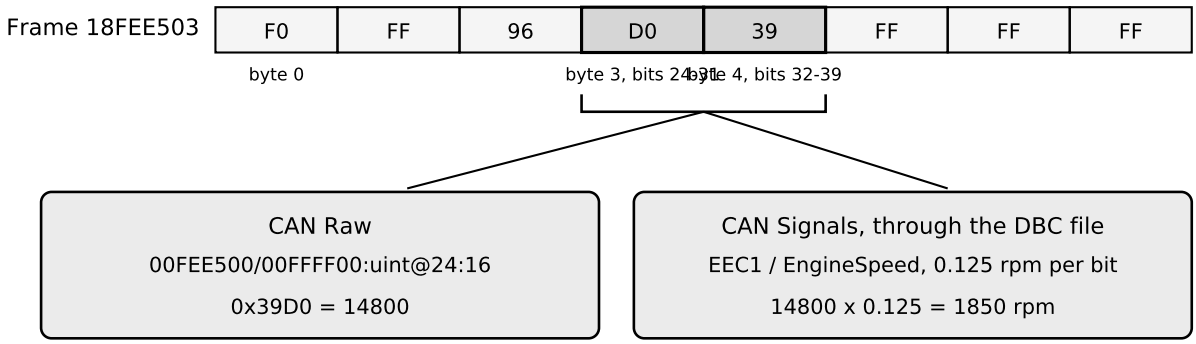
Amtiri

REFERENCE MANUAL

CAN

Generated on November 25, 2026



CAN is the bus inside vehicles, mobile machinery and much industrial equipment: a frame carries an identifier and up to eight bytes, and nothing about who sent it or for whom. Two modules read and write it, and differ in how a variable names its value.

Module	What it does	Typical use
`CAN-Raw` (CAN Raw Frames)	Takes slices of frames, named by identifier and bits, into variables; sends variables as frames	A controller nobody has a DBC file for, a frame found by watching the bus
`CAN-Signals` (CAN Signals)	Takes named signals into variables in engineering units, through the machine's DBC file; sends variables as messages	A machine whose manufacturer supplies the DBC file, J1939 engines

Enable, **Name** and the **Status and messages** tab work the same way in every module. They are described in Settings every module shares.

Reaching a bus

Both modules reach a bus the same way, and several modules on the same adapter share it: the port is opened once, and each module receives the frames its rows name. Two modules may not ask for one adapter at two bit rates.

Transport	Reaches the bus through	Systems
`slcan`	An SLCAN USB adapter - CANable, CANtact, USBtin - on a serial port. The module sets its bit rate	Every system the engine runs on
`socketcan`	A Linux SocketCAN interface: an appliance's own controller, or a USB adapter with a kernel driver. The bit rate is the system's, set with `ip link set can0 type can bitrate 250000`	Linux only
`loopback`	A bus inside the engine: frames sent come back to every module on it. For tests and demonstrations	Every system

A missing adapter is not a failure to open: the module opens, says the bus is unavailable, tries again every 2 seconds, and picks the adapter up when it appears. Through `socketcan` the controller's own state is reported too: error passive as a warning, bus-off as an error. Through `slcan` it is reported only by adapters that answer the status request, which several firmwares do not.

> **Note**
>
> Modules sharing one adapter do not hear each other's frames: the adapter does not echo what it sends. A value one module sends and another uses is better shared through its variable.

What is sent, and when

A forwarder is one value inside a frame, and several forwarders usually fill one frame. **Rows of one frame make one frame**: every forwarder of the same identifier, or of the same DBC message, fills it, and it is sent whenever one of their variables changes, built from the current value of each. A row with an interval also sends its frame on that clock, so a frame a controller expects every 100 ms keeps coming whether or not anything changed; the frame goes on the shortest interval any of its rows asks for. A variable that holds nothing yet leaves its bits at their fill value.

A plain CAN frame carries no sender, so sending needs no address of its own. A frame that cannot go while the bus is down is dropped: the bus's state already says why.

Values

A value lands only when it differs from the last one written, and no sooner than the row's `minIntervalMs` after it; the last value is never lost, it waits for the interval to end. Values are written through the engine's event queue, so a bus delivering hundreds of frames a second costs the engine one write per interval, not one per frame.

CAN Raw Frames

The `CAN-Raw` module names each value by the frame it is in and the bits it occupies.

Addresses

An address is `id[/mask]:type@bitOffset[:bitLength][:be]`.

- **The identifier** is hexadecimal. How many digits it is written with says which kind it is: up to three is an 11-bit identifier, more is a 29-bit one - which is how SLCAN and candump write them. `123` is 11-bit; `00000123` is 29-bit.
- **The mask** says which bits of a received identifier must match, as a controller's filter does. Left out, every bit must. `00FEE500/00FFFF00` takes that J1939 parameter group from any source at any priority. A forwarder sends one identifier, so its address takes no mask.
- **The type** is `bool`, `uint`, `int` (two's complement), `float` or `double` (IEEE 754). The length may be left out where the type fixes it: one bit, 32 or 64.
- **The bits** are numbered as a DBC file numbers them: bit `n` is bit `n % 8` of byte `n / 8`, bit 0 the least significant. A value is Intel, little-endian, from its least significant bit upward - as J1939 and NMEA 2000 send them. With `:be` it is Motorola, big-endian, and the offset is its most significant bit, as a DBC file's `@0` means.

Address	Reads
`18FEE500/1FFFFFFF:uint@24:16`	Bits 24 to 39 of that frame, as an unsigned number
`00F00400/00FFFF00:uint@24:16`	The same bits of J1939 group 61444, whoever sends it
`123:bool@7`	The top bit of the first byte of frame `123`
`321:float@32`	The last four bytes of frame `321`, as a 32-bit float
`321:int@7:12:be`	Twelve bits from the top of byte 0 into byte 1, Motorola, signed

A `bool` holds true or false, `uint` and `int` a whole number, `float` and `double` a real number. Scaling a raw value into a unit is a formula's job here; CAN Signals does it from the DBC file.

Settings

Key	Setting	Type	Default	Meaning
`transport`	Transport	text	`slcan`	`slcan`, `socketcan` or `loopback`, as Reaching a bus says
`serialPort`	Serial port	text	(empty)	`slcan`: the adapter's device, such as `/dev/ttyACM0` or `COM5`. The editor offers the ports it finds on the server
`bauds`	Bauds	int	`115200`	`slcan`: the adapter's serial speed. A USB adapter ignores it
`interface`	Interface	text	`can0`	`socketcan`: the Linux interface. `loopback`: the loopback's name
`bitrate`	Bit rate [bit/s]	int	`250000`	`slcan`: the bus speed the adapter is set to, 10000 to 1000000. J1939 runs at 250000, many machines at 500000
`collectors`	Collector	list of rows	empty	The values taken into variables
`forewarders`	Forwarder	list of rows	empty	The values sent from variables. The stored key is spelled `forewarders`

Collectors and forwarders

A collector row uses `field`, `address` and `minIntervalMs`. A forwarder row uses `field`, `address` and `intervalMs`.

Key	Column	Type	Default	Meaning
`field`	Variable	variable code	(none)	Collector: the variable written. Forwarder: the variable sent
`address`	Address	text	(empty)	The slice, as Addresses says
`minIntervalMs`	Minimum interval [ms]	int	(empty)	Collectors: the least time between two writes of the variable. Empty or 0: every change
`intervalMs`	Interval [ms]	int	(empty)	Forwarders: also send the frame on this clock. Empty or 0: on change only

The **Type / problem** column is not stored: it shows the variable type a slice yields, or what is wrong with its address, in the engine's words. A forwarder's frame is as long as its longest slice needs; bits no forwarder names are sent as zero.

Messages

All texts are in English; the full list is in Module messages. "then `ERROR`" follows `errorAfterSeconds` in Settings every module shares. `{bus}` and `{server}` name the transport, as `SLCAN on /dev/ttyACM0`, `SocketCAN can0` or `loopback test`; `{message}` is the frame's identifier.

When	Level	Text
Opening: a setting that cannot work	`ERROR`	`Configuration invalid: {reason}`: `'{transport}' is not slcan, socketcan or loopback`, `No serial port`, `No interface`, `An SLCAN adapter cannot run at {bitrate} bit/s`, `SocketCAN is available on Linux only, not on {os}`, or `{bus} is already open at {bitrate} bit/s`
Opening: an address that does not parse	`ERROR`	`{mapping}: {reason}`, such as `{mapping}: float is 32 bits, not 16`
Opening: a forwarder with a mask	`ERROR`	`{mapping}: a forewarder sends one identifier, so its address cannot have a mask`
The bus is up	`INFO`	`Connected to {server}`
The adapter could not be opened, or was lost	`WARNING`, then `ERROR`	`Disconnected from {server}`, then `Bus {bus} unavailable: {reason}`
The controller is error passive	`WARNING`	`{bus} is error passive`
The controller is bus-off	`ERROR`	`{bus} is bus-off`
The adapter refused a frame	`WARNING`	`Transmitting {message} failed: {reason}`
A value its variable's declared type refuses, or the event queue is full	`WARNING`	`Output {output} refused {value}: {reason}` or `Event queue full: {count} events pending, {dropped} dropped`

> **Tip**
>
> On Linux, the account the service runs as needs access to the adapter's serial device, usually through the group that owns it (`dialout` on Debian and Ubuntu).

Example

A J1939 engine read without its DBC file, and a fan command a controller expects every 100 ms:

```
Transport      slcan
Serial port    /dev/ttyACM0
Bit rate       250000

Collector    SPEED_RAW     00F00400/00FFFF00:uint@24:16      500
Collector    COOLANT_RAW   00FEEE00/00FFFF00:uint@0:8        2000
Forwarder    FAN_CMD       18FF2100:uint@0:8                 100
```

`SPEED_RAW` holds engine speed in units of 0.125 rpm and `COOLANT_RAW` the coolant temperature plus 40; a formula turns either into its unit. `FAN_CMD` goes out every 100 ms and whenever it changes.

CAN Signals

The `CAN-Signals` module reads a bus through the machine's **DBC file**, the database in which a manufacturer describes its messages and the signals inside each: where a signal is, its scale and offset, its unit, its range and its value table. An operator loads the file, picks the signals wanted, and gets engineering units in variables.

The DBC file

Load DBC file in the editor takes the file, up to 4 MB, and keeps its text **in the module's record**, so it travels with the configuration: a transfer, a copy to another engine or a restore from backup brings the database with it. A file in Windows-1252, as Vector's tools write it, is read as such. The editor then offers the file's messages, and each message's signals with their units, in the tables.

What is read from a DBC file:

Read	Passed over
Messages (`BO_`): identifier, name, length	Comments (`CM_`)
Signals (`SG_`): start bit, length, byte order, sign, scale, offset, minimum, maximum, unit	Attribute definitions and values, but the one below
Value tables (`VAL_`)	Environment variables, node lists, signal groups
Multiplexing: the multiplexor (`M`) and the signals it carries (`m1` , `m2`)	Extended multiplexing (`SG_MUL_VAL_`)
IEEE float and double signals (`SIG_VALTYPE_`)	
`BA_ "ProtocolType" "J1939"`	

A statement the parser cannot read stops the module opening, naming its line: a database that silently lost a signal would leave a variable that never changes.

J1939. A file that declares `ProtocolType` J1939 is matched by **parameter group number**: a message is read whichever source address and priority it arrives with. Its messages longer than eight bytes are put back together from the J1939 transport protocol, broadcast or connection-mode, and its frames are sent with unused bits set to one, which J1939 reads as "not available". Any other file is matched on the exact identifier, and sent with unused bits at zero.

Values

- A signal's value is its raw bits times the scale, plus the offset. A signal whose scale and offset are whole numbers holds a whole number; any other a real number. A float or double signal is read as IEEE 754 first.
- A signal with a **value table** holds its description as text - `On` , `Not available` - and a raw value the table does not list as its number, as text. Written, it takes either the description or the number.

- A **multiplexed** signal is read only from frames whose multiplexor carries its value, and written into a frame with the multiplexor set to it. The signals that are not multiplexed go in every frame of their message.
- A value outside the minimum and maximum the file sets is refused when written, and reported; when the file sets none, anything the bits can hold goes.

The editor's **Type / problem** column shows the type a signal yields: `string` for a value table, `int` or `float` otherwise.

Settings

Key	Setting	Type	Default	Meaning
`transport`	Transport	text	`slcan`	`slcan`, `socketcan` or `loopback`, as Reaching a bus says
`serialPort`	Serial port	text	(empty)	`slcan`: the adapter's device
`bauds`	Bauds	int	`115200`	`slcan`: the adapter's serial speed
`interface`	Interface	text	`can0`	`socketcan`: the Linux interface. `loopback`: the loopback's name
`bitrate`	Bit rate [bit/s]	int	`250000`	`slcan`: the bus speed the adapter is set to
`dbc`	DBC file	text	(empty)	The DBC file's text. Required: without it the module does not open
`dbcName`	DBC file	text	(empty)	The loaded file's name, shown in the editor
`collectors`	Collector	list of rows	empty	The signals taken into variables
`forewarders`	Forwarder	list of rows	empty	The signals sent from variables. The stored key is spelled `forewarders`

Collectors and forwarders

A collector row uses `field`, `message`, `signal` and `minIntervalMs`. A forwarder row uses `field`, `message`, `signal` and `intervalMs`.

Key	Column	Type	Default	Meaning
`field`	Variable	variable code	(none)	Collector: the variable written. Forwarder: the variable sent
`message`	Message	text	(empty)	The message's name in the DBC file, such as `EEC1`
`signal`	Signal	text	(empty)	The signal's name in that message, such as `EngineSpeed`
`minIntervalMs`	Minimum interval [ms]	int	(empty)	Collectors: the least time between two writes of the variable
`intervalMs`	Interval [ms]	int	(empty)	Forwarders: also send the message on this clock. Empty or 0: on change only

Only messages of up to eight bytes can be sent, and a multiplexor is not sent from a variable: it is set by the signals it carries.

Messages

All texts are in English; the full list is in Module messages. `{message}` is the DBC message, with its multiplexor value when it has one.

When	Level	Text
Opening: a setting that cannot work	`ERROR`	`Configuration invalid: {reason}`, as for CAN Raw
Opening: no DBC file	`ERROR`	`No DBC file`
Opening: the DBC file cannot be read	`ERROR`	`DBC file: line {n}: {reason}`
Opening: a message the file does not have	`ERROR`	`{mapping}: '{message}' is not a message of the DBC file`
Opening: a signal the message does not have	`ERROR`	`{mapping}: {message} has no signal '{signal}'`
Opening: a forwarder in a message longer than eight bytes	`ERROR`	`{mapping}: {message} is {n} bytes, and only a single-frame message is sent`
Opening: a forwarder for the multiplexor	`ERROR`	`{mapping}: {signal} is the multiplexor, set by the signals it carries`
The bus is up	`INFO`	`Connected to {server}`
The adapter could not be opened, or was lost	`WARNING`, then `ERROR`	`Disconnected from {server}`, then `Bus {bus} unavailable: {reason}`
The controller is error passive, or bus-off	`WARNING` or `ERROR`	`{bus} is error passive` or `{bus} is bus-off`
A value the file's range or value table refuses, or a frame the adapter refused	`WARNING`	`Transmitting {message} failed: {reason}`
A value its variable's declared type refuses, or the event queue is full	`WARNING`	`Output {output} refused {value}: {reason}` or `Event queue full: {count} events pending, {dropped} dropped`

Example

An excavator's J1939 bus with its manufacturer's DBC file:

```
Transport      slcan
Serial port    /dev/ttyACM0
Bit rate       250000
DBC file       excavator.dbc

Collector    ENGINE_SPEED   EEC1         EngineSpeed     500
Collector    COOLANT_T      ET1          CoolantTemp     2000
Collector    FAN_STATE      PROP_FAN     FanState
Forwarder    FAN_CMD        FanControl   FanSetpoint     100
```

`ENGINE_SPEED` holds rpm and `COOLANT_T` degrees, from whichever controller sends them. `FAN_STATE` holds `On`, `Off` or `Error`, as the file's value table names them. `FAN_CMD` goes out every 100 ms in the FanControl message, scaled as the file says.

Next steps

- NMEA
- Module catalogue

- Settings every module shares
- Module messages
- Connectivity

This page describes Data Orchester engine 6.20.0.