(/)

Getting Started

Table of Contents

Set the Jumpers

Setting Termination

Determine if you need to use the CANable's onboard termination. CAN bus requires a 120 Ohm terminating resistor on each end of the bus for proper operation, and a completely unterminated bus will not function at all. The CANable has a built in terminator you can use, but if your bus is already terminated make sure to disable onboard termination.

The **CANable 2.0** has a switch to enable and disable termination.

The **CANable Pro** has one jumper to control termination and a button to enter boot mode

- Jumper towards edge of board: termination disabled. Jumper towards inside of board: termination enabled
- To enter boot mode, hold down the small button near the USB connector while plugging into your computer

The original **CANable** has two jumpers: "Boot" and "Term".

- Jumper *towards* the screw terminals: bootloader disabled (normal operation). Jumper *away* from screw terminals: the CANable will enter the USB DFU bootloader when powered up.
- Jumper *towards* the screw terminals: the onboard 120R termination is enabled. Jumper *away* from the screw terminals: termination is disabled.

Connect to the Bus

Connect the CANH, CANL, and GND pins of your CANable to your target CAN bus. *You must connect ground for the CAN bus to function properly.*

Do not connect the 5v output of the original CANable or CANable 2.0 unless you need to power a target with 5v: this is an output only!

Plug it In

Make sure the Boot jumper is not in the "Boot" position, and then connect your CANable to your computer's USB port with a micro USB cable. *Make sure you're using a good USB data cable and not a charge-only cable*.

Install Drivers

Linux and Mac

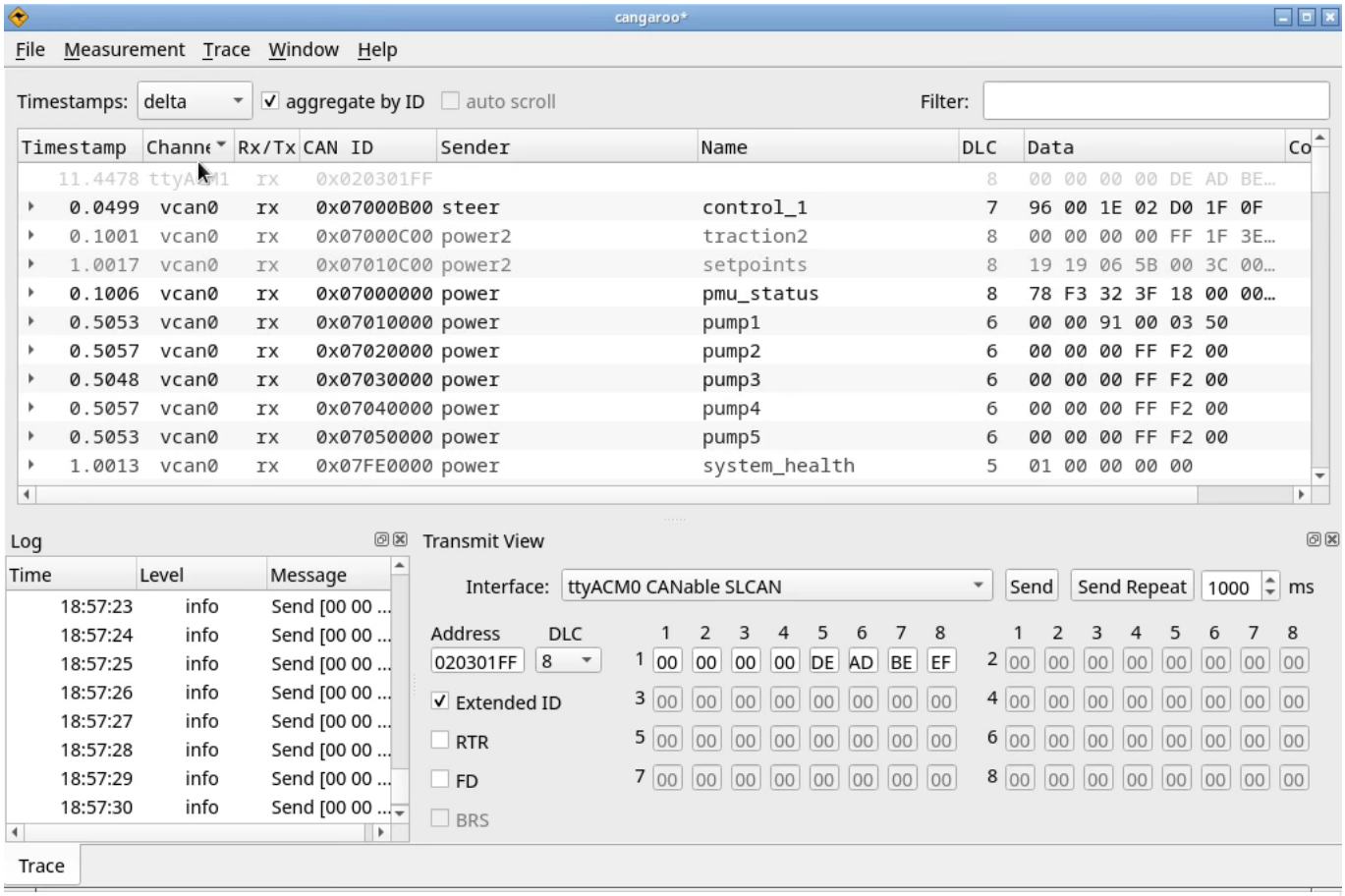
Drivers are not required on Linux and Mac. The CANable will appear as a USB CDC device: */dev/ttyACMX* or */dev/ttyUSBX* on Linux or */dev/cu.usbmodemXXXX* on on Mac.

Windows

Modern versions of Windows do not require a driver for USB-CDC devices, and the CANable will enumerate properly after plugging in.

For legacy Windows users, you may need to install an *.inf* file. You can download a zip of the driver (https://canable.io/utilities/windows-driver.zip), right-click on the *.inf* file, and click install. You may need to open your device manager, find the unknown *cantact* device, choose *Update Driver* and select the *.inf* file.[1]

Applications: Cangaroo



On Windows and Linux you can use with any CANable running either *candlelight* or *slcan* firmware. Cangaroo will auto-detect any CANable interfaces on startup.

Cangaroo has a number of features, including:

- Decoding messages using DBC files
- Transmission of standard and FD frames
- SLCAN and Candlelight firmware support
- Periodic transmission of CAN frames
- Listening to forwarded frames from CANblaster (https://github.com/normaldotcom/canblaster)

Downloads

- Cangaroo (Beta, CANable 2.0 support) for Windows (1/22/2023 ccdcb64) (https://canable.io/utilities/cangaroo-win32-ccdcb64.zip)
- Cangaroo (Legacy) for Windows (https://canable.io/utilities/cangaroo-win32-0363ce7.zip)
- Source Code (https://github.com/normaldotcom/cangaroo/).

Applications: SocketCAN on Linux

The CANable provides a socketCAN-compatible interface that can be brought up with *slcand*. This allows you to use all standard Linux CAN utilities like *candump*, *cansniffer*, and even *wireshark*. Bus speed is specified with the "-s" parameter where:

- -s0 = 10k
- -s1 = 20k
- -s2 = 50k
- -s3 = 100k
- -s4 = 125k
- -s5 = 250k

- -s6 = 500k
- -s7 = 750k
- -s8 = 1M

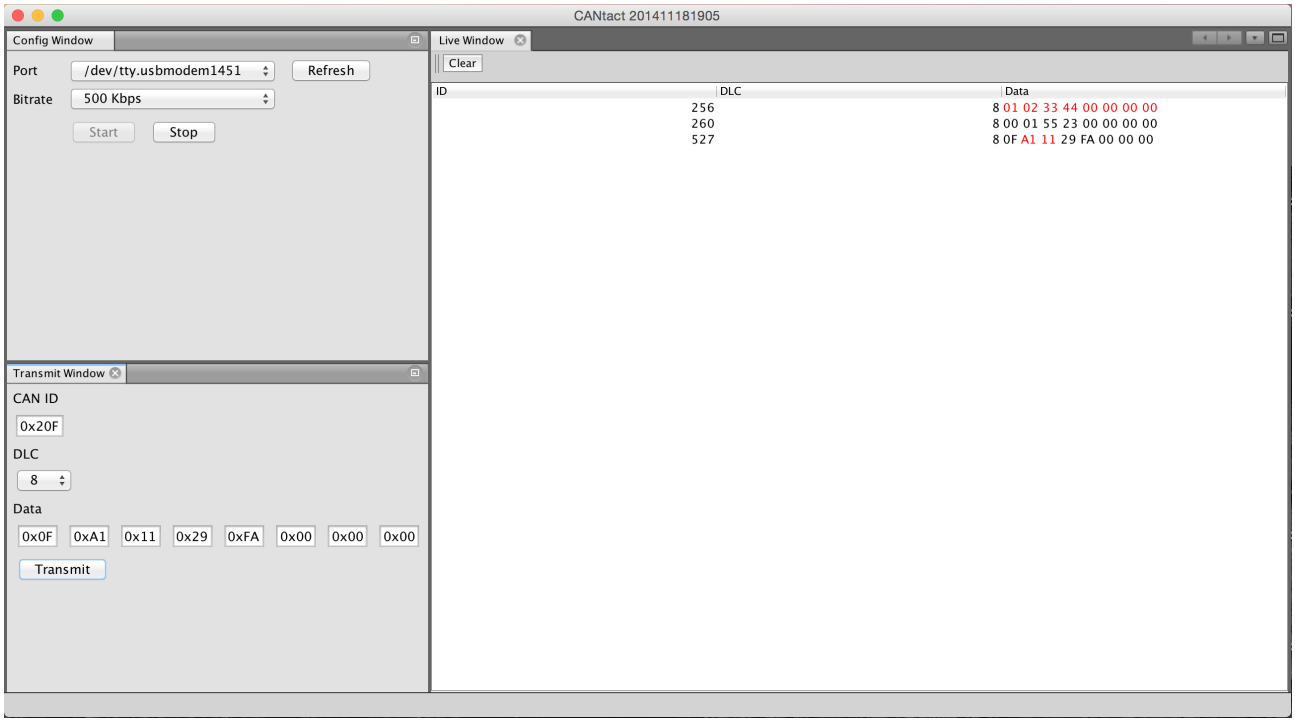
Just run *slcand* with the proper arguments for your bus speed, and a new CAN device should show up on your system. Don't forget to bring the interface up with *ifconfig* after running *slcand*! Now you can use any of the standard Linux CAN utilities to interact with the bus. Make sure that you specify the right TTY port, which you can check with the *dmesg* command after plugging in your CANable.

```
    sudo slcand -o -c -s0 /dev/ttyACM0 can0
    sudo ifconfig can0 up
    sudo ifconfig can0 txqueuelen 1000

    cansend can0 999#DEADBEEF   # Send a frame to 0x999 with payload 0xdea
dbeef
    candump can0                # Show all traffic received by can0
    canbusload can0 500000      # Calculate bus loading percentage on can0
    cansniffer can0             # Display top-style view of can traffic
    cangen can0 -D 11223344DEADBEEF -L 8    # Generate fixed-data CAN mess
ages
```

Applications: *cantact-app* on Windows and Mac

cantact-app is the easiest way to get up and running with your CANable on Windows and Mac.



cantact-app (https://github.com/linklayer/cantact-app) is a Java program for viewing real-time CAN bus traffic and sending CAN packets. This tool connects directly to the virtual serial port of the CANable (or CANtact) device, and doesn't require any other drivers. Just download the latest release (https://github.com/linklayer/cantact-app/releases/download/v0.3.0-alpha/cantact-v0.3.0-alpha.zip), connect to your CANable, and traffic should show up shortly.

Note: *cantact-app* runs on Linux but currently isn't able to discover serial ports.

Applications: Using CANable from Python with python-can

python-can is a python library that allows you to easily communicate on the CAN bus from Python. The library supports connecting to CANable/CANtact devices directly with via a serial connection on Windows or Linux and also can directly work with socketcan devices on Linux with the candlelight firmware.

Documentation on *python-can* is available here (https://python-can.readthedocs.io/).

Getting started is quite simple:

```
import can

# Candlelight firmware on Linux
#bus = can.interface.Bus(bustype='socketcan', channel='can0', bitrate=5000
00)

# Stock slcan firmware on Linux
bus = can.interface.Bus(bustype='slcan', channel='/dev/ttyACM0', bitrate=5
00000)

# Stock slcan firmware on Windows
bus = can.interface.Bus(bustype='slcan', channel='COM0', bitrate=500000)

msg = can.Message(arbitration_id=0xc0ffee,
                  data=[0, 25, 0, 1, 3, 1, 4, 1],
                  is_extended_id=True)

try:
    bus.send(msg)
    print("Message sent on {}".format(bus.channel_info))
except can.CanError:
    print("Message NOT sent")
```

Note: Previously, the CANard library was recommended for use with the CANable. CANard (legacy-canard-with-python.html) is still available but is not recommended for new projects.

Alternative Firmware: candleLight

There is a port of the candleLight (https://github.com/normaldotcom/candleLight_fw) USB to CAN firmware for CANable. The port works very well under Linux using the gs_usb driver. This firmware does not use slcan, so it is not interchangable with the stock firmware. However, the CANable appears as a CAN interface natively in Linux, works with the Cangaroo app (below) on Windows, and performs very well under high bus load conditions.

You can update your CANable to *candlelight* using the CANable Updater (https://canable.io/updater) site, or refer to Flashing New Firmware.

Note that Linux kernels <=4.9 had a bug in the gs_usb driver. I created a patch which is now in the mainline kernel, but if you need to compile the standalone module, you can use this custom version (https://github.com/normaldotcom/socketcan_gs_usb).

With the candlelight firmware, simply plug in the CANable and the device will enumerate as can0. Set the baud rate and bring the interface up with the following command, and you're good to go!

```
ip link set can0 up type can bitrate 500000
```

udev Rules

If you are using multiple CANables with the candleLight firmware, you may want to use udev rules to assign each serial number a fixed socketcan device name (can0, can1, etc) that persists between reboots and plugging/unplugging the device. This is easy with udev rules.

First, create a new udev rule file such as

```
/etc/udev/rules.d/99-candlelight.rules
```

This file will contain your rule. Place the serial number of your device (check dmesg after plugging it in, or use the usb-devices command) and desired device name in this file. No other values need to be changed. Add a line to this file for each device you would like to configure. I recommend setting the device name to can3 and higher, as devices without udev rules will still enumerate as can0, can1, etc.

```
SUBSYSTEM=="net", ATTRS{idVendor}=="1d50", ATTRS{idProduct}=="606f", ATTRS
{serial}=="000C8005574D430A20333735", NAME="can5"
SUBSYSTEM=="net", ATTRS{idVendor}=="1d50", ATTRS{idProduct}=="606f", ATTRS
{serial}=="000D8005574D430A20333735", NAME="can6"
SUBSYSTEM=="net", ATTRS{idVendor}=="1d50", ATTRS{idProduct}=="606f", ATTRS
{serial}=="000E8005574D430A20333735", NAME="can7"
```

Reboot your system or run the following commands and unplug/replug your device and the udev rule will assign the interface number after enumeration.

```
sudo udevadm control --reload-rules && sudo systemctl restart systemd-udev
d && sudo udevadm trigger
```

Firmware Sources

The stock CANable slcan firmware is available on Github (https://github.com/normaldotcom/canable-fw). Brief documentation on the slcan ASCII format is available in the README file of the repository.

Builds

The as-shipped build of the slcan and candleLight firmware are available for download here (http://canable.io/builds/).

Updating Firmware

Reprogramming your CANable is fairly easy. First, move the "boot" jumper into the boot position as labelled on the PCB and then plug it into your computer.

Web App

You can easily update your CANable by browsing to the CANable updater site (https://canable.io/updater) with Google Chrome. Follow the instructions to easily reflash your CANable.

ST DFU Tool

If you are running Windows, download ST's dfuse (http://www.st.com/web/en/catalog/tools/FM147/CL1794/SC961/SS1533/PF257916) tool and follow ST's guide (http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00155676.pdf) for installing the driver for the DFU device, generating a DFU file, and flashing the device.

dfu-util on Linux and Mac

If you're on Linux or Mac, install *dfu-util* from your distro's package manager or from brew on OSX. Run the following command to flash your device:

```
sudo dfu-util -d 0483:df11 -c 1 -i 0 -a 0 -s 0x08000000 -D canable-firmwar
e.bin
```

After flashing the firmware, return the boot jumper to its original position and plug/unplug your device. You are now running new firmware!

Questions and Support

If you have any problems getting your CANable up and running or if you have any questions, feel free to send me an email (mailto:support@canable.io).

1. Thanks to Colin O'Flynn (https://github.com/colinoflynn) for the signed driver ↩

Tags (/tags.html)

Eric Evenchick (http://evenchick.com)
Ethan Zonca (http://ethanzonca.com)

Openlight Labs Store
(http://openlightlabs.com/)