure…" dialog box to confirm. Press "yes" if the IDs are correct.



The GUI will briefly disable and then automatically refresh the device list.



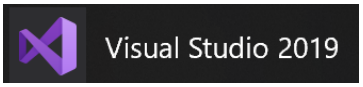
When the operation, is finished the ID will be updated and the interface will be selectable again.



8. Visual Studio – Getting Started

This section serves as a "Getting Started" guide that will start with creating a simple project, and then later will demonstrate adding code to control a Talon SRX with a gamepad.

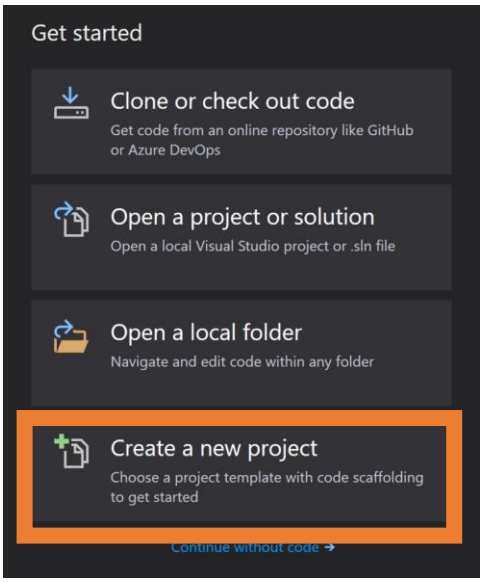
8.1. Creating your first project



Start by opening Visual Studio 2019 using the Desktop icon or by typing Visual Studio in the Windows Start Menu.

The first time opening Visual Studio after running the HERO-SDK-Installer, you may see the following prompt. This is just Visual Studio unpacking the HERO example templates. This can take 10-20 seconds.

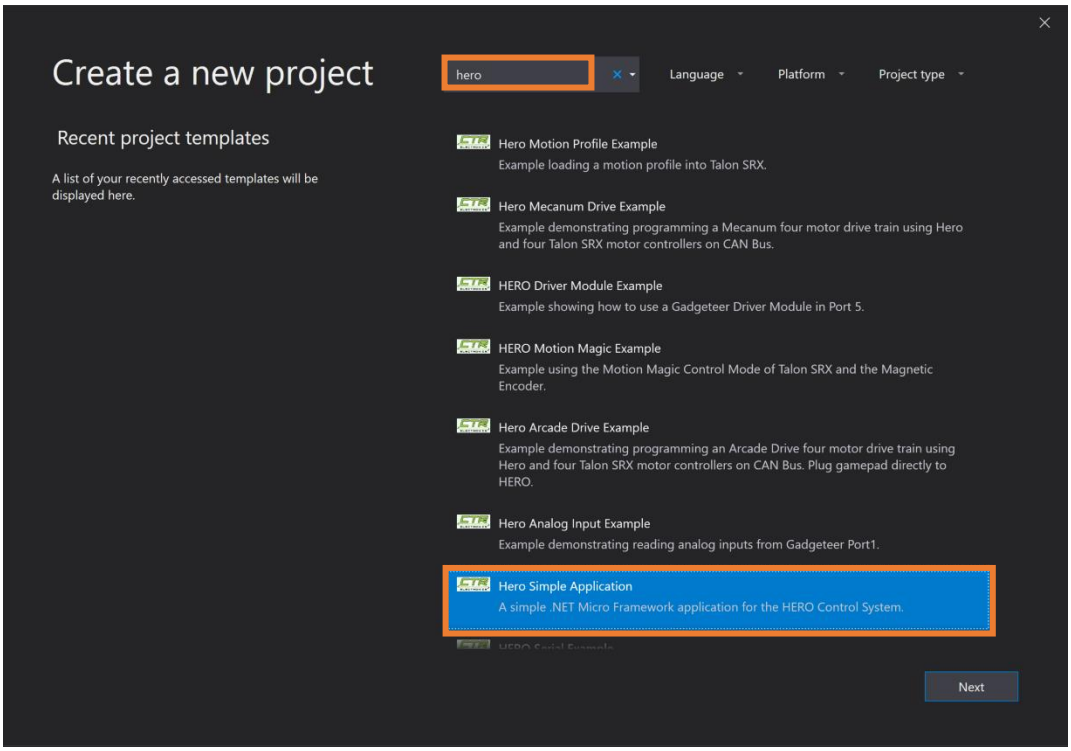
Start by opening Visual Studio and opening a new project from the start page.



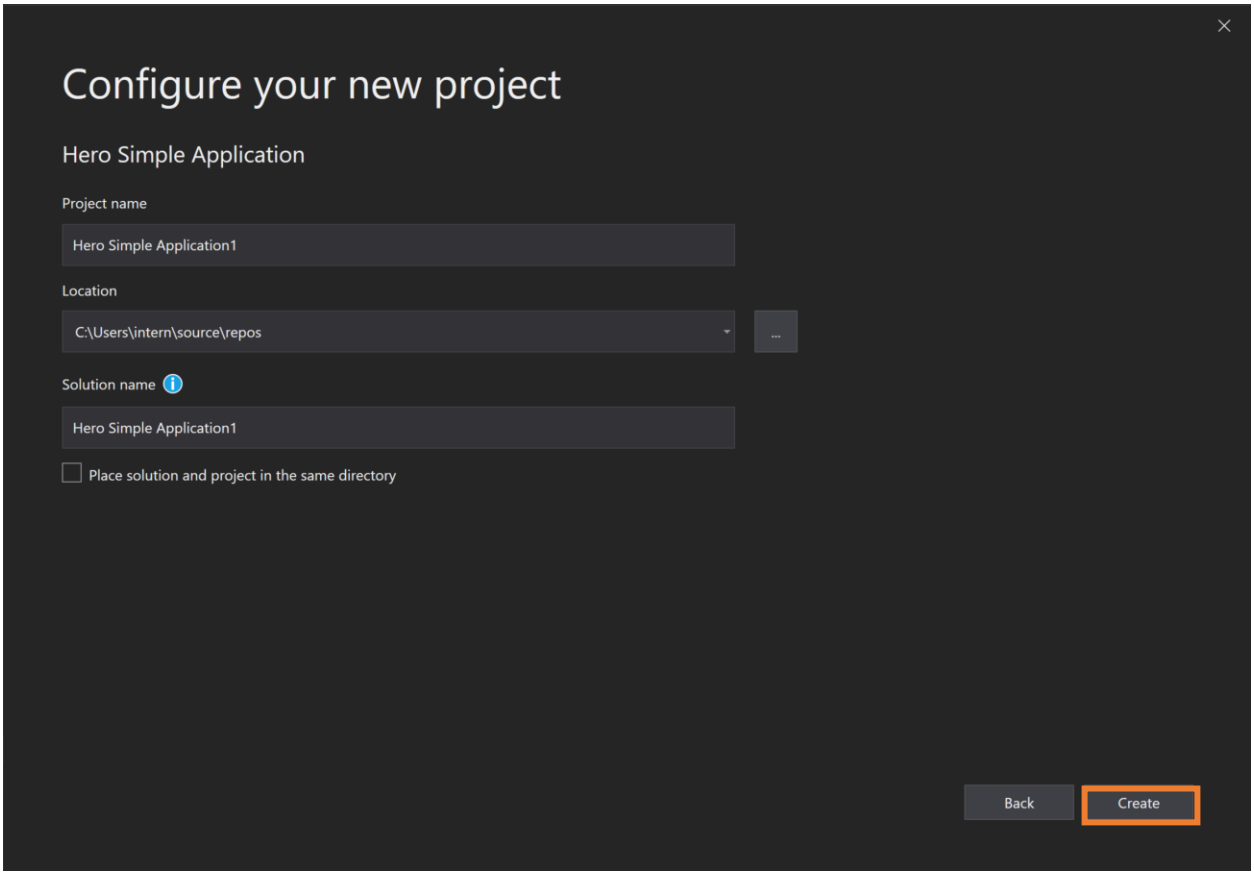
To begin, we will start with the HERO Simple Example. You can select this by typing "Hero" into the "Search for templates" box.

Then select the **HERO Simple Application**.

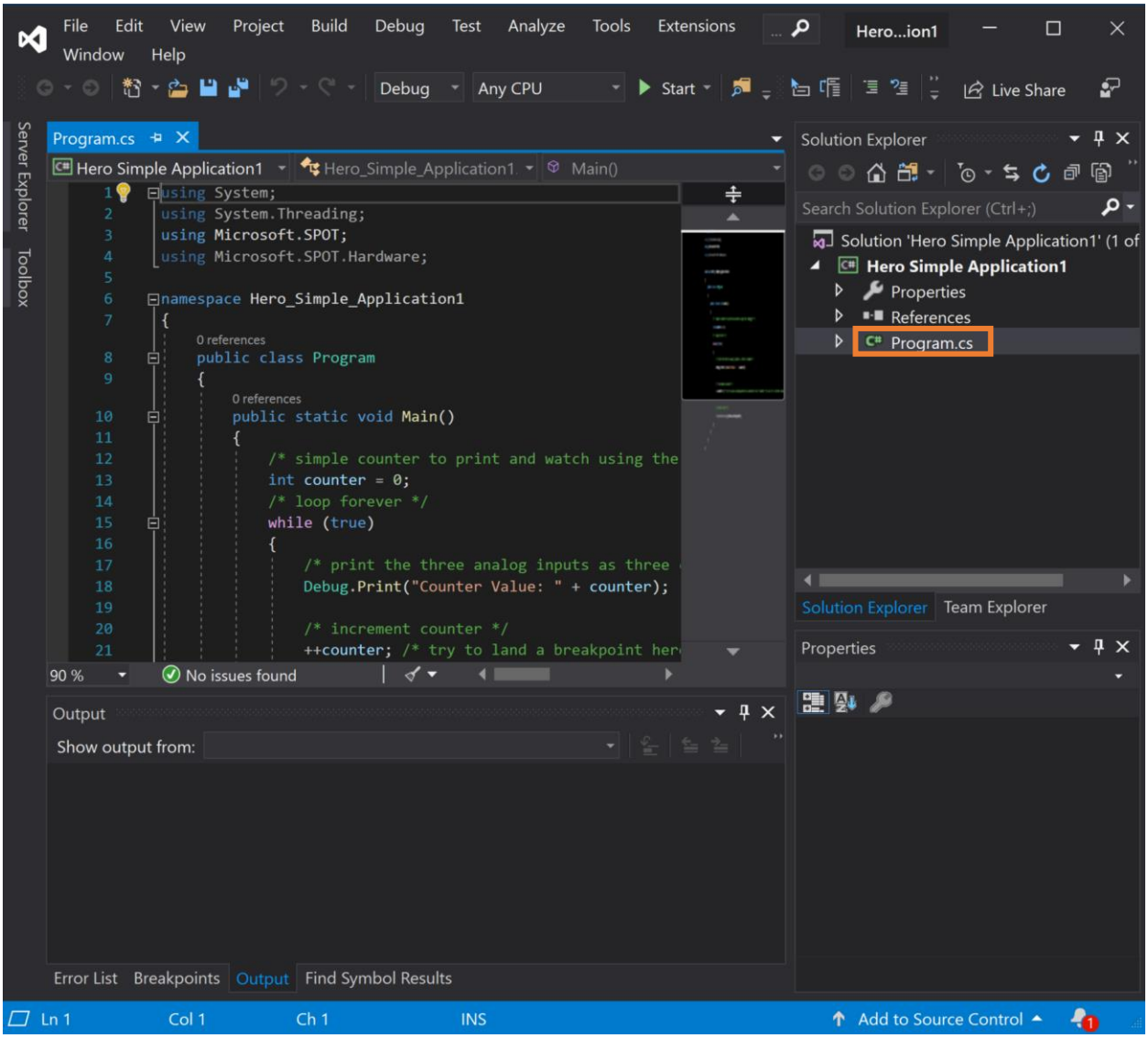
Then press "**Next**".



Name your project and then press "**Create**"

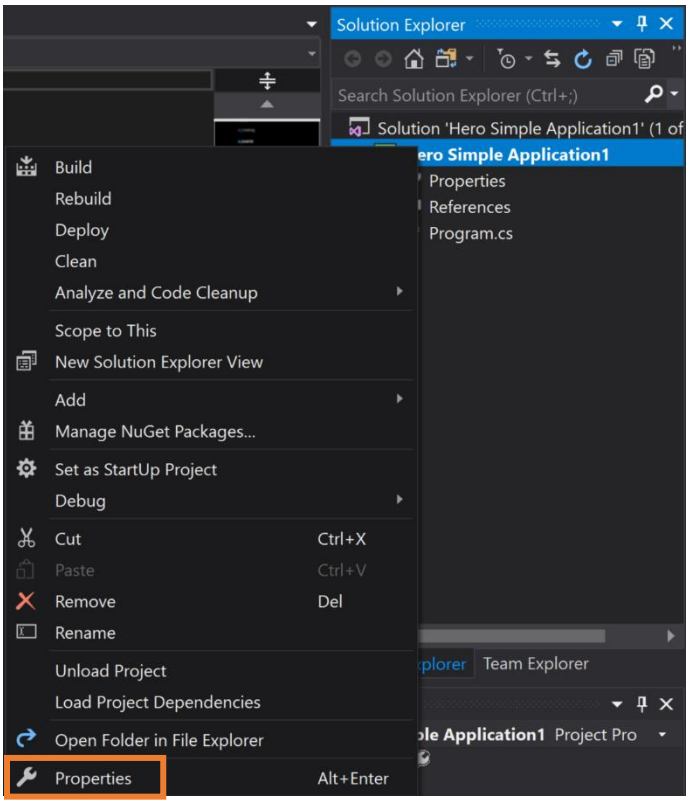


Double click on "**Program.cs**" in the **Solution Explorer** and the generated example should look like the screenshot below. This example will create a counter variable and print its incrementing value to the Output tab.



The next step is to debug/deploy this example to the HERO. To do these we will select the HERO as the target device (next section).

8.2. Selecting the HERO as the target device

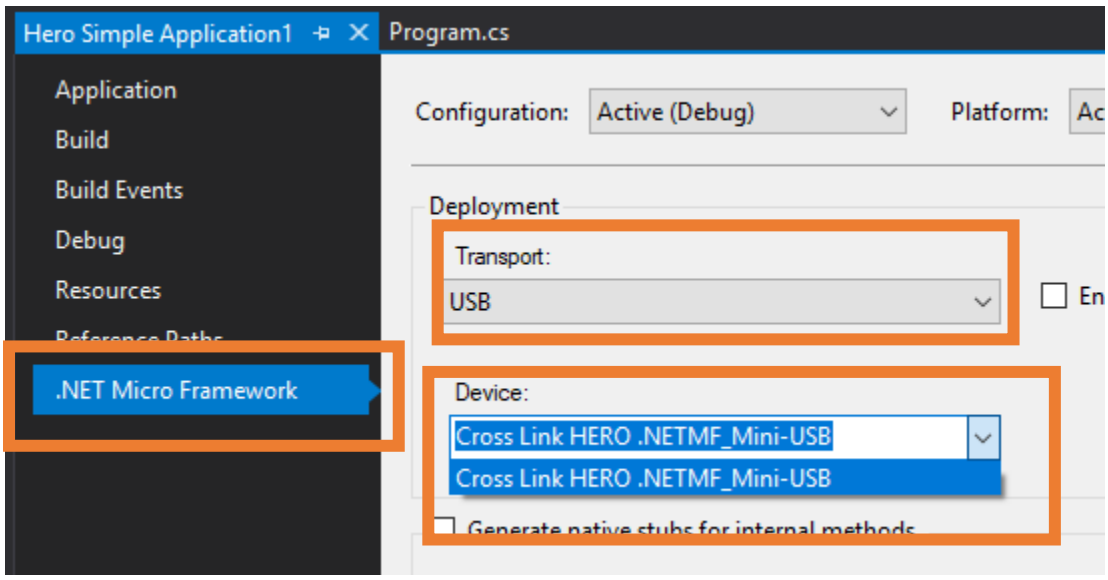


Right-click on the Project node and select Properties.

Select the **Micro Framework** Tab, select **USB as the Transport**, and (re)select the HERO as the device. It should appear as **"Cross Link HERO .NETMF_Mini-USB"**. **Always** click on the dropdown's down-arrow to repopulate the list of attached hardware, otherwise the selection may not take effect.

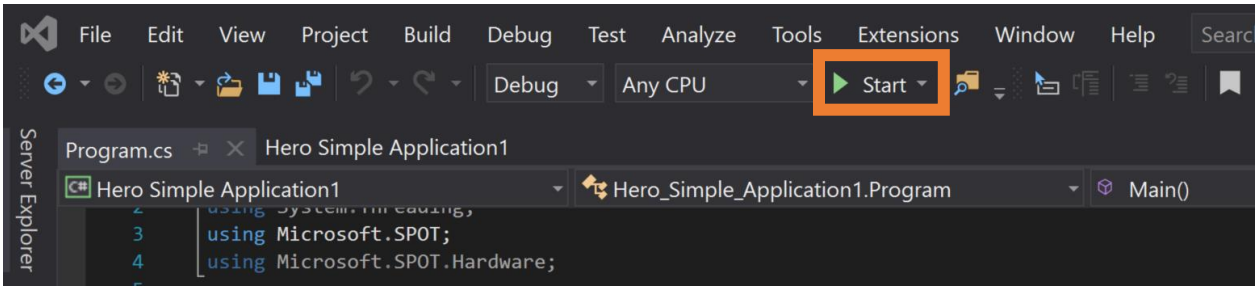
Requirement: Clicking the down arrow must be done even if the correct text is present in the text portion of dropdown.

Be sure to use the **mini USB Cable** and HERO's **NETMF** port.

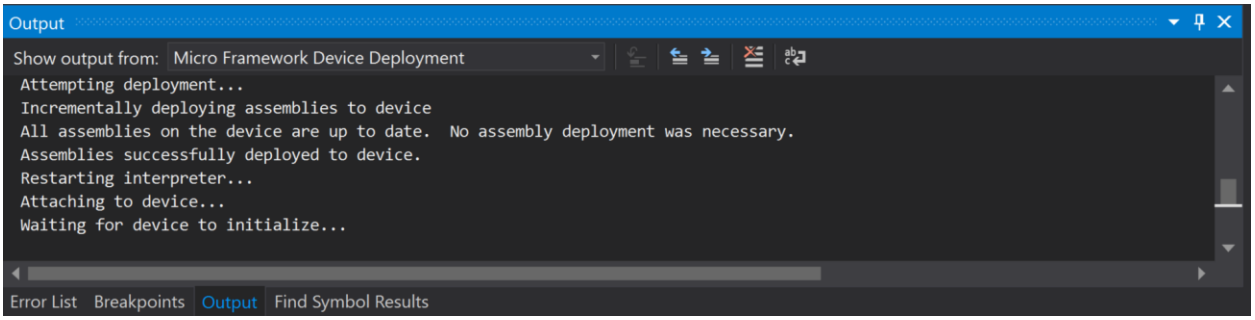


8.3. Deploying/Debugging your NETMF application

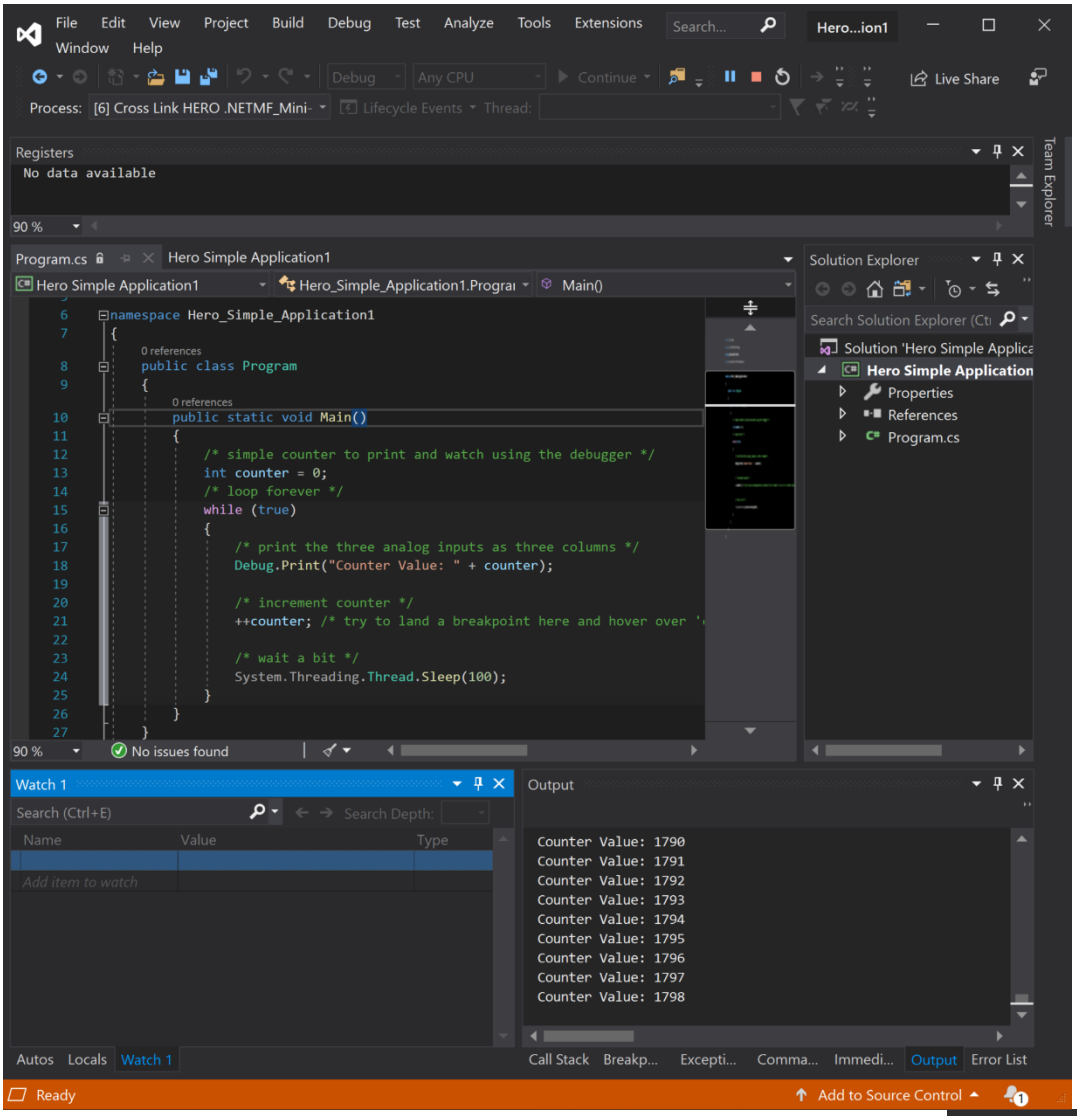
With the HERO selected as the target device, we can now press the "Start" button to launch the application.



The output window will first show connection status updates…



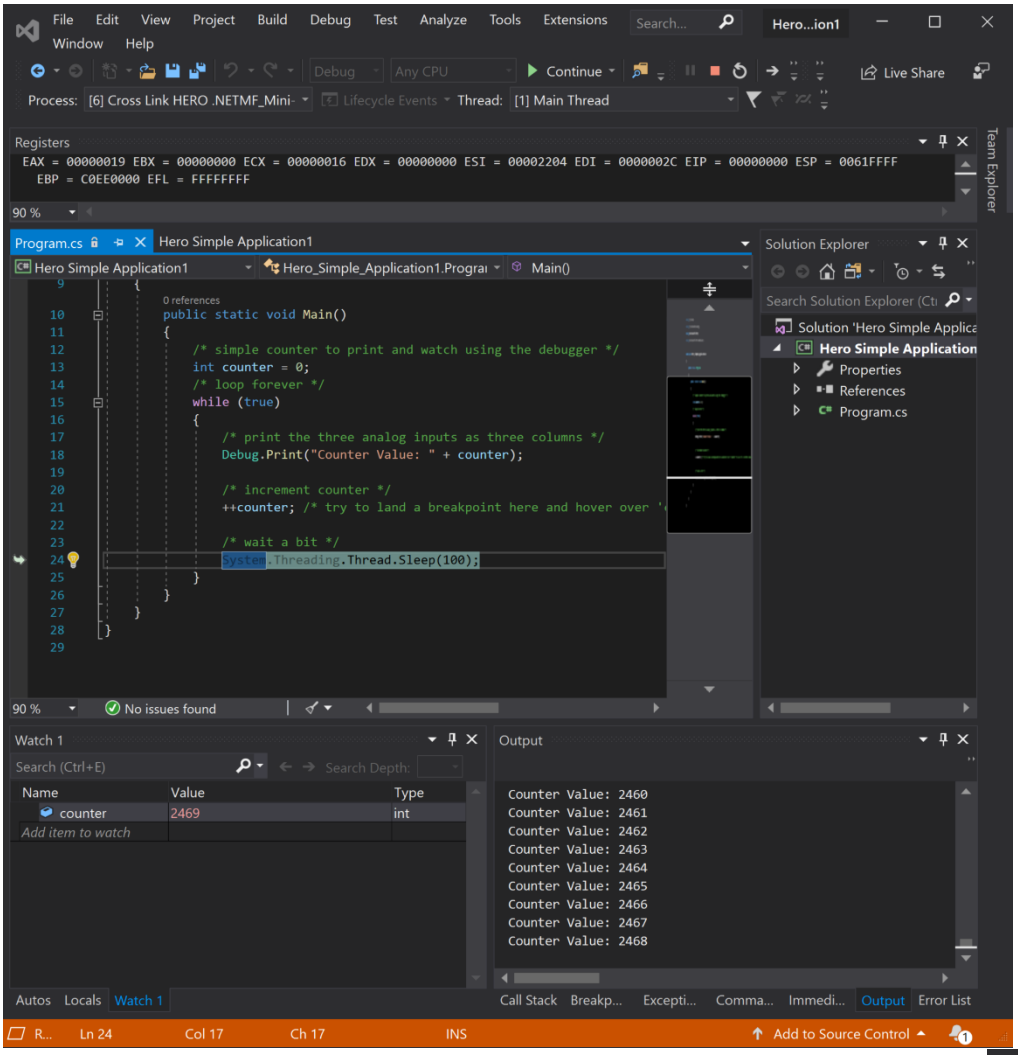
~10 seconds later Visual Studio will show the incrementing counter Value in the Output window. The HERO's STATUS LED will transition from blink **red** (no code) to blinking **orange** (running code but motor output is disabled).



Note that the "Break All" and "Stop Debugging" icons are now enabled. 

The  button can be used to pause the code execution, allowing you to see where it left off.

Then use  to resume execution.

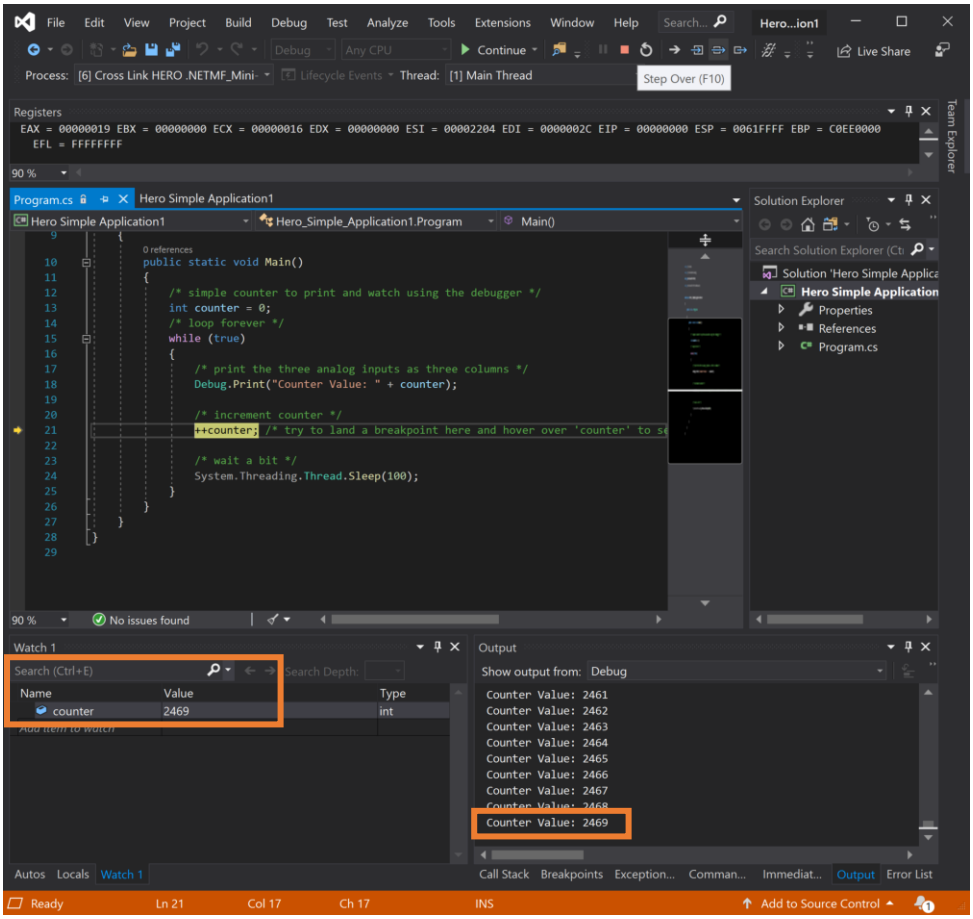


In this screenshot, the code stopped at the sleep function after pressing the  button. By adding the counter value to the Watch window, we can see the value inside the variable "counter".

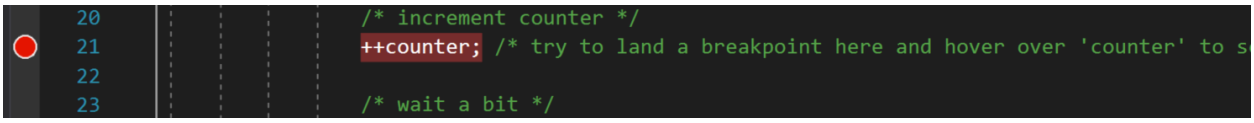
Notice that because of where we stopped, the counter value is 2469. However the last print command printed 2468. This is because the increment was done after the print step.

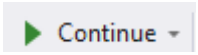
Also note that the HERO STATUS LED will blink red when the debugger has halted program execution.

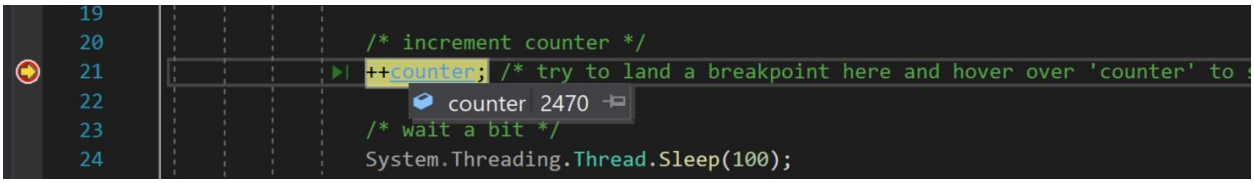
The stepping menu bar can also be used  to slowly execute the lines of code one line at a time. Press () to step over `Debug.Print("Counter Value: " + counter)` and note the newly printed line in the Output window matching the current value of "counter".



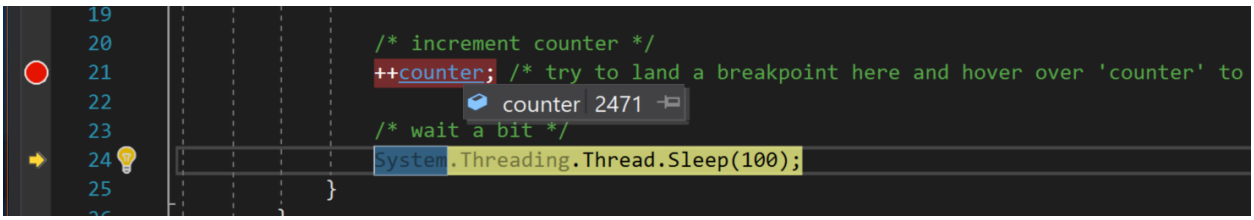
Now let's use a breakpoint to watch "counter" change. Left click in the left gutter on the line that increments "counter". You will see a red dot when this is done.



Press  to resume program execution and Visual Studio will automatically halt execution when this line is reached. Hover over "counter" with the mouse to see the latest value.



Then step over ( ), and re-hover to see its new incremented value.



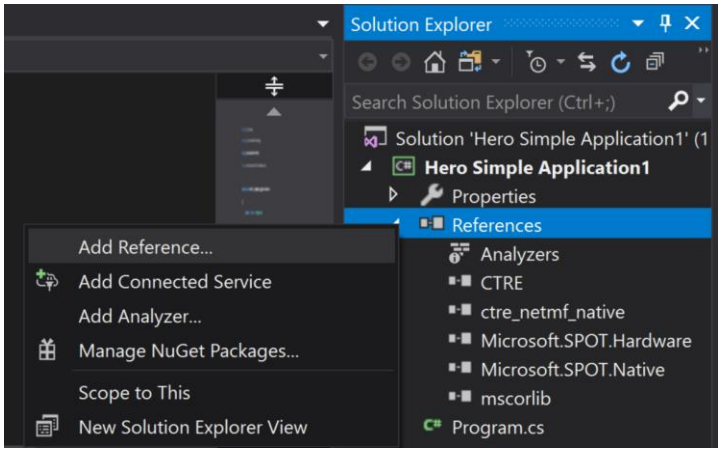
Press the stop icon to disconnect Visual Studio, and to allow code edits. Next we will add the CTRE Library Reference and control a Talon SRX (CAN Bus) using a USB gamepad.

8.4. Adding CTRE Libraries to NETMF projects.

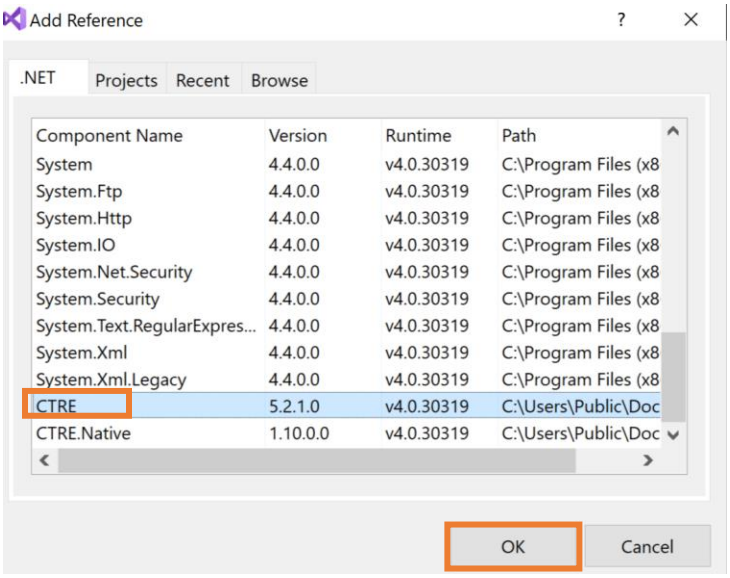
The CTRE Libraries are contained in the CTRE assembly file (at the time of writing this document). If the CTRE Libraries need to be added to an already existing NETMF project, you will need to add a reference to the installed CTRE assembly.

Generally speaking, the HERO examples will already have this done for you, but this is helpful to know when starting projects from scratch, or from other sources outside of the HERO-SDK.

This can be done by right-clicking on the "References" tree node in the Solution explorer, and clicking "Add Reference".



This will bring up "Add Reference" Dialog. Select "CTRE" and press "OK". CTRE is an available option because the HERO-SDK has already registed it.



8.5. Adding a USB gamepad

Let's add some code to get information from an attached gamepad. First, create a Gamepad object inside the first line of Main. We will call it "myGamepad". The `new` operators on the right side of the assignment (=) are used to create a new Gamepad object that will look at the HERO USB Host/Dev port. In the future, we will have other sources of gamepad information, so this helps delineate gamepads plugged into HERO versus remote-gamepad-strategies to be added in the future.

```
/* create a gamepad object */
CTRE.Phoenix.Controller.GameController myGamepad = new
CTRE.Phoenix.Controller.GameController (new CTRE.Phoenix.UsbHostDevice(0));
```

Then inside our while loop, we will check if a gamepad is connected and if so print the axis value at index '1'. Typically on most gamepad's, axis '0' is the first X-axis, and axis '1' is the first Y-axis. The axis value will range from -1 (stick pushed forward), to 1 (stick pulled back).

We will also call `Feed()`, which will signal HERO that it's okay to drive motor outputs. See Section 9.2 for more information on this function. This also gives us a safe means of disabling motor outputs (by disconnecting the gamepad).

The HERO STATUS LED will change to blinking **green** when a USB gamepad is inserted as a result of this change.

```
/* added inside the while loop */
if (myGamepad.GetConnectionStatus() == CTRE.Phoenix.UsbDeviceConnection.Connected)
{
    /* print the axis value */
    Debug.Print("axis:" + myGamepad.GetAxis(1));
    /* allow motor control */
    CTRE.Phoenix.Watchdog.Feed();
}
```

Finally, remove the original Debug.Print and change the sleep routine parameter to 10. Ten milliseconds is fast enough for reliable motor control and will call Feed() often enough to keep motor output enabled.

```
/* wait a bit */
System.Threading.Thread.Sleep(10);
```

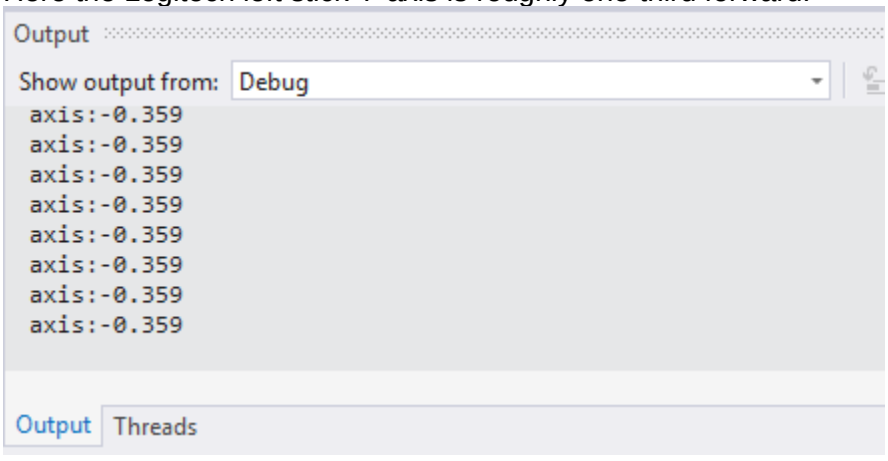
The entire example now looks like this…

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace Hero_Simple_Application5
{
    public class Program
    {
        public static void Main()
        {
            /* create a gamepad object */
            CTRE.Phoenix.Controller.GameController myGamepad = new
CTRE.Phoenix.Controller.GameController(new CTRE.Phoenix.UsbHostDevice(0));

            /* simple counter to print and watch using the debugger */
            int counter = 0;
            /* loop forever */
            while (true)
            {
                /* added inside the while loop */
                if (myGamepad.GetConnectionStatus() == CTRE.Phoenix.UsbDeviceConnection.Connected)
                {
                    /* print the axis value */
                    Debug.Print("axis:" + myGamepad.GetAxis(1));
                    /* allow motor control */
                    CTRE.Phoenix.Watchdog.Feed();
                }

                /* increment counter */
                ++counter; /* try to land a breakpoint here and hover over 'counter' to see it's current
value.  Or add it to the Watch Tab */

                /* wait a bit */
                System.Threading.Thread.Sleep(10);
            }
        }
    }
}
```

Now start this example and the Output window will show the current position of the Gamepad axis in the output window.

Additionally the HERO Status LED will blink **green** instead of **orange**.

Here the Logitech left stick Y-axis is roughly one-third forward.



8.6. Adding a Talon SRX

First we must create the Talon SRX. Use HERO LifeBoat to determine the device ID of the attached Talon SRX. Talons have a factory-default device ID of '0'. If it has not been done already, use LifeBoat to firmware update the Talon to non-FRC firmware (See Section 7.2.2).

```
/* create a talon, the Talon Device ID in HERO LifeBoat is zero */
CTRE.Phoenix.MotorControl.CAN.TalonSRX myTalon = new
CTRE.Phoenix.MotorControl.CAN.TalonSRX(0);
```

Next we will pass the axis value into the Talon's Set() routine. Although Talon supports various control modes, the default one simple takes a value between -1 (full reverse) and 1 (full forward) and applies the appropriate percent output to the motor. More information is available in the Talon SRX Software Reference Manual.

```
  /* pass axis value to talon */
  myTalon.Set(CTRE.Phoenix.MotorControl.ControlMode.PercentOutput, myGamepad.GetAxis(1));
```

The entire example now looks like this.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace Hero_Simple_Application5
{
    public class Program
    {
        public static void Main()
        {
            /* create a gamepad object */
            CTRE.Phoenix.Controller.GameController myGamepad = new CTRE.Phoenix.Controller.GameController(new
CTRE.Phoenix.UsbHostDevice(0));
            /* create a talon, the Talon Device ID in HERO LifeBoat is zero */
            CTRE.Phoenix.MotorControl.CAN.TalonSRX myTalon = new CTRE.Phoenix.MotorControl.CAN.TalonSRX(0);


            /* simple counter to print and watch using the debugger */
            int counter = 0;
            /* loop forever */
            while (true)
            {
                /* added inside the while loop */
                if (myGamepad.GetConnectionStatus() == CTRE.Phoenix.UsbDeviceConnection.Connected)
                {
                    /* print the axis value */
                    Debug.Print("axis:" + myGamepad.GetAxis(1));
                    /* pass axis value to talon */
                    myTalon.Set(CTRE.Phoenix.MotorControl.ControlMode.PercentOutput, myGamepad.GetAxis(1));
                    /* allow motor control */
                    CTRE.Phoenix.Watchdog.Feed();
                }

                /* increment counter */
                ++counter; /* try to land a breakpoint here and hover over 'counter' to see it's current value.
Or add it to the Watch Tab */

                /* wait a bit */
                System.Threading.Thread.Sleep(10);
            }
        }
    }
}
```

8.7. Deploying for release

When Visual Studio sends the application, it is already deployed in persistent flash. This means that it will auto start your application when the HERO is disconnected from Visual Studio. This also means that it will start within seconds of power booting.

9. Visual Studio C# Class Library

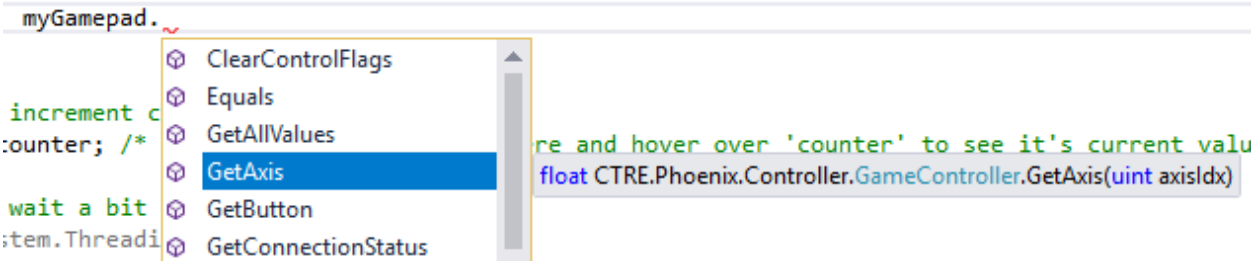
9.1. USB Gamepad/Joystick

To read the signals of a HID user device connected to HERO's Host/Dev USB port, create a Gamepad object.

```
CTRE.Phoenix.Controller.GameController myGamepad = new
CTRE.Phoenix.Controller.GameController(new CTRE.Phoenix.UsbHostDevice(0));
```

For Xbox/X-Input controllers, use the `Xbox360Gamepad` class instead of `GameController`.

Type the line "myGamepad." to see the available function list using the IntelliSense…



`GetAxis()` and `GetButton()` can be used to retrieve axis and button information. `GetConnectionStatus()` can be used to determine if a gamepad is present.

`GetAllValues()` can be used to retrieve a snapshot of all supported signals. This is helpful for signals that are not supported in the API yet (such as retrieving the VID/PID or POV-hat-switch values).

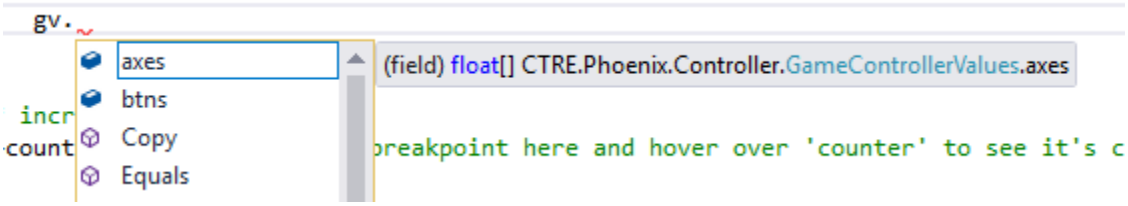
First create a `GamepadValues` object. This ideally would only be done once.

```
CTRE.Phoenix.Controller.GameControllerValues gv = new
CTRE.Phoenix.Controller.GameControllerValues();
```

Then fill its contents using `GetAllValues()`.

```
myGamepad.GetAllValues(ref gv);
```

Using IntelliSense, we can retrieve the updated signals.



9.2. Emergency Stop, Disabling motor drive

Since the HERO is a multi-purpose device for custom applications, the enable-disable procedure has to be flexible in meeting the requirements of the user. To meet this, a software watchdog has been added to the class library.

When a NETMF application is deployed to the HERO, the HERO's STATUS LED will blink **orange** indicating that code is running, but motor output (and PCM compressor/solenoid output) is disabled. You can still call the various set routines for Talon SRX and PCMs, however the motor outputs, solenoid outputs, and compressor output will be off (you may still modify the brake behavior of Talons).

In order to enable the HERO, the application must call `CTRE.Phoenix.Watchdog.Feed()` periodically. Ideally the application will call this routine when the HERO can confirm that it is safe to do so. For example, when using a USB gamepad to control a robot, the application should be written to only call `Feed()` when the USB gamepad is inserted. This can be done by periodically checking the gamepad connection and conditionally calling `Feed()`.

```
/* this is checked periodically.  Recommend every 20sm or faster */
if (myGamepad.GetConnectionStatus() == CTRE.Phoenix.UsbDeviceConnection.Connected)
{
    /* allow motor control */
    CTRE.Phoenix.Watchdog.Feed();
}
```

For bench-testing or applications that are considered safe, regardless of Talon state, it may be acceptable to always call `Feed()` thereby always leaving HERO in an enabled state. However we **highly** recommend coding a way to ensure you can reliably disable motor outputs under any circumstance.

When using a Logitech gamepad, it is beneficial to leverage the 'X' switch to disable motor outputs.

Other gamepad/joysticks may also have a hard switch that can be used to disable. For example a MODE button could be used to signal HERO to disable (in addition to gamepad presence).

Another example would be to measure an analog/digital signal that can behave as a kill switch.

9.3. CAN Devices (Talon SRX, PCM, PDP).

Classes are available for all CTRE CAN devices except Talon FX and CANCoder – these will be available in a future update.
The Talon SRX and Victor SPX classes are very similar to the C++ and Java classes documented at https://phoenix-documentation.readthedocs.io/en/latest/index.html.

9.4. Peripherals

9.4.1. Digital I/O

Pins labelled as GPIO pins in the Gadgeteer Port Table can be used for general Digital Input and Output.

9.4.1.1. CPU Pin Mappings

The CPU utilized by the HERO has all Pins from the CPU categorized under alphabetical Port headings. (eg. A, B, C, etc.)

The Cpu.Pin numbers in the Microsoft.Spot.Hardware library are in hexadecimal format (0x##) where the first hexadecimal digit is determined by the CPU Port (A = 0, B = 1, etc.) and the second digit by the Pin number within that Port.

Some example Cpu.Pin numbers are as follows:

CPU Port	CPU Port Pin	Cpu.Pin Reference
A	1	0x01
A	2	0x02
B	2	0x12
C	7	0x27

These Cpu.Pin numbers are then used to map the CPU Pins to Gadgeteer Port Pins in the CTRE.HERO.IO Library so that the user may reference Pins using the method outlined later in this section (Digital Outputs and Digital Inputs)

The full list of CPU Pin mappings for the HERO Gadgeteer Ports can be viewed in the CTRE.HERO.IO section of the HERO SDK which can be found on the CTRE GitHub.

9.4.1.2. Digital Outputs

Gadgeteer Port Pins are mapped in the CTRE.HERO.IO library such that a Digital Output may be created from the following example…

```
OutputPort digitalOut1 = new OutputPort(CTRE.HERO.IO.Port1.Pin4, false);
```

…where the Port and Pin are mapped to the corresponding CPU pin in CTRE.HERO.IO.
The second parameter of the constructor indicates the starting state of the output pin – 'true' for logic high (3.3 V), 'false' for logic low.

The Digital Output may then be set to high or low using the Write() function.

```
digitalOut1.Write(true); //sets Output to Logic High
```

9.4.1.3. Digital Inputs

Similar to a Digital Output, a Digital Input may be created using the following example:

```
InputPort inputTest = new InputPort(CTRE.HERO.IO.Port1.Pin5, false,
       Port.ResistorMode.Disabled);
```

It is important to note the ResistorMode input. When used as input, DIO pins on the HERO are 'floating' and can be at any value. In order to ensure accurate results, it is **highly** recommended to use a pull-up or pull-down resistor (depending on your application) and set the ResistorMode accordingly in your project.

Once set up, a digital input may be read as follows:

```
bool read0 = inputTest.Read();
```

9.4.2. Analog Inputs

The HERO Gadgeteer ports 1 and 8 each can be used for analog inputs. Each port contains three analog inputs giving a maximum total of six.

Creating an analog input can be done with the following example…

```
AnalogInput analogInput0 = new AnalogInput(CTRE.HERO.IO.Port1.Analog_Pin3);
```

…where the Port# is the desired HERO Gadgeteer port and the Analog_Pin# is designated in the CTRE.HERO.IO library for the appropriate Gadgeteer Ports.

The value can then be read using the `Read()` function. The return is a double-precision value between '0' and '1'. '1' represents full voltage (3.3V).

```
/* grab analog value */
double read0 = analogInput0.Read();
```

Each Analog_Pin# in CTRE.HERO.IO is mapped to the appropriate Analog Channel in Microsoft.Spot.Hardware. The mappings are as shown below.

Physical Pin	Channel
PORT 1 - Pin3	Analog Input 0
PORT 1 – Pin4	Analog Input 1
PORT 1 – Pin5	Analog Input 2
PORT 8 - Pin3	Analog Input 3
PORT 8 – Pin4	Analog Input 4
PORT 8 – Pin5	Analog Input 5

9.4.3. UART

Gadgeteer Ports 1, 4, and 6 can be used for UART communication. Each of these Gadgeteer Port utilizes a different COM Port on the CPU, so each Gadgeteer Port may be used as an independent UART bus.

The COM ports are mapped in CTRE.HERO.IO, and may be initialized using a desired baud rate as follows:

```
System.IO.Ports.SerialPort _uart = new
       System.IO.Ports.SerialPort(CTRE.HERO.IO.Port1.UART, 115200);
```

You may then read or write using the Write() and Read() methods which require a byte buffer for data, an offset to indicate the start of the desired data within the buffer, and the number of bytes to be sent or received. Similar to:

```
byte[] toWrite = new byte[8];

/*Insert Data into toWrite Here*/

_uart.Write(toWrite, 0, 8);
```

9.4.4. PWM

NOTE: Please update to the latest version of firmware to use PWM.

Gadgeteer Port 3 can be used for PWM output. Pins 4, 6, 7, 8, and 9 are supported for PWM.

Available PWM pins are mapped in CTRE.HERO.IO and can be initialized as follows:

Using Microsoft.SPOT.Hardware, Microsoft.SPOT.Hardware.PWM, and CTRE.HERO.IO:
NOTE: You will need to add Microsoft.SPOT.Hardware.PWM reference as detailed in 8.4

```
uint period = 50000; //period between pulses
uint duration = 1500; //duration of pulse
PWM pwm_9 = new PWM(CTRE.HERO.IO.Port3.PWM_Pin9, period, duration,
       PWM.ScaleFactor.Microseconds, false);
```

The period and duration are in the units set by the PWM ScaleFactor (in this case microseconds). The last field in the constructor sets the Boolean to invert the PWM signal.

The signal then needs to be started:

```
pwm_9.Start(); //starts the signal
```

The duration of the pulse can be changed as follows:

```
pwm_9.Duration = newDuration;
```

9.4.4.1. PWM Speed Controller Class

HERO has a class called **PWMSpeedController** that can be used to drive motor controllers with a PWM signal using PWM pins on the HERO. PWMSpeedController obeys the enable state of HERO as described in section 9.2.

When constructing a PWMSpeedController object, the first parameter is the PWM channel being used. An optional second parameter may be included to specify the period between pulses. The default period is 10 ms.

Example:

```
PWMSpeedController victor = new PWMSpeedController(CTRE.HERO.IO.Port3.PWM_Pin9);
```

The percent Vbus can be set to a value from -1 to 1 using the Set() method. Setting the percent Vbus to 0 would look like this:

```
victor.Set(0);
```

An example on how to drive a PWM controller using a joystick can be found on the CTRE Github.

Additionally, the PWMSpeedController class supports the following:

- **Invert** the direction of the motor using SetInverted(), which takes a Boolean argument. The current state of the inverted flag can be obtained as the returned argument of GetInverted().

- **Change the period** between pulses using SetPeriod(), which takes an unsigned integer argument. This argument is in units of milliseconds.

  Alternatively, the period may be set as the second argument of the constructor when creating a new controller object.

  The current value (in milliseconds) of the period can be obtained as the return argument of GetPeriod().

- **Enable/Disable:** The PWM signal may be enabled or disabled using Enable() and Disable(). When an object is constructed, the signal starts enabled by default. When disabled, the signal generates no pulses. If you wish to drive the controller to neutral, use Set(0) instead.

The full PWMSpeedController class may be found on the CTRE Github.

9.4.5. I2C

HERO has support for I2C devices. Both I ports on HERO utilize the same I2C bus, so while both connections may be used, all devices will receive the same data from HERO.

There is an example on the CTRE GitHub on how to set up and use I2C on hero. It can be found here.

An I2C device is created using an address and clock rate:

```
I2CDevice.Configuration SonarConfig = new I2CDevice.Configuration(DeviceAddress, ClockRate);
MyI2C = new I2CDevice(SonarConfig);
```

An I2C device can be read from or written to using the device's 1-byte address and the 1-byte container for data:

```
//I2C write function that takes the address and data
      private static void I2CWrite(byte Address, byte Data)
      {
          WriteCommand = new I2CDevice.I2CTraaction[1];
          WriteCommand[0] = I2CDevice.CreateWriteTransaction(new byte[2]);
          WriteCommand[0].Buffer[0] = Address;
          WriteCommand[0].Buffer[1] = Data;
          MyI2C.Execute(WriteCommand, 100);
      }
//I2C read function that takes the address, and buffer, and returns amount of
transactions
      private static int I2CRead(byte Address, byte[] Data)
      {
          ReadCommand = new I2CDevice.I2CTransaction[2];
          ReadCommand[0] = I2CDevice.CreateWriteTransaction(new byte[] { Address });
          ReadCommand[1] = I2CDevice.CreateReadTransaction(Data);
          ReadCheck = MyI2C.Execute(ReadCommand, 100);
          return ReadCheck;
      }
```

9.4.6. SPI

Like I2C, both SPI ports on HERO utilize a single bus. Both hardware ports may be connected, however the same master signal will be sent to all devices on either connection.

Examples of the SPI API can be seen in the Display Module example on the CTRE GitHub here.

9.5. Modules

Gadgeteer Modules from CTR Electronics have their own classes within the CTRE Class Library. They are located within CTRE.Gadgeteer.Module, and the source code can be found on the CTRE Github.

9.5.1. Driver Module

The CTRE Gadgeteer Driver Module is supported by the **DriverModule** class.

When construction a DriverModule object, the first and only argument must be the port definition of the gadgeteer port the module is physically connected to. These port definitions are contained in CTRE.HERO.IO.

Since the Driver Module requires a 'Y' type socket, the port provided must support 'Y' sockets. For HERO, these are ports 3 and 5. An example is shown below:

```
CTRE.Gadgeteer.Module.DriverModule driver = new
CTRE.Gadgeteer.Module.DriverModule(CTRE.HERO.IO.Port5);
```

The DriverModule class contains two methods to support functionality:

- Set() sets an output to a desired state and takes two arguments. The first argument is the numeric ID of the output being set, and the second is the state that output is being set to. The output can be in one of two states: driveLow and pullUp. Both are defined within DriverModule.OutputState. Since the Driver Module is a low side driver, setting the state to driveLow will provide power to and enable the device that is connected. Setting the state to pullUp will pull the voltage output back to the high voltage level, disabling the output.

  For example, to set output 3 to the driveLow state, you would call:
  `driver.Set(3, DriverModule.OutputState.driveLow);`

- Get returns the current state of an output and takes a single argument. This argument is the numeric ID of the output being checked.

- Examples of the Driver Module API can be seen in the Driver Module example on the CTRE GitHub here.

9.6. Timing

The Microsoft .netmf API offers an option for timing based on processor ticks.

The TimeSpan class offers definitions of the number of ticks in each millisecond, second, minute, hour, and day. It can be accessed as shown below:

```
long kTicksMs = TimeSpan.TicksPerMillisecond;
```

In addition, the current tick count can be accessed using DateTime.Now.Ticks, as shown:

```
long now = DateTime.Now.Ticks;
```

Together these can be used to trigger events after a set period of time has passed.

A generic example would look similar to this:

```
//Replace the number 100 with the desired number of milliseconds
const long kTimeoutPeriod = 100 * TimeSpan.TicksPerMillisecond;

long lastEvent = 0;

while (true)
{
    long now = DateTime.Now.Ticks;

    if ((now - lastEvent) > kTimeoutPeriod)
    {
        lastEvent = now;

        //DO EVENT

    }

    Thread.Sleep(10);
}
```

9.7. Utilities

The CTRE.Util class holds general utility methods that can be useful when programming HERO. The full functionality can be seen in the Utility.cs file of the SDK on the CTRE Github.

Disclaimer: The Util class is a holding place for general functions that don't currently belong as a specific method within another class. In later releases it may be deemed appropriate to move methods from Util to another, more permanent location. If this is the case, a deprecated version will be held in Util for at least one additional release, after which it will be removed from Util completely. These changes will be communicated in the HERO SDK Release Notes.

Currently `CTRE.Phoenix.Util` holds several methods.

10. HERO LED Table

HERO Condition	STATUS LED	COM LED	HERO Image
Not powered	**Off**	**Off**	
CAN Disconnected		**Red Blink**	Direct-Drive or NETMF
CAN Connected		**Green Blink**	Direct-Drive or NETMF
Change in USB Connectivity		**Orange Blink**	Direct-Drive or NETMF
No App	**Red Blink**		NETMF
App Running HERO Disabled	**Orange Blink**		NETMF
App Running HERO Enabled (Watchdog Fed)	**Green Blink**		NETMF
App Running HERO Enabled (Gamepad present)	**Red Blink**		Direct-Drive
Bootloader (DFU) Mode	**Solid Red**	**Off**	NA

NETMF (hero_netmf) – HERO is flashed with NETMF supported image. This means user can deploy NETMF applications using Visual Studio. To enable the HERO (and motor drive), application must call `CTRE.Watchdog.Feed()` periodically. When not called, the HERO (and motor drive) will disable outputs. This can be used to conditionally disable motor drive in a fashion best suited for the user's application (USB gamepad presence, button presses, IO, and other custom logic). See Section 9.2 for more information of motor disable.

Direct-Drive (hero_direct_drive) – HERO is flashed with the **hero_direct_drive** image. This means that HERO will grab the first Y-Axis on an attached USB HID device (gamepad or joystick) and will throttle all attached Talon SRXs on the CAN Bus.

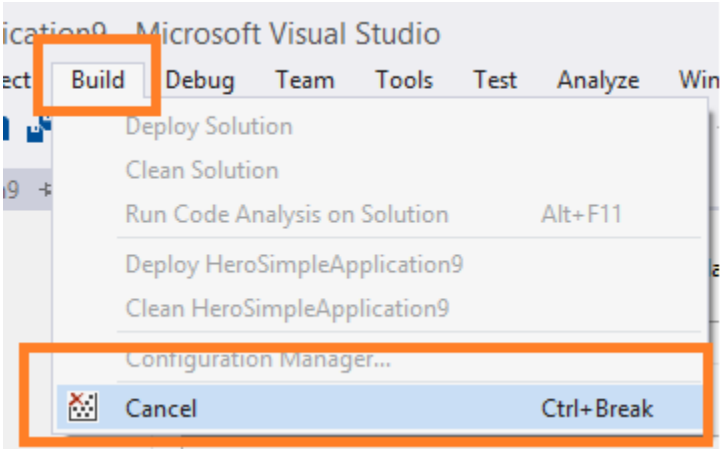
11. Troubleshooting Tips and Common Questions

11.1. Analog Reads seems to be stuck (around 0.2 – 0.3).

When an analog input is left unconnected, it will hover around these values. Make sure the correct ribbon cable is plugged into the correct slot.

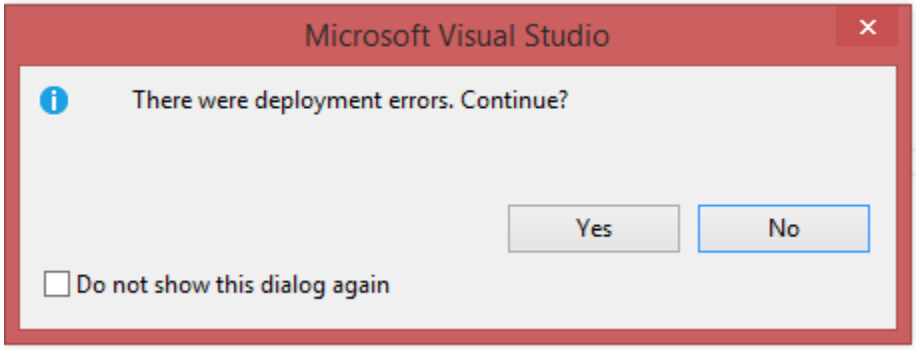
11.2. How to cancel a build

Often in this midst of developing, one might make a code-change, start the build/deploy process, and immediately realize that there were remaining necessary changes to make before attempting a worthwhile build. If there is time, you can cancel the build by pressing Build => Cancel or pressing the Ctrl + Break shortcut.

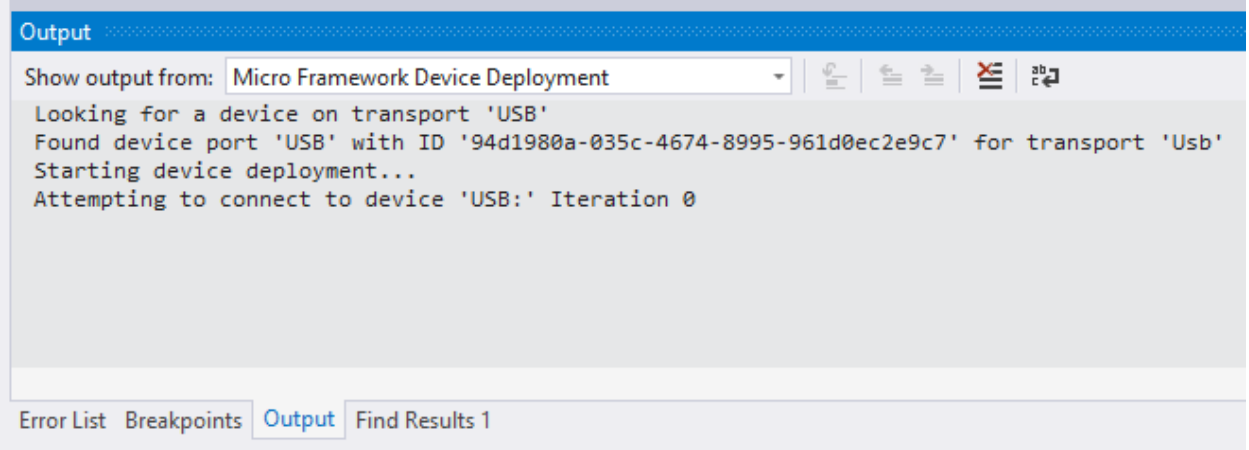


11.3. Visual Studio give me the error: "There were deployment errors. Continue?"

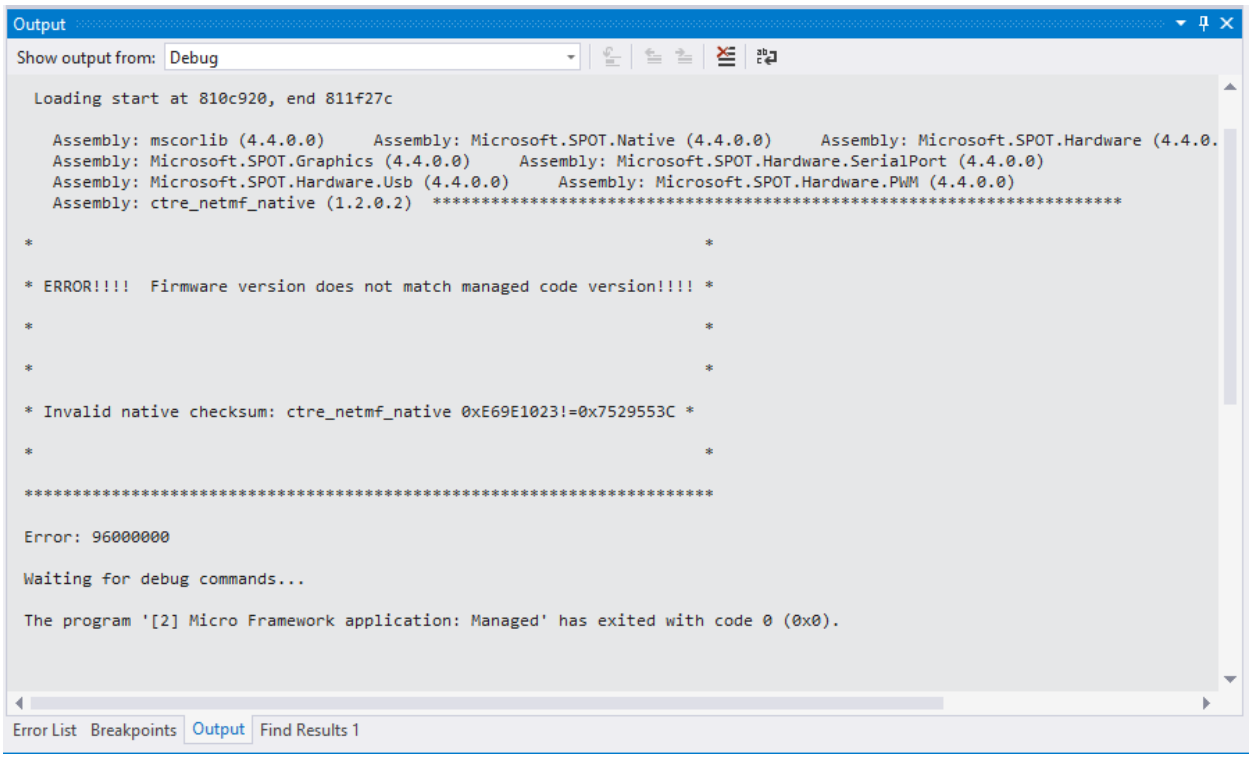
If you get a prompt indicating that the download failed, checked the output window for specific errors.



One common symptom is that connecting the device fails, causing several "attempting to connect" lines. This can happen if the HERO is not selected as the target device. See Section 8.2 for selecting the HERO as the target device.



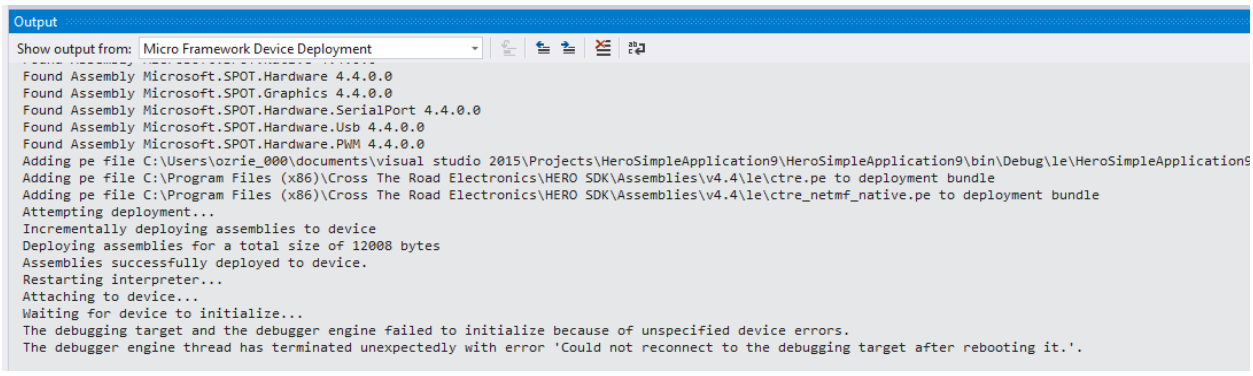
11.4. Visual Studio starts deploying but then errors saying the Firmware version does not match managed code version.



This can happen when there is a mismatch between the HERO's firmware and the HERO SDK installed on your PC. The simplest solution is to use HERO LifeBoat to re-image the HERO so that the firmware and SDK is synchronized again. See Section 7.1 for instructions on reimaging the HERO.

11.5. Visual Studio reports it could not reconnect to the debugging target.

This can happen if the PC has trouble reloading the USB drivers quickly. The simplest solution is to directly connect the USB device to the root hub of the PC/Laptop. Avoid using USB hubs if this problem is persistent.



11.6. Will any components of HERO be open-source?

The managed class libraries will likely be open-sourced via a GitHub account. HERO is meant to be a working example of how to support CTRE CAN devices in custom hardware platforms. Therefore it will be beneficial for integrators to "borrow" the CAN frame encoder/decoders for communicating with CTRE CAN devices. Users can then start with something that already works out-of-the-box with a minimal learning curve and no major price investment.

11.7. Why are there colored dots on the HERO? Is something wrong with the hardware?

There is nothing wrong with the HERO. In production each HERO goes through a number of test procedures to ensure quality. Each color represents a specific phase that the HERO went through.

11.8. My Talon SRX or PCM is blinking orange. They are in disabled mode even though the HERO is enabled (STATUS LED is green).

This is expected with the CAN device has FRC firmware. Use LifeBoat to flash non-FRC firmware.

11.9. HERO's STATUS LED should be blinking green (enabled) but occasionally blips orange.

The Feed() function may not be called often enough. The Watchdog times out at 100ms, so it is ideal to call Feed() several times faster than the timeout period. 20ms or faster should be sufficient.

11.10. So what's the difference between the hero_netmf firmware file and the hero_direct_drive firmware file? What is direct drive?

In order to develop a custom application in Visual Studio, you have to flash the hero_netmf firmware file into HERO. Then you can deploy/debug using Visual Studio 2015.

However, there are use cases where simply controlling one or more Talons directly with a hand held device is ideal. This could be for testing a mechanical prototype, or sanity checking electrical connections. So for those who would prefer to not write any software, they can flash the hero_direct_drive crf and simply plug in a HID gamepad into the HERO. HERO will drive all discovered Talons automatically.

This is analogous to a PWM tester being used to control a PWM Electronic Speed Controller.

Over time there will likely be additional utility firmware images that perform specific tasks, making CAN actuators even easier to use.

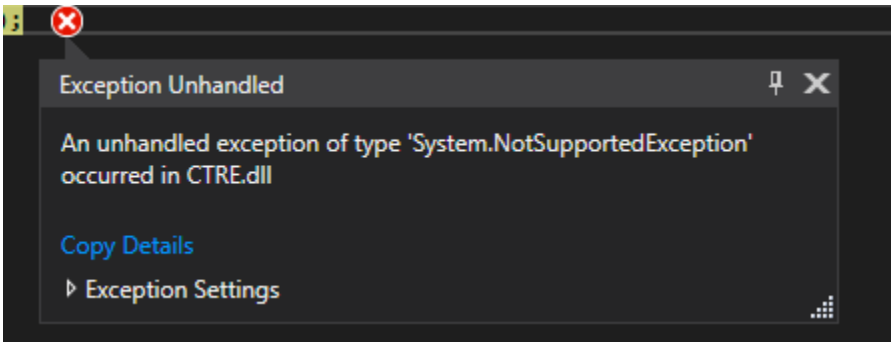
11.11. Why is this release marked "beta"?

~~Since the HERO-SDK installer leverages a beta build of Micro Framework 4.4, we decided to follow suite in the naming convention. Additionally we plan on releasing updates fairly frequently as we add new class objects to leverage, and respond to feedback. BUT there have been many requests for a solution to control our CAN devices outside of an FRC environment. We believe our preliminary beta release will meet these needs.~~

Releases are no longer marked "beta".

11.12. When I run a built-in example project, I get an exception.

By default, Micro Framework projects attempt to run in an emulator.
This emulator is not supported by the HERO libraries, so an exception is thrown when the program attempts to run a netmf or native HERO function.



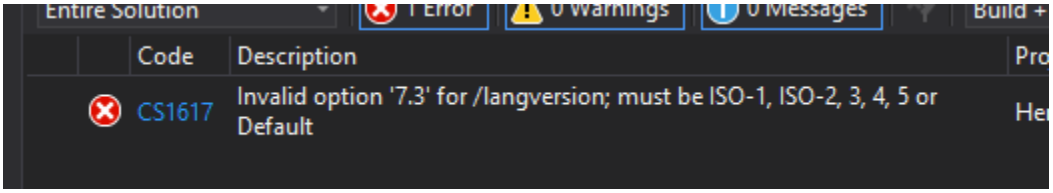
Exceptions known to be thrown in this scenario are included in (but not limited to) the list below:

- System.NotSupportedException
- System.ArgumentException

To fix this, make sure the project settings are configured to deploy to HERO over USB.

See Section 8.2 for details on setting this configuration.

11.13. Visual Studio gives me the error: "Invalid Option for /langversion"



This can happen with the latest version of Visual Studio 2019 when using older HERO C# Projects.

The workaround is to specify the language version in the *.csproj file by using a generic text editor to add the following lines as the first property group in the file:

```
<PropertyGroup>
  <LangVersion>5</LangVersion>
</PropertyGroup>
```

After modification, the contents of the csproj file should look similar to the following (the AssemblyName is custom to each project):

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Project DefaultTargets="Build" ToolsVersion="4.0" xmlns="
3    <PropertyGroup>
4      <LangVersion>5</LangVersion>
5    </PropertyGroup>
6    <PropertyGroup>
7      <AssemblyName>CANifier_Demo</AssemblyName>
```

Note that this issue is fixed in the latest example and template projects.

12. Functional Limitations

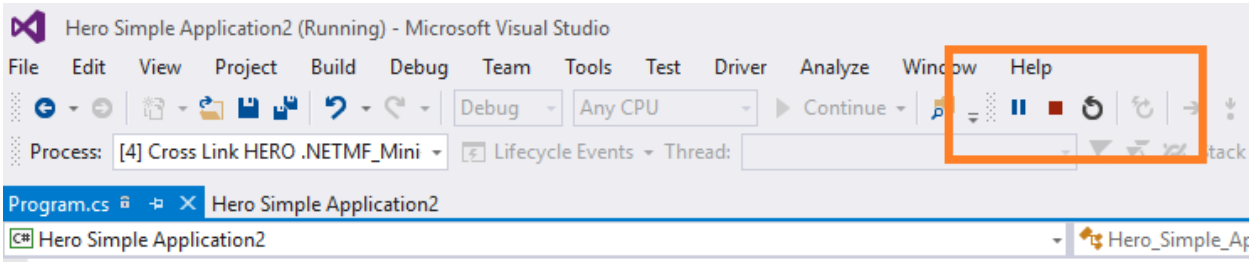
Functional Limitations describe behavior that deviates from what is documented. Feature additions and improvements are always possible thanks to the field-upgrade features of the HERO.

12.1. If HERO is disconnected from USB then reconnected, user may have to reselect the HERO as the target device in Visual Studio.

If Visual Studio loses track of the attached HERO, you may need to reselect it. This happens intermittently if you cycle disconnect/reconnect the mini USB connection to your PC. See Section 8.2 for instructions on how to select the HERO as the target device.

12.2. If HERO is disconnected during a debugging session, Visual Studio does not automatically terminate the debug session.

If the mini USB cable is disconnected during a debug session in Visual Studio, VS will not detect that the device has been disconnected. Instead it will continue to behave as though the program is "running" with pause/stop menu bar still visible…



The workaround is simply to press the "stop" button and wait for Visual Studio to close the session (which will take ~10 seconds). When you attempt to deploy your NETMF application again, you may need to reselect the HERO as the target device (see Section 8.2).

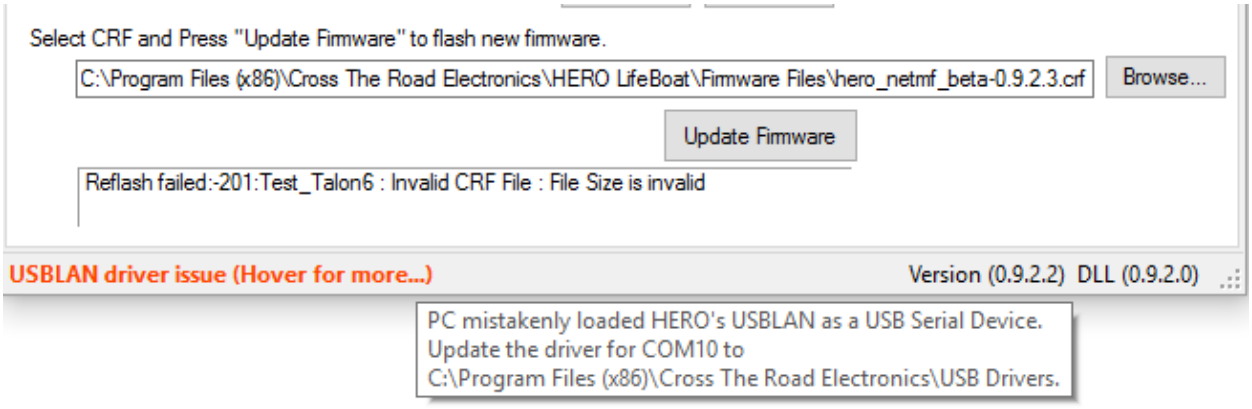
12.3. When I ran the Visual Studio 2015 installer, it seems stuck.

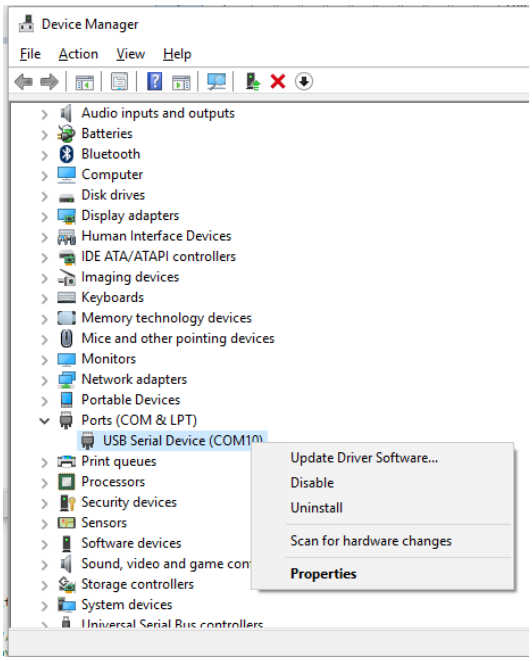
~~See Section 6.1.1~~

This has not yet been reported with Visual Studio 2017.

12.4. LifeBoat tells me there was a problem loading the driver, something about a "USB serial device"?

There is known issue in Windows 10 where composite USB devices that use CDC class are erroneously loaded as serial ports instead of loading the correct driver. If this happens LifeBoat will detect this condition and provide a helpful tooltip on how to resolve it.





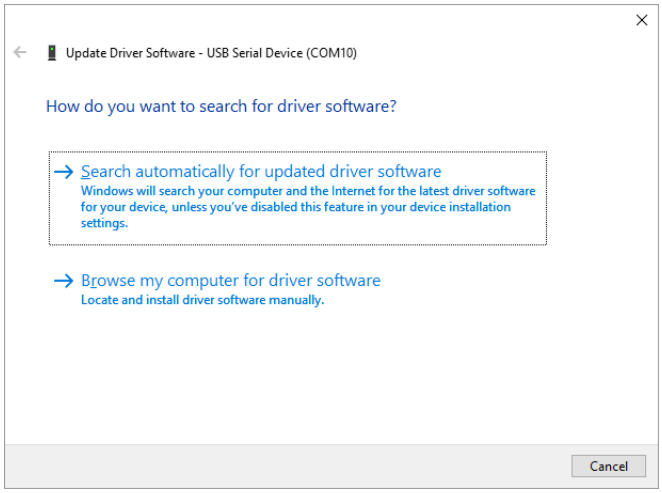
Find the erroneous entry in device manager and right-click to bring up the "**Update Driver Software**" option.

If you are not certain **which** USB Serial Device to correct, you can check which COM port number LifeBoat is reporting in the **tooltip**. In our example LifeBoat is asking the user to correct COM10.

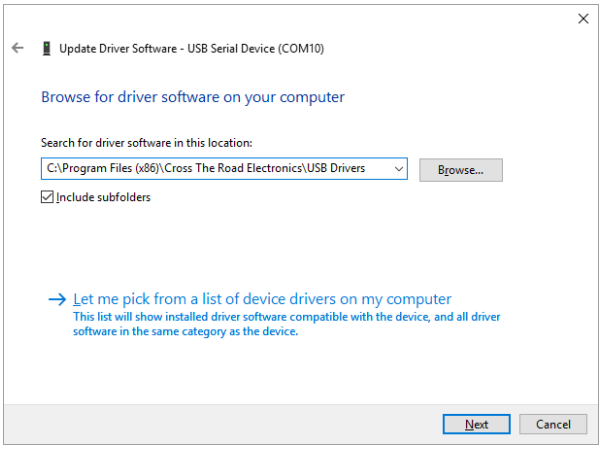
Or you can disconnect the HERO and see which COM number disappears.

Advanced users can also check the VID number under Properties => Details Tab => Property: Device instance path, which will hold a VID value of 29CA and PID of 3539.

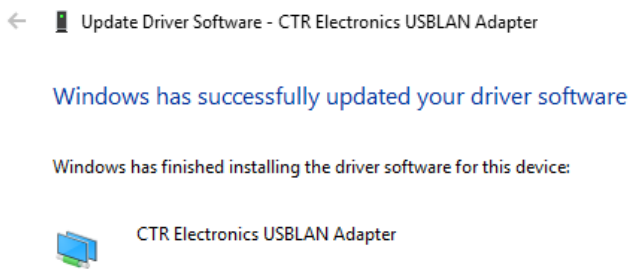
Next select "**Browse my computer…"**



Navigate to the USB Drivers in the HERO install path. Check "**Include subfolders**" and press "**Next**".



When Windows is finished updating to the correct driver, you will see this prompt…



12.5. Since the SDK installs a "beta" build of the Microsoft Micro Framework SDK, the development PC must not have the RTM release build from Microsoft.

A future software update will support the RTM version of the v4.4 Micro Framework SDK. For the time being, avoid installing any other versions of the .NET **Micro** Framework.

However, the PC can have .NET framework (not Micro) installed as these have no impact on the compiled HERO applications.

12.6. SPI, and I2C Examples are missing in the User Manual / Visual Studio Examples.

~~A future software and document update will have these interfaces supported.~~

These interfaces are currently supported. I2C can be found in Section 9.4.5 and SPI can be found in Section 9.4.6.

12.7. PDP Class needs to be added.

~~A future software and document update will have this included.~~

There is currently a PowerDistributionPanel class in the CTRE.Phoenix namespace.

12.8. Class library is C# only.

C# appears to be the most popular NETMF language used (compared to Visual Basic). However if there is sufficient feedback indicating that it would be worthwhile to support other languages as well, we will prioritize accordingly.

12.9. USB Host is HID only, no Xbox gamepad support.

~~A future software update will support this.~~

HID and X-Input gamepads are both supported.

12.10. No examples for the Visual Studio Emulator.

Will be available in a future update.

12.11. No way to see Battery Voltage in HERO LifeBoat.

Will be available in a future update.

12.12. A pristine, out-of-the-box HERO may reset if more than 10 CAN devices are on the CAN bus before it is field-upgraded.

This doesn't have any practical impact since the first step is to re-image the HERO (which is not prevented by this). However, it is noteworthy that a pristine, out-of-the-box HERO (a HERO that has never been re-imaged) will reset intermittently if it is attached to a CAN bus with more than 10 CAN devices. This is expected and will not prevent the re-imaging process. Once imaged, this symptom will no longer occur. See Section 7.1 for instructions on how to reimage a HERO using LifeBoat.

12.13. Certain Digital Input Pins are not pulled to the expected value when using PullUp/PullDown Resistor Settings

Port 5 – Pin 8 will always be pulled to logic high regardless of the resistor value setting and should only be used in an application where that is desired.

The two greyed-out pin sets below have been fixed as of Installer Revision 4.4.0.24, but will still apply to previous SDK revisions. Please update to the most recent SDK release for the most up-to-date functionality.

Port 8 – Pin 6 currently has unpredictable behavior as a Digital Input and should not be used.

Port 6 – Pins 4 & 5 will always be pulled to logic low regardless of the resistor value setting and should only be used in an application where that is desired.

12.14. Gadgeteer Port 3, Pins 7 and 8 are not available for PWM use.

This has been fixed with the latest firmware and SDK revisions (firmware rev. 0.10.0.1 and SDK rev. 4.4.0.10), which can be found in installer version 4.4.0.25 and later. Please update to the most recent SDK release for the most up-to-date functionality.

12.15. I'm using PWM and my application won't deploy.

This is a known issue with older firmware versions. Please update to the latest version of firmware and the SDK.

12.16. Lifeboat closes as soon as it is opened.

Certain antivirus software may interfere with Lifeboat and prevent it from opening properly.

If this happens, check your antivirus software and firewall settings.

Ensure that there are exceptions within both of these to allow Lifeboat to run.

12.17. When I try to open a project, Visual Studio says the project type is unsupported / I opened an existing project and Visual Studio says it needs to be migrated.

Both of these conditions indicate that the netmf project type for HERO is not installed properly in Visual Studio. This can happen for one of two known reasons:

1. Visual Studio has updated.
2. The netmf plug-in for Visual Studio failed to install properly during the installation of Phoenix

Case 1 is the most common. Re-running the Phoenix Installer or manually re-installing the *.vsix file appears to solve the issue in both cases.

12.18. When running a project with CAN devices, I keep getting "Error 1 CTR: Error Code 1"

This a side effect of the way HERO firmware tracks whether a CAN message is stale.

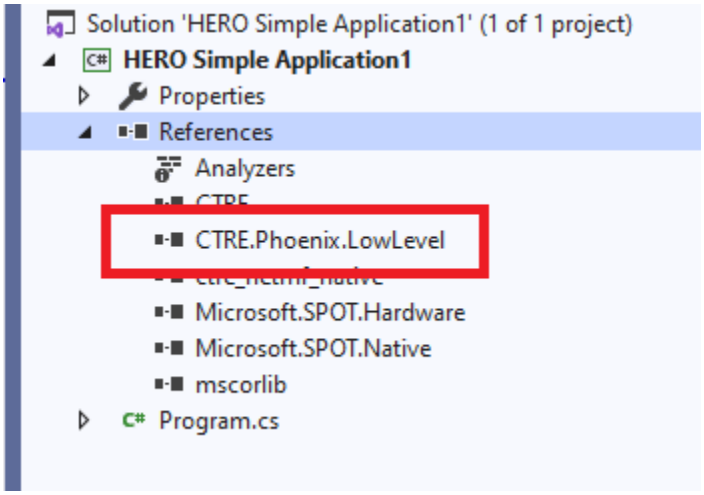
If the CAN message has been read more than once since the latest frame has been received, a warning code of 1 (message stale) is returned and printed.
HERO firmware has always reported this warning code, but the updated Phoenix libraries now print all non-zero error/warning codes (instead of just negative error codes).

A new version of HERO firmware is coming soon that will instead report a CAN Timeout error only if the message has not been received for more than 255 ms.

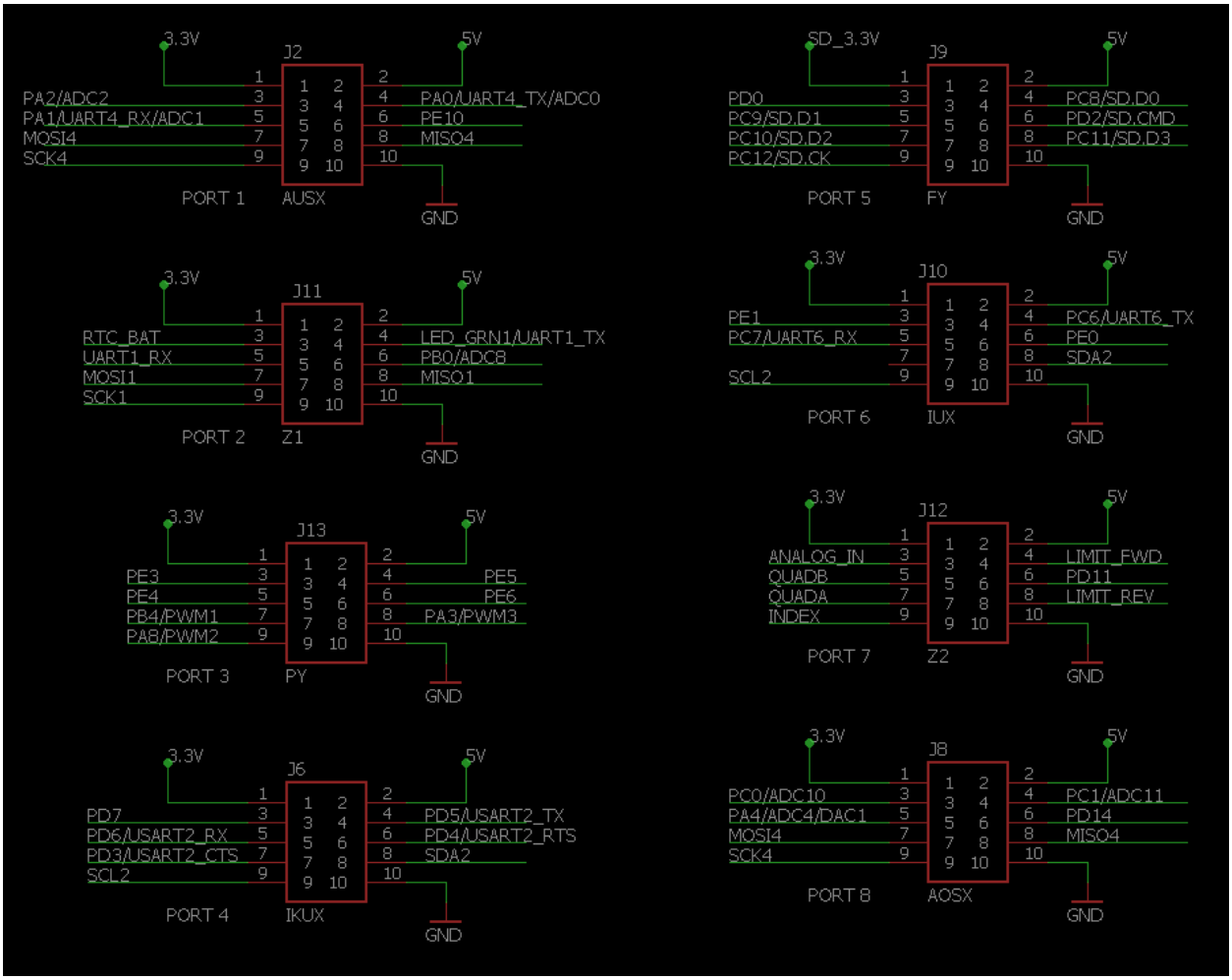
12.19. Phoenix Versions 5.18.4 and later require an additional C# library

With the release of Phoenix v5.18.4, and additional library was created called "CTRE.Phoenix.Lowlevel". This means all pre-existing netmf projects require manually adding the library to the project references. New projects created with v5.18.4 or later will automatically include this reference.

Use the instructions in Section 8.4 to add the "CTRE.Phoenix.Lowlevel" library. Your project references should then list the additional reference as shown below.



13. Hardware References



14. CRF Firmware Revision Information

Model	Version	Date	Description
HERO (NEMTF)	1.1.0.0	4/17/2018	hero_netmf-1.1.0.0-BroadcastEnable.crf Watchdog: Updated to work with new non-FRC firmware.
Talon SRX/Victor SPX	11.8	4/17/2018	TalonSrx-Application-11.8-PhoenixNonFRCUpdate.crf VictorSpx-Application-11.8-PhoenixNonFRCUpdate.crf Non-FRC firmware that matches the functionality of the 3.8 FRC firmware.
HERO (NETMF)	0.10.0.1	7/21/2013	hero_netmf_beta-0.10.0.1.crf PWM: Added functionality to enable Watchdog control. PWM: Added additional PWM channels (5 total available)
Talon SRX	10.13	7/21/2013	TalonSrx-Application-10.13-nonFRC-CustomParams.crf Added new signals for two custom parameters. Current Limit feature added.
Talon SRX	10.11	5/11/2016	TalonSrx-Application-10.11-nonFRC-MotionMagicAndNewPersis Motion magic control mode added. Persistent storage has been redesigned so persistent settings will be lost after updating to this version.
HERO (NETMF)	0.9.7.0	5/11/2016	hero_netmf_beta-0.9.7.0.crf USB Gamepad: Added support for larger USB device-reports. Various performance improvements for CAN periodic transmits.
HERO (NETMF)	0.9.2.3	1/6/2016	hero_netmf_beta-0.9.2.3.crf First public release of the NETMF firmware for HERO.
HERO (Direct-Drive)	0.0.3.0	1/6/2016	hero_direct_drive-0.0.3.0.crf First public release of the Direct-Drive firmware for HERO.
Talon SRX / PCM	10.00 10.00	1/6/2016	TalonSrx-Application-10.00-nonFRC.crf PCM-Application-10.00-nonFRC.crf First public release of non-FRC firmware for PCM and Talon SRX.

More comprehensive revision information can be found in the RELEASE_NOTES file.
The file is already installed here:
"C:\Users\Public\Documents\Cross The Road Electronics\RELEASE_NOTES.txt"

15. Document Revision Information

Version	Date	Description
1.2	3-Oct-2023	Updated VS 2019 install link
1.1	12-Nov-2020	Added Section 12.19
1.0	7-Apr-2020	Revised Manual for compatibility with Visual studio 2019 Removed Section 1.2, Moved 1.3, 1.4 up Revised Section 6,6.1,7.1,7.2.1 Revised Section 8.1,8.2,8.3,8.4,9.3,9.4.4,9.4.5,9.4.6 Added Section 11.13
0.12	22-May-2018	Added Section 12.18.
0.11	21-May-2018	Revised Manual for compatibility with Phoenix 5.4.3.0 Added Section 4.3, 12.17
0.10	10-May-2018	Added Section 11.12
0.9	6-Mar-2017	Added Sections 9.4.5 and 9.4.6. Revised Section 12.6.
0.8	25-Aug-2016	Moved 12.15 to 12.16, added new 12.15. Revised Section 7.1.1. Revised Section 9.4.4.
0.7	21-Jul-2016	Added Section 9.4.4.1 Revised Section 9.4.4 and 12.14.
0.6	30-Jun-2016	Added Sections 9.4.4, 9.5, 9.6, 9.7, 12.14, and 12.15. Revised Section 12.13.
0.5	5-May-2016	Added Section 9.4, Section 12.13
0.4	3-Mar-2016	Added Section 13
0.3	6-Jan-2016	Initial Release