BusEye CL2 CAN Data Logger

Overview

The Perspic BusEye CL2 (CL2 = "Can Logger" 2-channel) is a data acquisition device designed for troubleshooting CAN networks and system behavior. It connects to two CAN Bus networks and records data to an SD card. An onboard real-time clock timestamps the data for thorough analysis. The device is easy to use yet powerful enough to diagnose complex system issues and log data for detailed review. The BusEye CL2 is configured dynamically from a JSON file on the SD card, making configuration changes quick and easy.

The BusEye CL2 comes standard with mounting flanges and a DT/AT automotive waterproof plug, with an optional M12 plug. The standard device is rated IP54 (resistant to splashing water), with an optional IP68 rating (resistant to submersion).

Features

- 2 Channel CAN Bus data recorder
- Configurable via JSON file on SD card
- Records to CSV file or raw data file
- Log file analysis available in CSV or graphing via ASAMMDF GUI

- Optional baud rates from 5 K bit/s to 1 Mbit/s
    - 5, 10, 50, 100 125, 250, 500, 800, 1000 Kbit/s options
    - Future support: CAN-FD Compatibility
- Optional 120 Ohm termination resistors with solder jumper configuration
- RTC (Real-Time Clock) for time-stamped data acquisition
    - Settable via file on SD Card or USB-C connection to computer
- Wide input voltage range of 6-28v DC
    - Reverse polarity protection
- IP54 rating (splash-resistant); IP68 (submersion-resistant) optional

Applications

CAN Bus data transmission is widely used in vehicles, mobile equipment, and industrial communication systems. Modern vehicles often have multiple CAN Bus networks to pass data between ECUs, sensors, and control devices. Troubleshooting system behavior is essential for ensuring proper operation of CAN Bus systems. Acquiring CAN Bus data is critical for understanding system behavior during testing, design, and reverse-engineering.

Enterprise-grade data acquisition systems are typically expensive, but the BusEye CL2 delivers advanced features at a fraction of the cost of comparable solutions.

Device Specifications

Electrical

- *Input Voltage*: 5-28 V DC (See Power Supply Recommendations)
- *Current Draw*: 30-60 mA
- *CAN Baud Rate*: 5, 10, 50, 100, 125, 250, 500, 800, 1000 Kbit/s (configured in config.json)
- *CAN Resistors*: Optional, ships with no resistance, solder jumper to enable termination resistors
- *Logging Performance*: Processes up to 1,200 messages per second.
- *Serial Debug Interface*: baud rate 115,200; displays information about device status. Send 'h' character to view menu.
- *USB Interface*: USB-C for console debug and SD Card file access
- *RTC Backup Time*: Up to 6 years

Power Supply Requirements

RTC functionality is sensitive to power supply glitches. If power supply cycles on and off within 20 ms, the device may lock up while switching from standard operation into battery backup mode.

When utilizing battery backup for RTC functionality, ensure power supply switching occurs no more frequently than 20 ms to prevent lockup.

If a lockup occurs, reset the device by removing and reinserting the CR2032 battery, then reset the RTC via USB or the NOW.txt file.

Pin Functions

6-Pin Deutsch/AT Style

Pin Number	Function
1	CAN1_H
2	CAN1_L
3	VCC/PWR
4	GND
5	CAN2_L
6	CAN2_H

Mating Connector Part Number: AT06-6S (DT06-6S)

M12 A-Key Style

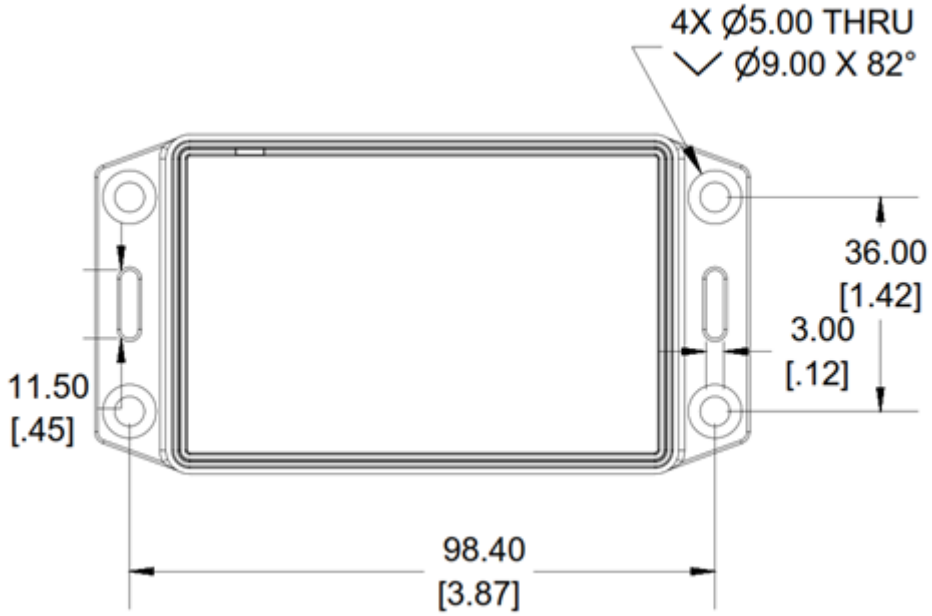
Pin Number	Function
1	N/C
2	VIN
3	GND
4	CAN1_L
5	CAN1_H

Mating part number example: T4110001051-000 (M12 A-Key Female, Straight, TE Part)

Physical Specifications

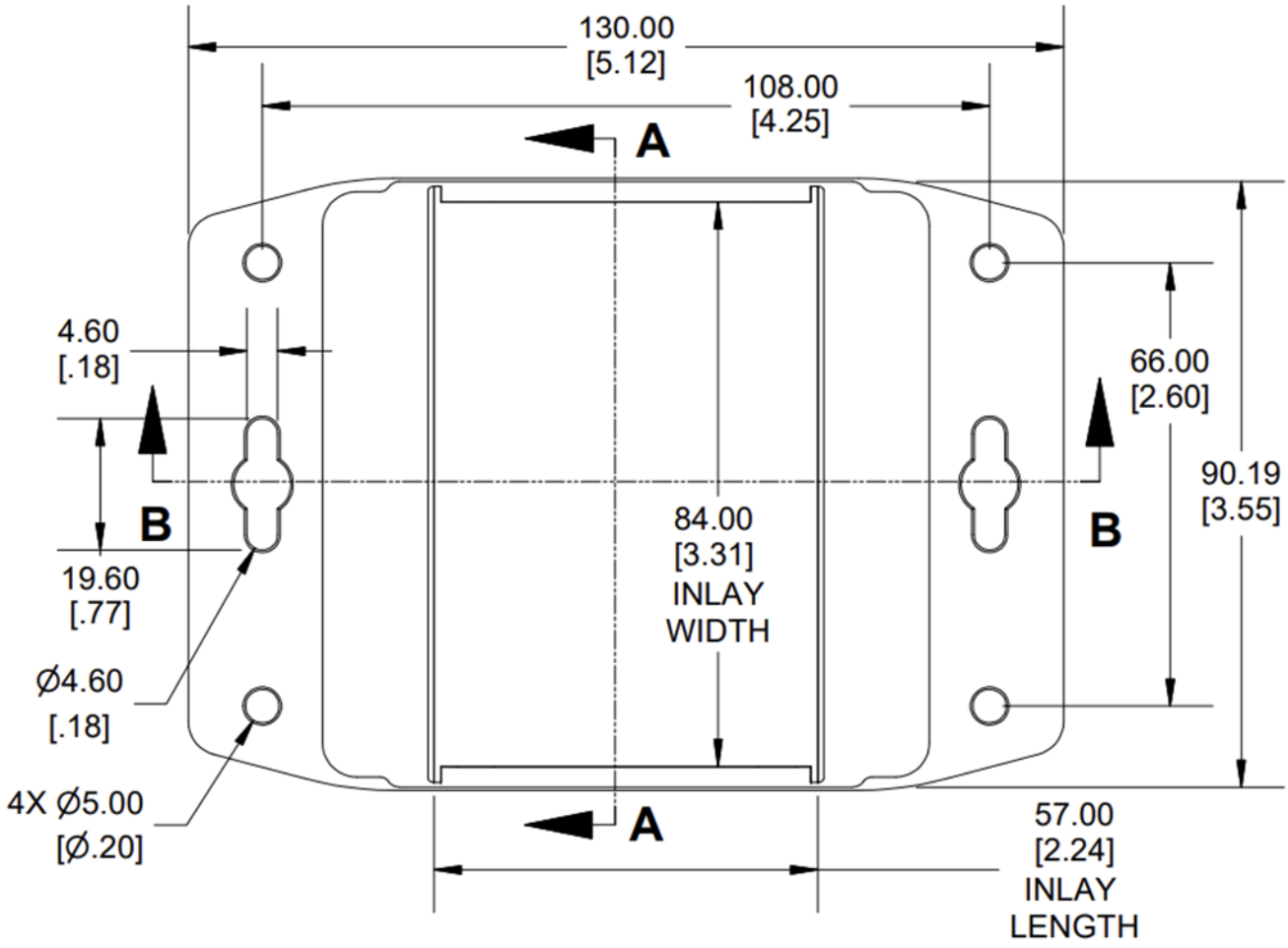
Physical - IP54 ABS Enclosure (Standard)

Dimension	Measure (Pigtail)
Width	5.8 cm
length	13 cm (measured from strain relief) 8.6 cm (case only)
height	3 cm
Weight	110 g
Cable Length	15 cm
IP Rating	IP54
Hole Pattern	Rectangular 4x 5mm countersunk through holes
Temperature	-20 to 80 C
Humidity	0 to 95% (Non-Condensing)



Physical - IP68 Polycarbonate Enclosure

Dimension	Measure (Pigtail)
Width	9 cm
length	13 cm (measured from flanges) 9 cm (case only)
height	3.4 cm
Weight	200 g
Cable Length	15 cm
IP Rating	IP68
Hole Pattern	Rectangular 4x 5mm countersunk through holes
Temperature	-40 to 120 C
Humidity	0 to 100%



Device Operation

Config File

A config file is required to start logging. Configuration is set via a JSON file. Any parameters not set, misspelled, or not found in the JSON file will revert to the default values shown below.

> Note: all parameters are case sensitive, all parameters use lowercase with underscores.

> Note: Errors in the json file could cause the device to fail to start. We recommend checking the file with JSONLint before uploading to ensure proper JSON formatting.

> Note: Maximum configuration file size is 16 KB

Config File Parameters:

- max_file_size - Integer. Maximum file size before creating a new log file. Unit: bytes (Default: 1,000,000,000, ~1GB)
- unit_type - String. Unit Type, project type or model name. Must be 20 characters or less. (Default: "")
- unit_number - String. Unique equipment or unit identifier (e.g., serial number). Must be 20 characters or less. (Default: "")
- log_type - String (3 Characters). Options: `"CSV"`, `"DAT"`, `"BLF"`(future). (Default: "CSV")

  `DAT` is a text-based data format that results in a smaller file size but is not readable in Excel.
- timestamp_format_seconds - Boolean. If `true`, timestamps are in seconds since log file start. If `false`, timestamps are in ISO 8601 format (Default: `true`)

  > Note: ISO Timestamp accuracy is to the second, which may result in loss of message order.
- overwrite_logs - Boolean. If `true`, when log file limit (>9999) is reached or SD card space is full, older logs will be deleted to free space. (Default: false)
- print_filter_number - Boolean. If `true`, the log file will include the filter number that matched the CAN frame. If not filter matched, the filter number is 127. (Default: false)

Can Bus Configuration (`can_1`, `can_2`)

- baud_rate - Integer. CAN Bus baud rate in Kbits/s (Default: 500)
- bus_name - String. Name or description for the bus. Must be 20 or fewer characters. (Default: "can_x" where x is bus number)
- accept_standard_non_matching_frames - Boolean. (Default: true)
  - If true: Frames that do not match filters are accepted and logged.
  - If false: Frames that do not match the filters are dropped.

> If no standard filters are defined: all frames are evaluated on this boolean and this flag evaluates to "log all standard frames"

- accept_extended_non_matching_frames - Boolean. (Default: true)
    - If true: Frames that do not match filters are accepted and logged.
    - If false: Frames that do not match the filters are dropped.

> If no extended filters are defined: all frames are evaluated on this boolean and this flag evaluates to "log all standard frames"

- bus_enabled - Boolean. (Default: true)

  If false: the CAN hardware remains off.

- listen_only - Boolean. (Default: false)
    - If True, the CAN hardware will only listen to the bus; it will not transit or acknowledge messages.
    - Do not enable this if the logging device is one of two devices on the bus.
- filters - JSON list of filter elements for the given bus (Default: none)

Filter Elements

Filter elements are processed in top-to-bottom order. Once a matching filter element is found the frame is processed with that filter element. If no matching filter is found for a frame the default behavior defined in "accept_<standard/extended>_non_matching_frames" determines whether the frame is accepted or rejected.

> Note: Short-forms can be used to reduce the JSON file length when using many filters (see example config).

- type (or "t") - String. Options: `"range"`, `"r"`, `"dual"`, `"d"` or `"classic"`, `"c"`) range: Eny ID between ID1 and ID2 is matched. dual: Either ID1 or ID2 are matched. classic: ID1 is the filter; ID2 is the mask.
- ID1 - Integer or hex string.
- ID2 - Integer or hex string.
- action (or "a") - String. Options: `"accept"`, `"a"`, `"reject"`, `"r"`, or `"accept_priority"`, `"ap"`) accept: Log the CAN frame, reject: Do not log the CAN frame. accept_priority: Add the CAN frame to the beginning of the buffer so that it is logged first.

Classic Filters

The classic filter uses an ID (ID1) and a mask (ID2) to filter CAN frames through bitwise comparison. The mask determines which bits to compare: a 1 in the mask means the corresponding bit in the frame ID must match the filter ID, while a 0 means the bit is ignored. For example, with a filter ID of 0b10101010 and a mask of 0b11110000, a frame ID of 0b10101100 would match because the upper 4 bits align with the filter ID, while the lower 4 bits are ignored due to the mask. This allows flexible filtering by focusing only on relevant parts of the frame ID.

Example Config

```
{
    "max_file_size": 10000000,
    "unit_number": "test_unit",
    "unit_type": "test_type",
    "passthrough" : false,
    "can1":{
        " ": 500,
        "bus_name": "test_bus",
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "bus_enabled": true,
        "listen_only": false
        "filters": [
            {
                "type": "range",
                "ID1": 123,
                "ID2": 234,
                "action": "accept"
            },
            {
                "t": "r",
                "ID1": 123,
                "ID2": 234,
                "a": "a"
            }
        ]
    },
    "can2":{
        " ": 500,
        "bus_name": "test_bus_2",
        "accept_standard_frames": true,
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "bus_enabled": true,
        "listen_only": false
    },
    "overwrite_logs": true,
    "log_type": "DAT",
    "timestamp_format_seconds": true
}
```

Time Setting

Setting Time by Console

To set a new RTC time via console, connect the USB-C plug to a computer. A COM port and a disk drive (if SD Card is inserted) will appear on the computer.

On Windows

Run the PowerShell Script: ./set-time-CAN-Logger.ps1 download set-time-CAN-Logger.ps1 here

- Enter the COM port number for the logger.
- Monitor the output - it will display all COM data from the device, including bus configurations.
- Confirm the last line of output contains: `"Set RTC to new Time: <current time>"`

On Linux

Run the Bash script: ./set-time-CAN-Logger.bash download set-time-CAN-Logger.bash here

- This script does the same as the powershell script but is untested on linux

Setting Time by File on SD Card

To set a new RTC time, create a file named `NOW.TXT` on the SD card. The file must contain a single line with Unix time (number of seconds since January 1st, 1970) as an integer. On boot, after detecting the SD Card, the device will search for `NOW.TXT` before starting logging.

If the time set in `NOW.TXT` is before Jan 10, 2021 the RTC ti will not be set (as this predates the device's manufacture).

Once the RTC is set, a message confirming the update will be sent to the console, and `NOW.TXT` will be deleted.

Steps to Set Time:

1. Choose a time 1-2 minutes in the future and generate a unix timestamp for that time.
2. Write the integer to a text file named `NOW.TXT` on the SD Card.
3. Insert the SD card into the device.
4. Power up the device as close to the desired time as possible.
5. Verify that time was set correctly by checking the log file's creation date (in UTC) or the first line header of the log file.

> NOTE: Ensure the RTC battery is installed before setting the time to retain the settings after restarting the device.

Data Log Formats

Every log file start with two header lines.

1. A JSON object outlining parameters and settings used for the data log.
2. A header line defining the data format of the following lines.

Two data formats are available: `CSV` and `DAT`. Each CAN message is recorded as a single line in the log file, with messages from both buses written to the same file.

Timestamps for each message are recorded in seconds since the start of logging, unless the setting timestamp_format_seconds is `false`, in which case a full ISO timestamp is printed in every line.

Metadata Format

The first line of every log file is a JSON object containing metadata about the log session, including:

- Start time of logging
- CAN Bus configuration
- Log Type (CSV or DAT)
- Unit type and unit number set in the configuration file

> NOTE: If RTC battery is not present every reboot will default datetime to 2020-01-01. (Also occurs on RTC Battery failure)


```
{
    "can_1": {
        " ": 1000,
        "bus_enabled": true,
        "bus_name": "test_bus",
        "bus_number": 1,
        "accept_extended_non_matching_frames": true,
        "accept_standard_non_matching_frames": true
    },
    "can_2": {
        " ": 1000,
        "bus_enabled": true,
        "bus_name": "test_bus_2",
        "bus_number": 2,
        "accept_extended_non_matching_frames": true,
        "accept_standard_non_matching_frames": true
    },
    "log_start_time": "2020-01-01T00:04:15.",
    "log_type": "DAT0.1",
    "unit_number": "test_unit",
    "unit_type": "test_type"
}
```


CSV Format

The CSV format is easy to read and can be directly opened with any spreadsheet software

> Note: The JSON header line may need to be deleted to properly open the CSV file in spreadsheet software.

```
timestamp,CAN_BUS,CAN_EXT,CAN_ID,CAN_LEN,Data0,Data1,Data2,Data3,Data4,Data5,Data6
,Data7,filter
0.000,1,0,24,8,FF,FF,FF,AA,BB,CC,DD,EE,127
0.002,2,0,7E8,8,06,41,A4,FF,FF,FF,00,00,127
```

DAT Format

The DAT format reduces file size by approximately 30% by minimizing the number of characters per message.

Format is: `<timestamp>-<CAN Bus Number>-<CAN ID in Hex>#<DATA BITS in Hex>`

- CAN Bus number is 1 or 2
- CAN IDs will be zero-padded to the number of bits in the ID (E.G. extended frames will always show 8 characters, standard frames will show 3 characters).
- Data bits will be zero-padded to the length of DLC (E.G. if only 4 bits are sent but DLC is 6 there will be two leading zero bytes, or four characters).

Example:

```
timestamp-CAN_BUS-CAN_ID#Data0..7
13.557-2-021#FFFFFFAABBCCDDEE
```

USB Connection

The BusEye CL2 has an onboard USB-C connector that provides console and file services over USB. When connected to a Windows PC, it will appear as a USB Composite Device with a Removable Disk and COM port.

- USB Vendor ID 0xAD50
- USB Product ID: 0x60C4.

USB File Connection

When connected to a computer via USB, the CAN Logger's SD card appears as a standard USB drive, allowing easy configuration and file transfer.

> Note: File transfers over USB may be slow. For large transfers transfers, it's recommended to remove the SD card and transfer files directly.

USB Console

The USB device console allows you to view settings, adjust basic parameters, and configure logging. Simply connect the USB-C cable to a computer, and the data logger will appear as a console device. Use a console program like (PuTTY)[https://www.putty.org/] or (Tera Term)[https://teratermproject.github.io/index-en.html] to manually access menus and set the time.

Command 'h': Show help message

Command 'c' Reload CAN Bus Configuration from config file and restart logging (useful for modifying config through USB)

Command 't': Show current RTC Time

Command 'j': Show Logging and CAN Bus Configuration

Command 'k': Show Logging and CAN Bus Status

Command 'l': Show LED Status

Command 'p': Toggle CAN Message passthrough from BUS1 to BUS2

Command 'r' 'g' 'b': Toggle RGB LEDs

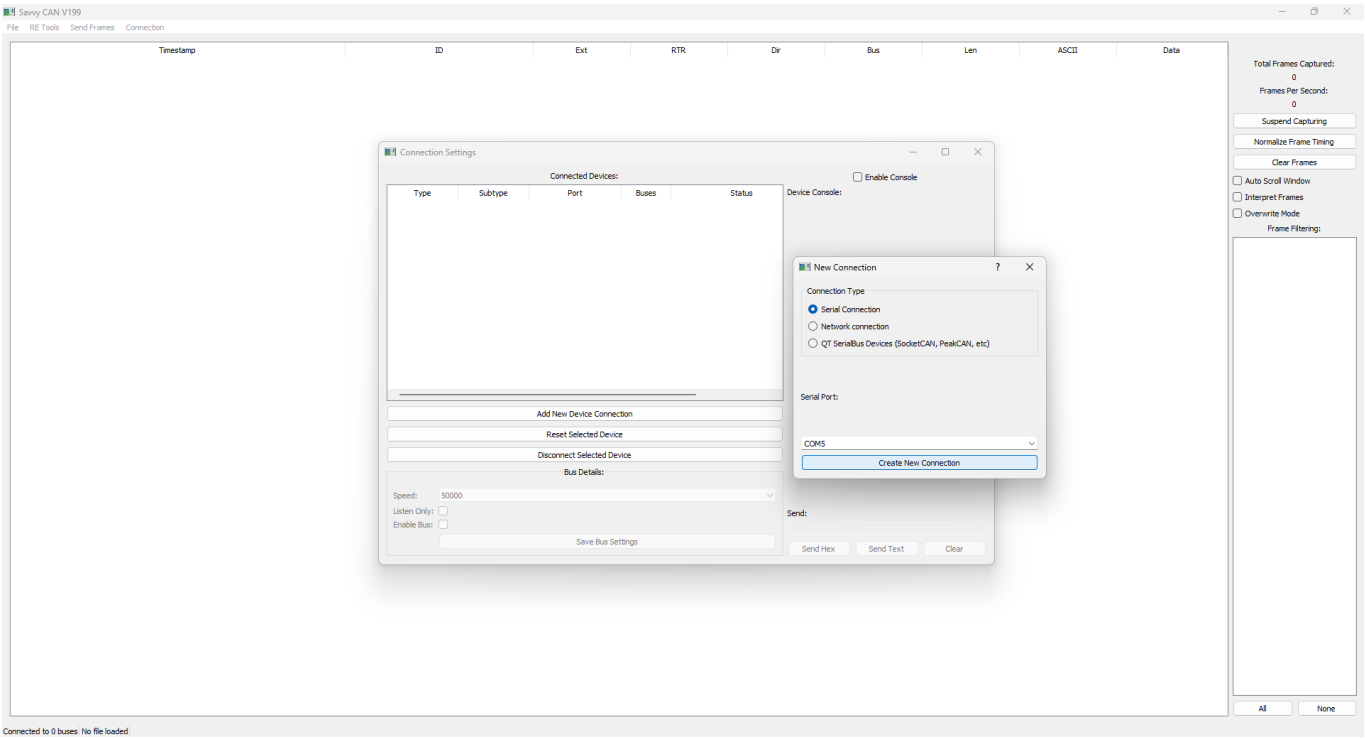
Command '1' '2': Toggle sending CAN Data to USB Serial in human-readable format

Command 'v': Print CAN Logger Software Version

Command 'ç': Toggle to toggle CAN Serial setup

Can Serial Setup

The USB Console can be function as a *GVRET* device and can be used with SavvyCAN. This is particularly useful in passthrough mode for retransmitting messages (see Passthrough Mode section).



More information: GVRET Github

> Note: CAN Bus numbers in *GVRET* mode are 0 (Bus 1) and 1 (Bus 2). Note: CAN Bus speeds are not overwritten in *GVRET* mode - busses must be configured in the JSON file before connecting to *GVRET*. Note: When the device is configured in *GVRET* mode, the USB console is locked and cannot be used for other configuration.

Advanced Device Operation Modes

Viewing CAN Data on the Console

Once the device busses are configured, CAN data can be printed to the USB console to troubleshoot connection issues. Send commands `1` and `2` to toggle printing of raw CAN messages to the console.

Passthrough Mode

CAN Passthrough Mode forwards CAN messages from one bus to the other. To enable passthrough:

- Set "passthrough" to `true` in *config.json*
- OR send command `p` to the USB console toggle passthrough mode.

> Note: Baud rates configured in *config.json* is not overwritten by serial requests. The busses will operate at the configured baud rates in *config.json*.

Performance and System Behavior

Bus Errors

If the logger detects protocol errors, it will put the CAN hardware into sleep mode to prevent bus arbitration errors. In this state, logging will be disabled, and a warning message will be printed to the serial port.

SD Card Removal

The SD Card can be removed at any time. When Removed, the device will pause CAN busses and wait for the SD Card to be reinserted. Once reinserted, the configuration file will be read, and the device will reinitialize with the new configuration.

Performance considerations

At CAN message rates above about 1,200 messages per second, performance may be affected. Before dropping frames, the logger will buffer as many messages as possible. The alternating nature of CAN1/CAN2 file writing can cause multiple CAN1 messages to be logged before CAN2 messages received at the same time. At higher bus loads, timestamps may not follow strict chronological order.

Tested Performance Numbers:

- Test setup: 1 Mbit/s baud rate, standard IDs, full frame data (8 bytes).
- Two CAN Busses logging simultaneously:
  - 600 messages/sec per bus -> 100% of messages are logged. (Total 1,200 messages/sec)
- One CAN Bus logging:
  - 800 messages/sec -> 100% of messages are logged.

> NOTE: Actual performance may vary based on other factors. Exceeding the tested message rate may cause frames to be dropped from log.

Advanced Configuration and Device Internals

LED Status Lights

- Green LED (beside power connector) - Indicates that internal device power is on.
- Status LED (beside USB-C Connector) - Indicates device runtime status:
  - Red - No SD Card is present
  - Green - Device is operating normally
  - Blue (flash) - Device is writing data to the SD Card

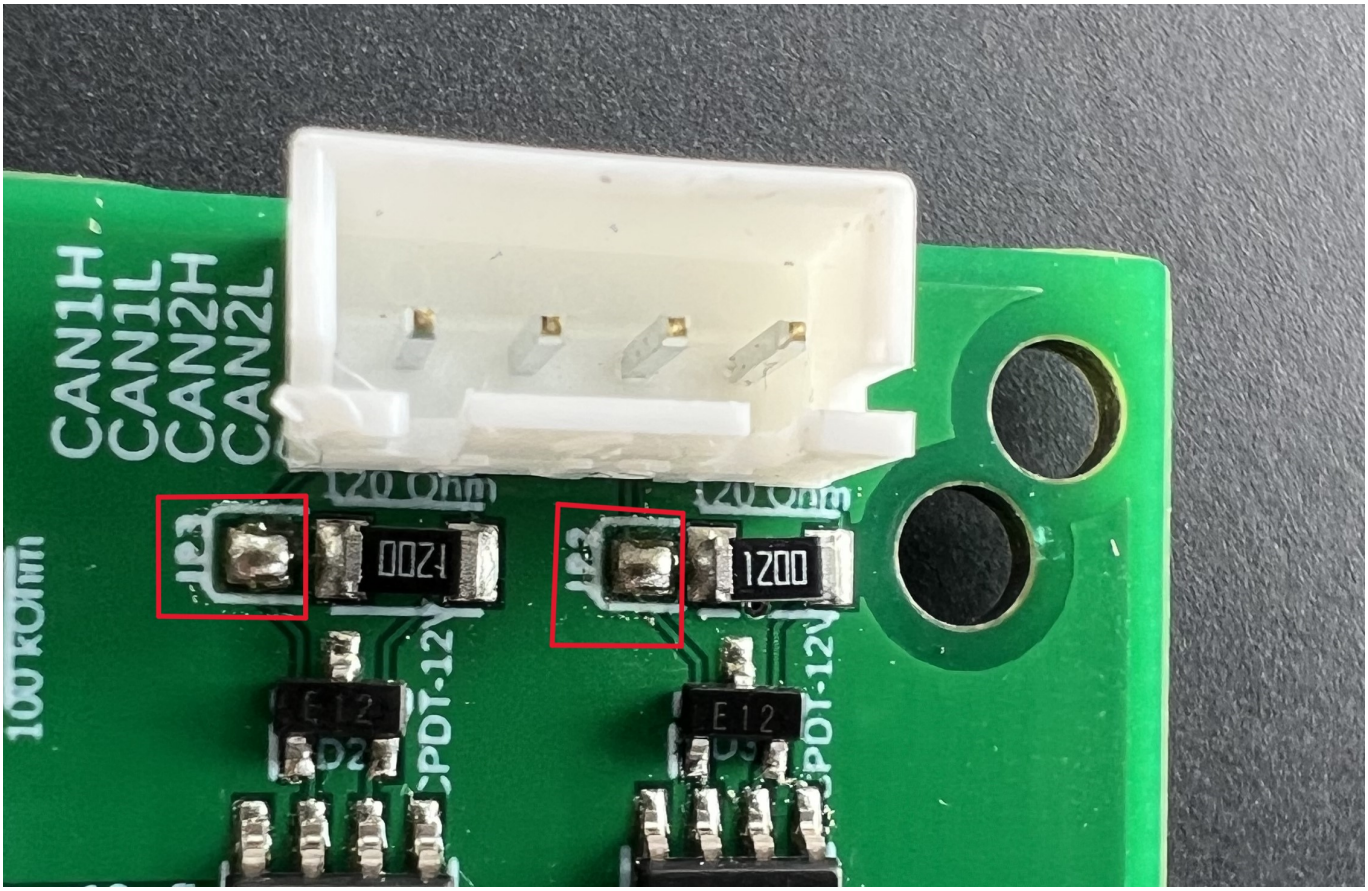
> Note: If the green power LED is on but the Status LED is off, the device may be locked in battery backup state. Remove and replace the RTC battery to reset.

RTC Battery

A CR2032 battery holder on the bottom side of the PCB holds the RTC battery. To replace the RTC battery, remove the PCB from the case and push the battery out of the holder.

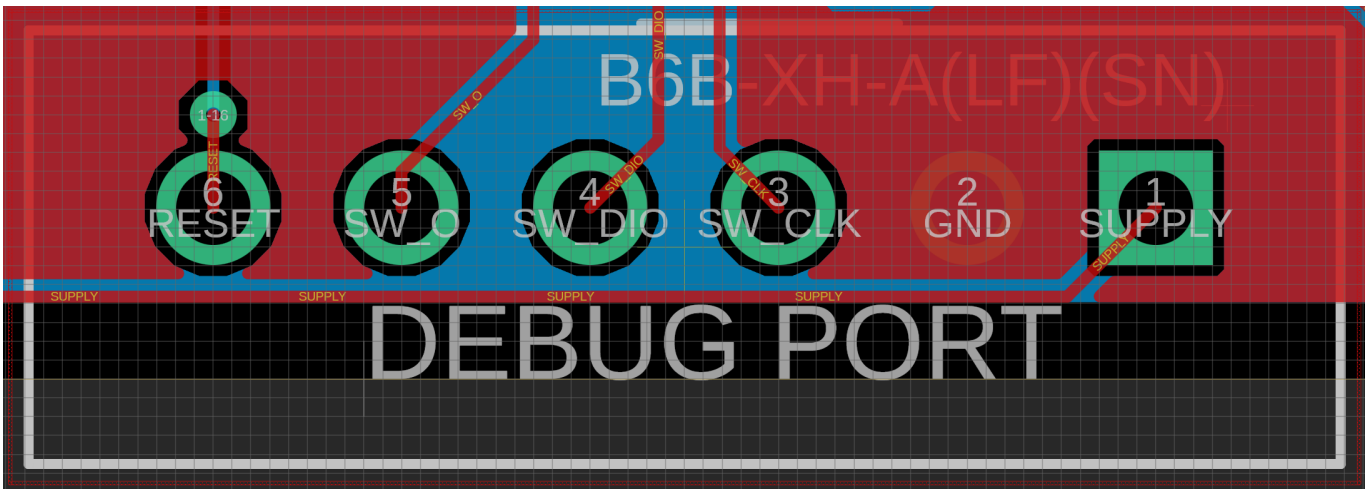
CAN Termination Resistor Jumpers

Solder the jumpers if termination resistance is required for the node. (Default: unsoldered)



Debug Interface Pinout

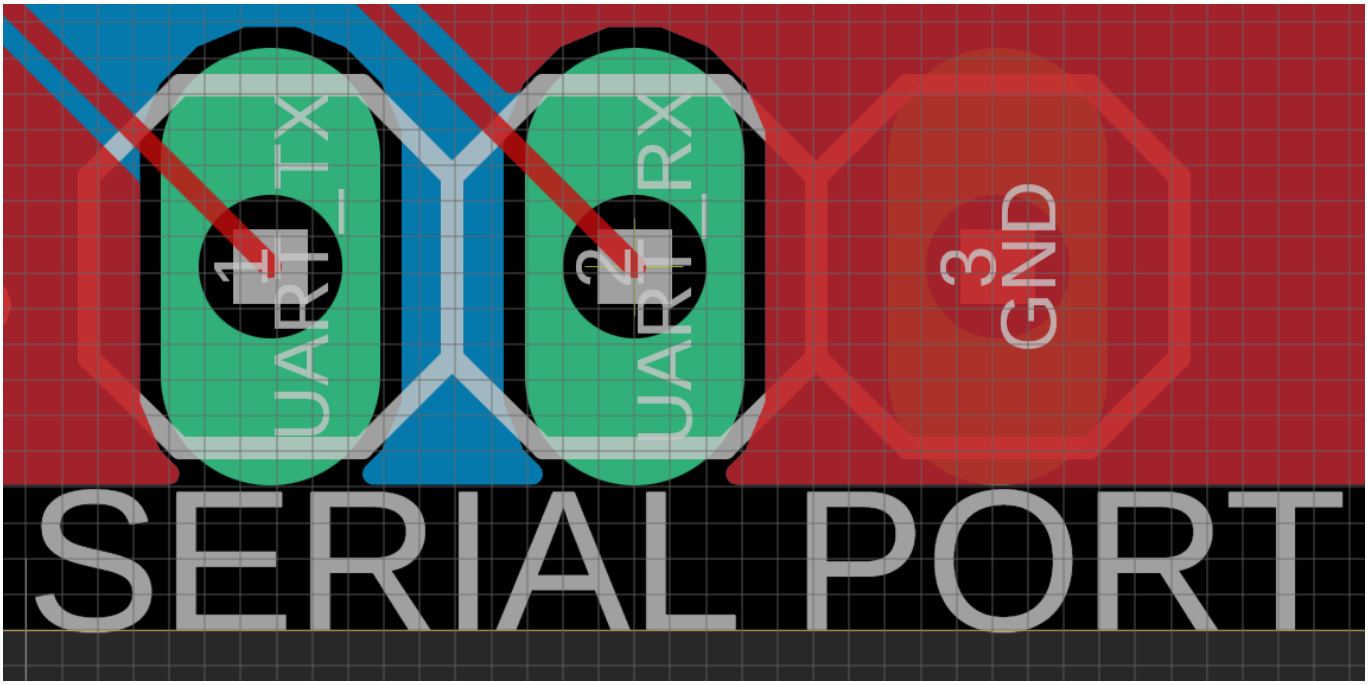
The debug interface follows MPLAB PICKit 4 pinout. The PICKit 4 can be directly connected to re-flash the device.



FUNCTION	BOARD PIN	PICKIT4 PIN
RESET	6	1
SW_O	5	4
SW_DIO	4	8
SW_CLK	3	5
GBD	2	3
SUPPLY	1	2

Serial Debug Interface Pinout

Serial pins are provided running at 112,500 baud rate. The serial console provides some debug messages above and beyond the USB Console.



FUNCTION	BOARD PIN
TX	1
RX	2
GND	3

Tests and Certifications

RF Emissions: Tested per CISPR22 Edition 5.2 2006-03 Class A
EMC: Tested per IEC 61000-4-2 Edition 3.2 2010-04 Level 2
ESD Immunity: Tested per IEC 61000-4-2 Edition 2.0 2008-12 Class 1

Part Numbers

Part Number	IP Rating	Wiring Option	Temperature Range	Connector
10-24001	IP54	15 cm pigtail	-20 to 80 C	Deutsch 6-Pin AT
10-24011	IP68	15 cm pigtail	-40 to 120 C	Deutsch 6-Pin AT

Don't see the configuration you need? Contact Us for specialty configurations and custom connectors

Definitions

- **CAN (Controller Area Network)** – A robust vehicle bus standard designed to allow micro-controllers and devices to communicate without a host computer.
- **Baud Rate** – The speed at which data is transmitted over the CAN network, measured in Kbit/s (e.g., 250 Kbit/s, 500 Kbit/s).
- **CAN Frame** – A single message sent on the CAN bus, containing an ID, data length, and data payload.
- **Standard CAN ID (11-bit)** – The shorter CAN identifier format, using 11 bits for the message ID.
- **Extended CAN ID (29-bit)** – The longer CAN identifier format, using 29 bits for the message ID.
- **DLC (Data Length Code)** – Specifies the number of bytes in a CAN frame's data payload (0-8 bytes for CAN 2.0, up to 64 bytes for CAN-FD).
- **Termination Resistor** – A 120Ω resistor placed at both ends of a CAN bus to prevent signal reflections.
- **Listen-Only Mode** – A mode where the CAN logger passively listens to messages without transmitting or acknowledging them.
- **Passthrough Mode** – A mode where messages received on one CAN bus are retransmitted to the second CAN bus.
- **CSV (Comma-Separated Values)** – A simple file format for logging CAN messages, readable by spreadsheet software.
- **RTC (Real-Time Clock)** – A hardware clock used to timestamp CAN messages, retaining time even when powered off.
- **JSON (JavaScript Object Notation)** – A text-based configuration file format used for device settings.
- **GVRET (General Vehicle Reverse Engineering Tool)** – A protocol used by software like SavvyCAN for real-time CAN data viewing.
- **SavvyCAN** – A software tool for viewing, analyzing, and sending CAN messages using compatible hardware.
- **IP54 Rating** – Indicates the device is protected against limited dust ingress and splashing water.
- **IP68 Rating** – Indicates the device is fully protected against dust ingress and long-term water submersion.
- **USB Console** – A serial communication interface used for configuring and monitoring the CAN logger.
- **Power Cycling** – The process of turning the device off and on to reset or apply new settings.

Version Changelog

V1.0.0

Initial Release Version

V1.1.0

Add id filtering for incoming CAN Frames