BusEye CL2 CAN Datalogger — User Manual

Figure 1: BusEye CL2

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 a micro-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 micro-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).

Quick Start

- Prepare SD card: format FAT32, create `config.json` at root (see example below), insert card.
- Wire power and CAN: connect VIN/GND and CAN_H/CAN_L. Ensure proper bus termination (120 Ω at each end, 60 Ω total on the bus).
- Set time (recommended): either `NOW.TXT` method or USB console `time set`.
- Power the device: logging starts automatically when SD is detected and configuration is valid.

- Verify: connect USB-C, open a serial terminal, issue `status` command to see current log file; check the SD card for new log files.

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
- 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 networks are 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.
- *Storage Media*: microSD Card (Class 10 or better recommended), up to 32 GB supported
- *USB Interface*: USB-C for console debug and SD Card file access
- *RTC Backup Time*: Up to 6 years

Pin Functions

6-Pin Deutsch/T Style

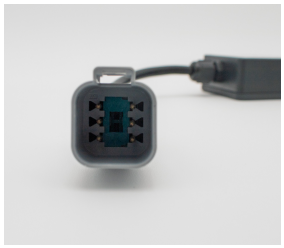

Figure 2: Deutsch/AT 6 Pin Male Plug

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

AMP Example: AT06-6S (DT06-6S)

M12 A-Key Style

Figure 3: M12 Male Plug

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

TE Example: T4110001051-000

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)

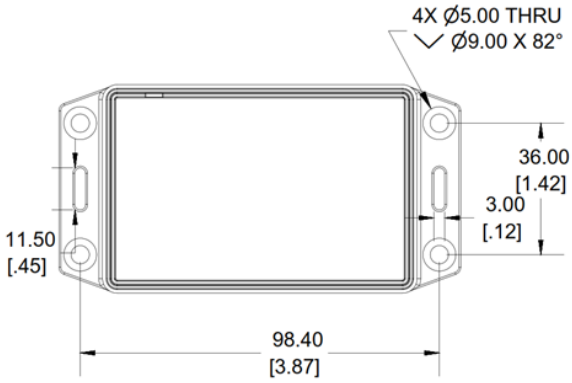

Figure 4: IP54 Flange Mounting Pattern

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%

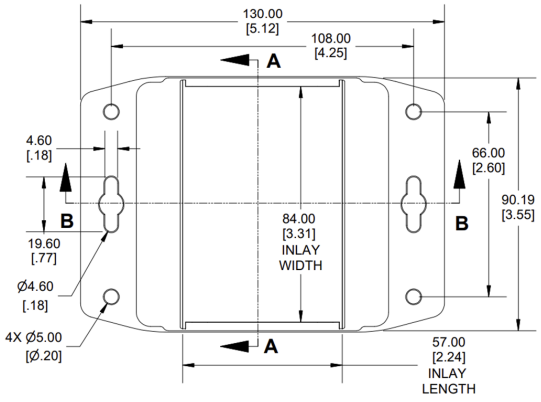

Figure 5: IP68 Flange Mounting Pattern

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 revert to defaults.

- All keys are case sensitive; use lowercase with underscores.
- Validate JSON before use (e.g., with JSONLint). Invalid JSON prevents logging.
- Maximum configuration file size: 16 KB

Top-level parameters:

- `unit_number` (string): Unique unit identifier, <= 20 chars. Default: ""
- `unit_type` (string): Project/model name, <= 20 chars. Default: ""
- `overwrite_logs` (bool): If `true`, oldest logs are deleted to free space when needed. Default: `false`
- `log_type` (string): `CSV` or `DAT`. Default: `CSV` (DAT is compact text; CSV is spreadsheet-friendly)
- `max_file_size` (int): Max bytes before starting a new file. Default: 1,000,000,000 (~1 GB)
- `passthrough` (bool): Bridge CAN1+CAN2 in real time while also logging. Default: `false`
- `timestamp_format_seconds` (bool): If `true`, timestamps are seconds since start; if `false`, ISO 8601. Default: `true`
  - Note: ISO timestamps include milliseconds but are second-ordered; very high traffic can appear re-ordered.
- `print_filter_number` (bool): Include matched filter index in each record. Default: `false` (127 means no filter matched)

Per-bus configuration (keys: `can1`, `can2`):

- `bus_name` (string): Label for the bus, <= 20 chars. Default: `can_<n>`
- `bus_enabled` (bool): Enable hardware for this bus. Default: `true`
- `baudrate` (int): Kbit/s. One of 5,10,50,100,125,250,500,800,1000. Default: 500
- `listen_only` (bool): Silent monitor; no ACK/tx. Default: `false`
- `scan_enable` (bool): Reserved; not implemented. Default: `false`
- `sensitivity` (string): `none|low|medium|high` (error-degrade sensitivity). Default: `medium`
- `allow_rtr` (bool): Allow logging of RTR frames. Default: `true`
- `accept_standard_non_matching_frames` (bool): Log unmatched 11-bit frames. Default: `true`
- `accept_extended_non_matching_frames` (bool): Log unmatched 29-bit frames. Default: `true`
- `standard_filters` (array): 11-bit filter entries.
- `extended_filters` (array): 29-bit filter entries.

Filter entries (processed top-to-bottom; first match wins):

- `type` or `t`: `range|r`, `dual|d`, `classic|c`
  - range: match ID in [ID1..ID2]
  - dual: match either ID1 or ID2
  - classic: mask filter; ID1=filter, ID2=mask
- `ID1`, `ID2`: integer or hex string (`"0x..."`)
- `action` or `a`: `accept|a`, `reject|r`, `accept_priority|ap`

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 Configuration

For more examples, see Appendix A.

```
{
  "unit_type": "vehicle-test",
  "unit_number": "CL2-042",
  "log_type": "CSV",
  "max_file_size": 100000000,
  "can1": {
    "bus_name": "powertrain",
    "baudrate": 500,
    "allow_rtr": true,
    "accept_standard_non_matching_frames": true,
    "accept_extended_non_matching_frames": true
  },
  "can2": {
    "bus_name": "body",
    "baudrate": 250,
    "allow_rtr": true,
    "accept_standard_non_matching_frames": true,
    "accept_extended_non_matching_frames": 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. See section *USB Connection* for more details on connecting to the console.

To set the time run the command `time set YY-MM-DDTHH:MM:SS`. For example, to set the time to June 1st, 2021 at 14:30:00, send the command: `time set 21-06-01T14:30:00`.

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 time 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 format of the following CAN data records.

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:

- The full logging configuration including unit type/number, CAN bus settings, and logging options
- A unique UUID for the log session
- Log start time (ISO 8601 format)
- Log Type (CSV or DAT) and version

  Note: If RTC battery is not present every reboot will default datetime to 2016-01-01 (also occurs on RTC battery failure).

```
{
  "unit_type": "vehicle-test",
  "unit_number": "CL2-042",
  "max_log_file_size": 100000000,
  "log_type": "CSV",
  "overwrite_logs": false,
  "timestamp_format_seconds": true,
  "print_filter_number": false,
  "passthrough": false,
  "can1": { "bus_name": "powertrain", "bus_enabled": true, "baudrate": 500,
            "listen_only": false, "scan_enable": false, "sensitivity": "high",
            "allow_rtr": true, "accept_standard_non_matching_frames": true,
            "accept_extended_non_matching_frames": true,
            "standard_filters": [ {"t":"r","ID1":"0x100","ID2":"0x1FF","a":"a"} ],
            "extended_filters": [] },
  "can2": { "bus_name": "body", "bus_enabled": true, "baudrate": 250,
            "listen_only": false, "scan_enable": false, "sensitivity": "high",
            "allow_rtr": true, "accept_standard_non_matching_frames": true,
            "accept_extended_non_matching_frames": true,
            "standard_filters": [], "extended_filters": [] },
```

```
  "uuid": "...",
  "log_start_time": "2020-01-01T00:04:15.123",
  "log_type": "CSV0.1"
}
```

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]
0.000,1,0,024,8,FF,FF,FF,AA,BB,CC,DD,EE
0.002,2,0,7E8,8,06,41,A4,FF,FF,FF,00,00
```

When `print_filter_number=true`, an extra trailing `filter` column is included.

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
```

When `print_filter_number=true`, a trailing `-<filter>` is appended

> e.g. `13.557-2-021#FFFFFFAABBCCDDEE-5`

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, it's recommended to remove the SD card and transfer files directly.

Termination and Wiring

- Ensure 120 Ω termination at both physical ends of the CAN bus. If CL2 is at an end, use the solder jumper to enable its 120 Ω terminator.
- Use twisted pair for CAN_H/CAN_L; keep stubs short (<30 cm typical).
- Do not enable `listen_only=false` if the logger is one of only two nodes; ensure there are at least two non-listen nodes to provide ACKs.

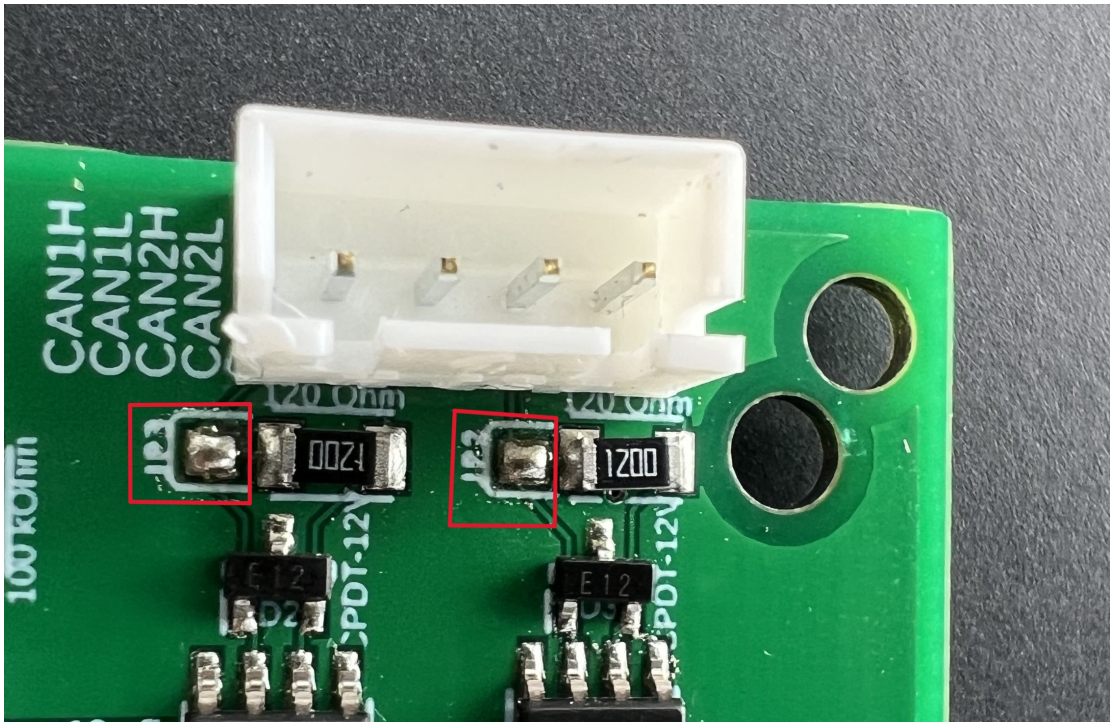

Figure 6: Termination Jumper

Firmware Update

See `BusLink+BusEye Firmware Update` for step-by-step IPE programming instructions (PICkit, device selection, and verification).

Safety & Warnings

- Power input: 6–28 VDC nominal; observe polarity. Device includes reverse polarity protection but improper wiring can still cause damage.
- ESD: Handle with care when covers are open; avoid static discharge to connectors.
- Vehicle systems: Use caution when connecting to live vehicle networks; avoid interfering with safety-critical systems.

Troubleshooting

- Device logs but timestamps read 2016-01-01: Install/replace RTC coin cell and set time (NOW.TXT or console `time set`).
- No new log files: Verify SD is FAT32, present and writable; check `status` for filesystem state; confirm `config.json` is valid JSON.
- No CAN data: Check wiring and bus termination; confirm `listen_only` and `baudrate` are set properly;

ensure traffic exists on the bus.
- USB drive slow/unreliable: Prefer removing SD for large transfers.

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 or Tera Term to access the CLI and set the time.

> Note: changes to **can 1**, **can 2** or **passthrough** settings will be save to config.json immediately after the change. A note will be printed in the console when this occurs.

Command **help**, **h** or **?**: Show help message

Command **version** or **v**: Print the software version

Command **status** or **stat**: Show advanced CAN Bus status

Command **config** or **cfg**: Show or set CAN Bus configuration as JSON string (See Configuration command section)

Command **passthrough** or **pt**: Show or toggle CAN passthrough mode. E.g. `passthrough` (show), `passthrough on` (set), `passthrough off` (unset)

Command **led** or **l**: Show or set LED status. E.g. `led` (show), `led r on`, `led g off`, `led b tog`

Command **can** or **c**: Show or set CAN Bus configuration. (See CAN command section)

Command **rotatelog** or **rl**: Read config and rotate to a new log file (logging continues)

Command **reboot** or **rb**: Reboot the device

Command **xE7**: Send this special character to toggle CAN Serial setup (GVRET mode)

Configuration Command

The configuration command allows you to view or set the CAN Bus configuration using a JSON string. The configuration is stored on the local memory after each change and is loaded on boot.

config or **config show**: Print the current configuration JSON String.

config set: Set the configuration using a JSON string. After typing `config set` and pressing enter, paste the JSON string. Once the device receives the final closing bracket, the device will validate the JSON and apply the new configuration. (CAN Configuration can be sent "minified" to reduce size, or pretty-printed for readability, printouts are always minified).

> Note: we recommend modifying the configuration using the `can` command first, then copying the current configuration using `config show` before uploading it to the device. This ensures a valid configuration. Direct editing of the JSON file is possible, but the JSON must be valid.

config reset: Reset the configuration to factory defaults.

CAN Command

The CAN command allows you to view or set individual CAN Bus settings.

> Note: shortcuts for CAN command arguments are available and can stack. For example, `can 1 baud 500` can be shortened to `c 1 b 500`.

can help or **can h** or **can ?** - Show help message for CAN commands.

can settings or **can s** or **c settings** or **c s** - Show a summary of the current CAN Bus settings.

can <1|2> baud or **can <1|2> b** - Show or set the baud rate for CAN Bus 1 or 2. Supported baud rates are: 5, 10, 50, 100, 125, 250, 500, 800, 1000 (in Kbit/s). Example: **can 1 baud 500** or **c 2 b 125**

can <1|2> scan or **can <1|2> s** - Show or set the baud rate scanning for CAN Bus 1 or 2. Supported options are:

- `on` - Enable baud rate scanning
- `off` - Disable baud rate scanning

NOTE: Baud rate scanning is not yet implemented in this software version.

can <1|2> mode or **can <1|2> m** - Show or set mode: `normal|listen`

can <1|2> accept or **can <1|2> a** - Show or set acceptance of unmatched frames: `std|ext|both`

can <1|2> rtr or **can <1|2> r** - Show or set RTR handling: `on|off`

can <1|2> filter or **can <1|2> f** - List, add, remove, or clear filters (standard/extended); see on-device `can help` for usage

can <1|2> print or **can <1|2> p** - Show or set printing frames to console: `on|off`

can <1|2> sensitivity or `can <1|2> sens` - Show or set the bus degrade sensitivity for CAN Bus 1 or 2. Supported options are:

- `none` - No automatic degrade
- `low` - Low sensitivity, degrade into "CAN_SUSPECT" state only (disable retransmits on transmit errors)
- `medium` - Medium sensitivity, degrade into "CAN_DEGRADED" state (listen-only mode)
- `high` - High sensitivity (default), degrade into "CAN_OFFLINE" state (disable all tx and rx, put can into sleep mode until attempting recovery)

Sensitivity controls how aggressively the device reacts to CAN errors and how far it will automatically degrade bus activity to protect the network. Higher sensitivity detects smaller bursts of protocol errors or rising error counters sooner and allows escalation to stricter behaviors: High may fully take the bus OFFLINE (no TX/RX) until attempting recovery; Medium will at most enter listen-only (DEGRADED) to observe traffic without transmitting; Low only limits retransmissions briefly (SUSPECT); None disables automatic degradation and keeps normal operation regardless of errors.

Choose a higher sensitivity for noisy or safety-critical networks where it's preferable to back off quickly rather than risk adding load to a faulted bus. Use lower sensitivity on stable networks or in scenarios where maintaining transmission through transient disturbances is more important than aggressive protection, for example when a tool or node is actively scanning baud rates. Sensitivity is set per bus and can be adjusted live via the CLI or saved in the configuration JSON.

can <1|2> mode or **can <1|2> m** - Show or set the mode for CAN Bus 1 or 2. Supported modes are: - `normal` - Normal mode (default) - `listen` - Listen-only mode

can <1|2> accept or **can <1|2> a** - Show or set the acceptance filter for CAN Bus 1 or 2. Supported options are: - `std` - Accept only standard (11-bit) CAN IDs - `ext` - Accept only extended (29-bit) CAN IDs - `both` - Accept both standard and extended CAN IDs (default) - `none` - Accept neither standard nor extended CAN IDs (only log frames specifically accepted in filters)

NOTE: Accept refers to "non-matching" frames based on filters. If filters are configured they will take precidence over this setting. All frames that do not match a filter will be accepted or rejected based on this setting. See filter

diagram below for flow chart on how filtering works.

can <1|2> filter or **can <1|2> f** - Show or set the CAN ID filtering for CAN Bus 1 or 2.

can <1|2> filter list - Print all filters for the selected CAN Bus.

can <1|2> filter add - Add a new filter to the selected CAN Bus.

- `<frame type>` - `std` for standard (11-bit) CAN IDs, `ext` for extended (29-bit) CAN IDs
- `<filter style>` - `range` to match all frames between ID1 and ID2, `dual` to match ID1 or ID2, `classic` for a mask-based filter
- `<id1>` - The first ID in the filter (for `range` ID1 must be less than ID2, for `dual` ID1 and ID2 can be in any order, for `classic` this is the filter ID)
- `<id2>` - The second ID in the filter (for `classic` this is the mask)
- `<action>` - `accept` to accept matching frames, `reject` to reject matching, `accept-priority` to accept and set highest priority (0) for matching frames

Example: `can 1 filter add std range 0x123 0x456 reject`

Short form example: `c 2 f add s r 0x123 0x456 r`

can <1|2> filter delete - Remove a filter from the selected CAN Bus by its index number. The index number can be found using `can <1|2> filter list`.

can <1|2> filter clear - Remove all filters of the specified frame type from the selected CAN Bus. `<frame type>` can be `std`, `ext` or `all`.

Filtering Diagram

The diagram below illustrates how CAN ID filtering works. Filters are evaluated in the order they are listed, and the first matching filter determines the action taken. If no filters match, the default acceptance setting is applied.

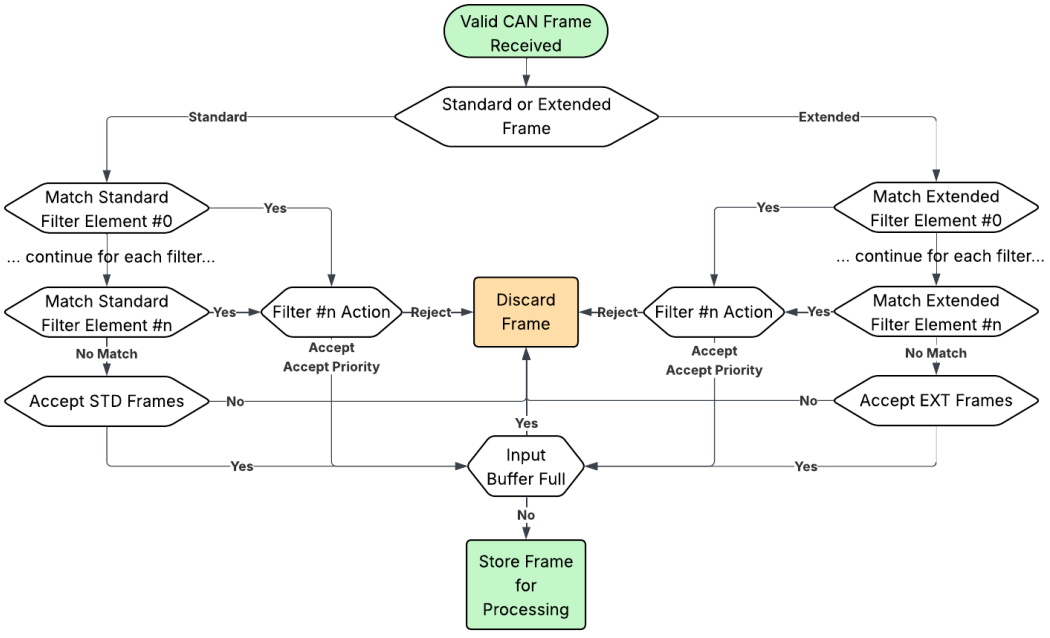

Figure 7: Filtering Diagram

Can GVRET Functionality

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 and viewing them in realtime on a PC (see Passthrough Mode section).

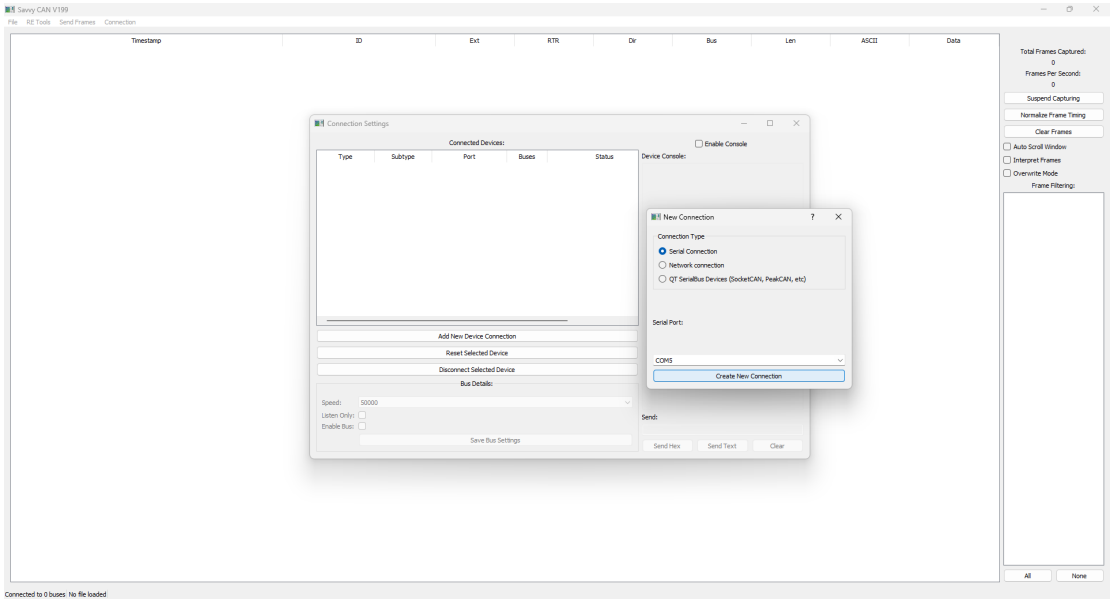

Figure 8: SavvyCan Connection

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 `can 1 print on` and `can 2 print on` to toggle printing of raw CAN messages to the console.

Passthrough Mode

Passthrough mode can be used to retransmit CAN messages received on one bus to the other bus. This is useful for diagnosing issues on a CAN Bus by inserting the logger as a repeater node between two existing nodes.

When passthrough mode is enabled, every message received on CAN1 is retransmitted on CAN2, and vice versa. This allows the logger to act as a transparent bridge between two CAN nodes while logging all traffic. Furthermore, baud rates for each bus can be configured independently, allowing the logger to connect nodes operating at different speeds.

To enable passthrough mode set the 'passthrough' option to true in the configuration JSON or send the command `passthrough on` via the USB console. To disable passthrough mode, send the command `passthrough off`.

(Note: when setting passthrough mode via console, the configuration file is NOT updated as this is considered a debugging function. For permanent changes, edit the configuration JSON file).

Performance and System Behavior

Bus Errors

If the logger detects protocol errors, it will put incrementally move the the CAN hardware into warning and degraded states to prevent bus arbitration errors for other nodes. The maximum degraded state is determined by the `sensitivity` setting for each bus. E.G. if sensitivity is set to `medium`, the bus will degrade into *CAN_DEGRADED* state at worst, and will not go offline.

CAN_OK: No errors detected.

CAN_SUSPECT: 3 errors within 50ms, transmit error count $>$ 96 or receive error count $>$ 50. In this mode the CAN hardware will transmit new frames once and will not retry if transmission fails.

CAN_DEGRADED: 10 errors within 50ms, transmit error count $>$ 128 or receive error count $>$ 100. In this mode the CAN hardware will be in listen-only mode and will not transmit any frames or respond with any ack bits.

CAN_OFFLINE: CAN Hardware went to "BUS OFF" state, Transmit error count $>$ 254, or receive error count $>$ 126. In this mode the CAN hardware will be in sleep mode and will not transmit or receive any frames.

The device will attempt to recover from degraded states automatically as follows.

From *CAN_OFFLINE*: Wait between 0.25 seconds incrementing to 8 seconds, then attempt to reinitialize the CAN hardware in *CAN_DEGRADED* mode. If the bus remains offline, it will retry every 8 seconds until the bus recovers.

From *CAN_DEGRADED*: Wait 1.5 seconds, then attempt to reinitialize the CAN hardware in *CAN_SUSPECT* mode. If the bus remains degraded, it will retry every 1.5 seconds until the bus recovers.

From *CAN_SUSPECT*: Wait 0.5 seconds, then attempt to reinitialize the CAN hardware in *CAN_OK* mode. If the bus remains suspect, it will retry every 0.5 second until the bus recovers.

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, message order in the log file may not follow strict chronological order, but timestamps will be preserved.

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.

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	Case Type + IP Rating	Wiring Option	Temperature Range	Connector
10-24001	Rectangular IP54	15 cm pigtail	-20 to 80 C	Deutsch 6-Pin AT
10-24010	Open Case IPXX	15 cm pigtail	-20 to 80 C	Deutsch 6-Pin AT
10-24011	Square IP68	15 cm pigtail	-40 to 120 C	Deutsch 6-Pin AT

Visit our website at perspic.ca for the latest part numbers and configurations.

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 full console funcitonality over USB-C Add can sensitivity setting per bus Add CAN Frame ID filtering for incoming CAN Frames

Power Supply Requirements

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

Appendix A: Example Configuration Files

Example 1: Basic Logging on Both Busses

- Both busses enabled
- Logging all frames at 500 Kbit/s
- 1 GB max file size
- DAT format
- Overwriting old logs when full.

```
{
    "unit_type": "test_type",
    "unit_number": "test_unit",
    "overwrite_logs": true,
    "log_type": "DAT",
    "max_file_size": 1000000000,
    "passthrough": false,
    "timestamp_format_seconds": true,
    "can1": {
        "bus_name": "can_1",
        "bus_enabled": true,
        "baudrate": 500,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "medium",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "standard_filters": [],
        "extended_filters": []
    },
    "can2": {
        "bus_name": "can_2",
        "bus_enabled": true,
        "baudrate": 500,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "medium",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "standard_filters": [],
        "extended_filters": []
    }
}
```

Example 2: Custom Baud Rates and Logging All Frames

- Both busses enabled
- CAN1 at 500 Kbit/s, CAN2 at 250 Kbit/s
- Logging all frames
- High sensitivity degradation on both busses
- Passthrough mode enabled

```
{
    "unit_type": "Car",
    "unit_number": "VIN123",
    "overwrite_logs": false,
    "log_type": "CSV",
    "max_file_size": 1000000000,
    "passthrough": true,
    "timestamp_format_seconds": true,
    "print_filter_number": true,
    "can1": {
        "bus_name": "can_1",
        "bus_enabled": true,
        "baudrate": 500,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "high",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "standard_filters": [],
        "extended_filters": []
    },
    "can2": {
        "bus_name": "can_2",
        "bus_enabled": true,
        "baudrate": 250,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "high",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": true,
        "accept_extended_non_matching_frames": true,
        "standard_filters": [],
        "extended_filters": []
    }
}
```

Example 3: Advanced Filtering Configuration

- Both busses enabled
- CAN1 at 500 Kbit/s, CAN2 at 250 Kbit/s, passthrough disabled
- No sensitivity degradation on either bus
- Custom filters on both busses. Only frames matching accept filters will be logged.

  CAN1: Standard IDs: accept range 123 to 456;

  CAN1: Extended IDs: accept dual 0xFFFDDD or 0xDDDFFF

  CAN2: Standard IDs: accept with priority range 0x123 to 0x456

  CAN2: Extended IDs: accept dual 0xDEAD or 0xBEEF


```
{
    "unit_type": "",
    "unit_number": "",
    "overwrite_logs": false,
    "log_type": "CSV",
    "max_file_size": 1000000000,
    "passthrough": false,
    "timestamp_format_seconds": false,
    "print_filter_number": true,
    "can1": {
        "bus_name": "can_1",
        "bus_enabled": true,
        "baudrate": 500,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "none",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": false,
        "accept_extended_non_matching_frames": false,
        "standard_filters": [
            {
                "type": "r",
                "id1": "0x7B",
                "id2": "0x1C8",
                "action": "a"
            }
        ],
        "extended_filters": [
            {
                "type": "d",
                "id1": "0xFFFDDD",
                "id2": "0xDDDFFF",
                "action": "a"
            }
```

```
        ]
    },
    "can2": {
        "bus_name": "can_2",
        "bus_enabled": true,
        "baudrate": 250,
        "listen_only": false,
        "scan_enable": false,
        "sensitivity": "none",
        "allow_rtr": true,
        "accept_standard_non_matching_frames": false,
        "accept_extended_non_matching_frames": false,
        "standard_filters": [
            {
                "type": "d",
                "id1": "0x123",
                "id2": "0x456",
                "action": "ap"
            }
        ],
        "extended_filters": [
            {
                "type": "d",
                "id1": "0xDEAD",
                "id2": "0xBEEF",
                "action": "a"
            }
        ]
    }
}
```