CANivore
User's Guide

Revision 1.0

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. Device description

The CTRE CANivore is a USB to CAN FD adapter, capable of providing a CAN FD network to a Windows PC or a roboRIO-Linux. Each CANivore will allow you to add an additional CAN bus to the roboRIO and control supported CTR Electronics devices from your robot project.

The CANivore provides the following advantages:

- Ability to **add additional CAN Buses** for supported CTR-Electronics devices to the FIRST-roboRIO.
- Ability to use CTR-Electronics devices **by running robot application in Windows** (desktop simulation + actual hardware) when using FRC C++/Java.
- Use **Phoenix Tuner** to field-upgrade, configure, and test devices **without a roboRIO**.
- **Exceed** the **bandwidth limitations** of the existing native legacy **CAN bus on the RoboRIO**.
- Individually **name CAN buses** so software works every time regardless of boot-up order.
- **Improve CAN bus wiring** –no longer need to wire single bus to every corner of the robot.
- **Onboard ESP32** for **custom applications** and access to **Wi-Fi/Bluetooth/BLE**.

1.1. Kit Contents

- 1 CANivore
- 1 USB A to C cable

1.2. Features

- **USB to CAN FD** adapter for Windows and roboRIO-Linux for supported CTR-Electronics devices. (Note 1)
- Each CANivore adds an entirely new CANFD network to the roboRIO
- Can be used to **control and update devices** from a **Windows PC**
- Integrated **ESP32** for **wireless capabilities (Wi-Fi/Bluetooth)** and programmable control
- Windows PC use case **does not** require **custom USB drivers**
- **CANivore kernel driver** already **included into NI roboRIO image.**
- Ability to set **custom name** for each CANivore CAN bus
- Modern **USB Type-C connector**
- Includes a **USB A-to-C cable** – perfect for the FIRST roboRIO use case.
- Can be **USB powered** without requiring additional power wiring
- Can be **powered from DC supply or battery (up to 28V)** for **standalone operation**
- Can be powered by **both** USB and power connector for **redundant power supply**
- **Reverse** Input Power **Protection**
- Supports **USB Hub**
- Supports **Hardware Attached Simulation** for FRC C++/Java (FRC Simulation on desktop with actual hardware)
- **Polycarbonate housing** prevents debris from entering inside device
- **Voltage measurement** of V^+ and 5V rails
- **CAN bus utilization** measurement
- **Multi-color LEDs** for Status, Communication, Wi-Fi, and Bluetooth
- Reliable **Weidmuller** connector for power and CAN
- **Robust bootloader** and **reliable field-upgrade** (no physical button required, no "stuck states" that requires user intervention)
- W**irelessly check, configure, and field-upgrade** CANivore using roboRIO Wi-Fi and Phoenix Tuner.
- W**irelessly check, configure and field-upgrade** attached CAN devices using roboRIO Wi-Fi and Phoenix Tuner.
- **Software-selectable** termination resistor (120 Ω)

Note 1: A future design goal will be to include support for more common Linux platforms. The focus of the initial release was for FIRST Robotics, but the CTRE CANivore kernel driver can be ported to other Linux systems.

1.3. Electrical Specifications

Symbol	Parameter	Condition	Min	Typ.	Max	Unit
General Specs						
Tamb	Ambient temperature		-40		+85	°C
V^+	Supply voltage		5.2	12	28.0	V
Weidmuller Powered Current Ratings						
Isupp	Supply Current	DC supply 12.0V, No CAN	36		40	mA
Isupp	Supply Current	DC supply 12.0V, CAN Connected, no traffic	29		32	mA
Isupp	Supply Current	DC supply 12.0V, CAN Connected, ~50% traffic	29		32	mA
Isupp	Supply Current	DC supply 12.0V, CAN Connected, ~50% traffic & ESP broadcasting SSID	78		98	mA
USB Powered Current Ratings						
Isupp	USB Current	USB Supply 5.0V, No CAN	63		78	mA
Isupp	USB Current	USB Supply 5.0V, CAN Connected, no traffic	46		60	mA
Isupp	USB Current	USB Supply 5.0V, CAN Connected, ~50% traffic	46		60	mA
Isupp	USB Current	USB Supply 5.0V, CAN Connected, ~50% traffic & ESP broadcasting SSID	150		200	mA
ESD Rating						
	ESD Protection Contact Discharge				±30	kV
	ESD Protection Air-Gap Discharge				±30	kV

1.4. General specifications

Outside Dimensions	2.17" x 1.55" x 0.76"
Weight without cables	1.16 ounces (32.89 grams)
Supported Communication Protocols	CAN FD USB

1.5. LED States

The CANivore features 2 tri-color LEDs (STAT & CAN) and 2 mono-color LEDs (Wi-Fi & BT).

LED Name	Behavior	Blink Style	Description
STAT	**Red**	**Double-Blink** (Note 2)	Device powered through V^+/V^-, but **USB not plugged in**.
	Red	**Fast-Strobe** (Note 3)	USB plugged in, but **no USB communication** *(USB not enumerated or USB suspended)*
	Orange	**Double-Blink** if V^+/V^- is **not** powered (Note 2) **Fast-Strobe** if V^+/V^- is powered (Note 3) ***TIP: Use this to confirm the V+/V- wiring!***	**Good USB** connection, CAN streaming is **disabled**
	Green		**Good USB** connection, CAN streaming is **enabled**
	Green / Orange	LED is never off – one of the two colors is **always illuminated**.	CANivore is in **Bootloader**. Most likely device was unplugged during field-upgrade (Note 4). Use Phoenix Tuner to field upgrade latest CRF firmware file. Alternatively, if CANivore already has application firmware, disconnect all power sources, then re-connect to cold-boot device.
	Orange / Red		CANivore has hardware damage
Wi-Fi	**Green**	**Blink** (Note 1)	Wi-Fi is enabled or ESP32 custom application is allowed to use Wi-Fi
	Off		Wi-Fi is disabled
BT (Bluetooth)	**Green**	**Blink** (Note 1)	Bluetooth is enabled or ESP32 custom application is allowed to use Bluetooth
	Off		Bluetooth is disabled
CAN (CAN bus)	**Solid Red**	LED is never off.	**Voltage too low** for CAN bus. CAN communication may **not** be reliable
	Red	**Double-Blink** if termination is disabled (Note 2)	**No CAN** communication
	Orange	**Fast-Strobe** if termination is enabled. (Note 3)	Reserved for CAN 2.0b legacy mode
	Green	***TIP: Use this to confirm termination setting!***	**CAN FD** active

Note 1: "Blink" means LED transitions between on and off slowly at a fixed rate.

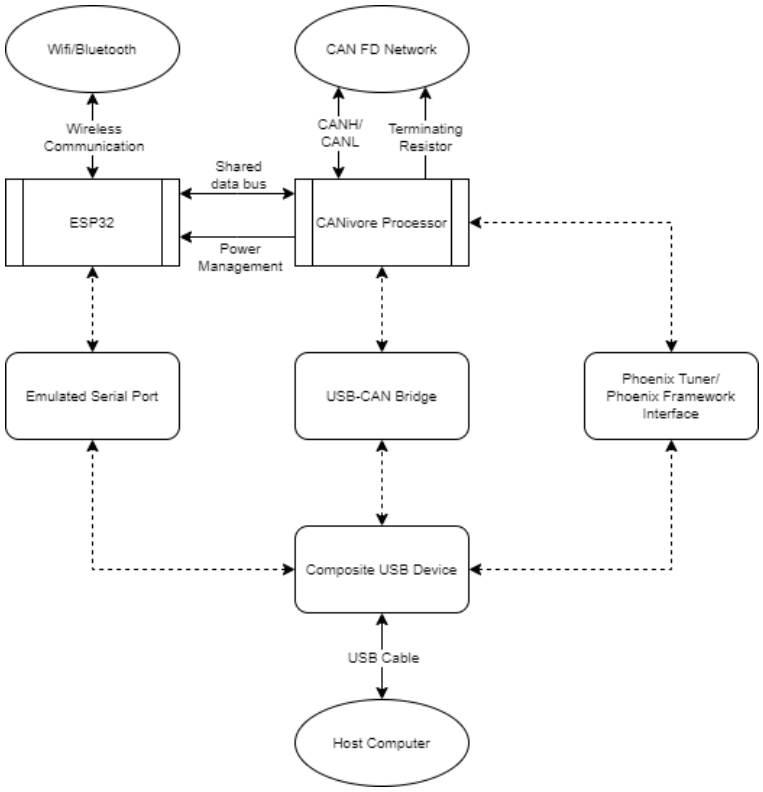
Note 2: "Double-Blink" means LED transitions between on and off twice, then pauses.

Note 3: "Fast-Strobe" means LED spends very little time off. It will appear excited or mostly-on.

Note 4: If using USB features, ensure proper signal path between CANivore USB connector and the root USB controller (including all the hubs and data/power connections in between).

2. Block Diagram

The diagram below shows the organization of the CANivore peripherals.



3. CAN FD Bus

CAN FD is the next iteration of the CAN bus data link layer. The FD stand for Flexible Data-Rate, meaning that portions of the frame is transmitted at higher data rates (up to 10Mbps). This means less bus time for more data. Additionally, a single frame can now hold up to 64 data bytes.

This is critical for many reasons:

- **Lower bus utilizations** because of less frame overhead and higher bitrates.
- **More devices** before hitting maximum bus utilization.
- Less time spent designing how to split signals across multiple frames
- Larger frames mean **potential firmware features that were not previously possible** with CAN 2.0B.

More information on CAN FD is available at https://ctr-electronics.com/can-fd, including which CTR-Electronics devices support it.

CANivore automatically configures the CAN bus network to be compliant with CAN FD. As a result, end-users do not need to configure bit timing information.

3.1. Terminating Resistor

Inside the CANivore is a 120Ω terminating resistor which can be software enabled. This setting **defaults to ON**, and therefore can immediately be used to terminate one extreme end of the CAN Bus.

Note: The other extreme end must **also be terminated with a 120Ω resistor**. We recommend soldering a 120Ω through hole resistor between the CANH (yellow) and CANL (green) wires at the extreme end of the bus.

4. CANivore Processor

At the center of the CANivore is a **120MHz Cortex M4F processor**. This dedicated controller implements all USB and CAN bus related features. This includes:

- Communication with CTR-Electronics CANivore-USB kernel driver
- Field-upgrade and other Phoenix Tuner features
- General configuration of the device (termination resistor, device-name, ESP32 enable/disable, etc.)

5. ESP32 Co-Processor

The CANivore features an **ESP32WROOM32D** as a co-processor for custom user code.
This processor was chosen because:

- Native support for Wi-Fi and Bluetooth/BLE
- Popularity in the embedded systems and hobbyist community
- Increases the maximum potential long-term feature-set of the CANivore

The ESP32 is not necessary when using CANivore to add additional CAN buses for supported Phoenix devices, as this is implemented by the CANivore core processor. However, the shared data bus between CANivore processor and ESP32 allows for the potential to provide CAN bus features to customer user applications.

After the initial 2022 release of the product, examples will be posted for sending arbitrary CAN frames. Moving forward, CTR-Electronics will be investing in the development of advanced ESP32-based features. As features are added, more examples and documentation will be posted accordingly.

5.1. Serial port

When CANivore is USB connected to a PC, a virtual serial port will appear in the peripheral list. This serial port is virtually connected to the serial inputs on the ESP32. This means that the software-tools that already exist for updating and debugging the ESP32 will also work on CANivore. User will have to specify the COM port as they would normally do with typical ESP32 breakouts.

5.2. Wi-Fi

Wi-Fi (2.4 GHz) is integrated into the ESP32. The latest documentation can be found under the **ESP32WROOM32D** resources provided by **espressif.com**.

5.3. Bluetooth/BLE

Bluetooth/BLE is integrated into the ESP32. The latest documentation can be found under the **ESP32WROOM32D** resources provided by **espressif.com**.

6. Software

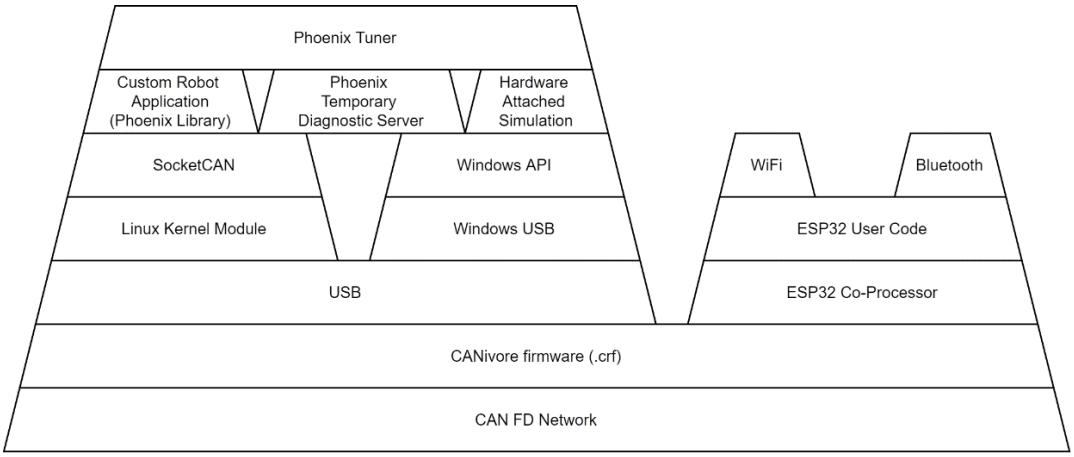
Although this user's guide primarily covers hardware-related aspects to the CANivore, outlining the software interactions may be useful when first becoming familiar with the device.

6.1. Latest Documentation

Software-developers are **encouraged** to review the software documentation for the latest information, particularly the "Phoenix Framework Documentation" section. However general software guidance is also provided below.

6.2. Software stack-up

Software stack-up below covers software-related interactions of the CANivore.



6.3. Basic Setup

Basic tasks such as field-upgrade, device-naming, and configuration can be done with several configurations.

Software Tool	Description	Diagnostic Server Address (Note 1)
Phoenix Tuner	CANivore is connected to Windows PC via USB.	"CANivore-usb"
	CANivore connected to Windows PC via USB. User is running "Hardware Attached Simulation".	"localhost"
	CANivore is connected to roboRIO. Phoenix Tuner connected to roboRIO.	roboRIO: Team number / IP Address
caniv	Using SSH console with roboRIO	NA

Note 1: The Diagnostic Server Address is specified in the "Robot Controller Install" tab of Phoenix Tuner.
Note 2: caniv is a command line tool in the roboRIO-Linux system.

See https://ctr-electronics.com/documentation for the latest information, particularly the "Phoenix Framework Documentation" section.

6.3.1. Basic Tasks – Field-upgrade

Like other devices manufactured by CTR-Electronics, device firmware is stored in a CRF File. CRF is automatically installed with Phoenix Framework and is also available for download on the CTR-Electronics website (CANivore product page).

6.3.2. Basic Tasks - Naming the CAN bus

CANivore can be "named" by the user with a unique custom name. These impacts:

- The name of the device in Windows Device Manager
- The name of the device in Phoenix Tuner
- The name of the device when executing "**caniv –list**"
- The CAN bus name to pass into software objects in Phoenix API

This also allows developers to identify what devices are on what CAN bus.
CANivore supports custom names up to 32 standard ASCII characters.

Naming the CANivore can be done through the CTRE provided **caniv** tool or through **Phoenix Tuner**.

6.3.3. Basic Tasks - Programmable Terminating Resistor

The termination resistor can be turned on and off through the CTRE provided **caniv** tool or through **Phoenix Tuner**.

Note: Terminating resistor **defaults to ON.**

Termination mode can be confirmed in Phoenix Tuner and via the COMM LED.

Note: Measuring the termination resistance via ohmmeter is not possible when CAN bus is active. A software button may be added in the future to allow for measuring resistance if the need arises.

6.4. Windows PC

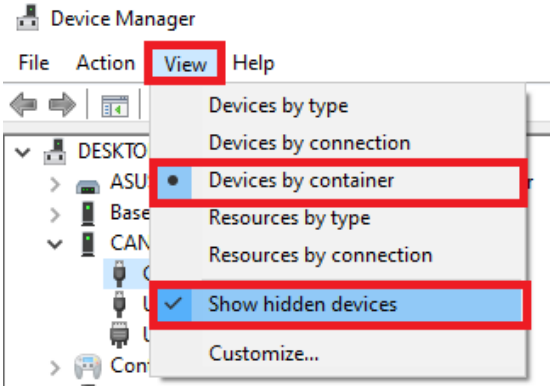
CANivore communication is done in a **driver-less fashion**. Meaning **any modern Windows PC** will enumerate the CANivore, and report it as successfully connected in device manager.

Note: It is generally required that only one application can connect to a CANivore in Windows. However, when desktop-simulating an FRC application, Phoenix Tuner still functions if the Diagnostic Server Address is set to "localhost". This allows for simultaneous use of Phoenix API (in the desktop robot application) and Phoenix Tuner (diagnostics).

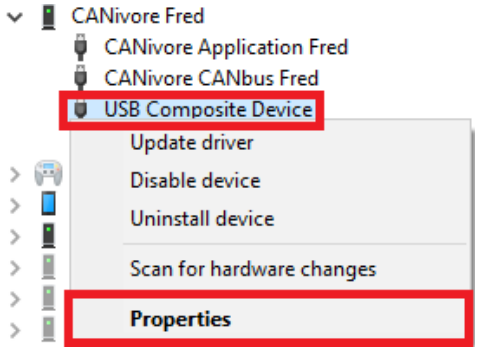
6.4.1. Windows PC - Device Manager

To confirm the CANivore is connected in Windows, perform the following steps:

1. Open Device Manager. This can be done with the shortcut "Win+R", typing "devmgmt.msc", and pressing 'enter'.
2. Change Device Manager to view "Devices by container" & enable "Show hidden devices". This makes it easier to identify CANivores connected to the computer & shows any devices that Windows is hiding, such as a misbehaving CANivore.



3. Navigate to your CANivore and right-click->Properties on the USB Composite Device that is under it. If you have not renamed the CANivore, it will show as "CANivore Default Name", otherwise it will be named "CANivore <custom name>".



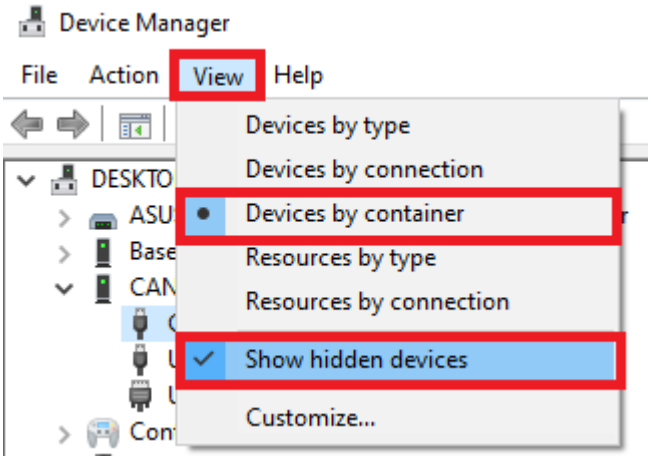
6.4.2. Windows PC - Installation

There is no additional software installation beyond installing Phoenix Framework.
Phoenix Tuner and Phoenix Diagnostics Server already supports CANivore natively.

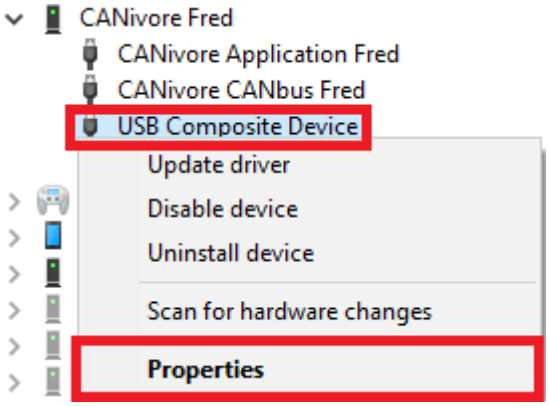
6.4.3. Windows PC - Driver Uninstall / Reinstall

To uninstall the CANivore windows driver, perform the following steps:

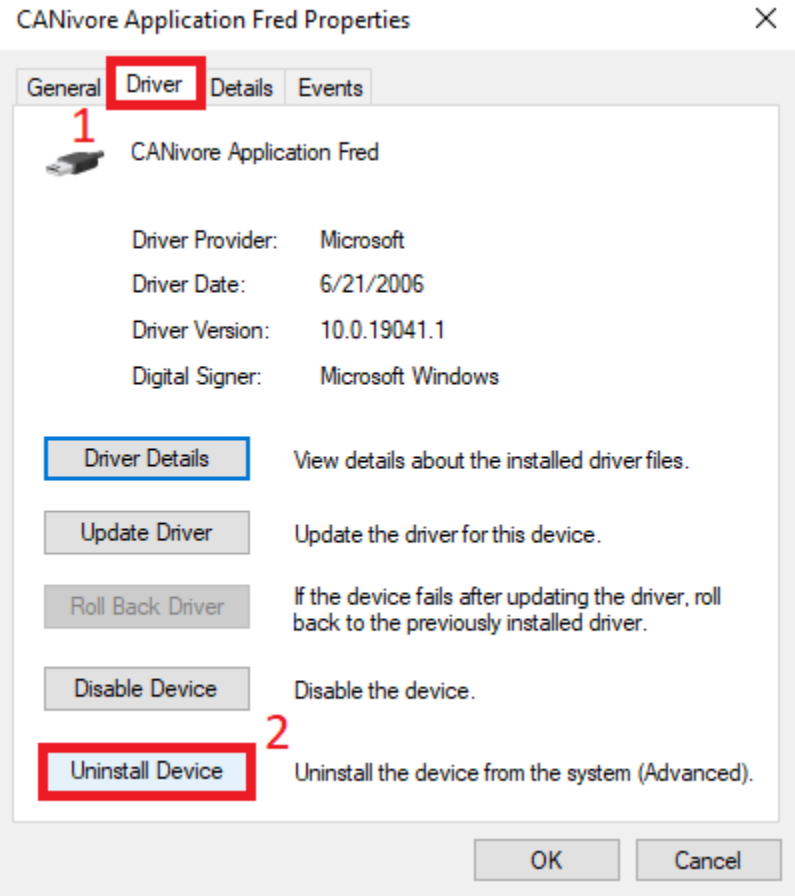
1. Open Device Manager. This can be done with the shortcut "Win+R", typing "devmgmt.msc", and pressing enter.
2. Change Device Manager to view by container & enable "Show hidden devices". This makes it easier to identify CANivores connected to the computer & shows any devices that Windows is hiding, such as a misbehaving CANivore.



3. Navigate to your CANivore and right-click->Properties on the USB Composite Device that is under it. If you have not renamed the CANivore, it will show as "CANivore Default Name", otherwise it will be named "CANivore <custom name>".



4. Navigate to "Driver" and click "Uninstall Device". This will uninstall all the CANivore drivers from the computer so that the next time CANivore is plugged in, it cleanly gets the new drivers.



This is everything necessary to uninstall a CANivore

6.4.4. Windows PC – USB Known Issue

In *extremely rare circumstances*, Windows may fail to completely enumerate the CANivore USB interface due to a Windows-specific issue in the USB host stack. If this happens, Phoenix Tuner is unable to detect or communicate to the CANivore despite it appearing in Device Manager. If this occurs, **uninstall the CANivore driver, unplug the CANivore, and plug it back in**.

For this issue to occur:

- Must be the first time CANivore is inserted into a PC (or first insert after renaming CANivore).
- USB enumeration is interrupted **without** physically disconnecting the device (processor reset event).

These conditions typically cannot be reproduced by the end-user and are only known because of validation-testing done by the CTR-Electronics team.

6.5. Linux / SocketCAN

To support Linux-based systems, CTR-Electronics has elected to use SocketCAN for the CAN bus software interface.

6.5.1. Linux / SocketCAN - What is SocketCAN

SocketCAN is a commonly used set of drivers and networking stack that allows for sending/receiving CAN frames in Linux. Because the roboRIO is a Linux system, we opted to bring SocketCAN into FIRST Robotics. In fact**, CTR-Electronics has supported SocketCAN with Phoenix Framework for *years*** (particularly with industrial customers) using various hobbyist style SocketCAN-USB products.

However, while supporting our industrial customers with SocketCAN, the list of problems we've encountered with existing USB-to-CAN hardware solutions grew to such a degree that developing the CANivore became the only solution to ensure a quality experience, in industry and in FIRST Robotics.

Some of those **limitations** are:

- No way to **name each network** to ensure **network name is the same after booting up**. User applications must specify "can0" vs "can1" vs "can2", and **the numbers are not the same each time**.
- SocketCAN hardware **does not work on power up**. They **require configuration** every time they are inserted.
- **Disconnect/reconnect events** will cause **CAN bus stream to stop**.
- Poor quality firmware – lost frames,
- Field-upgrade not robust – devices rendered "bricked" or require button-on-boot recovery.
- **Confusing LED sequences** not tailored for troubleshooting wiring.
- **Termination** is often **not selectable** or **requires mechanical jumper**.
- Performance issues

CTRE-Electronics has solved these common problems by providing:

- CANivore USB-to-CAN Adapter
- Phoenix Framework (API supports picking CANivore by name)
- CANivore's SocketCAN compatible kernel driver integrated into 2022 roboRIO Image.

6.5.2. Linux / SocketCAN – Extensions

For robust operation in the FIRST use-case, the following extensions were implemented:

- **CAN bit timings** set via SocketCAN are **not honored**. **CANivore will determine the correct bit timings** independently, ensuring they cannot be set incorrectly.
- CTR-Electronics **kernel driver is part of the NI FRC 2022 image**. If mid-season updates are necessary, they will be installed via Phoenix Tuner.
- Console features such as **candump** are functional, however they can consume substantial CPU when printing to the shell.
- Users should **not** "bring-up" CANivore networks via shell commands ("ifconfig" for example) or startup scripts. **Phoenix API library will do this for you**. If you manually "ifconfig down" a CANivore bus, Phoenix API will not re-initialize it for you.
- CANivore is rigorously tested with CTR-Electronics supported devices only. Use cases outside of this were not considered for the initial 2022 release of this device.
- **Phoenix API requires user to specify the CAN bus name** when creating a software object for supported devices like Falcon 500 or CANcoder. Developers may also use the socket CAN names such as "can0", but these may not be reliable when using more than one SocketCAN bus as the number-suffix may change with each robot cold-boot. It is recommended to use the CANivore custom name instead.

6.5.3. Linux / SocketCAN – Installation

Custom CTR-Electronics kernel driver is part of the NI FRC 2022 image. If mid-season updates are necessary, they will be installed via Phoenix Tuner.

There is no additional software installation beyond installing Phoenix Framework.
Phoenix Tuner and Phoenix Diagnostics Server already supports CANivore natively.

6.6. Programming ESP32

The emulated serial port provided by CANivore allows the existing ESP32 software tools to deploy to the CANivore.
Example software tools include:

- Espressif IDF (Visual Studio extension)
- ESP32 nanoFramework
- ESP32 Arduino

  Note: Typically, user must specify the serial COM port when using these tools.

Examples and documentation for programming the ESP32 will be provided in future updates.

6.6.1. ESP32 - Enable/Disable

The ESP32 can be enabled or disabled via the CANivore Processor. **This setting defaults to OFF**.
This setting allows for:

- Basic power management features (turn off ESP32 to reduce current)
- Ability to comply with FIRST game rules if Wi-Fi/Bluetooth features must be enforced off. (See latest FIRST rules for requirements).

Note: When using ESP32 software tools (**Espressif IDF** for example), CANivore will conveniently boot the ESP32 automatically. But if the persistent setting is still set to disable (as seen in Phoenix Tuner), ESP32 will **not** boot on the next power cycle.

6.7. Hardware Attached Simulation

In the FIRST Robotics space, "Simulation" refers to **compiling an embedded robot application** for the **desktop PC**, thereby allowing testing on the local machine. Any underlying driver that requires interaction with hardware is usually abstracted and re-implemented so that the behavior is as close as reasonably possible to actual hardware.

When the CANivore is **directly connected to Windows PC** [(Note 1)] via USB, there is opportunity to **run with true hardware**, while still having the benefits of local testing and debugging on the host PC. This is referred to as "Hardware Attached Simulation".

With this mode of operation, users can:

- Control supported Phoenix devices with code running on their development PC, providing a better debugging experience.
- Update or configure newly unboxed Phoenix devices (make sure to select "localhost" in Phoenix Tuner's Diagnostic Server Address).
- Use the same Phoenix Tuner features as though you were deployed on the roboRIO.

This is accomplished by modifying the Phoenix vendor json such that, every "windowsx86-64" platform entry is moved to the dependencies entries that already contain "linuxathena".

In other words, there should be "windowsx86-64" everywhere you see "linuxathena", and nowhere else.

A former json example will be provided in the future as manual editing can be prone to error (forgetting to delimit with commas is common).

Note 1: Supporting Linux desktop in future is possible. The CANivore USB kernel driver must be ported to Linux-x86-64.

7. Wiring

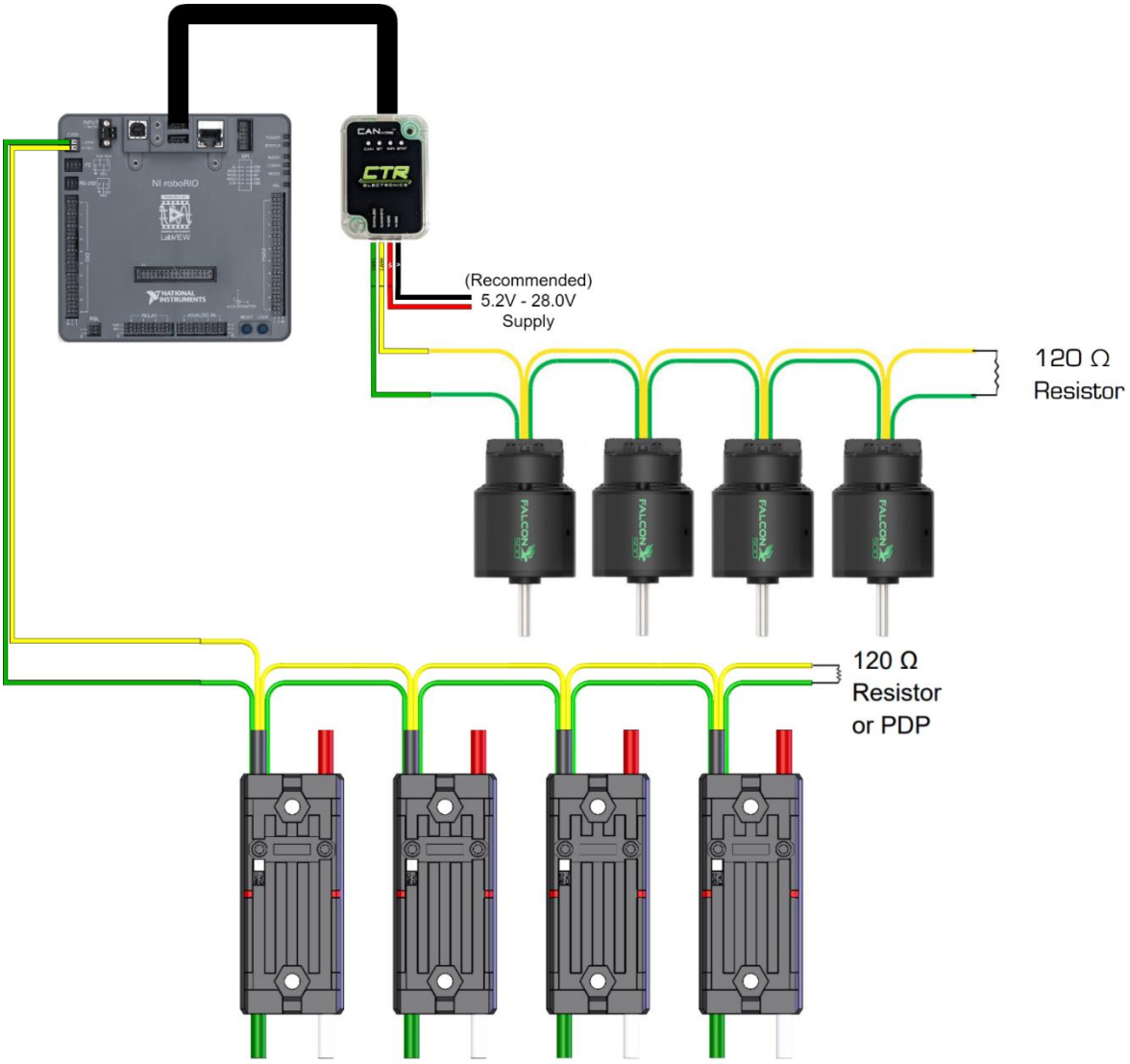
The CANivore provides its own separate CAN FD bus. This means the CANivore should only be connected to the host via the USB cable, and not through another CAN connection. Power can be supplied to the CANivore through USB, or optionally through the V^+ and V^- Weidmuller ports. CAN is connected to CANH and CANL Weidmuller ports, with yellow to CANH and green to CANL.

Common use cases are described in the sections below.

7.1. RoboRIO Wiring

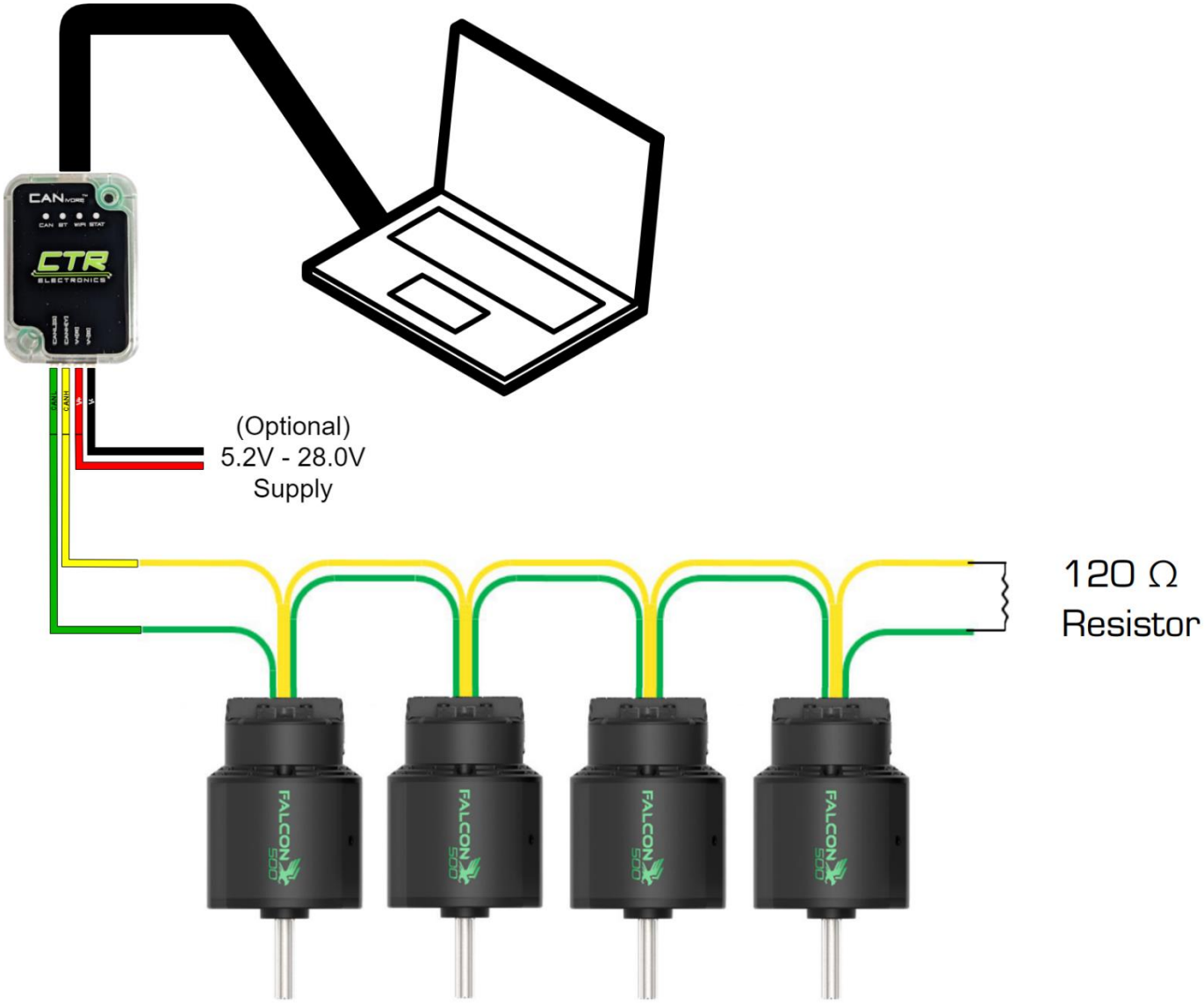
This is the most common wiring configuration for use with an FRC robot. In this configuration, connect the CANivore to the USB A port on the RIO with the supplied USB C Cable, the CAN FD devices through CANH and CANL Weidmullers. Optionally add auxiliary power to the CANivore through the V+ and V- Weidmullers.

A diagram of the wiring is below for reference.



7.2. Hardware Attached Simulation Wiring

The CANivore wiring schematic is slightly different in the Hardware Attached Simulation configuration. In this configuration, the user can connect and control CAN FD devices while simulating robot code on their programming laptop.



7.3. Network Topologies

Since CAN FD is an extension of CAN 2.0, the wiring requirements are similar.

- CAN bus must still have **120Ω of termination at each extreme end**. (Note 1)
- CAN **bus stub lengths** must be short (typical **maximum** of **one foot**).
- **Daisy chaining still recommended** as this reduces stub length to near zero.

In the past with CAN 2.0, teams with very short CAN buses *may* have been able to get some degree of success without ensuring the above requirements are met, such as with star or ring topologies. However, as transmission frequencies increase (such as in CAN FD), it is more important to ensure proper bus topology (daisy chain or short stub lengths).

This is the reason CTR-Electronics incorporates true daisy-chaining (Note 2) in all modern CAN bus products.

Note 1: CANivore can act as one of the terminating resistors with its software-selectable terminating resistor.

Note 2: True daisy chaining means bridging the CANH and CANL lines within the product, and not inside a connector (critical failure point) or inside of breakout far away from the devices.

8. FAQ

8.1. Is there a way to tell if the device is present/powered?

To determine visually if the device is powered and functioning, check the built-in LED, see Section 1.5. LED States.

8.2 How do I tell CAN is connected?

The CAN LED displays the state with regards to CAN. See the LED table in Section 1.5. LED States.

8.3 How do I tell CANivore is powered?

The STAT LED displays the general status of CANivore. See the LED table in Section 1.5. LED States.

8.4 How do I tell CANivore is connected to USB?

The STAT LED displays the USB connection status of CANivore. See the LED table in Section 1.5. LED States.

8.5 How do I field-upgrade the CANivore?

Field-upgrading a CANivore can be accomplished either through the CTRE provided software Phoenix Tuner or through **caniv**. See Section 6.3.

8.6 What do I need to do to use this in FRC?

Following the rules from the 2020 season, no device is allowed to emit wireless signals. This is accomplished by configuring the ESP32 on the CANivore off using Phoenix Tuner. This is sufficient to meet the rules from the 2020 season. Additionally, FRC teams should always confirm what is considered "legal" per the latest FRC competition rules.

8.7 Do I need to install SocketCAN or USB-drivers on to the RIO to use this?

No, every roboRIO image starting with the 2022 season comes with the necessary software to connect and use the CANivore. If there is a mid-season update, Phoenix Tuner will be updated to provide it.

8.8 What happens if CANivore disconnects and reconnects during a match?

Phoenix software constantly monitors the status of every CANivore connected. If a CANivore disconnects during a match, Phoenix will detect it and report the disconnect event to the Driver Station. On reconnection, Phoenix will automatically configure the CANivore to resume operation. No custom scripting is necessary.

8.9 What happens if a non-FD device is connected to CANivore?

The non-FD device will send error frames which causes every device on the CAN bus to blip red. Removing the non-FD device will resolve this.

8.10 What is the ESP32 for?

The ESP32 is a co-processor on the CANivore that allows the user to upload custom code for general use. This feature will be expanded on in future updates.

8.11 How can I program the ESP32?

The ESP is programmable through the emulated serial port exposed through USB. For more information on the ESP, see Section 6.6.

8.12 How do I select a device on this CANivore's bus?

Phoenix API supports an additional parameter for supported device constructors that allow the user to pass in the CAN Bus name. If the name matches the name configured for the CANivore, Phoenix will interface to the CANivore and use it to communicate to the constructed device.

8.13 What is Hardware Attached Simulation?

See Section 6.7.

8.14 Connecting CANivore to my Windows Computer doesn't show up in Tuner

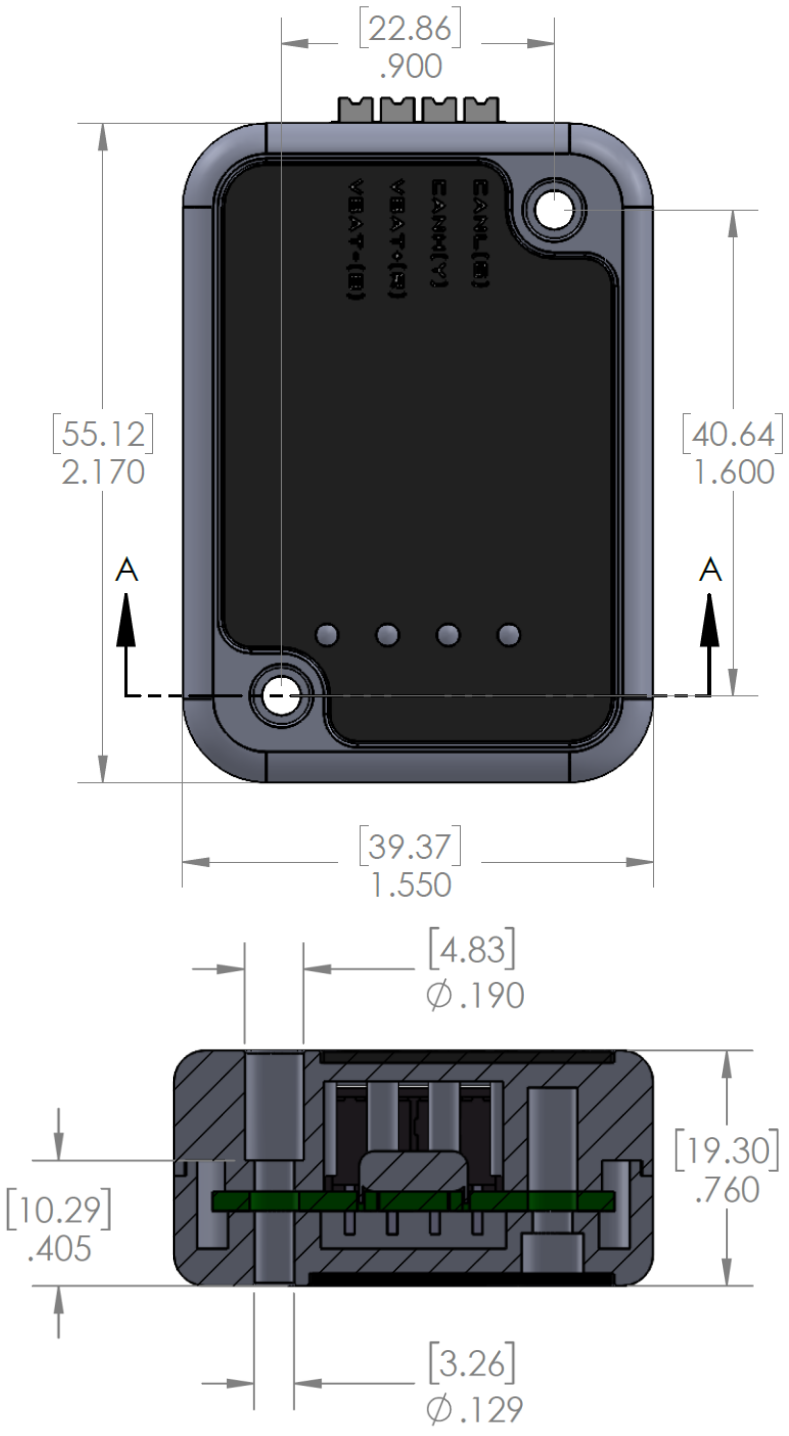
Check device USB LED to confirm connectivity, see Section 1.5. LED States.

If COMM LED is red, recheck all cabling in between CANivore connector and USB root hub (host controller).
If using a USB hub, rule it out by removing it and re-testing.

If COMM LED is healthy, check Device Manger (Section 6.4.1).
If device is present, then confirm the correct string is set in Phoenix Tuner for diagnostic server (Section 6.3).
This is how Tuner knows if you want to use a locally connected USB CANivore or a CANivore that is remote connected to a roboRIO.

In the unlikely event that Windows failed to enumerate the CANivore correctly, a workaround is provided in Section 6.4.4.

9. Mechanical Drawings



10. Revision History

Revision	Date	Description
1.0	21-Jan-2021	Initial Creation.