CURRENT ELECTRIC VEHICLES

CEV CONFIGURATOR MANUAL

The complete guide to the CEV Configurator — how to drive the application, and what every vehicle setting does. Part 1 tours the app's screens and workflows; Part 2 walks each configuration form, setting by setting; Part 3 follows worked examples of complete vehicles through the journeys they make. Written for installers and shops.

Generated 2026-09-22 · apps/configurator/docs/user-manual.md

Contents

Using the Configurator 5

- Projects 5
- The Setup Wizard 6
- DBC Files 7
- Incoming CAN Signals 9
  - The CAN signal editor 11
- Devices 12
  - Analog input 14
  - Digital input 15
  - Frequency input 16
  - Digital output 17
  - PWM output 18
  - Analog sensor 19
  - Frequency sensor 20
  - CAN sensor 21
  - Script sensor 22
  - Auxiliary throttle 23
  - Coolant pump 24
  - Fan 25
  - Warning light 26
- Calibration Curves & Tables 27
  - The 2D curve editor 29
  - The 3D surface editor 30
- The Script Editor 32
- The Issues List 32
- Connecting to a Vehicle 34
- Flashing the VCU 36
- Updating Firmware 37
- The Live Dashboard 39
- Torque Pipeline 41
- Custom monitoring views 43
- Per-Cell Battery Monitoring 44
- Recording & Replaying Drives 45
- Comparing Configuration with the Vehicle 47
- Faults 48

Auto calibrate pedal 49
Configuring the Vehicle 50
Supported hardware 50
Traction inverters 50
Battery management systems 51
DC-DC converters 51
On-board chargers 52
Charge inlet controllers 52
Power distribution units 52
Motion sensors 52
Sensor and control inputs 53
Vehicle 53
Identity 53
Drivetrain layout 54
Mass & geometry 56
CAN Buses 57
Bus bitrates 57
Outbound messages 59
Script CAN gates 59
HV Power 61
Contactors 61
Precharge & sequence 65
Interlock loop 71
BMS 74
Drivetrain 76
Inverters 76
Gearbox 83
Gear detection / limiter 87
Charging & Power 93
DC-DC Converters 93
On-Board Chargers 97
Fast Charging 103
Gensets 107
Power Distribution 115
Pedal & Brake 118
Accelerator pedal 118
Brake 123

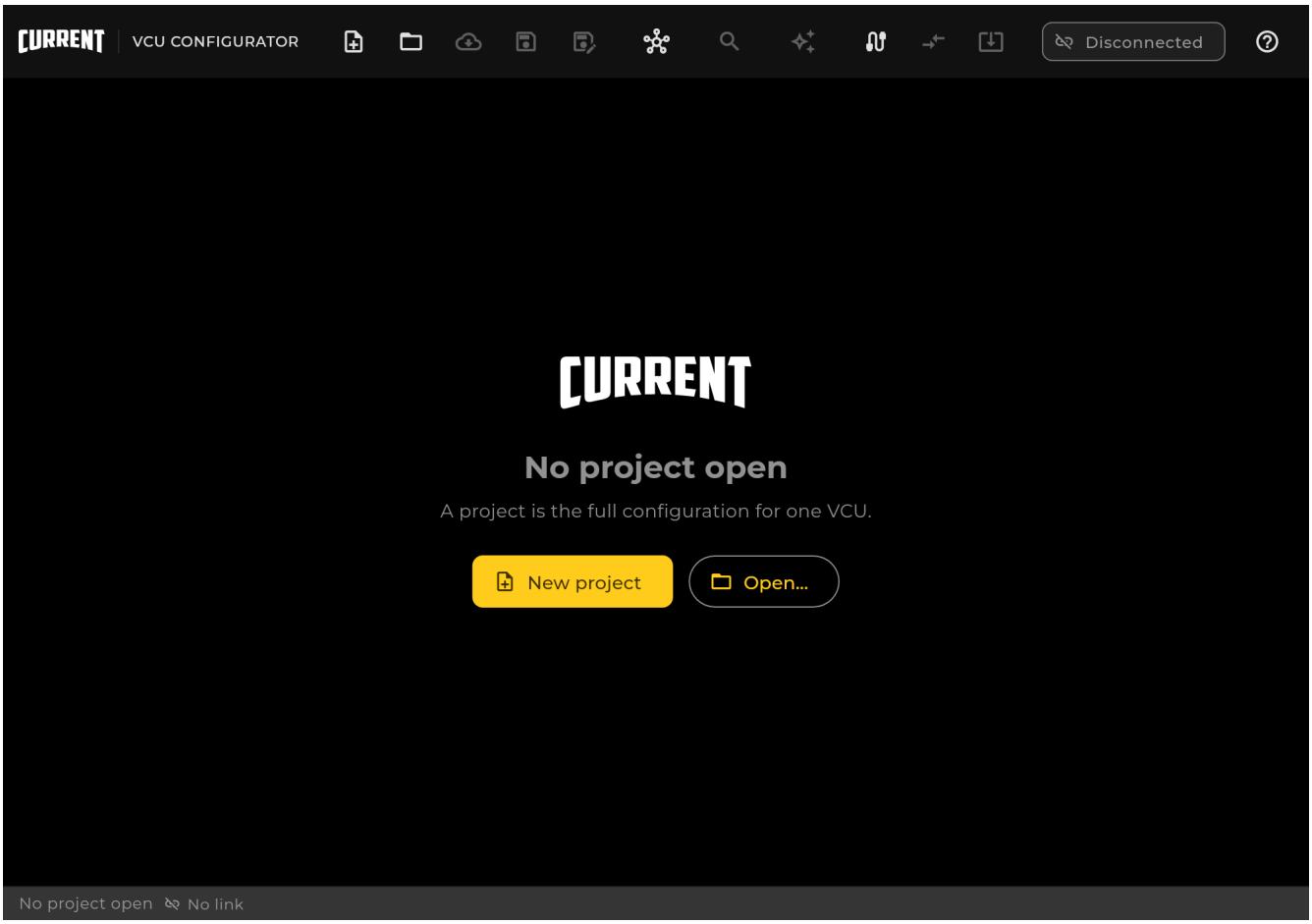
Torque Arbiter 128
Drive Profiles 132
Direction (P/R/N/D) 136
Cruise 142
Hill hold / descent 146
Creep 150
Traction control 153
Torque vectoring 157
Distribution 160
Brake regen 163
Brake light 165
Safety monitor 169
Dynamics 173
IMU 173
Wheel speed 175
Steering 179
Suspension 184
Producers 189
Tools 194
Recipes 194
Diagnostics 194
Worked Examples **194**
How the vehicle changes state 194
Being allowed in a state is not the same as being started 195
Driving 195
Charging, and going wrong 196
Key-on to rolling 197
Key-off and shutting down 199
Separate accessory and stop controls 199
Ignition shutdown and recipe requests have different limits 199
Accessory power without high voltage 200
Coasting in neutral 201
Charging from a wall socket 202
Fast charging and safe teardown 204
When something goes wrong 205

Using the Configurator

How to drive the Configurator application itself — creating a project, connecting to a vehicle, flashing a configuration, the live dashboard, and the tools you use to author a build. This first part is a tour of the application's screens and workflows; the second part, Configuration Screens, walks the per-subsystem forms. For what each individual vehicle setting does, see the Vehicle Build Manual.

Projects

A project is the complete configuration for one vehicle, saved as a single file. Create, open, and save projects, and keep several open at once in tabs.



What a project is. A project holds everything that makes up one VCU's build — every setting, the CAN databases it draws on, and any drive recordings you capture. It lives in one file you can save, copy, and hand off. When nothing is open the app shows a welcome screen with **New project** and **Open…** to get you started.

Creating and opening. **New** starts a fresh project, already seeded with the shop's default CAN databases, and asks whether to set the vehicle up with the wizard or in the full editor. **Open** loads an existing project file. Both are on the toolbar and on the usual keyboard shortcuts.

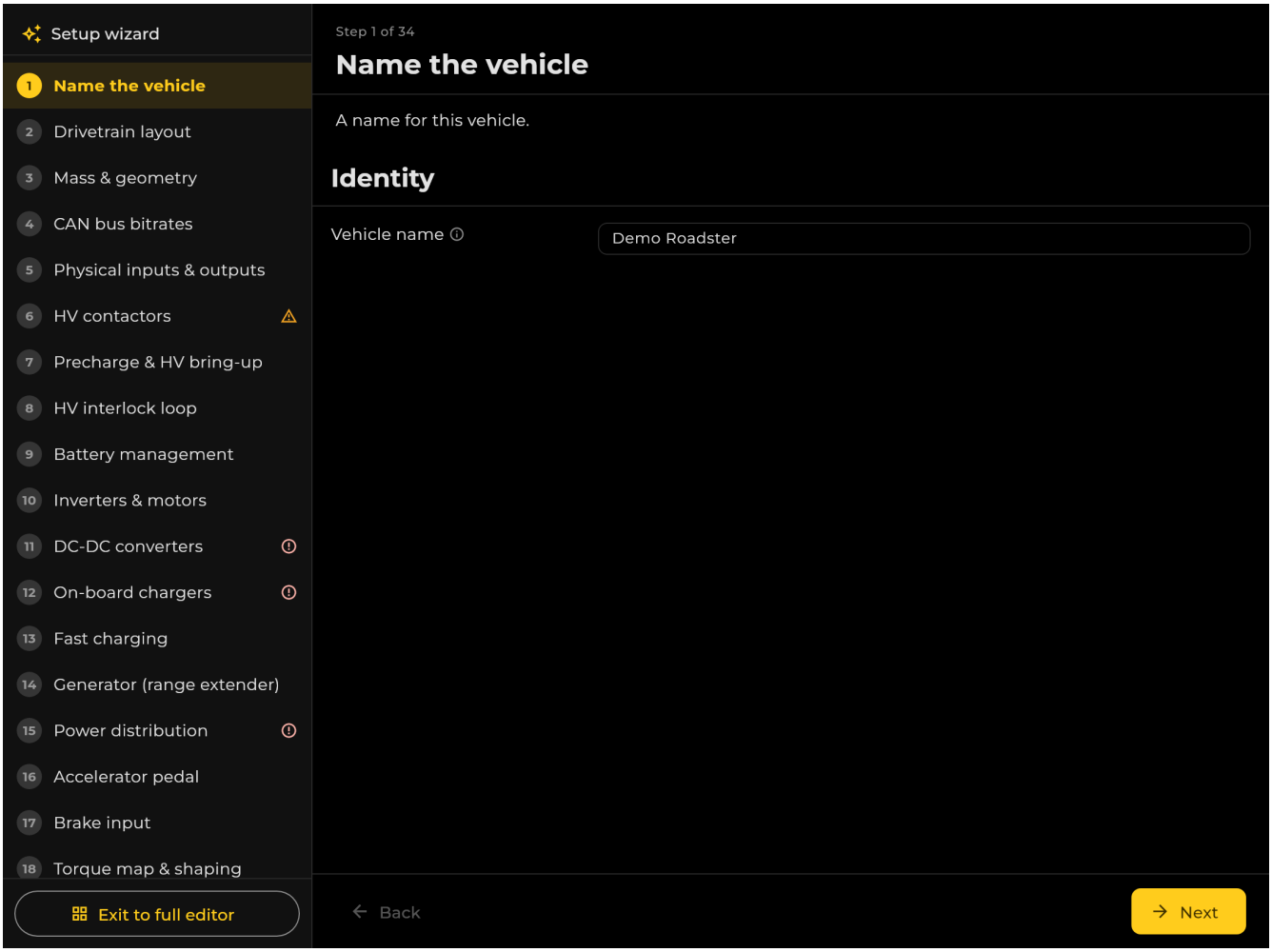
Saving. **Save** writes the open project back to its file; **Save As** writes a copy under a new name. The app commits whatever field you were editing before it saves, so an in-progress change is never lost. A project with unsaved edits warns you before you close it.

Working in tabs. Each open project gets its own tab across the top; click a tab to switch, or its close button to shut it. You can keep several vehicles open side by side — handy for comparing two builds, or copying a setup from one to another.

Inside a project. An open project has two views you switch between: **Configuration**, where you author the build across a tree of sections with the documentation alongside, and **Monitoring**, the live dashboard and drive recorder. You can drag the panes in each view into whatever arrangement suits you, and the app remembers it per project.

The Setup Wizard

Configure a new vehicle step by step. The setup wizard walks you through the settings a build needs, in order, one screen at a time, and finishes only when the configuration has no errors.



Starting the wizard. When you create a new project the app asks whether to set the vehicle up with the wizard or jump straight into the full editor. Choose the wizard for a guided build. You can also enter or resume it at any time from the **Setup wizard** button on the toolbar. The wizard and the full editor work on

the same project, so anything you do in one is already there in the other — **Exit to full editor** and come back whenever you like.

Working through the steps. The left rail lists the steps in order and numbers them; the current step is highlighted and the header shows "Step N of M". Each step is the same form you would use in the full editor, with a short overview and a **How this works** link for the detail. Use **Back** and **Next** to move through them, or click any available step in the rail to jump straight to it.

Subsystems it asks about first. For anything a vehicle might or might not have, the wizard asks "Does this vehicle have X?" before showing its settings. Choose **No, skip** to pass it by, or **Yes, configure** to open its form. A subsystem you have already set up opens straight to its form.

Steps that aren't ready yet. A step whose prerequisites are not in place is greyed out and locked, with a note about what it needs — for example, a feature that depends on a wheel-speed source stays locked until you have provided one. Configure the prerequisite and the step unlocks itself.

Vehicles that don't drive. Not every installation propels itself — a fire-pump boat, a battery box, a stationary generator skid. If you have not given the installation a drive motor, the wizard does not ask about driving it: the accelerator pedal, the brake and the whole torque section (drive profiles, direction selector, cruise, hill assist, creep, traction control, vectoring, regen and the safety monitor) are all skipped, and the build is complete without them. Add a drive motor and they join the sequence.

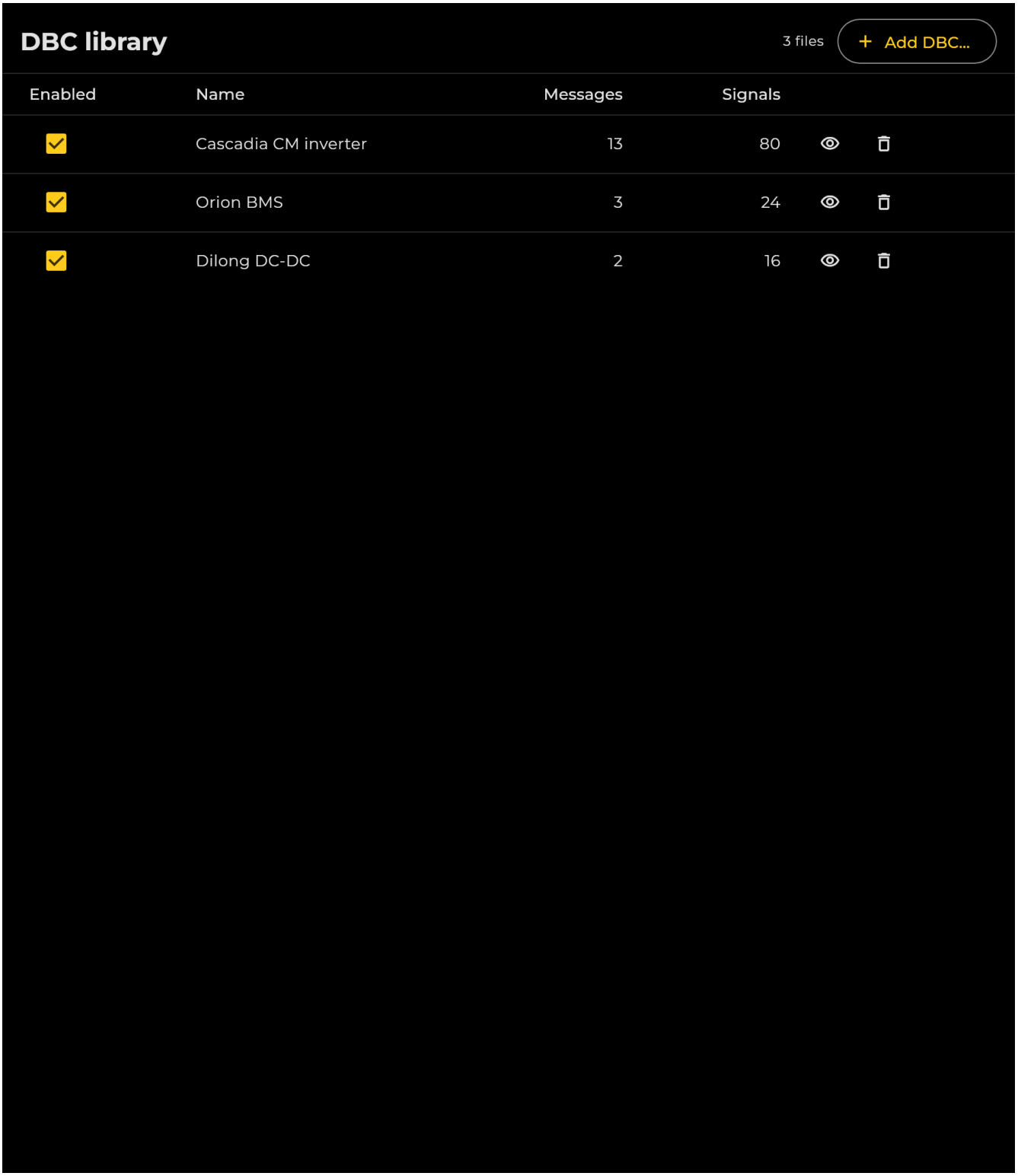
Nothing is hidden from you. Those sections stay in the left navigation the whole time, greyed out. Open one and it tells you what it is for, why it does not apply to this vehicle, and which setting would enable it, with a button to go there. Settings you have not enabled are always visible and always explain themselves — you should never have to already know a feature exists in order to find it. The one exception runs the other way: a skipped step comes back into the wizard if it still holds an unresolved problem, so a warning can never end up somewhere you cannot reach it.

Watching for problems. Each step carries a badge when it has an issue — a red mark for an error, an amber one for a recommendation — so you can see at a glance what still needs attention as you work.

Review & finish. The last entry in the rail is **Review & finish**: a single list of everything still outstanding across the whole vehicle. Click any item to jump to the step that fixes it. **Finish setup** turns on only when no errors remain — the same completeness bar the pre-flash gate holds — so a vehicle that finishes the wizard is ready to flash.

DBC Files

Attach the CAN databases that describe your vehicle's messages so the Configurator can offer their signals by name instead of a hand-typed bit layout. These files are authoring aids — they help you build the configuration and are never written to the vehicle.



What a DBC is for. A DBC file is a CAN database — a vendor's or integrator's description of the messages on a bus and the signals packed inside them. Attach the ones that match your vehicle's devices and the Configurator can offer their signals by name in the 'Configure CAN signal' picker, so you rarely have to type a bit layout by hand.

The attached list. One table lists every DBC attached to this project, with its message and signal counts. The checkbox enables or disables one without removing it — a disabled DBC stays attached but its signals drop out of the picker. Click a file to browse its contents in a window: each message with its id and size, expandable to the signals it carries and where each one sits.

Adding one. **Add DBC…** loads a database file from disk. The Configurator reads it as it loads and reports how many messages and signals it found; if the file reads as empty it asks before attaching, so a wrong or unparseable file doesn't slip in silently.

Per-project, with a shared starting set. New projects begin from the app's shared DBC library, and everything you do here applies only to this project from then on. When you add a DBC the app offers to copy it into that shared library too, so future new projects start with it already attached.

Authoring only. These databases are used only while you build the configuration — to name signals and lay them out. They are not sent to the vehicle; the signals you actually define carry their own layout with them.

Incoming CAN Signals

A CAN signal is a single value carried on the bus — a pedal reading, a gear code, a temperature. Define each one once in this registry and reference it by name wherever a sensor or selector reads it.



One definition, many uses. A CAN signal describes where a value lives on the bus — which message it rides in, which bits it occupies, and how to turn those bits into a real number. This registry holds them all in one table — each row shows a signal's bus, its bit layout, and everything that uses it. Define a signal once and point at it by name from anywhere that reads a CAN value, so correcting the layout fixes every use at

once. Each signal keeps a small reference number in the first column; it is assigned for you and never changes.

Adding a signal. **Add signal** opens the signal editor; click a row to reopen it later. Changes apply when you press **Done** — **Cancel** leaves the signal exactly as it was. If your project has DBC files attached, **Choose from DBC…** is the fast path — pick the signal by name and its message, bit position, byte order, and scaling all fill in for you. With no DBC you can type the layout by hand: the message id, which CAN bus the signal arrives on, the start bit and length, and the scale and offset that turn the raw value into engineering units. You can also create a signal on the spot from the picker in any setting that reads a CAN value — it lands here in the registry just the same.

Duplicating a signal. The copy icon on a row opens the editor on a copy of that signal, named 'Name_Copy'. Change the bits or the scaling to make it a new signal on the same message. If you save the copy with the layout unchanged, it describes the same signal as the original, so the registry keeps only the original and tells you so.

Named values. When a signal came from a DBC that labels its codes — a gear selector reading 'Park / Reverse / Neutral / Drive', say — those labels appear as a reference list, so you can see the exact codes a gear or direction match should target.

Naming a signal. Each signal has a name. Picking from a DBC adopts the name the database uses; when you build the layout by hand you type one. Scripts and recipes read a signal by its name together with its message id — that pair is how they find it — so the name is worth keeping recognisable, and renaming it here changes what they must refer to. Capitalisation does not matter. Two signals riding the same message cannot share a name; the editor will not accept the second one.

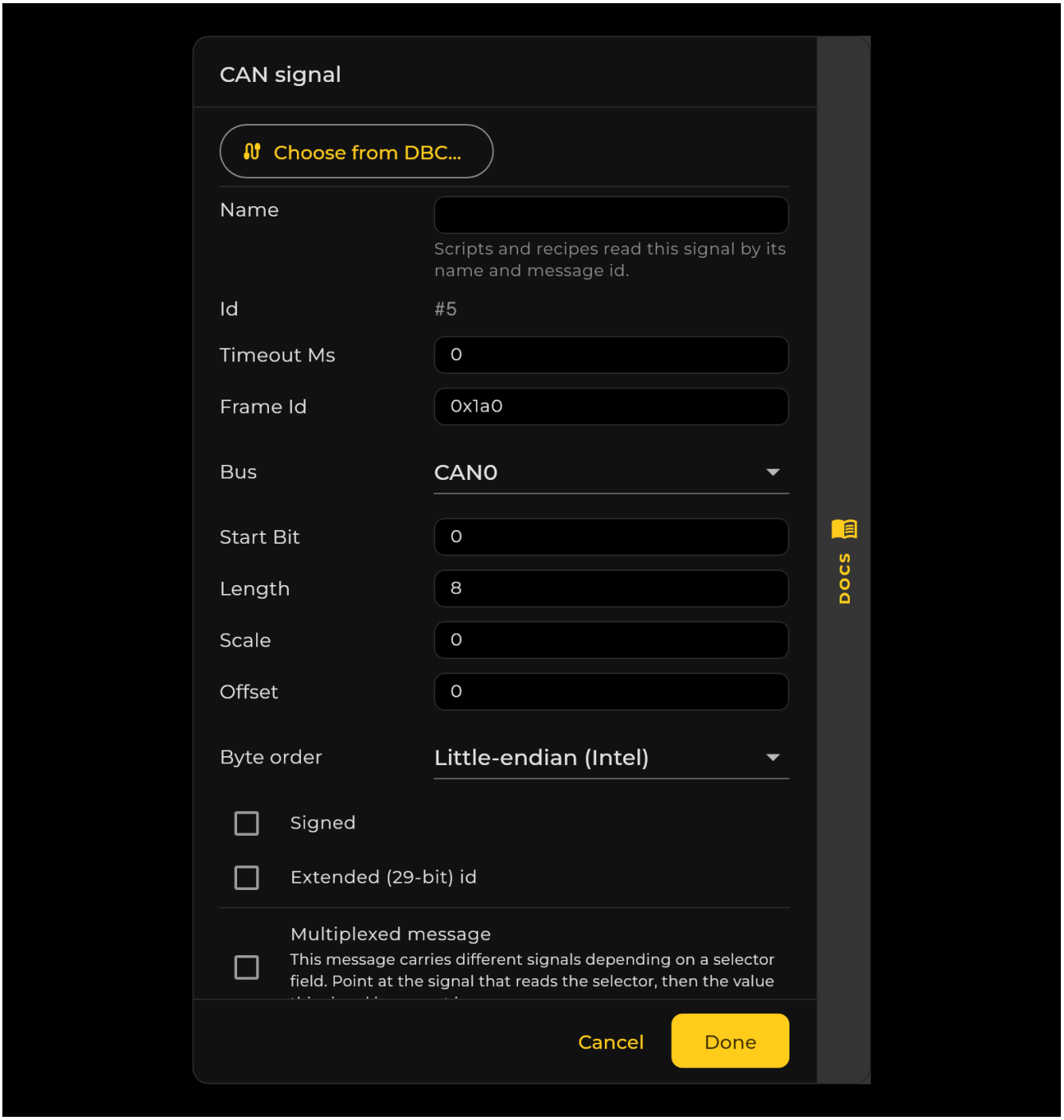
Signals that time out. You can tell a signal what to assume when the bus goes quiet: turn on 'assume a value on timeout' and give it a fallback, so a dropped message reads as a known value instead of a stale one.

Multiplexed messages. Some messages pack different signals into one frame, chosen by a selector field. Mark the signal as multiplexed, point at the signal that reads the selector, and give the selector value this one appears in.

Seeing what uses a signal. The 'Used by' column lists what references each signal, and the editor shows the same 'referenced by' list. A signal that is still in use can't be removed until you clear those references — so you never leave a sensor or selector pointing at nothing.

The CAN signal editor

Every CAN signal is defined in one editor, opened from Add signal or Edit. If a matching DBC is attached, pick the signal by name and its message id, bit position, length, byte order, and scaling all fill in for you; otherwise type the bit layout by hand. Set a staleness timeout and, optionally, a value to assume when the bus goes quiet, and — for a multiplexed message — the selector signal and the value this one rides under. The signal's name sits at the top: taken from the database file on a pick, typed in otherwise. It is what scripts and recipes use to read the signal, alongside the message id.



Devices

The named inputs, outputs, sensors, and actuators your vehicle uses — plus the major subsystems — each bound to a pin, a CAN signal, or a subsystem slot. Scripts and other settings refer to a device by its name.



What a device is. A device gives a name to something the VCU reads or controls: an analog, digital, or frequency input; a digital or PWM output; a sensor fed from a pin, a CAN signal, or a script; a coolant pump, a fan, or a warning light. Installer scripts and other settings refer to a device by its name, so naming things clearly here pays off everywhere else.

One table, two kinds of rows. **Subsystems** — inverters, contactors, chargers, and the like — sit at the top, tagged 'from settings' and read-only: they are created and named in their own settings, and appear here only so scripts and settings can point at them. **Open setting** jumps to where one is configured. Every row below them is managed right here, and the Connection column shows where each device is wired — its pin, its CAN signal, or the script that feeds it.

Adding a device. **Add device** asks what kind, then opens its editor in a window; click a row to reopen it later. **Save** applies your changes and **Cancel** discards them. Give it a name, then fill in what that kind needs — a pin for an input or output, a reading unit and calibration curve for a sensor, a CAN signal for a CAN sensor, the inputs and response curve for a pump or fan, or the thresholds and behaviour for a warning light.

Duplicating a device. The copy icon on a row opens the editor on a copy of that device, named 'Name - Copy'. The copy keeps every setting of the original. Its pin starts empty, because a pin does one job. **Save** adds the copy; **Cancel** adds nothing.

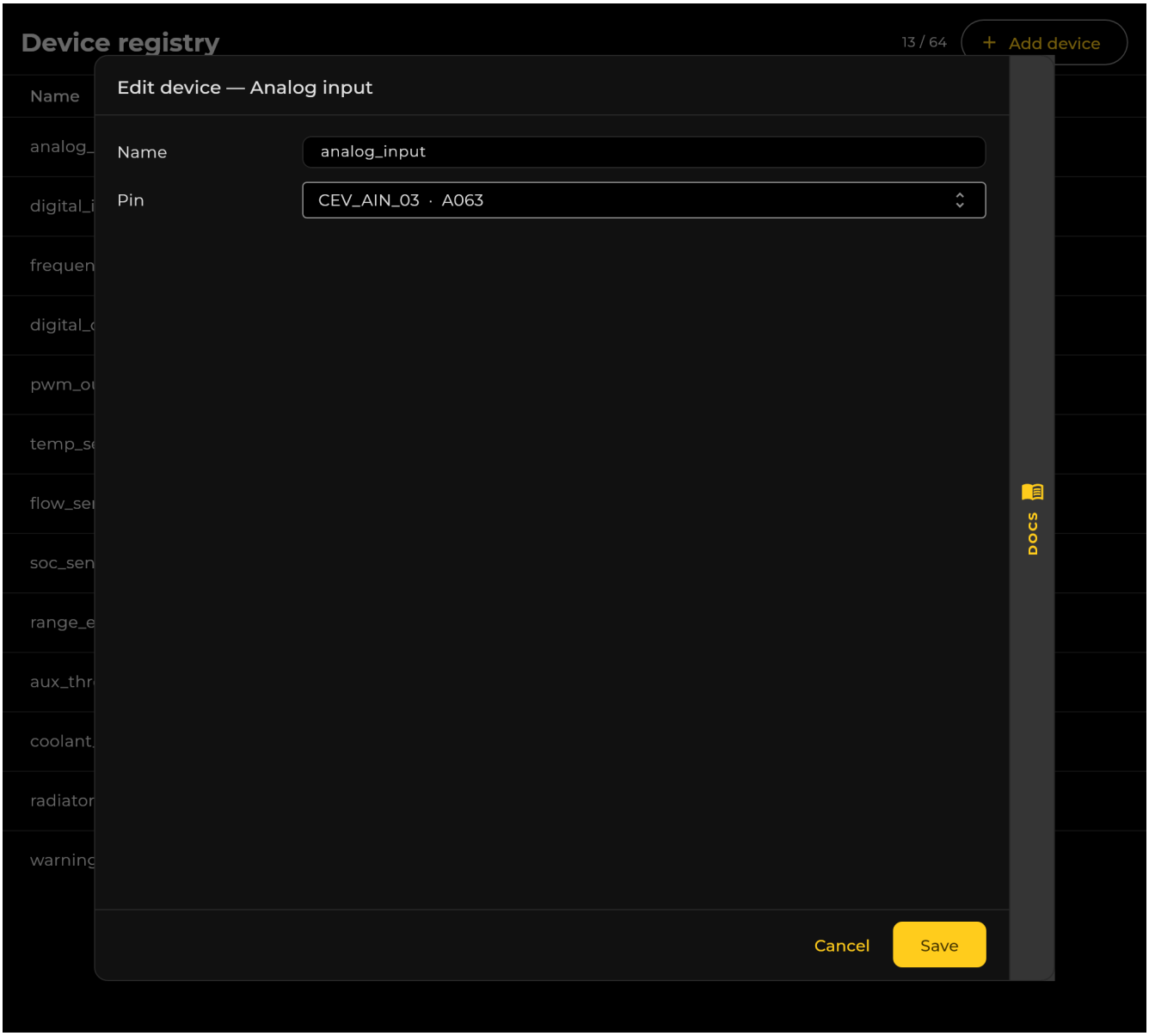
Picking a signal. When a pump, fan or warning light asks which value to follow, the list shows your devices first, then the vehicle's own readings. A reading from a system that is not configured on this vehicle sits last, marked 'Not configured'. You can still pick it, and the app asks first, because it reads as no value until that system is configured. The Issues list warns about every such choice. Picking a signal opens a second step in the same window: you can add math steps to the reading (multiply, offset, clamp, map a range, run it through a curve), say what unit the result is in, and see the name a script can read it by. Leave it empty to use the reading as it is.

Pins are exclusive. A physical pin does one job. The pin picker offers only pins that can do what the device needs, and shows any pin already claimed by another device or a setting as disabled, with a note about what holds it — so two things can't quietly claim the same wire.

Removing. A device that a setting still refers to can't be removed; the app names what points at it so you can clear that first. The counter at the top right shows how many devices you've used against the limit.

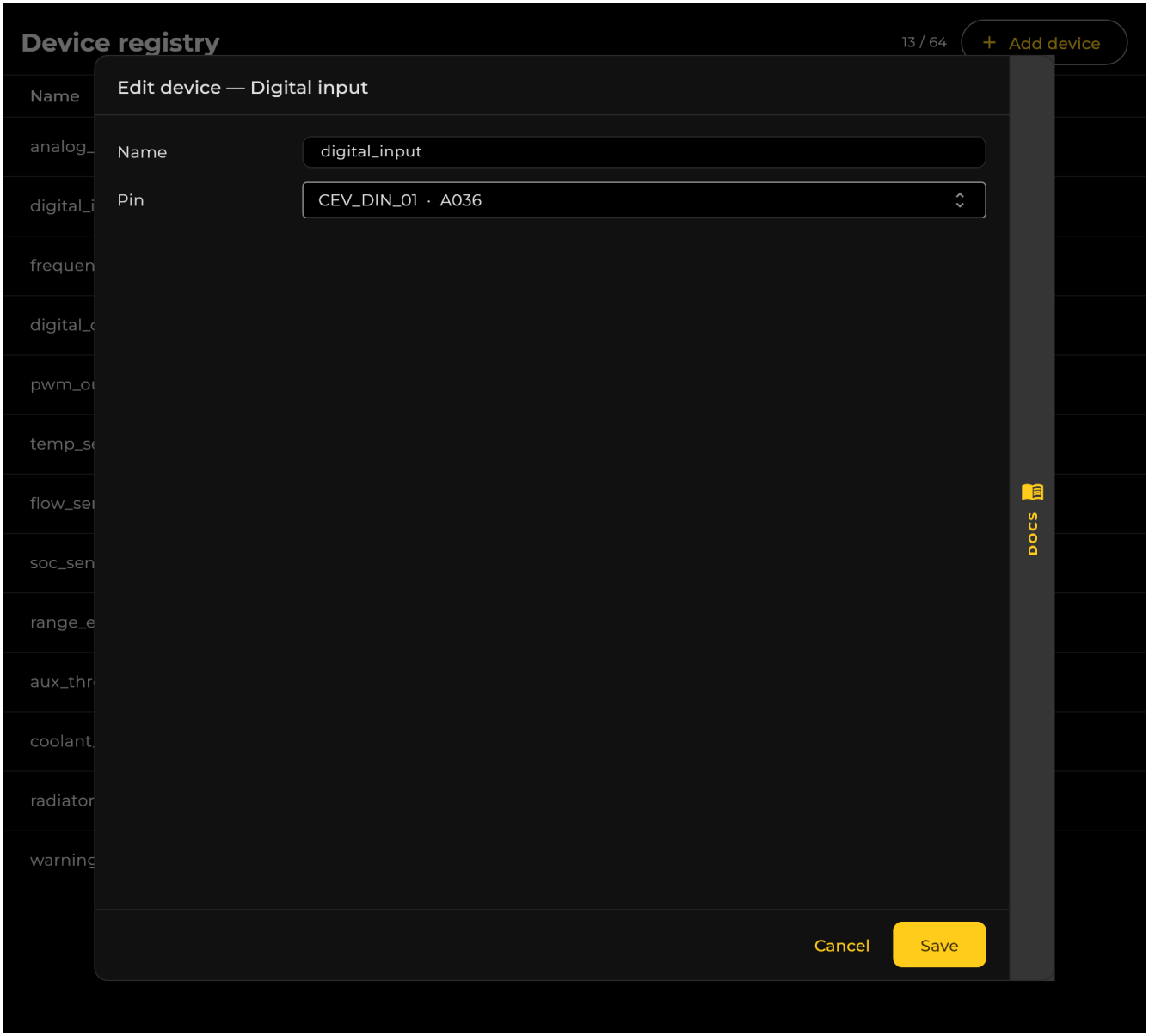
Analog input

A named raw analog voltage input read straight off a pin — the value is the voltage as-is, with no unit or calibration applied. Use it when a script or another setting just needs the raw reading from an analog pin. The editor sets its name and the pin it reads.



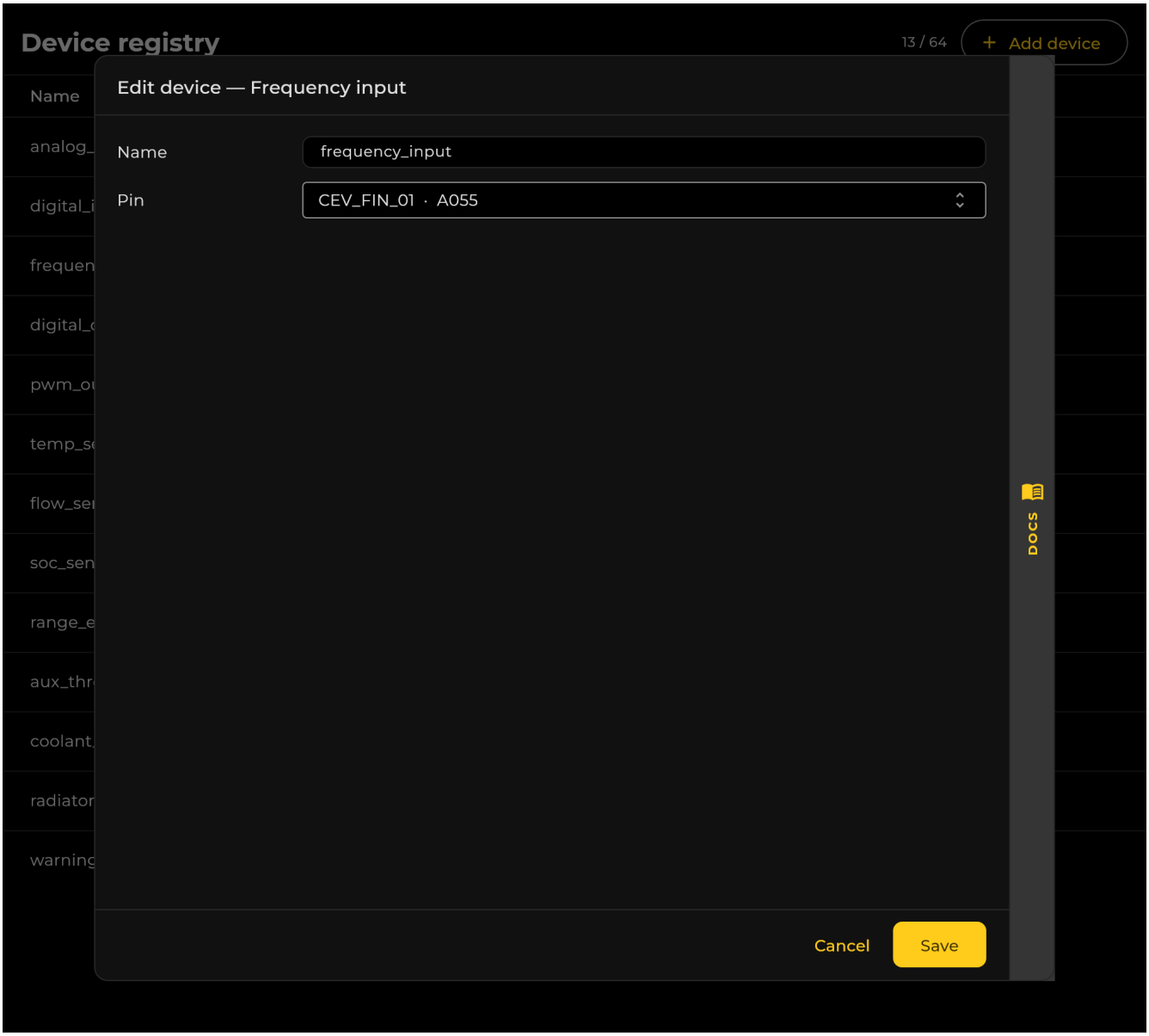
Digital input

A named on/off input read from a pin — high or low, nothing more. Use it for a simple switch, contact, or logic-level signal that a script or setting reads. The editor sets its name and the pin it reads.



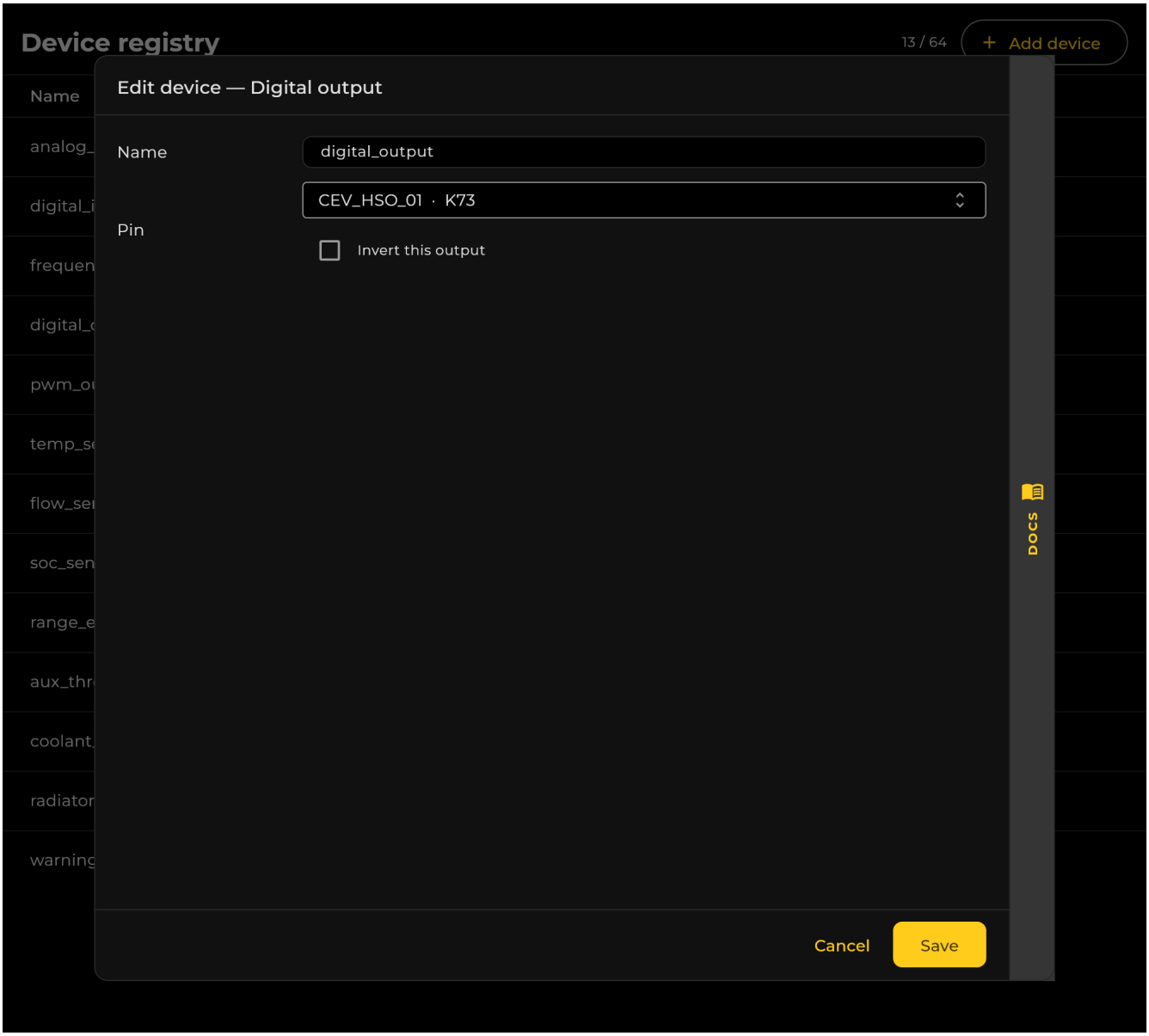
Frequency input

A named input that reads the pulse rate on a pin as raw frequency, with no unit or calibration applied. Use it where a script or setting needs the raw frequency from a pulse source. The editor sets its name and the pin it reads.



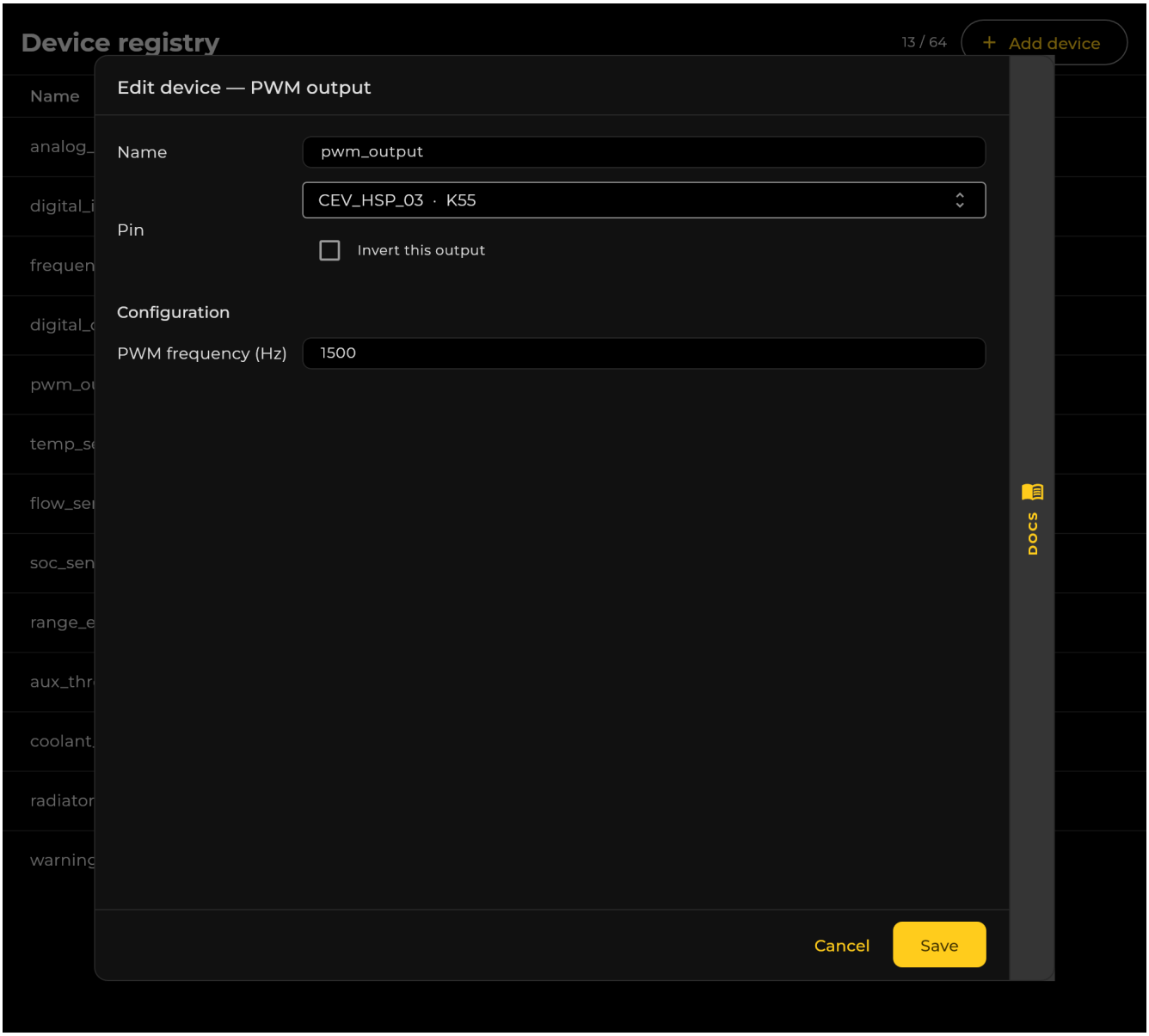
Digital output

A named on/off output driven on a pin — a script or setting commands it high or low. Use it to switch a relay, lamp, or load that only needs to be turned on or off. The editor sets its name and the pin it drives.



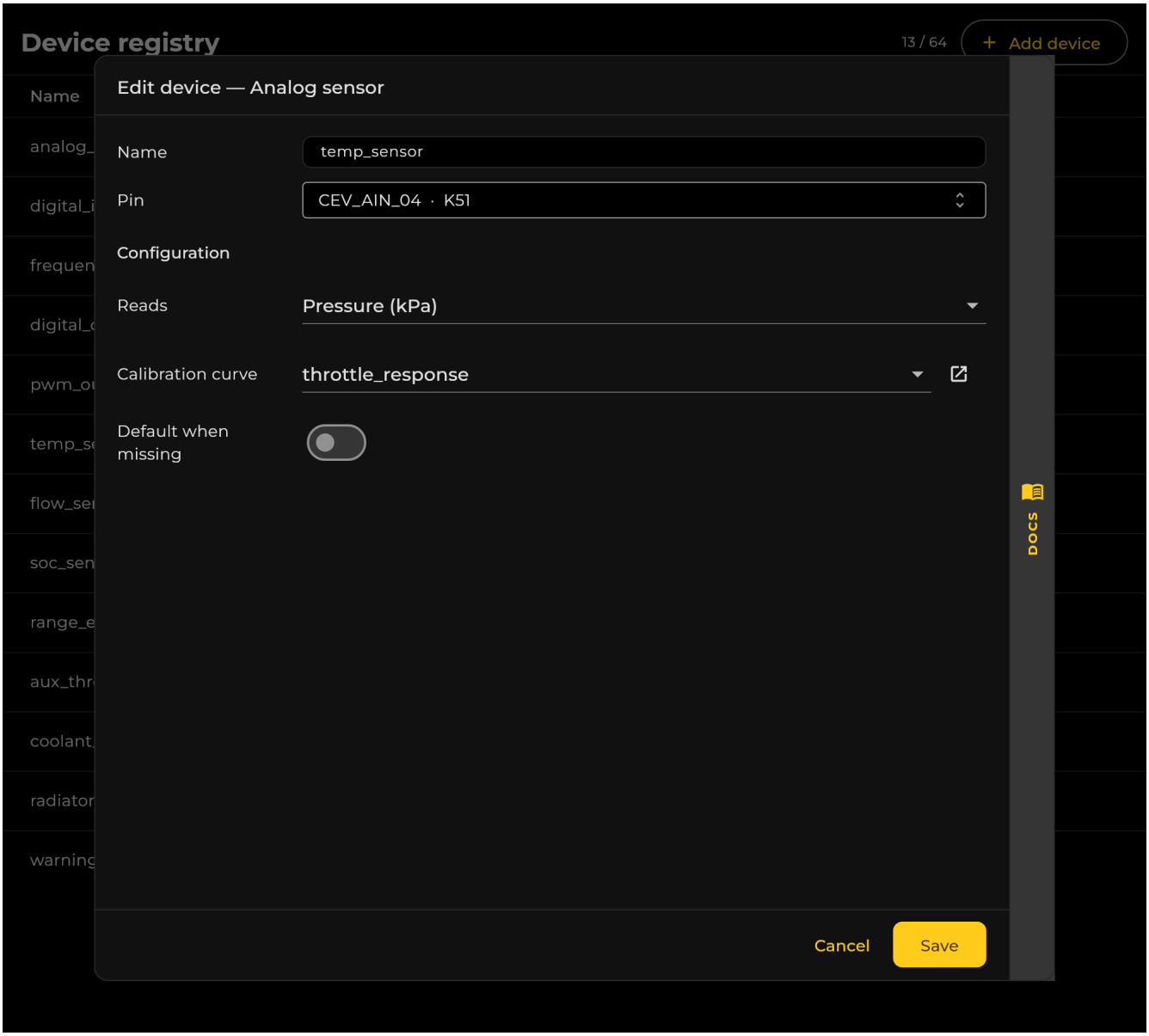
PWM output

A named variable-duty output on a pin — a script or setting sets how hard it drives, anywhere from off to full. Use it to dim a lamp, run a proportional valve, or drive any load that takes a variable duty. The editor sets its name, the pin it drives, and the PWM frequency it switches at.



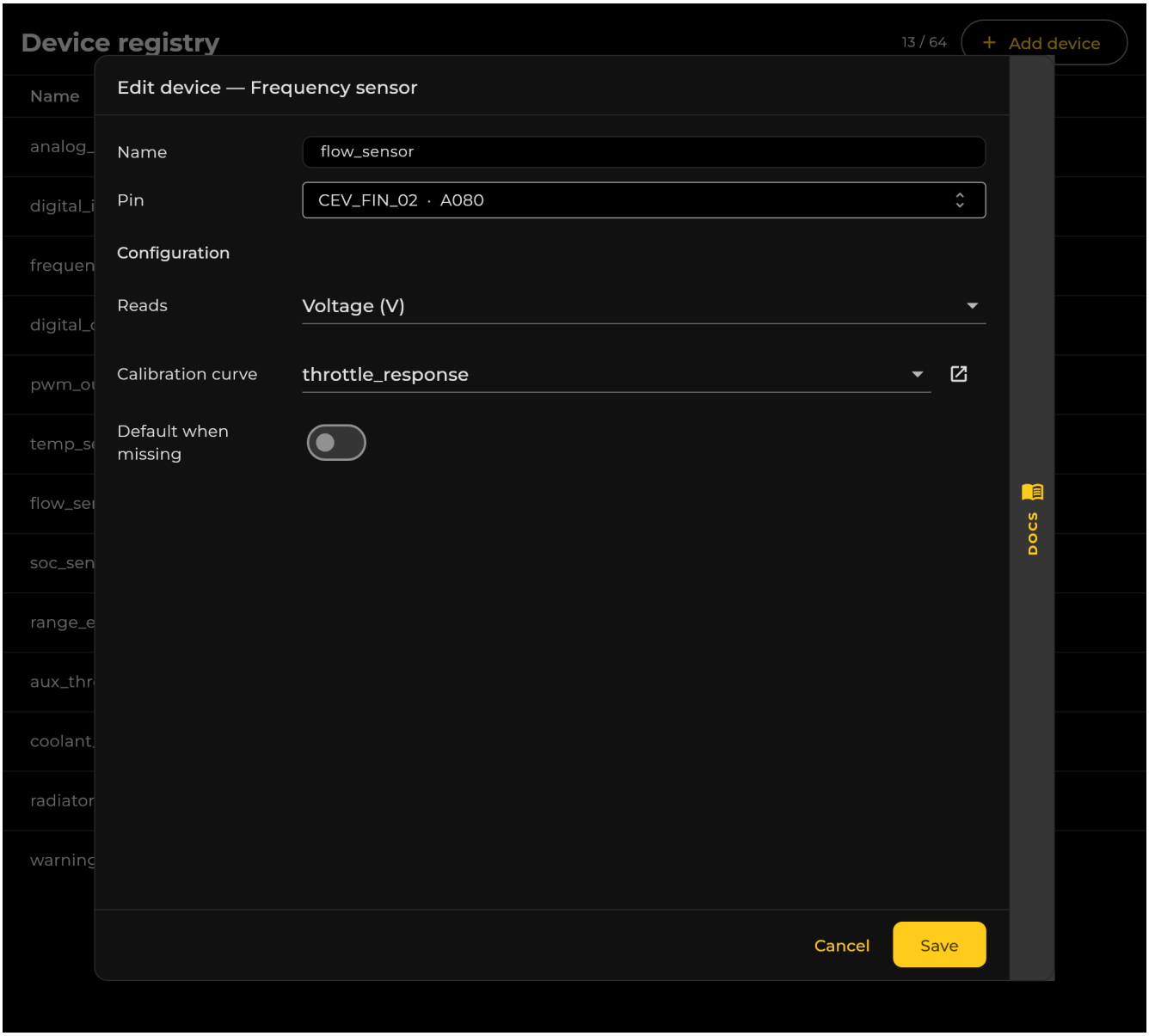
Analog sensor

An analog pin turned into a real reading: the raw voltage is passed through a calibration curve and reported in a unit you choose. Use it for a temperature, pressure, level, or position sensor wired to an analog pin. The editor sets its name, the pin, the unit it reads in, the calibration curve that converts voltage to that unit, and an optional default to fall back on when the reading is missing.



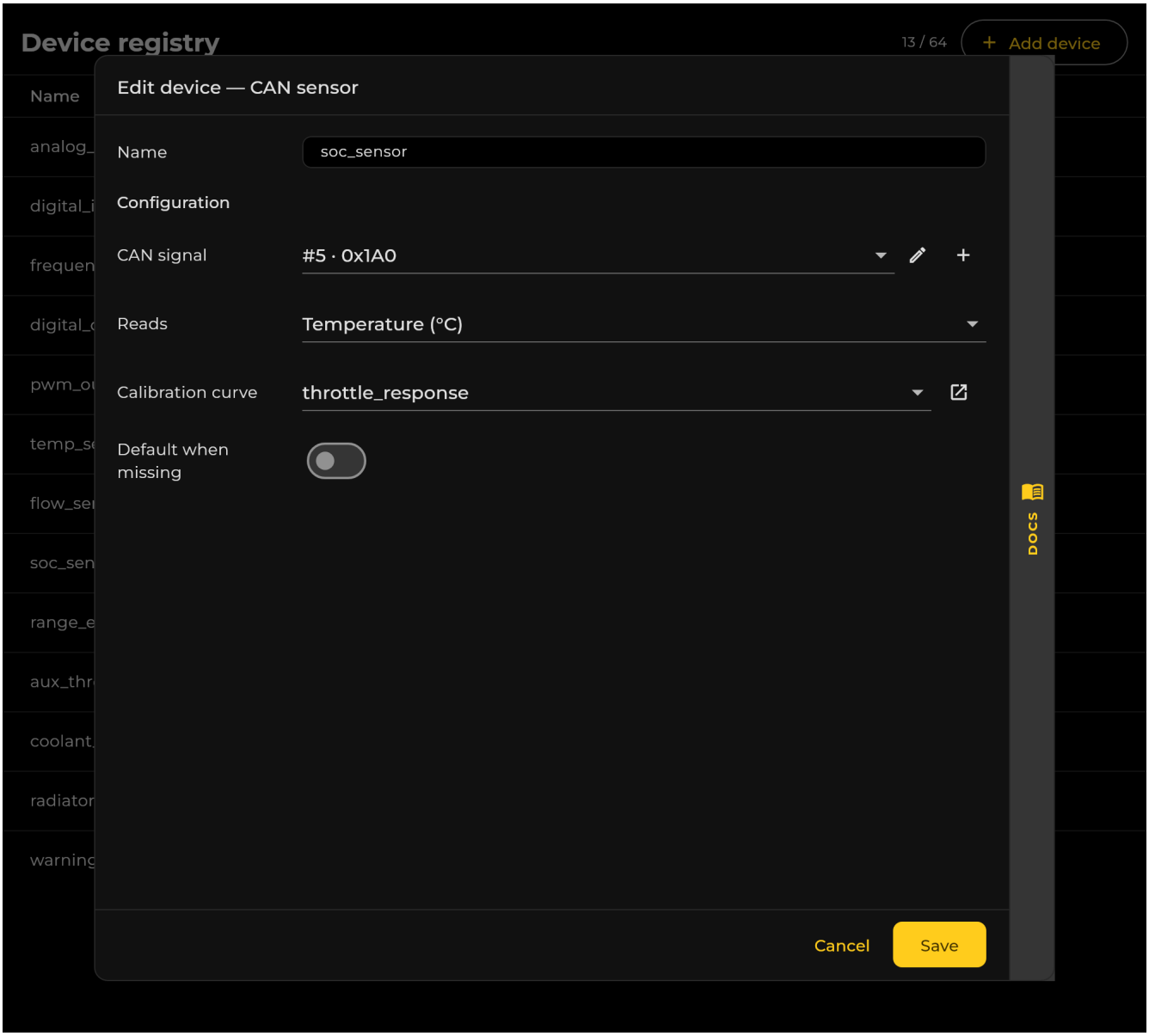
Frequency sensor

A frequency input turned into a real reading: the raw pulse rate is passed through a calibration curve and reported in a unit you choose. Use it for a flow, speed, or RPM sensor that outputs a pulse train. The editor sets its name, the pin, the unit, the calibration curve that converts frequency to that unit, and an optional default for when the reading is missing.



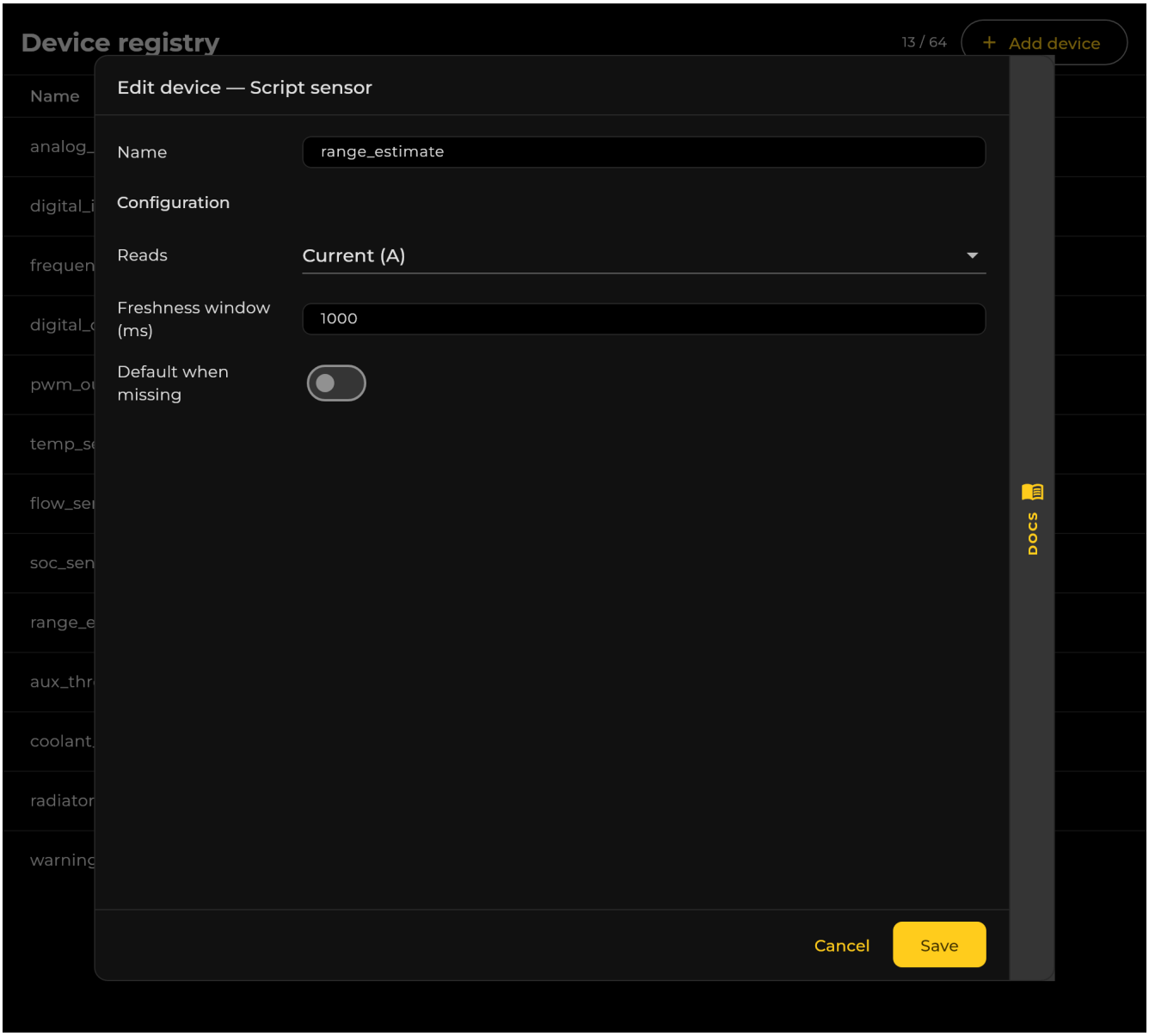
CAN sensor

A reading taken from a value on the CAN bus rather than a pin, passed through a calibration curve and reported in a unit you choose. Use it to expose a signal broadcast by another device on the bus as a named sensor. The editor sets its name, the CAN signal it reads (picked from the signal registry or defined on the spot), the unit, the calibration curve, and an optional default for when the signal is missing.



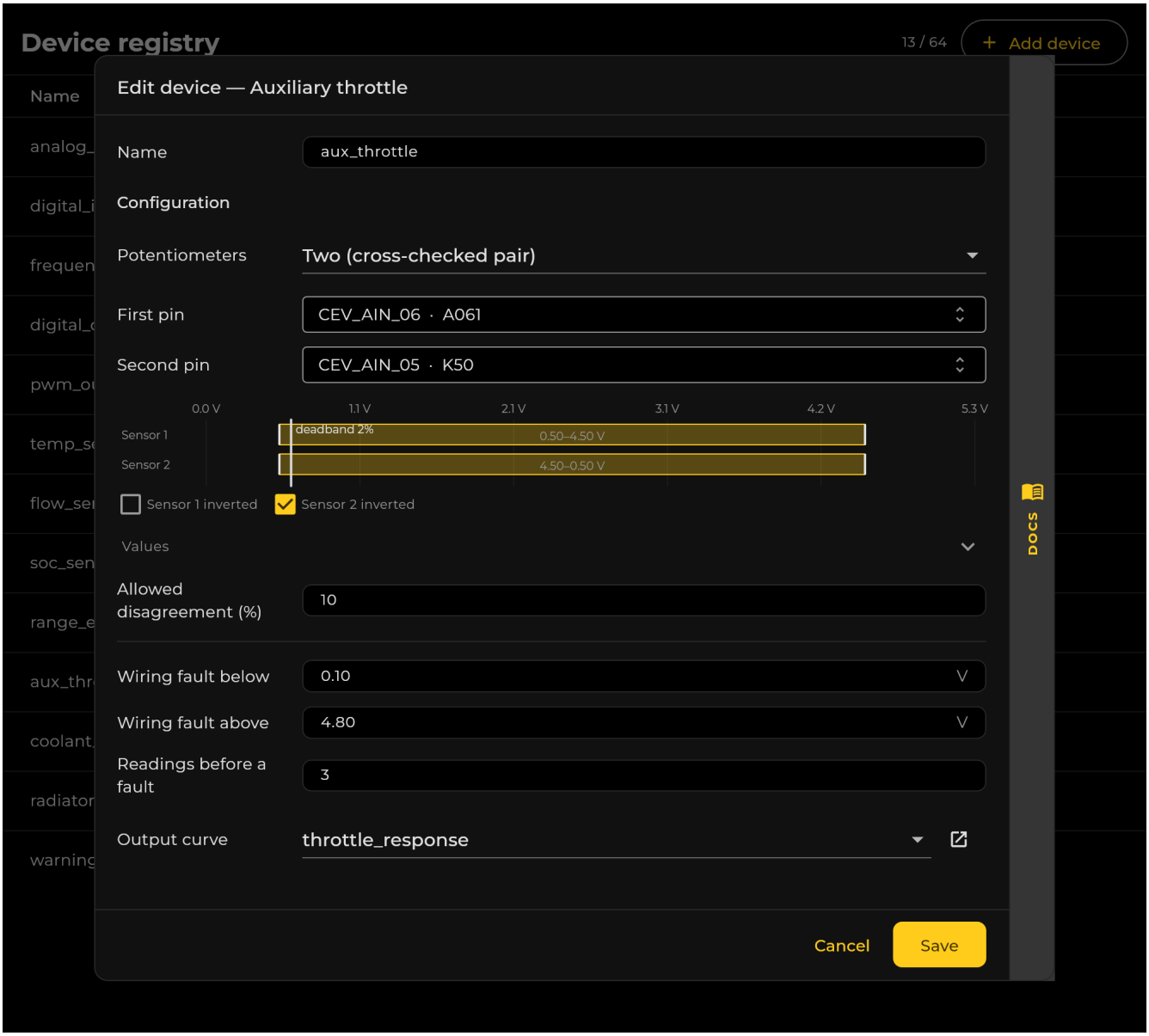
Script sensor

A sensor whose value a script publishes — there is no pin or CAN source; the script writes the reading itself. Use it to surface a computed value to the rest of the configuration as though it were a real sensor. The editor sets its name, the unit it reads in, a freshness window (how long a published value stays valid before it counts as missing), and an optional default for when it is.



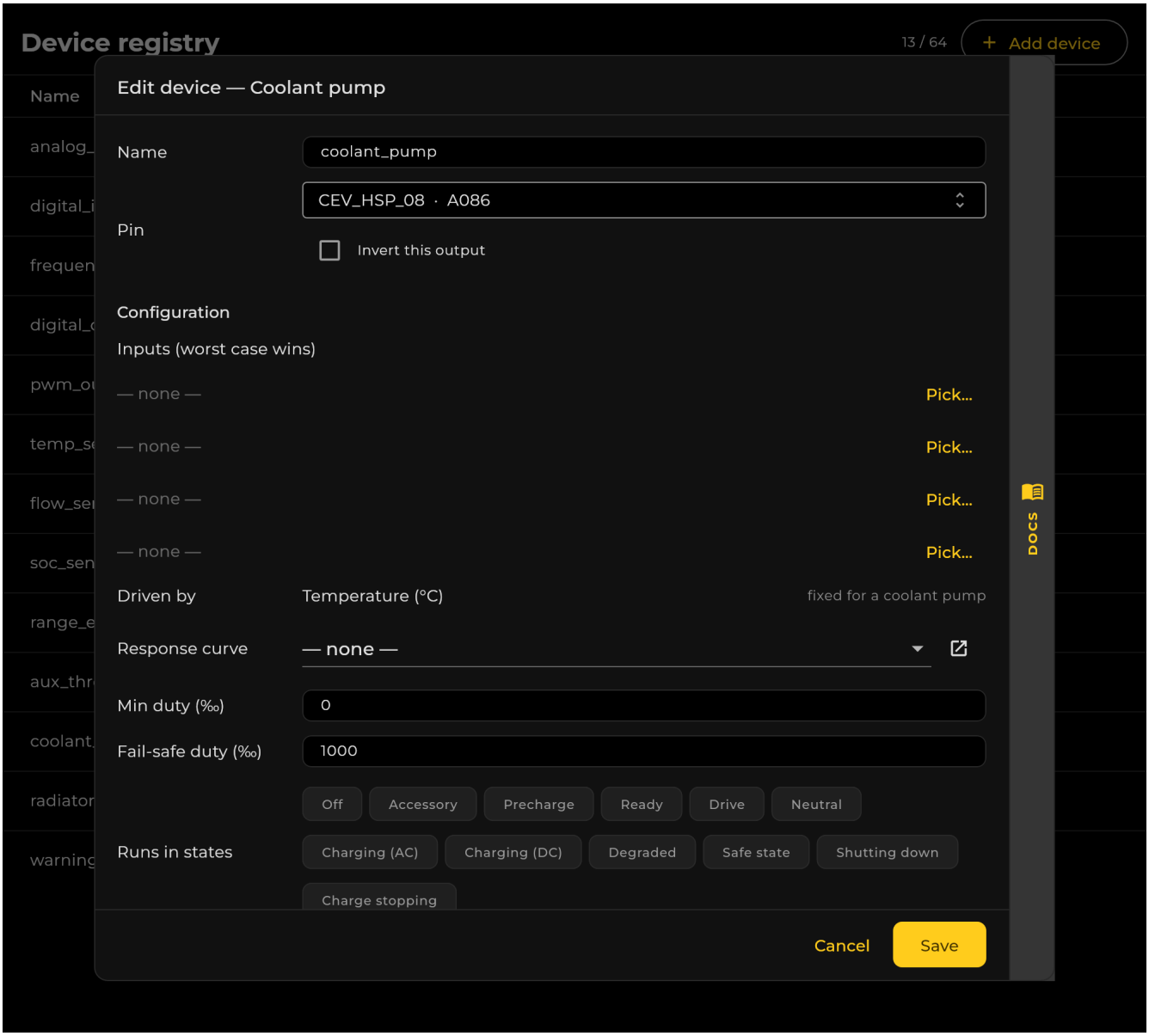
Auxiliary throttle

A hand or foot throttle — a lever or pedal on a potentiometer — read as a 0–100% position that scripts and recipes can use to drive something, typically an accessory motor. **This is not the accelerator.** The accelerator has its own dedicated channels, its own settings, and the safety systems that go with driving the vehicle; a throttle here only publishes a reading and can never ask for torque. Its channels are refused if you point them at the accelerator's. It is set up like the accelerator, though, because the hardware is the same: choose one potentiometer or a cross-checked pair, record the reading with the lever released and again at full travel (either direction — a sensor whose voltage falls as you press is fine), and set the window outside which a reading means broken wiring. A pair also gets an allowed disagreement between the two. Optionally shape the output through a curve; with no curve the output is the travel percentage. A reading that fails any of those checks reads as nothing at all rather than a stale or invented number.



Coolant pump

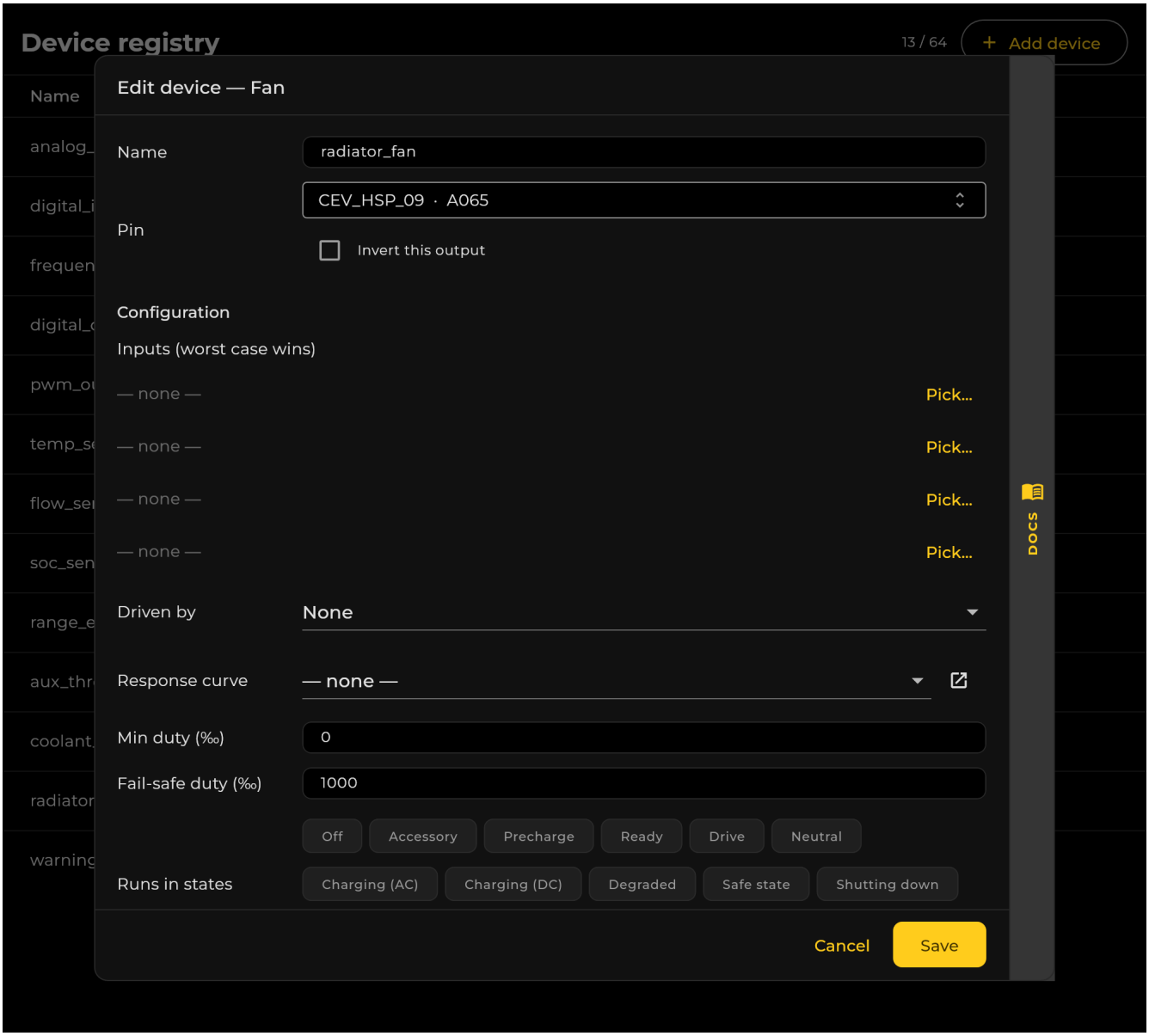
A pump the VCU drives itself on a pin from one or more live inputs — such as a coolant or motor temperature — instead of a fixed command, ramping its output with demand and falling back to a safe duty if the inputs go missing. Use it for a controlled coolant or cooling pump. A pump always works in °C, so its inputs are temperature readings or plain numbers read as °C.



The pump takes up to four inputs, each a source and the quantity it reads. The highest demand among them wins, and that value is run through a response curve to decide how hard the pump runs, in the unit you set under 'Driven by'. A minimum duty keeps the pump turning whenever it is active, and a fail-safe duty takes over if the inputs drop out, so cooling never simply stops. 'Runs in states' selects which vehicle states the pump operates in, and the PWM frequency sets how fast the drive switches.

Fan

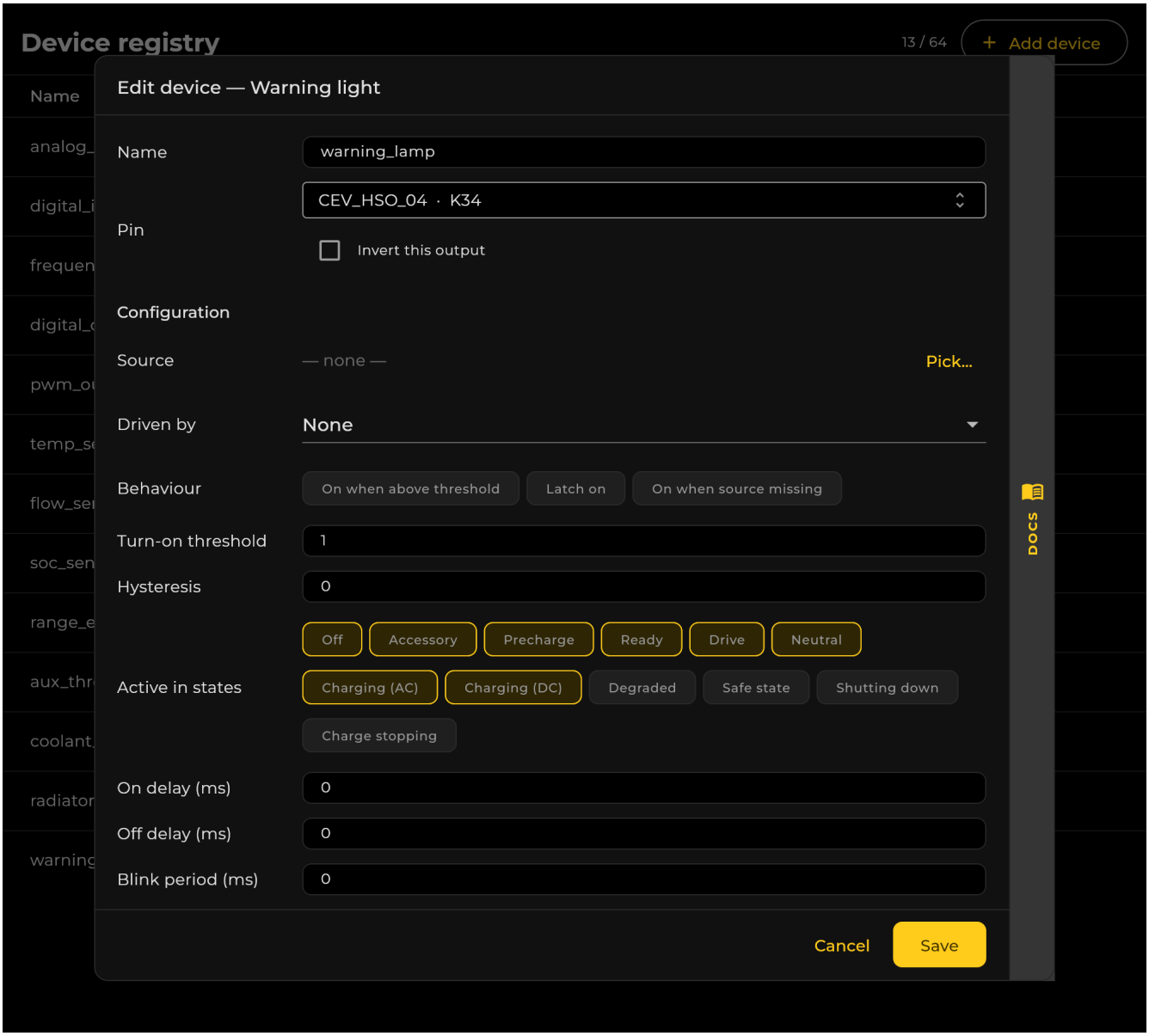
A fan the VCU drives itself on a pin from one or more live inputs, on the same control model as the coolant pump: it ramps with demand and falls back to a safe duty if the inputs go missing. Use it for a radiator, cabin, or component cooling fan. 'Driven by' follows the unit of the inputs you pick: one real unit sets it, and only plain-number inputs (a switch state, a flag) leave it to you, read in the unit you choose.



Its editor is the coolant pump's: up to four inputs with the highest demand winning, a response curve that turns that demand into fan output in the unit set under 'Driven by', a minimum duty that keeps it turning while active, a fail-safe duty for when the inputs go missing, the vehicle states it runs in, and the PWM frequency of the drive.

Warning light

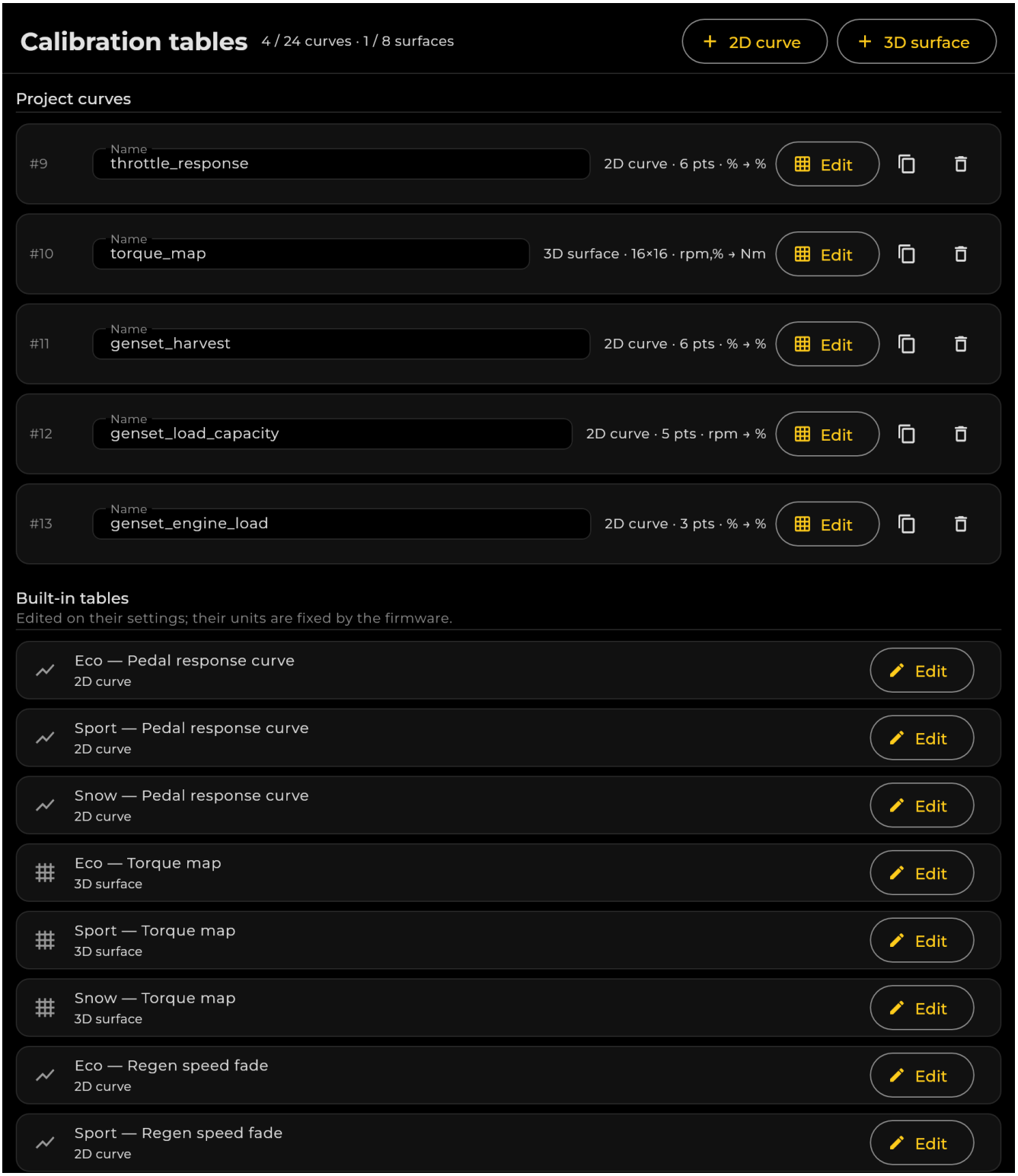
A lamp the VCU drives on a pin from a chosen source and a threshold — it lights when the source crosses the level you set. Use it for a dashboard tell-tale or indicator tied to any reading. The editor sets its name, the pin, the source and its quantity, the threshold and behaviour, blink and delay timing, and which vehicle states it is active in. The threshold is typed in the source's unit; a plain-number source is read in the unit you choose.



'Source' picks the value that drives the lamp and 'Driven by' names its quantity. 'Behaviour' chooses how it reacts — on when the value is above the threshold, latched on once tripped until cleared, and whether to light when the source is missing. The turn-on threshold and a hysteresis band set where it switches on and off cleanly; on- and off-delays absorb brief flickers; a blink period makes it flash rather than hold steady. 'Active in states' limits it to certain vehicle states, and it can be tied to a diagnostic event so the lamp reflects a logged fault.

Calibration Curves & Tables

Create and shape the calibration curves and surfaces the vehicle reads — from a sensor's calibration to the torque map. A 2D curve maps one input to one output; a 3D surface maps two inputs to an output.



What a table is. Calibration comes in two shapes. A **2D curve** maps one input to one output — a line through a handful of points. A **3D surface** maps two inputs to an output — a grid of points. Everything from a sensor's calibration to the accelerator's shaping to the torque map is expressed this way, so once you can read one editor you can read them all.

Project curves. The top list is the curves and surfaces you create yourself. **2D curve** and **3D surface** add a new one; give it a name, then press **Edit** to open the editor. The small **#** beside each entry is a reference number the app assigns automatically — it is shown for information only, and you never type it. Everywhere else in the app you pick a curve by its name, not its number. The copy icon adds a duplicate with the same points, units and axes, named 'Name - Copy'. The trash icon removes one you no longer need.

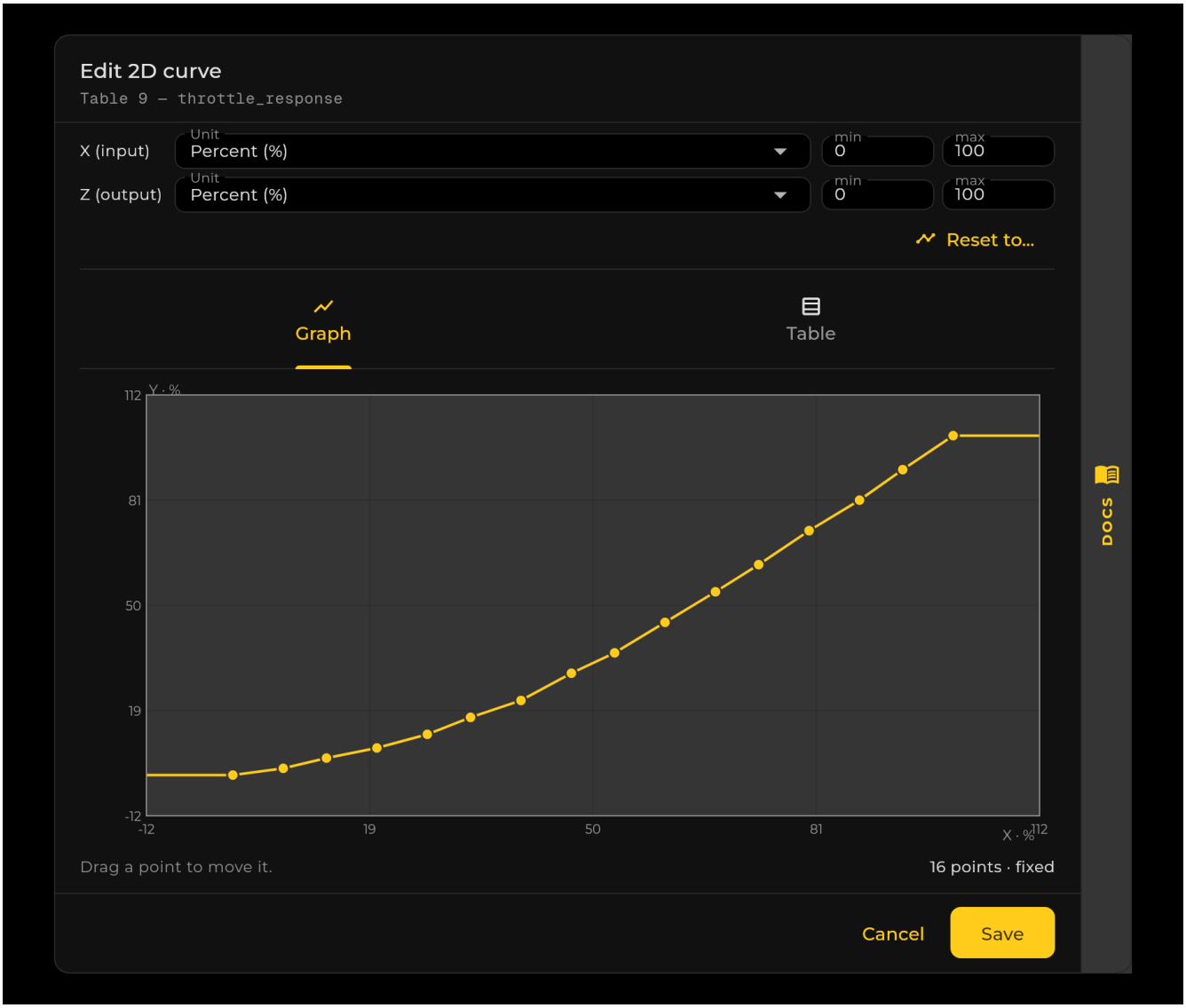
Built-in tables. Below your own curves are the fixed tables that belong to specific vehicle systems — the torque map, the pedal shaping, and the like. You can reshape any of them, but you cannot add or delete them, and the units on their axes are fixed by the setting they belong to. **Edit** opens the same editor.

Editing a table. The editor shows the curve or surface both as a graph you can drag and as numbers you can type. Add points where you need more detail, remove ones you don't, and set each axis's range and units — the graph redraws as you go. Values are shown in familiar vehicle terms, not internal counts.

Live readout. While a vehicle is connected and sending live data, a built-in table's editor draws a moving marker showing exactly where the vehicle is reading that table right now — so you can watch the operating point track the pedal or the motor while you tune. The marker fades out if the live data goes stale, rather than freezing on a last value.

The 2D curve editor

The 2D curve editor shapes a single relationship — one input mapped to one output — as a line running through a series of points. Reshape it two ways: drag a point around on the graph, or open the table beside it and type each point's exact numbers. Each axis carries its own unit and range, so you decide what the horizontal input and the vertical output mean and how far each one runs. When a vehicle is connected and streaming, a live marker rides along the line, showing exactly where the vehicle is reading the curve right now.

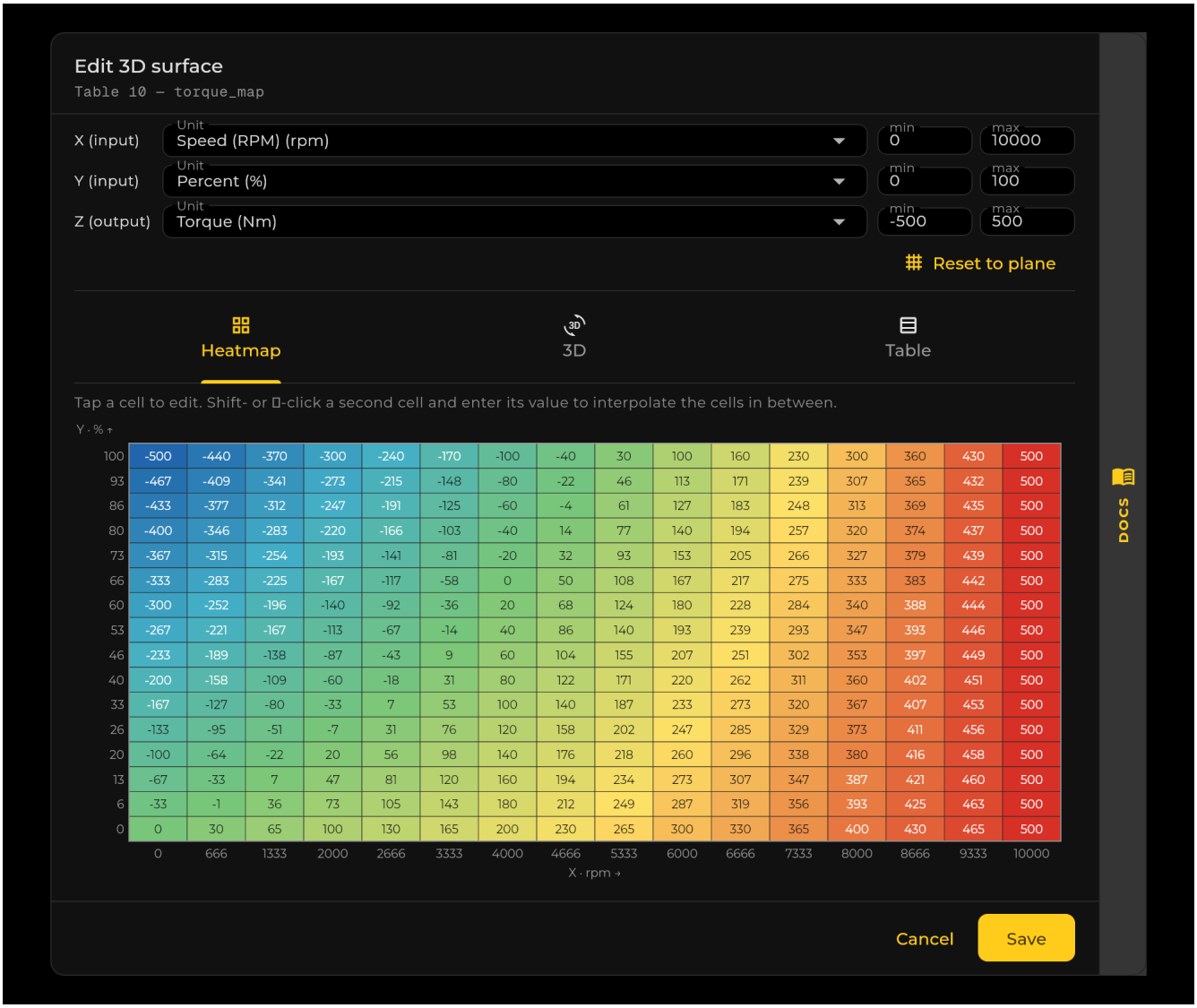


Use whichever tab fits the moment. The graph is for feel — grab a point and slide it until the shape looks right, and the line redraws under your finger. The table is for precision — type the input and output values straight in when you already know the numbers. Set each axis's unit and its low and high limits first, so the graph is scaled to the range you care about and every value reads in plain vehicle terms rather than raw counts. If a curve gets away from you, reset it to a preset shape — a straight line, a soft S-curve, or a progressive or degressive sweep — and start shaping again. When a curve belongs to a particular vehicle system its units are locked to match that setting, so you shape the line but never change what the axes stand for.

The 3D surface editor

The 3D surface editor shapes a map with two inputs and one output — a grid of cells where each pair of input values lands on an output value. View and edit it three ways from the tabs along the top: a colour Heatmap, a rotatable 3D view, and a plain Table of numbers. Tap any cell to type its value; to fill a whole block at once, set one corner cell, then shift- or ⌘-click the opposite corner and enter its value to blend

every cell between the two. Each of the three axes — the two inputs and the output — has its own unit and range, and Reset to plane returns the whole surface to a smooth, evenly-sloped starting point.



The Heatmap tab is where most of the work happens: cells are coloured by value so the whole map reads at a glance, and tapping one opens an entry right on top of it. The real time-saver is the corner-to-corner blend — put a value in one cell, then shift- or ⌘-click a far cell and give it a value, and the editor fills in a smooth ramp across every cell in the rectangle between them, so you set a whole region in two clicks instead of typing each square. The 3D tab shows the same map as a surface you can drag to rotate, which makes a stray bump or dip easy to spot — but it is view-only, so make changes on the Heatmap or Table. The Table tab lists the raw grid with the input breakpoints down the side and across the top. Set each axis's unit and its low and high limits so the grid is laid out over the range you want, and when a surface starts out wrong, Reset to plane wipes it back to a clean, evenly-sloped ramp to build up from. When a vehicle is streaming, a live marker on the Heatmap and 3D views shows where the map is being read right now; and where a surface belongs to a particular system, its grid size and units are locked to match that setting.

The Script Editor

Write and organize the installer scripts that run on the vehicle. One entry-point script plus any modules it uses, with syntax highlighting and completion that knows the devices and curves you have configured.

What the script editor is. This is where you write the installer scripts that run on the vehicle — the small programs that wire up custom behavior for a particular build. A project holds one or more script files; you edit them here, and they are carried inside the project.

The entry point and modules. Every project has one script named **main.lua** — the entry point. It is flagged, always present, and cannot be deleted. Any other files you add are modules that the entry point can pull in, so a large build can be split across several files instead of one. The tabs across the top switch between files; the close button on a module removes it. The entry point has no close button, on purpose.

Adding files. **New** creates an empty script. **Open…** brings an existing script in from disk as a new file in the project. **Replace…** swaps the contents of the file you are currently looking at for one from disk, leaving the rest of the project untouched. A name that would collide with an existing file is given a number automatically, so nothing is ever overwritten by accident.

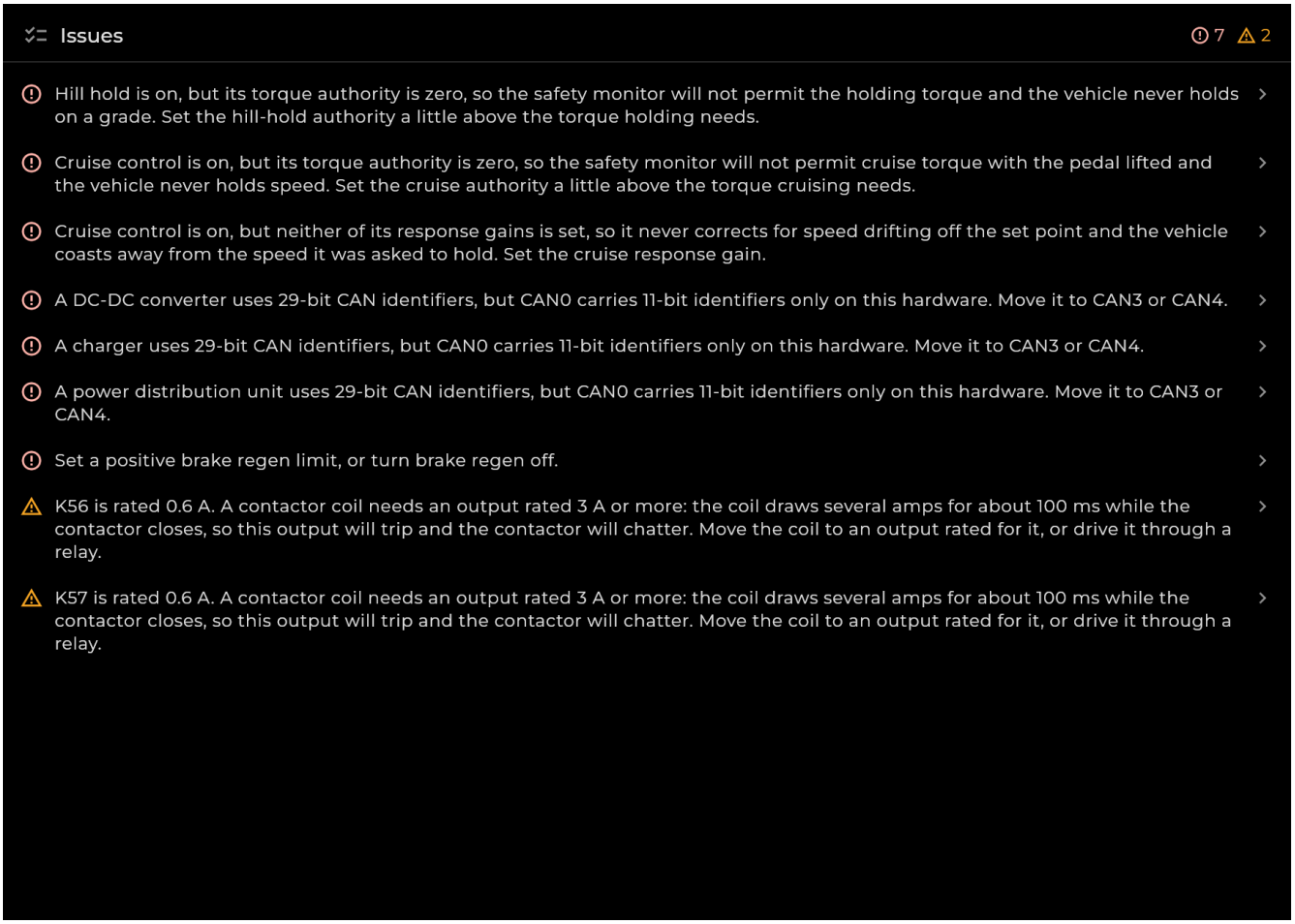
Writing scripts. The editor colours the code as you type and suggests as you go. The completions know the vehicle's scripting vocabulary plus the devices and curves you have configured in this very project, so the names it offers are the real ones you can reference — not a generic list. Indentation follows the structure of the code for you.

Generated files. Recipes appear here as read-only files. When a Recipe is enabled, the shared Recipe helper has a read-only Next flash tab. After a VCU readback, an On VCU tab shows the helper currently stored on that unit. You cannot replace or delete these generated files.

Getting to the vehicle. Your scripts travel inside the project and are packaged into the configuration when you flash it. There is no separate step to send them — flashing the VCU carries the current scripts along with the rest of the configuration.

The Issues List

A live checklist of everything standing between your project and a safe flash — the completeness and consistency problems found in your configuration, split into errors that block flashing and warnings that don't.



What the list shows. The Issues list is the Configurator's running verdict on your project. Every time you change a setting it re-checks the whole configuration and lists what's wrong: settings that are required but still empty, values that fall outside their safe range, and combinations that don't hang together — a feature switched on while a source it depends on is left unset. Errors are listed first, then warnings, with a count of each shown in the header.

Errors versus warnings. An **error** (red) is a problem that makes the build unsafe or incomplete to send to a vehicle — a missing required setting, a value out of range. A **warning** (amber) is something worth a second look that won't stop you: an unusual but legal choice. The distinction is what drives the flash gate: **errors block both flashing and export; warnings do not.** As long as a single error remains, the vehicle cannot be flashed.

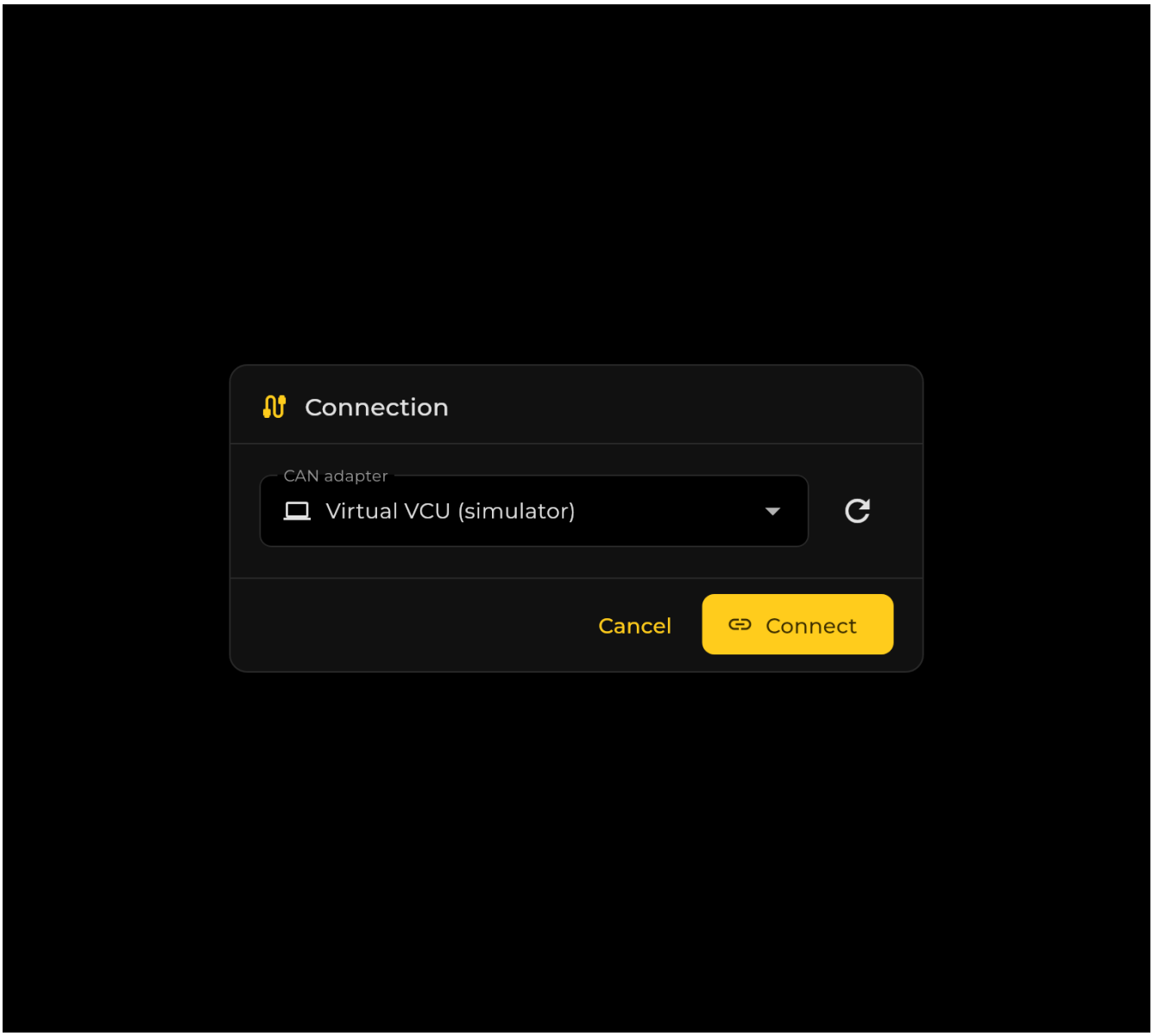
Jump to the fix. Every item is a link. Click it and the app jumps straight to the setting that raised it, with the field in view, so you never have to hunt for where a problem lives. Fix the setting and the item clears itself from the list — the list is always live, re-evaluated as you work, never a stale snapshot you have to refresh by hand.

A clean list means ready to flash. When there are no errors and no warnings at all, the list shows an all-clear — no issues, ready to flash. That is your signal that the project is complete and consistent enough to send, because it is the very same check the flash gate runs: a clean Issues list is your assurance the flash won't be blocked. Warnings left on the list still let you flash, but read them first.

When there's nothing to check. With no project open the pane just prompts you to open one. The checks run against a project's configuration, so there is nothing to report until you have a project loaded.

Connecting to a Vehicle

Link the Configurator to a VCU over CAN so you can read its live state, flash a configuration, and watch the dashboard. You connect through a CAN adapter — a USB dongle or an on-board interface — or work offline against the built-in simulator.



Opening a connection. The **Connect…** button on the toolbar opens the Connection window. Its **CAN adapter** dropdown lists every adapter the app can currently see: named channels on a Kvaser, Vector or IXXAT interface (IXXAT on Windows only), PEAK PCAN devices, the serial-port dongles (CANdapter, CANable, USBtin, CANtact and the like), and — on Linux — any on-board CAN network interface. If the driver for one of these is not installed the app shows it greyed with a short note why, rather than failing.

Pick the adapter your VCU is wired to and press **Connect**. The **Rescan** button next to the dropdown looks again for adapters you plugged in after opening the window.

Working offline. The dropdown always includes a **Virtual** adapter. It runs a simulated VCU inside the app so you can build a configuration, validate it, and explore the dashboard with no hardware attached — useful for authoring a build before the vehicle is on the bench. The virtual link behaves like a real one: it goes through the same connect sequence and reports a vehicle name and schema version.

The connect sequence. Connecting is staged, and the window shows each step as it happens: **Opening link** brings the adapter up; **Vehicle name** and **Schema version** read the VCU's identity and confirm the configuration format the firmware expects matches what this build of the app produces; **Telemetry** confirms the vehicle will answer the dashboard's live-data requests. When all four steps pass, the window shows the connected vehicle's name and schema version and the dashboard begins updating. If a step fails the window stops there and names it — most often the vehicle's identity and schema could not be read, which points at the wrong adapter, a wiring problem, or a VCU that is not powered.

Reading the steps honestly. A step only earns a tick when it actually confirmed something. A step that ran but got no answer is marked as unconfirmed instead, and the window says so rather than reporting a clean connection — so the ticks tell you what the vehicle really did, not merely how far the sequence got.

When nothing at all comes back. Opening the adapter succeeds as soon as the driver hands over a channel, which it will do even when the adapter is not actually on the bus. If the connection comes up but no traffic whatever is seen, the window says so and names the usual cause: **a CAN adapter can only belong to one place at a time**, and if you are also running a virtual machine on this computer it has very likely taken the adapter for itself — hand it back to this computer, or shut the virtual machine down, and connect again. Failing that, check that the adapter is on the diagnostic bus, at the right bit rate, and that the bus is terminated. This case looks exactly like a dead vehicle otherwise, so it is worth ruling out first.

Staying connected. Once connected, the diagnostic tools light up: the live dashboard, comparing the vehicle's configuration against your project, and flashing. The dashboard keeps the live data current by asking the vehicle for it repeatedly; if the vehicle goes quiet the dashboard greys its gauges rather than showing stale numbers. **Disconnect** drops the link; after a failure, **Retry** re-runs the sequence against the last adapter you picked.

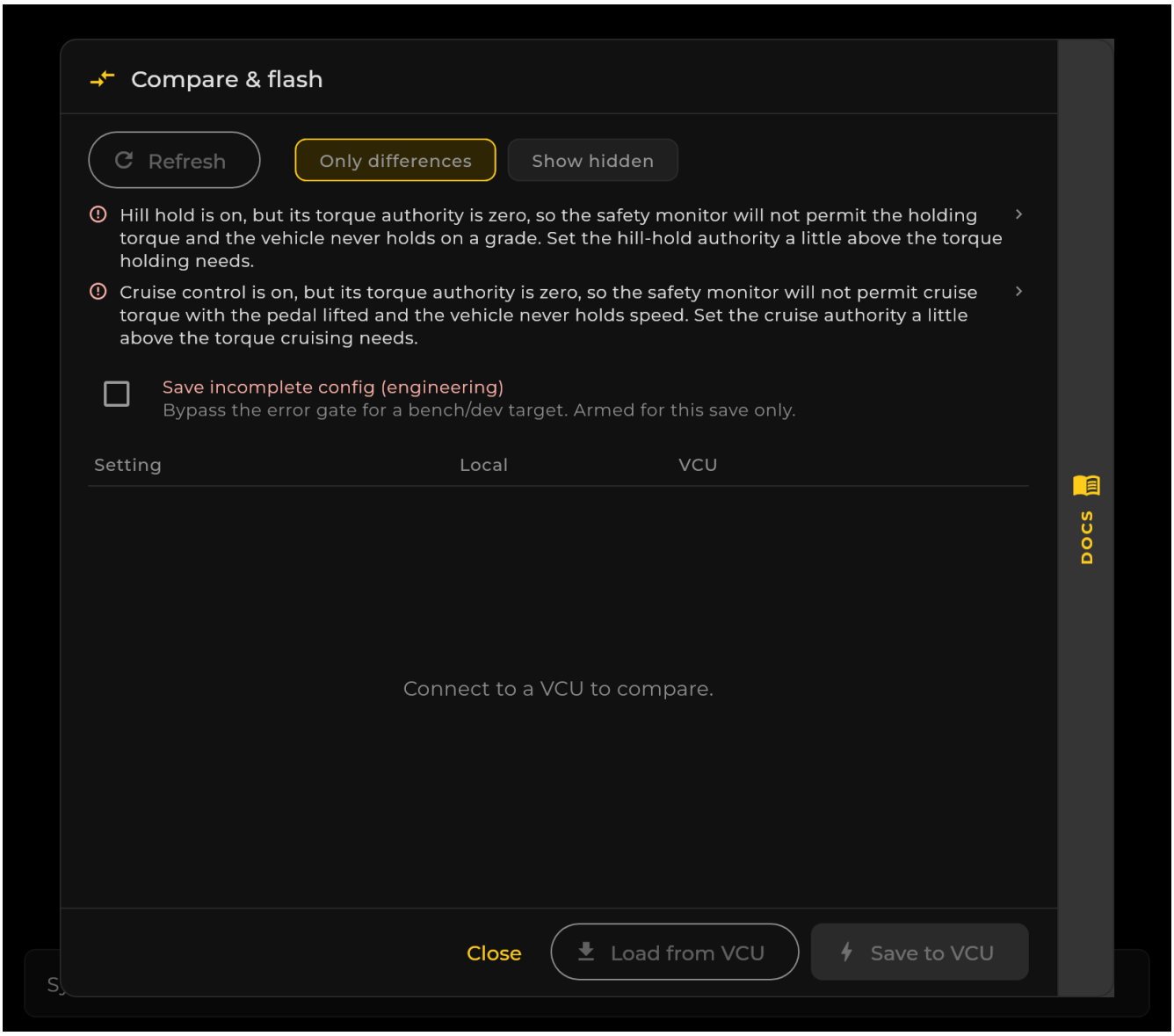
Schema mismatch. The schema version the app reads back is the contract between this build of the Configurator and the firmware on the VCU. If the two do not match, the two disagree about the shape of the configuration, and a flash built against the wrong shape is not safe to send. If the vehicle's format is *older* than this app's — an older firmware that has since been updated — you do not have to reconfigure it: open the compare window and choose **Migrate from VCU…** to carry its settings forward and review what changed. If the vehicle's format is *newer* than this app understands, the app will say so and stop; update the Configurator and try again.

Sharing the link. While the app is connected, it also serves the adapter to other tools on this computer, and to the build machine's virtual machine, over a local network socket, so a diagnostic script can use the same dongle without you disconnecting. The connection window shows the address it listens on, how many tools

are attached, and which one, if any, currently holds the bus. While a tool holds the bus the dashboard pauses its readings and the compare and flash actions wait; they resume the moment the tool lets go.

Flashing the VCU

Write your configuration to the VCU. The Configurator builds the image from your project, checks it, and either saves it to a file or flashes it live over the connection — with a completeness gate that stops an unfinished or unsafe build from reaching the vehicle.



What gets flashed. The Configurator builds a single configuration image from your project and writes it to the VCU, which reads it at start-up. The **Flash** panel shows the built image at a glance — its size, a validity check on its contents, and where each part of the configuration sits inside it — so you can confirm a real image was built before you send it.

The pre-flash gate. This is the app's most important safeguard. Before it will build or flash, the Configurator runs the full validation pass over your project — the same issues you see in the Issues list. **If there are any**

errors, both flashing and export are blocked, so a half-configured vehicle cannot be sent to hardware by accident. Clear the errors — each one links to the setting that raised it — and the buttons enable themselves.

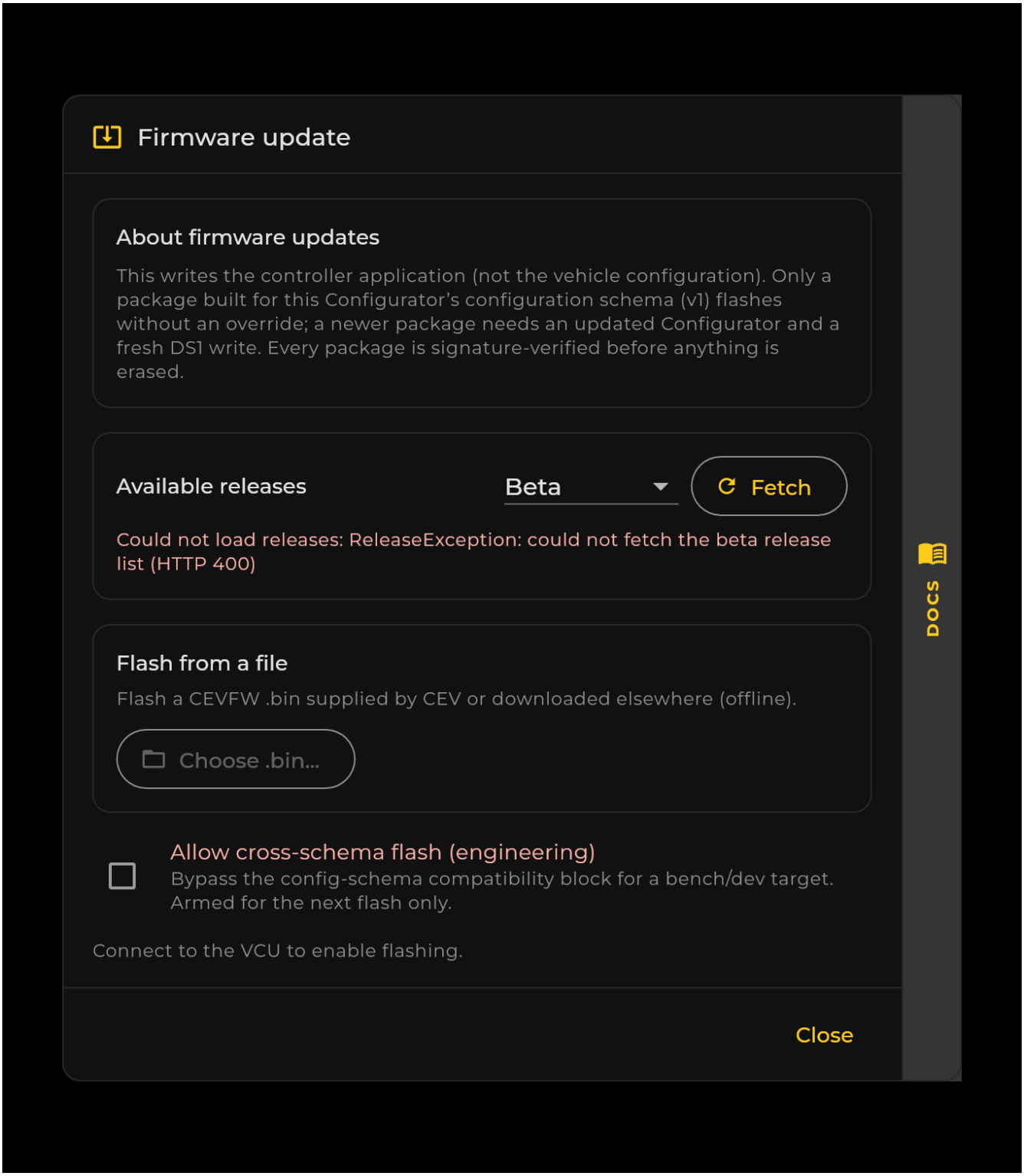
The engineering override. There is a deliberate escape hatch for bench and development work: arming the per-flash **engineering override** lets a flash proceed despite errors. It is intentionally awkward. It covers one flash only and disarms itself immediately afterward, so the very next flash is gated again, and the result is called out as an override — both on screen and in the confirmation — together with the number of errors that were bypassed. Never use it to hand off a finished vehicle; use it only on a bench, when you know exactly why an error is present and safe to ignore.

Flashing live. With a vehicle connected and the gate clear, **Flash to VCU** writes the image. The panel shows progress as it goes — entering the programming session, then a transfer percentage — and finishes by resetting the VCU so it boots the new configuration. If the VCU refuses the image, the panel says so, and the vehicle keeps running its previous configuration untouched.

Exporting to a file. **Export image…** saves the built image to disk instead of flashing it. Use it to hand an image to another tool, to archive exactly what a vehicle was given, or to flash from a machine that is not running the Configurator. Export obeys the same pre-flash gate as a live flash — an image with unresolved errors is not written.

Updating Firmware

Update the controller's application software from a published release or a supplied file. Every package is signature-checked before anything is written, and a package built for a different configuration version is blocked so the firmware and the vehicle settings can't drift apart.



Firmware versus configuration. This screen updates the controller's application software — the program that actually runs the vehicle. That is a different thing from the vehicle settings you build on the Flash screen: settings tell a given firmware how this vehicle is wired and tuned, while firmware is the program itself. You update firmware far less often than settings, usually only when CEV publishes a new release.

Where firmware comes from. There are two sources. **Available releases** lists the firmware CEV has published, on a Production or Beta channel — press **Fetch** to load the list, and each entry shows its version and the configuration version it expects. Alternatively, **Flash from a file** takes a firmware package (a .bin) that CEV supplied directly or that you downloaded on another machine, which lets you update a controller with no internet connection.

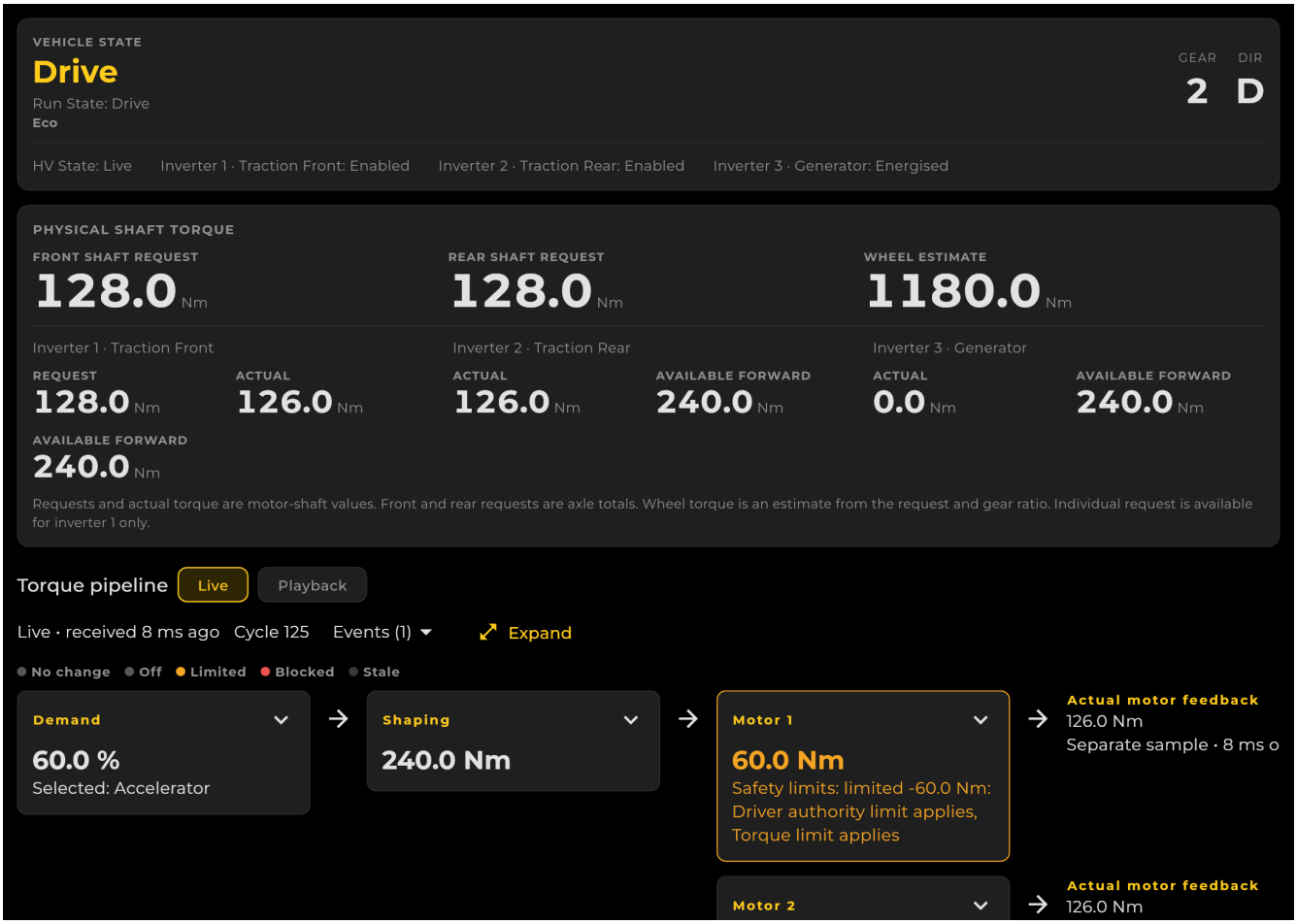
The compatibility rule. Firmware and settings share a configuration version, and they must agree. A release that matches this Configurator's configuration version is marked **compatible** and can be flashed straight away; one built for a different version is marked **needs override** and is blocked, because newer firmware that changes the configuration version needs a matching Configurator and a fresh settings write afterward. The Configurator is the leading edge: update the app first, and the matching firmware unlocks.

Checks before anything is written. Safety comes first: before it erases a single byte, the app verifies two signatures on the package — one proving it came from CEV unaltered, and one proving the controller will accept and boot it — and it confirms the package is built for this controller model. If any check fails, the update is refused and the controller is left exactly as it was.

Flashing, the override, and the read-back. Connect to the vehicle first. Then pick a compatible release, or choose a file, and press **Flash**: the panel shows progress as it transfers, resets the controller so it starts the new firmware, and reads the running version back so you can confirm what landed. The **engineering override** is a bench-only escape hatch that lets a mismatched configuration version through for one flash; like the override on the Flash screen it re-arms itself off after a single use, so it can never quietly become the default.

The Live Dashboard

Watch the connected vehicle's live data — speed, motor, battery, temperatures, and I/O — on a set of gauges and charts. Available only with a vehicle connected.



What the dashboard shows. Open the **Monitoring** view to see the vehicle's live readings laid out as gauges and time charts: road and motor speed, torque, pack voltage and current, state of charge, temperatures, pedal and I/O values, and the current vehicle state. The vehicle only reports this data while the Configurator is connected and asking for it, so the dashboard comes alive on connect and costs nothing when no tool is attached.

It shows only what this vehicle has. The dashboard is built from your configuration: a gauge appears only for a subsystem you actually configured — the motors you defined, the battery, the chargers, and the pins and signals you assigned. You won't see a dead gauge for hardware this vehicle doesn't have, so the layout reflects the real vehicle in front of you.

Fresh data only. Each reading is watched for freshness. If the vehicle stops reporting a value — a sensor drops out, a module goes quiet, or the link is interrupted — its gauge greys out to a "no data" state rather than holding the last number, so a stale reading can never be mistaken for a live one.

Devices. The **Devices** view lists every contactor and every device in your project by the name you gave it, with the wire it is on (the pin alias and the connector pin, for example HSO_07 · K52). For each contactor it shows whether the vehicle has commanded it closed, what its feedback contact reports, the live level of the feedback pin, and the coil driver's own diagnostic. The card above them shows the precharge sequence: the high-voltage state, whether the bus is live, the pack and DC-link voltages, and the last precharge time.

Drive Mode. Any dashboard view can be popped out full-screen into a separate window as a **Drive Mode** display — a large, glanceable readout for a test drive on a second screen — while the main window stays on the configuration.

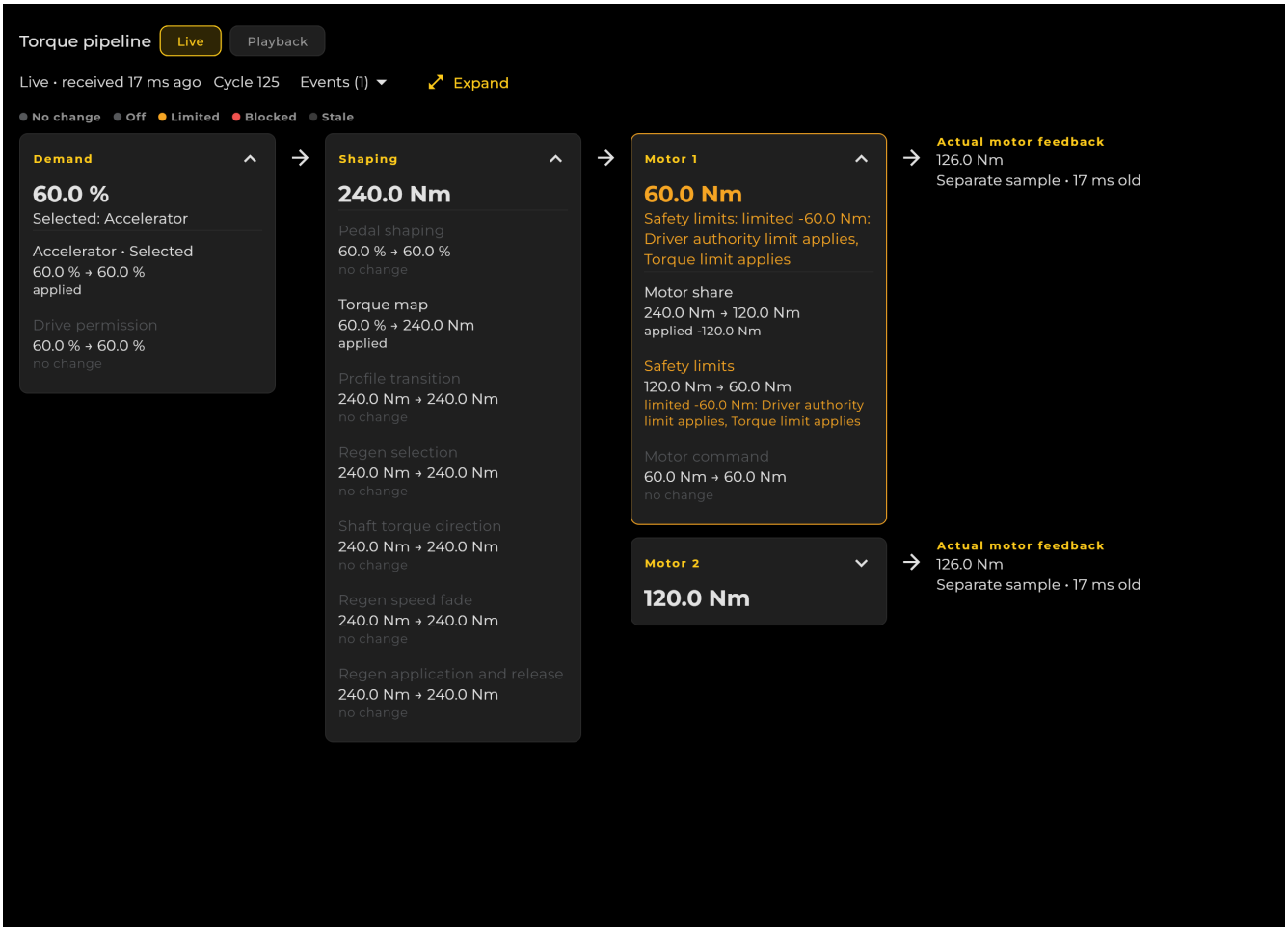
Recording. To capture a drive for later review, use the **Recorder** alongside the dashboard; it saves the live data into your project so you can replay it on these same gauges.

Run State. The Drive view groups **Vehicle State**, **Run State**, **HV State**, and **Inverter State** in its top banner. HV State shows the contactor sequence, such as Precharging, Live, or Opening. Inverter State shows the status reported by each configured inverter, such as Energised, Enabled, or Shutdown. Each reading shows **No data** when its own telemetry is too old. Only configured subsystems appear. **DIR** shows the selected P/R/N/D direction. **GEAR** shows a numbered transmission gear and appears only when gear sensing is configured. **Run State** sits below the vehicle state. This is the Bosch controller's own state: **PreDrive**, **Drive**, or **PostDrive**, with separate startup and final shutdown states. Bosch **Drive** means the controller is running; the vehicle can still be **Off** or **Ready**. **PostDrive** starts the vehicle's shutdown sequence. The controller can remain powered for a short time after the high-voltage contactors open. **Unavailable** means no state sample is available or the firmware does not provide it. **No data** means the last reading is too old to show as current.

Torque readings. The Drive view shows a **Torque** panel below the state banner. Each configured inverter has **Actual** motor-shaft torque and **Available forward** torque, in Nm. For a traction motor on inverter 1, **Request** shows the current motor-shaft torque command. Individual requests for other inverters are not available. **Front request** and **Rear request** show the sum of the motor-shaft requests on each configured axle. **Wheel estimate** applies the configured gear ratios to the torque requests; it is not a measured value. Signed values keep their direction. Each value becomes a dash when its data is missing, invalid, or too old; a dash does not mean zero torque.

Torque Pipeline

Trace each torque request through its sources, limits, and motor commands.



Open Monitor, then Drive, to inspect the torque pipeline. Read from left to right: demand sources, torque shaping, and a branch for each traction motor. Expand a group to see the input, output, and reported reason at each step. Demand before the torque map uses percent. Torque after the map uses Nm. Selected sources are marked. A skipped step has no evaluated value.

Select a step to see the captured speed, pedal authority, active profile, and motor limits where applicable. Open configuration switches to the Configuration tab and shows the related vehicle setting. These links are unavailable during playback. A limit can be normal control behavior. The view also identifies blocked requests and refused motor commands.

Each step carries a state word and a color, and the legend under the title shows each color. A step that changed the value shows in the normal text color. No change means that the step passed its value through. Off means that the feature is inactive, disabled by a script, or not evaluated. Limited shows the reduction in the warning color. Blocked shows the reason in the error color. A stale sample turns the whole view gray. Group titles use the accent color.

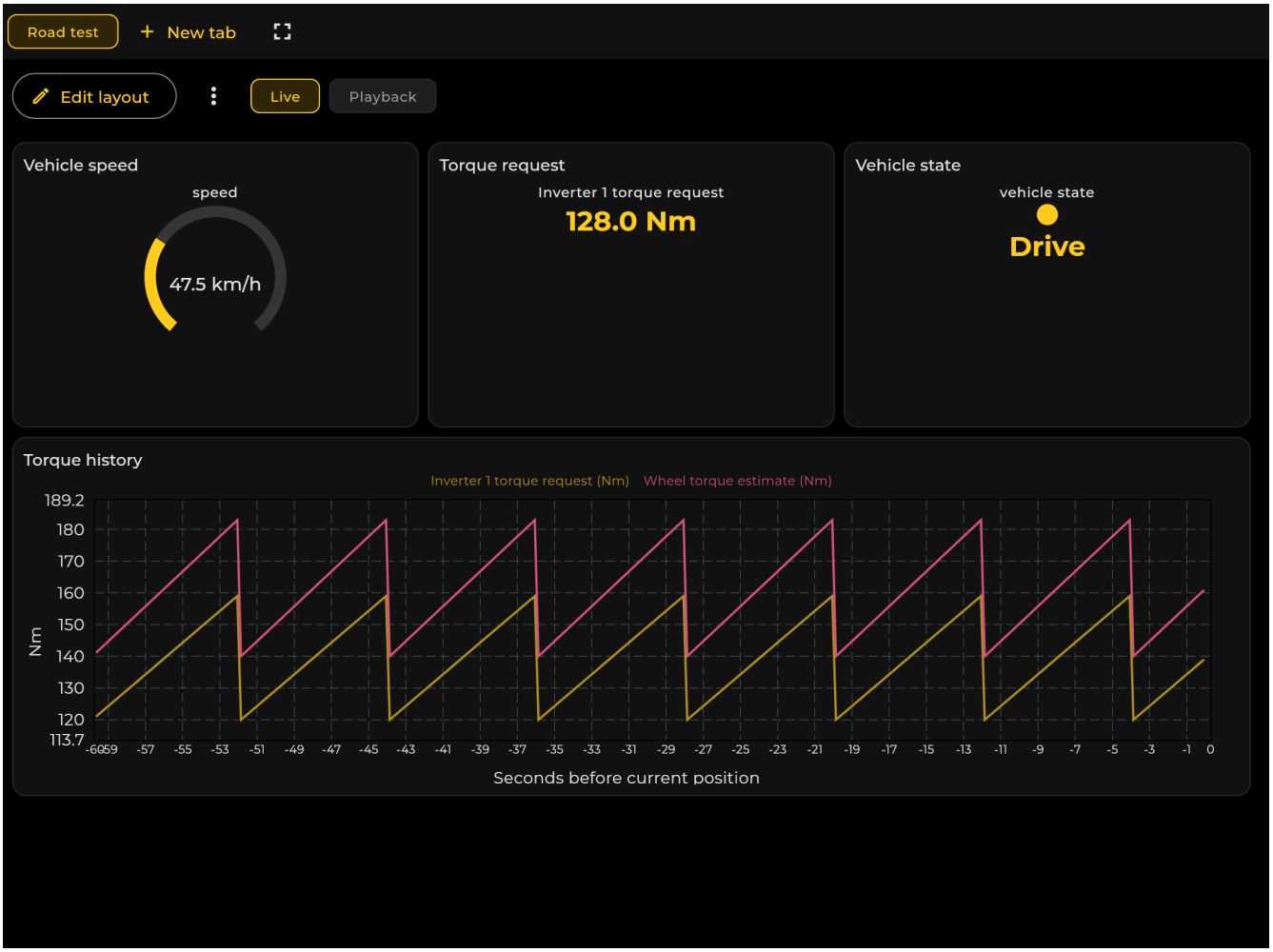
Actual motor feedback is a separate reading with its own age. Do not treat a command as proof of delivered torque. Stale values keep an explicit age label. Older firmware and recordings show Pipeline diagnostics unavailable when they contain no pipeline data.

Select Events to inspect a brief limit or block. Each event contains the whole pipeline from one control cycle. Events occur when a limit or block starts, changes reason, or ends. The controller keeps two events between reads. The app keeps the newest 128 received events. A history gap means that some events were lost. Motor feedback is unavailable for an event because it was not captured with that event.

Select Playback, choose a recording, and use Play, Pause, or the time slider. The pipeline uses recorded values at that position. Use Recorder to capture new drives. Other playback views share the recording and time position. Use the horizontal scrollbar in a narrow panel, or select Expand for a larger view.

Custom monitoring views

Build a monitoring tab from readings, cards, and charts, then use it with live data or a recorded drive.



Open **Monitoring → Custom views** and select **New tab**. Start with a blank view or copy the readings from a standard monitoring group into editable cards. Give the tab a name. Standard tabs remain available. Copies contain reading cards; the standard per-cell and CAN traffic tables remain in their own tabs.

Edit layout unlocks the grid. Select **Add widget**, search for a vehicle reading, a named device reading, or a published script value, and choose a number, gauge, bar, state indicator, chart, or card. A card groups related readings; its checkboxes let you hide individual members. A chart accepts up to six readings across two compatible unit groups. Script values are unitless unless their source provides a unit.

Set the widget label, compatible display units, gauge or bar limits, and warning threshold and color where available. A warning color is a display setting; it does not change the vehicle's fault limits. Chart windows range from one second to ten minutes. Drag a widget header to move it and its corner to resize it. With a widget focused, use arrow keys to move it, or Shift plus an arrow key to resize it. Narrow windows reflow the grid without changing the saved layout.

Select **Done** to accept all layout and widget changes. The project then has unsaved changes; use the normal **Save** command to write them to disk. **Cancel** discards the current layout draft. Tab options let you rename, duplicate, reorder, or delete a custom tab. Views travel with the vehicle project when you share it.

Select **Drive Mode** to open the same saved custom tabs in a full-window display. Each custom tab selects either **Live** or **Playback**. Playback uses one recording and one time position for every widget in the tab. Select a recording, play or pause it, change speed, or move the time slider. Other tabs that use playback share that recording and position. Live vehicle updates do not enter a playback view.

Live charts retain up to ten minutes while you switch monitoring tabs. History starts when a reading is selected for a saved chart and clears when you close the project. Up to 128 distinct chart readings can collect history in one project. A disconnect leaves a gap. This temporary history is separate from drives saved with Recorder.

A missing reading keeps its place and explains whether it was not received, not configured, not published, or not captured. A stale reading retains its last value with a visible age, while its chart shows a gap. Playback can show only data in the recording. New recordings preserve device identities; older recordings without that information cannot reliably display device-bound widgets. Current vehicle readings and captured script values remain available where their data is present.

Per-Cell Battery Monitoring

See inside the traction battery one cell at a time — voltage, temperature, internal resistance, and which cells are balancing — to spot imbalance, weak cells, and thermal hotspots. This live view appears only for a battery whose BMS reports per-cell data.

What the cell view shows. A traction battery is built from many individual cells wired together. Most of the time you watch the pack as a whole — total voltage, current, state of charge — but when you need to look deeper, the **Cells** view opens the battery up and shows every cell at once. It is where you chase an imbalance, hunt down a weak or lagging cell, check for a thermal hotspot, and see exactly which cells the pack is balancing right now.

When it appears. Cells is one of the **Monitoring** dashboard's views, offered only for a vehicle whose battery management system (BMS) reports data for each individual cell. Many packs report only pack-level numbers — total voltage and current, state of charge, and the hottest and coldest readings — and for those batteries the Cells view simply is not shown. If your vehicle has a cell-reporting BMS, the view is there whenever you are connected and monitoring.

If it is blank. Reporting every cell is almost always a feature you switch on in the BMS itself, not something set in the vehicle or the Configurator. So it is normal to have a cell-capable battery and still see an empty Cells view telling you the cell broadcast may be disabled — the battery can report cells but is not currently

sending them. Turn the per-cell broadcast on in the BMS's own settings and the view fills in. There is nothing to configure here; the view discovers the cells and their count on its own.

The voltage view — grid or bars. The main display is every cell's voltage. A headline across the top calls out the lowest cell, the highest cell, and the spread between them — the spread is the single clearest measure of pack balance, and it is flagged when it grows wide. Two layouts show the same cells: **Grid** lays them out as a coloured heat-map from coolest (lowest) to hottest (highest), so a run of weak or high cells stands out as a patch of odd colour; **Bars** draws each cell as a bar for reading heights side by side. In both, the lowest and highest cells are outlined so you can find them at a glance, and any cell the pack is actively balancing is marked with a dot. Hover a cell to read its exact voltage and its position in the pack.

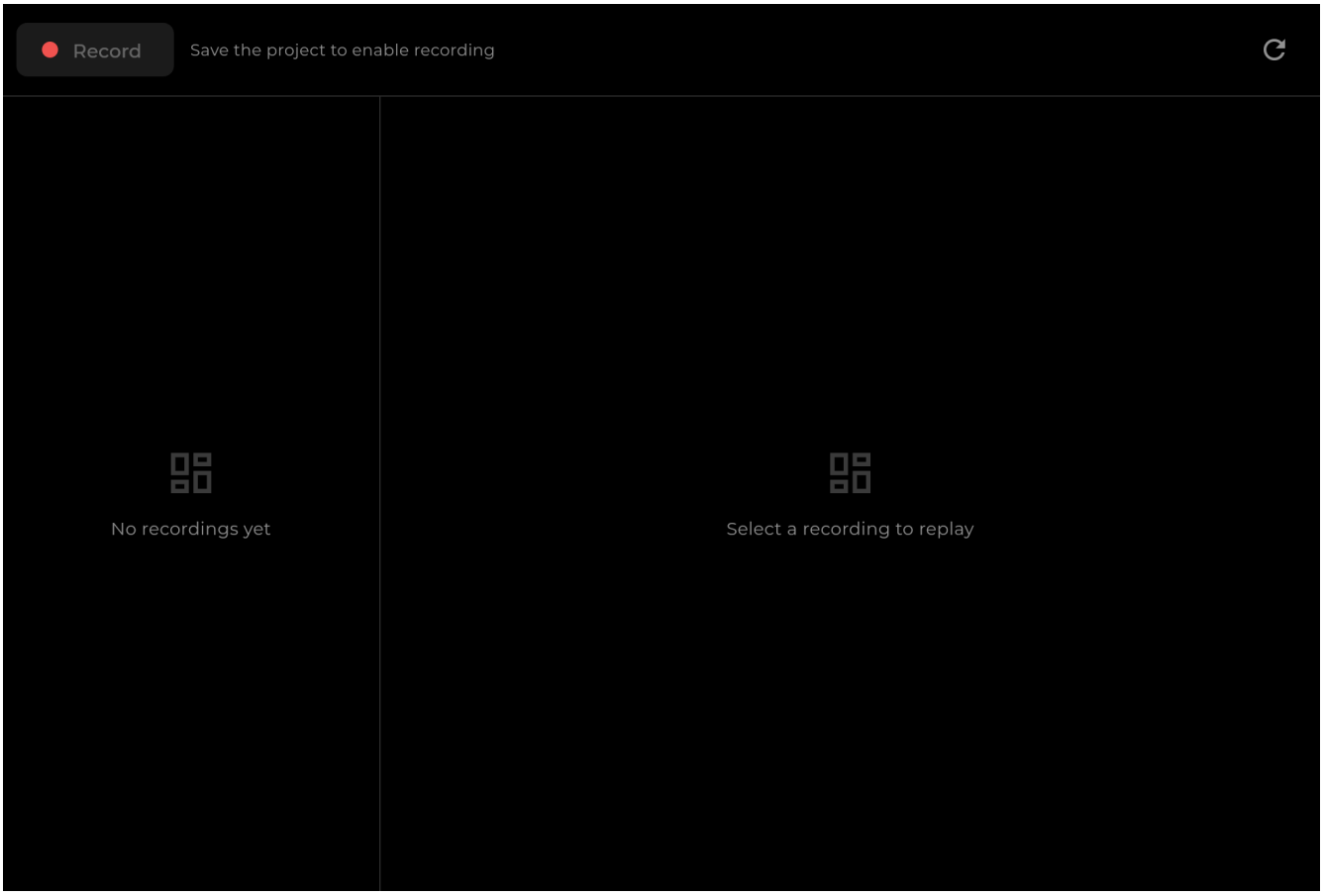
Temperatures. Beneath the voltages is a temperature strip — one square per temperature sensor in the pack, coloured coolest to hottest, with the overall range called out beside it. It is the fast way to catch a thermal hotspot: a sensor running hotter than the rest stands out immediately.

Resistance and open-cell voltage. Behind the **Resistance & Open-V** sub-tab are two more per-cell measures, shown when the battery reports them. Each cell's internal resistance — a rising value is an early sign of a tiring cell — and its open-cell voltage, both drawn as the same heat-map with a lowest / highest / spread headline. These are requested only while that sub-tab is open, so they add nothing until you look at them.

Live and light. Everything on this screen is live and pulled only while the view is on screen, so it costs the vehicle nothing when you are not watching. If a reading stops arriving it greys out rather than freezing on a stale number, so you always know whether you are seeing current data or none at all. Like every dashboard view, Cells can be popped out full-screen into its own window for a closer look.

Recording & Replaying Drives

Capture a live drive and play it back later. Record the vehicle's telemetry into your project while you drive, then scrub through it on the same gauges — and export it for other tools.



Recording a drive. Open the **Recorder** in the Monitoring view. With the project saved and a vehicle connected, **Record** starts capturing the live telemetry; the button becomes **Stop** and shows a running frame count so you can see it working. Press **Stop** to finish. Each recording is named with the date and time and is saved inside the project file, so it travels with the build.

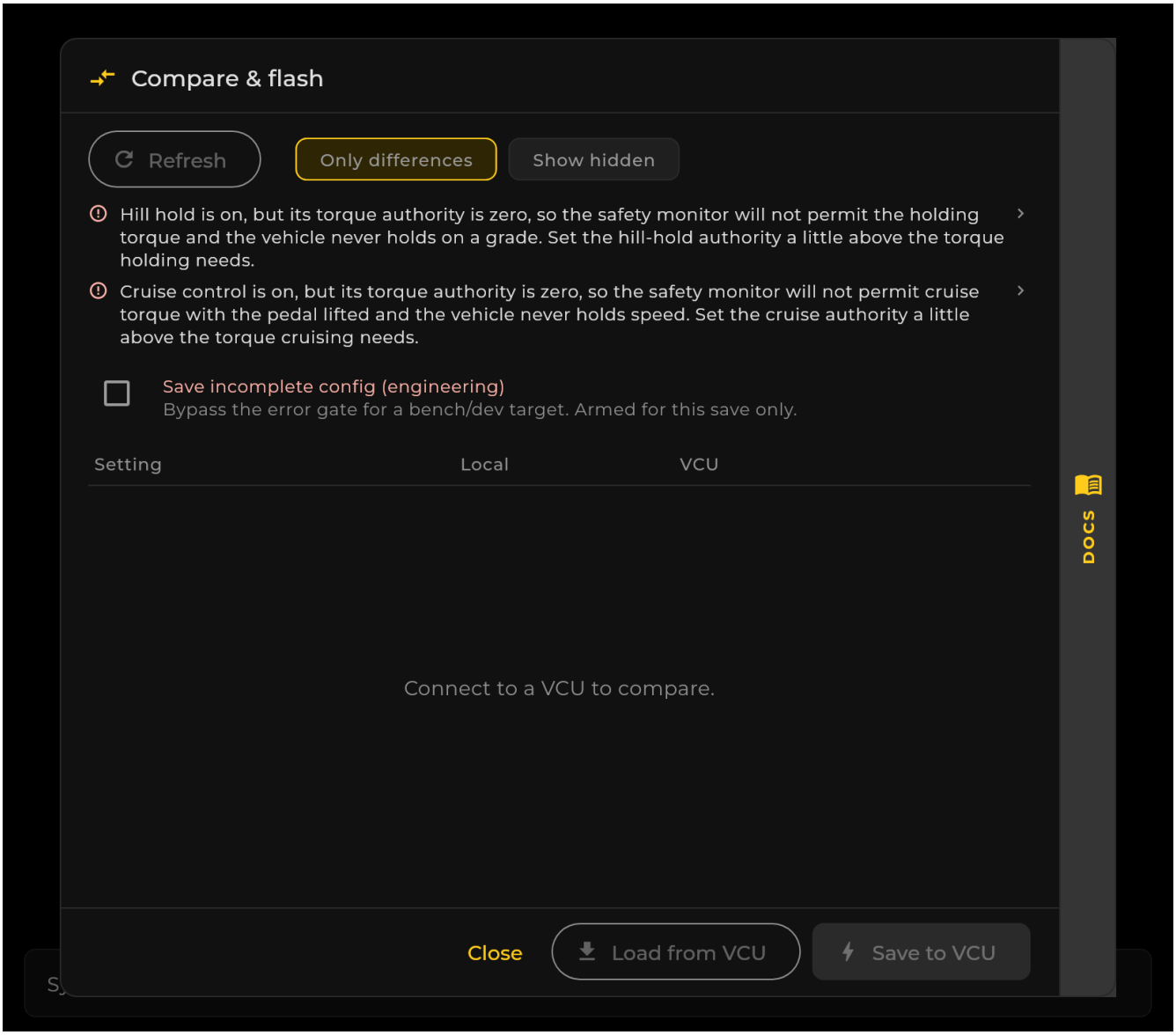
The recordings list. Every drive you capture appears in the list on the left with its frame count, and stays there across sessions. A very long drive is trimmed automatically so it can't grow the project file without bound, keeping the readings that actually move. Use the menu on any recording to open, export, or delete it.

Replaying. Select a recording to load it into the playback pane, which drives the very same gauges as the live dashboard — so a captured drive looks exactly as it did on the road. Play and pause, drag the scrubber to any moment, and change speed from a quarter-speed up to four times. Gauges grey out over stretches where a reading wasn't updating, just as they would live.

Exporting. From a recording's menu, **Export CSV…** writes a spreadsheet-friendly table of every frame, and **Export CAN log…** writes a standard CAN log — for analysis in another tool, or to archive a drive alongside the build. **Delete** removes a recording from the project for good.

Comparing Configuration with the Vehicle

Compare the configuration in your project against what the connected vehicle is actually running, field by field, and copy values either way. Read the live settings off the VCU, spot the differences, and push or pull individual values or the whole configuration.



Opening the comparison. With a vehicle connected and a project open, a **Compare VCU** badge sits in the toolbar. It reads the vehicle's live configuration and tells you at a glance whether the project is **In sync** or how many settings **differ**; tap it to open the comparison window. **Refresh** re-reads the vehicle at any time.

Reading the two columns. The window lists settings in three columns — the setting, its **Local** value (from your project) and its **VCU** value (what the vehicle is running right now) — and highlights every row where the two disagree. Leave **Only differences** on to see just what has changed, or turn it off to review everything; **Show hidden** reveals the settings the app normally manages for you.

Copying a single value. Each row has two arrows: push your project's value up to the vehicle, or pull the vehicle's value down into your project. Some settings are read-only on the vehicle and can only be pulled. A push takes effect on the running vehicle right away.

Copying everything at once. **Load from VCU** brings the vehicle's entire live configuration into your project — useful for capturing the exact setup of a vehicle you did not build here.

When the vehicle's settings were saved by older firmware. A vehicle that has been running an earlier firmware version stores its settings in an older format. Those cannot be compared setting-by-setting against your project — the same slot means something different on each side — so the window says so, and **Load from VCU** becomes **Migrate from VCU…**. Choosing it brings the vehicle's settings forward to the format this version of the app uses, and shows you what happened before anything is applied: how many settings carried over as they were, which ones are new in this firmware and are therefore sitting at their defaults, and which could not be brought across at all. Nothing enters your project until you accept it, and nothing reaches the vehicle until you **Save to VCU**. The settings that need a second look stay listed in the Issues pane afterwards — go through them before you save. If instead the vehicle is running *newer* firmware than this app understands, it will tell you so and stop; update the Configurator and try again.

Making changes stick. Pushing a single value changes what the vehicle is running at that moment, but it is temporary — a power cycle would lose it. **Save to VCU** stores the whole configuration on the vehicle so it survives a restart; use it once you are happy with the settings. Your project on this computer stays the record of truth, so save it too — after pulling values down, in particular. Comparing is for reconciling a vehicle against a project, not a substitute for flashing a finished build.

Faults

Browse every fault the VCU can report, and — with a vehicle connected — read which ones are currently active and clear its stored fault memory.

The fault catalog. The list shows every fault the VCU can report, whether or not a vehicle is attached. Each row carries a code, the vehicle system it comes from, how serious it is, the reaction the vehicle takes when it happens, and a plain-language description of what it means. This side is always available, so you can browse it offline to understand what a given fault stands for before you ever plug in.

Severity and reaction. Two columns tell you how much a fault matters. **Severity** runs from Info through Warning and Critical to Fatal, colour-coded so the serious ones stand out. **Reaction** is what the vehicle actually does about it — from simply noting the fault, through reducing what it will do, to dropping into a safe state or refusing to arm. The reaction is fixed for each fault; it is not something you set here.

Reading what is active. Connect to a vehicle and **Read live DTCs** asks the VCU which faults are currently set. Active faults are highlighted, tagged ACTIVE, and sorted to the top of the list; the rest drop away unless you turn off **Active only**, which brings the full catalog back with the inactive rows dimmed for context.

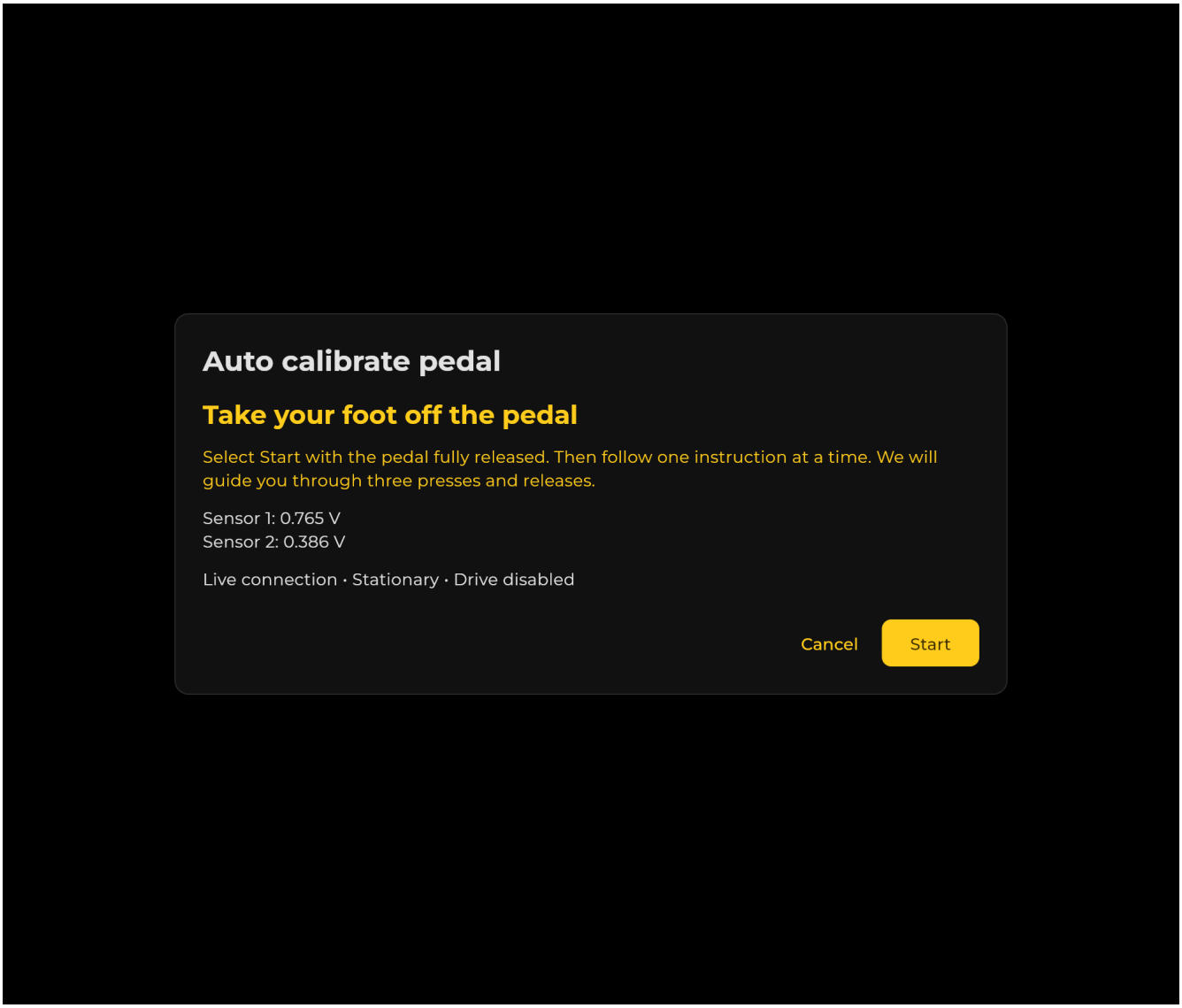
Codes the app does not recognize. If the vehicle reports a code the catalog does not list, it still appears — marked **(unmapped)** — rather than being hidden. This almost always means the vehicle's firmware and this build of the app are a version apart; align them and the code will name itself.

Clearing faults. **Clear DTCs** erases the vehicle's stored fault memory and immediately reads it back, so you see what truly remains. A fault whose cause is still present will come straight back — clearing is how you tell a stale record apart from a live problem, not a way to silence one.

The live Faults dashboard also reads cleared fault history from the VCU. It shows each retained fault, occurrence count, and last occurrence time since boot. This catches brief pedal faults even after the pedal is released. History lasts until the VCU restarts; older cleared records may be replaced when its memory is full. Clearing stored diagnostic codes does not erase this since-boot history. Older firmware shows a history-unavailable notice.

Auto calibrate pedal

Measure the pedal sensor endpoints over three guided cycles, review the result, and apply it to the project.



Open **Auto calibrate…** on the Accelerator pedal page. Select the single- or dual-sensor pedal type before opening the window. Connect to the live VCU, stop the vehicle, select Park or Neutral, and ensure drive is

disabled. The window checks vehicle speed and traction-inverter feedback; a Park indication alone is not enough. Missing or stale safety readings prevent calibration.

Select **Start** with the pedal fully released. Keep your foot off while the released position is measured. When prompted, press slowly all the way to the mechanical stop and keep it there until the instruction changes. When asked to take your foot off, release completely and wait for the next instruction. Repeat for the three guided cycles; do not cycle the pedal continuously. Each step advances after at least five fresh readings spanning one second or more. The sample counter shows progress; shared diagnostic polling can make a hold take several seconds. Live voltages show each required sensor. Press to the mechanical stop each time: software can recognize steady readings but cannot prove you reached the stop.

The capture stops if the vehicle moves, drive becomes enabled, the connection changes, readings become stale, or project settings change. A bad electrical reading, too little travel, inconsistent endpoints, or disagreement between the sensors rejects the result. Read the reason, correct the cause, and select **Retry**. Cancel discards the capture.

After three valid cycles, compare the current and proposed rest/full-travel voltages and inversion. The individual cycle measurements remain visible for review. **Apply** updates only these pedal settings in the project. Fault thresholds, maximum sensor disagreement, deadband and fault confirmation count stay unchanged. Save the project and flash its configuration with the normal commands. Both pedal control and the safety monitor use these sensor endpoints.

Configuring the Vehicle

What every setting in each configuration form does — organised the same way as the left-hand navigation. Each form is shown on a richly-configured example vehicle.

Supported hardware

The makes and models this firmware drives, by subsystem, named as the Configurator names them. A subsystem left at *None* is absent: that is not a fault, and the features that depend on it stand down. Each subsystem's own section below describes its settings.

Traction inverters

Up to five inverter slots, each with its own vendor. Every family below is driven through the same torque and speed commands, so the rest of the configuration does not change with the inverter.

Selection	Make and models	Protocol	CAN identifiers	Notes
H-Protocol (Genus)	Turntide, formerly Sevcon: Genus Gen4 and Gen5	H-Protocol over CAN with 29-bit identifiers	29-bit; CAN3 or CAN4 only	The VCU runs the energise and enable sequence and reads the 16-bit fault code.

Selection	Make and models	Protocol	CAN identifiers	Notes
Cascadia CM	Cascadia Motion, BorgWarner: CM200DX, CM200DZ, CM350DZ and CM350 SiC, with the iM-series motors	Cascadia RMS protocol, 11-bit or J1939 29-bit identifiers, base id configurable	11-bit or 29-bit, a protocol option; 29-bit needs CAN3 or CAN4	Oil-cooled and liquid-cooled variants are selected in the protocol options.
CANopen CiA-402	Any drive that implements the profile: Sevcon Gen4 and Gen5, Turntide, and other CiA 402 drives	CANopen CiA 402 drive profile, node-addressed	11-bit; any bus	The VCU runs the drive's boot script and starts it on the network.
Steinmetz	Steinmetz: MC0-500	Steinmetz CAN protocol v1.5, standard 11-bit fixed identifiers	11-bit; any bus	
H-Protocol (Kendal/ Gen6)	Turntide: Kendal v3 (Scafell and Hawkshead) and Gen6 Size 9	H-Protocol with a flat control word and node-addressed identifiers	29-bit; CAN3 or CAN4 only	Faults arrive as alert records.
Nissan Leaf	Nissan: EM61 (ZE0 and AZE0) and EM57 (ZE1) traction inverters, 80 kW and 160 kW	Nissan EV-CAN, with the vehicle controller and battery controller emulated by the VCU	11-bit; any bus	Pick the generation and the 160 kW variant in the protocol options.

Battery management systems

One battery management system per vehicle. It is the source of the charge and discharge limits, the pack voltage and the cell extremes.

Selection	Make and models	Protocol	CAN identifiers	Notes
Orion	Ewert Energy Systems: Orion BMS 2	Orion standard CAN map, with the per-cell broadcast for cell monitoring	11-bit; any bus	
Thunderstruck	Thunderstruck Motors: Thunderstruck BMS through its master control unit	Thunderstruck MCU CAN protocol v1.0	11-bit; any bus	
Static	No battery controller on the bus: Any pack without a CAN battery management system	None. Fixed charge and discharge limits come from the configuration, with optional derating by pack voltage.	No CAN link	Cell monitoring and pack diagnostics are not available.

DC-DC converters

High-voltage to 12 V converters, one or more slots. Two units of the same make on one bus need distinct message ids.

Selection	Make and models	Protocol	CAN identifiers	Notes
Dilong	Dilong: Dilong combined charger and DC-DC units, DC-DC side	Dilong OBC and DC-DC CAN protocol v1.8 at 500 kbit/s	29-bit; CAN3 or CAN4 only	

On-board chargers

AC chargers, one or more slots. The charger reports its AC input and its DC output, and the VCU commands the charge limits from the battery.

Selection	Make and models	Protocol	CAN identifiers	Notes
Dilong	Dilong: Dilong combined charger and DC-DC units, charger side	Dilong OBC and DC-DC CAN protocol v1.8 at 500 kbit/s	29-bit; CAN3 or CAN4 only	

Charge inlet controllers

DC fast-charge inlet controllers. The controller talks to the charging station. The VCU supplies the battery limits and sequences the contactors.

Selection	Make and models	Protocol	CAN identifiers	Notes
AMP (Auto Motive Power)	Auto Motive Power: AMP EVCC	AMP EVCC CAN interface 5.7	11-bit; any bus	

Power distribution units

Smart 12 V distribution with per-channel switching, current readback and fault reporting over CAN.

Selection	Make and models	Protocol	CAN identifiers	Notes
Current Labs	Current Labs: Current Labs PDU, 8 channels	Current Labs PDU CAN protocol	29-bit; CAN3 or CAN4 only	

Motion sensors

An inertial sensor for the vehicle dynamics features. The model decides which axes exist, and each axis maps to a CAN signal.

Selection	Make and models	Protocol	CAN identifiers	Notes
Bosch MM5.10	Bosch: MM5.10	Bosch sensor CAN frames	11-bit; any bus	Five axes, yaw rate and roll rate, no pitch rate.
Bosch MM5.10-R	Bosch: MM5.10-R, the ruggedized MM5.10	Bosch sensor CAN frames, same as the MM5.10	11-bit; any bus	Five axes, yaw rate and roll rate, no pitch rate.
Bosch MM7.10	Bosch: MM7.10	Bosch sensor CAN frames	11-bit; any bus	All six axes.
Generic (CAN)	Any inertial sensor that broadcasts on CAN: Any	Every axis is a CAN signal you map yourself	As set on the CAN signal it reads	All six axes are available when the sensor sends them.

Sensor and control inputs

Subsystems that read a sensor or drive an engine through a choice of wiring rather than a vendor protocol.

Subsystem	Selection	How it is read	Notes
Wheel speed	Frequency (tone wheel)	Tone-wheel sensors on the frequency inputs, one per corner	
Wheel speed	CAN	Per-corner speeds from CAN signals, for example from an ABS module	
Steering angle	Analog	One voltage on an analog input, centre at mid voltage, scaled to road-wheel angle	
Steering angle	CAN	A CAN signal scaled to road-wheel angle	
Suspension travel	Analog	Four analog inputs, one per corner, with a zero and a full-bump calibration each	
Suspension travel	CAN	Four CAN signals, one per corner, in millimetres	
Genset engine control	Generic	Per-function output channels for run or ignition, starter and throttle, each digital, PWM or CAN	With no engine control selected the genset is uncontrollable and the VCU only harvests its output.

Vehicle

Vehicle-wide settings: a name, how many drive motors it has, and the physical dimensions the control software needs to estimate grade, traction, and stability.

Identity

A name for this vehicle.

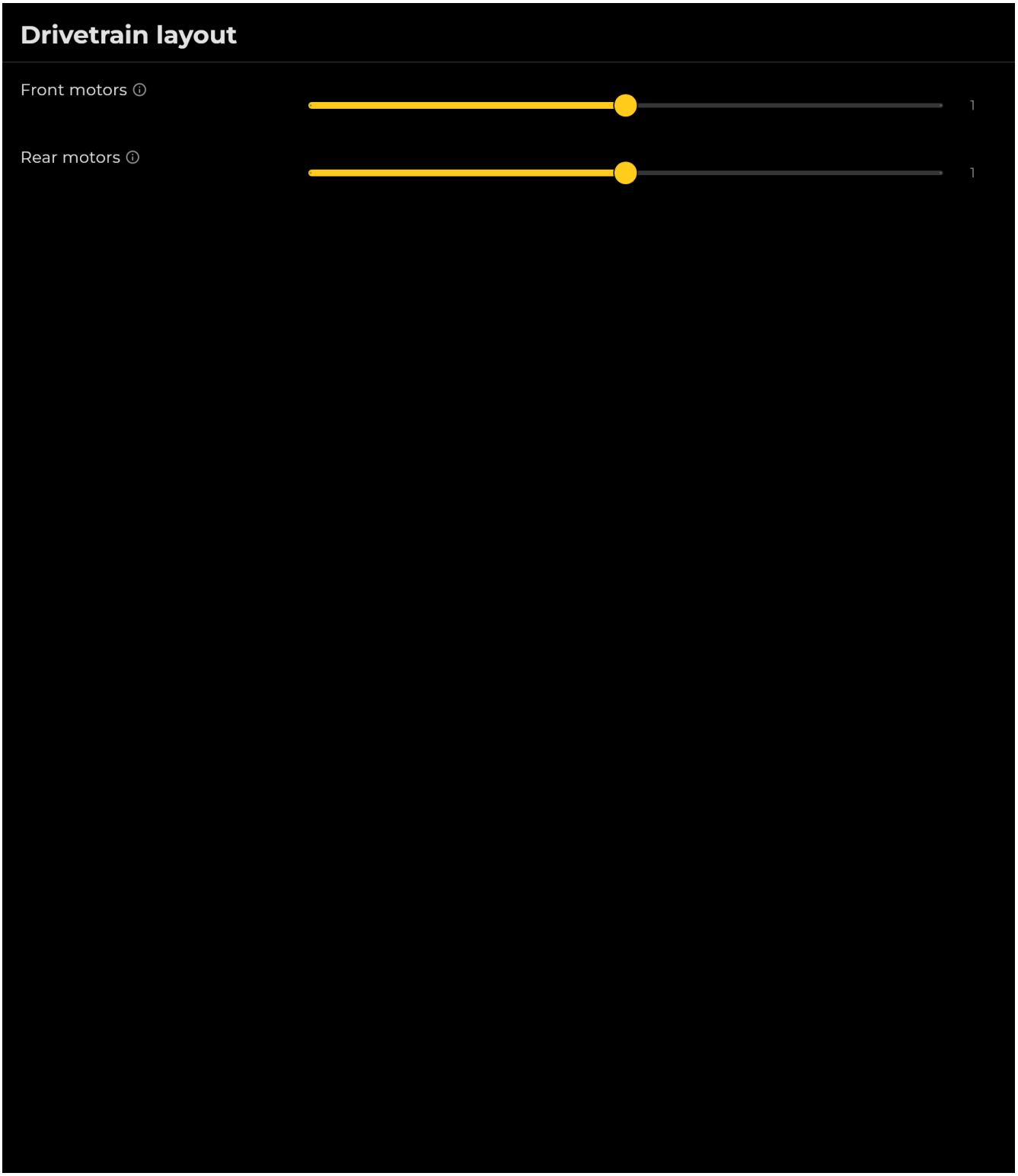


Settings

- **Vehicle name** — A name for this vehicle. Appears on the project tab and in readouts.

Drivetrain layout

How many drive motors are on the front and rear axles. This shapes which drivetrain features are available.

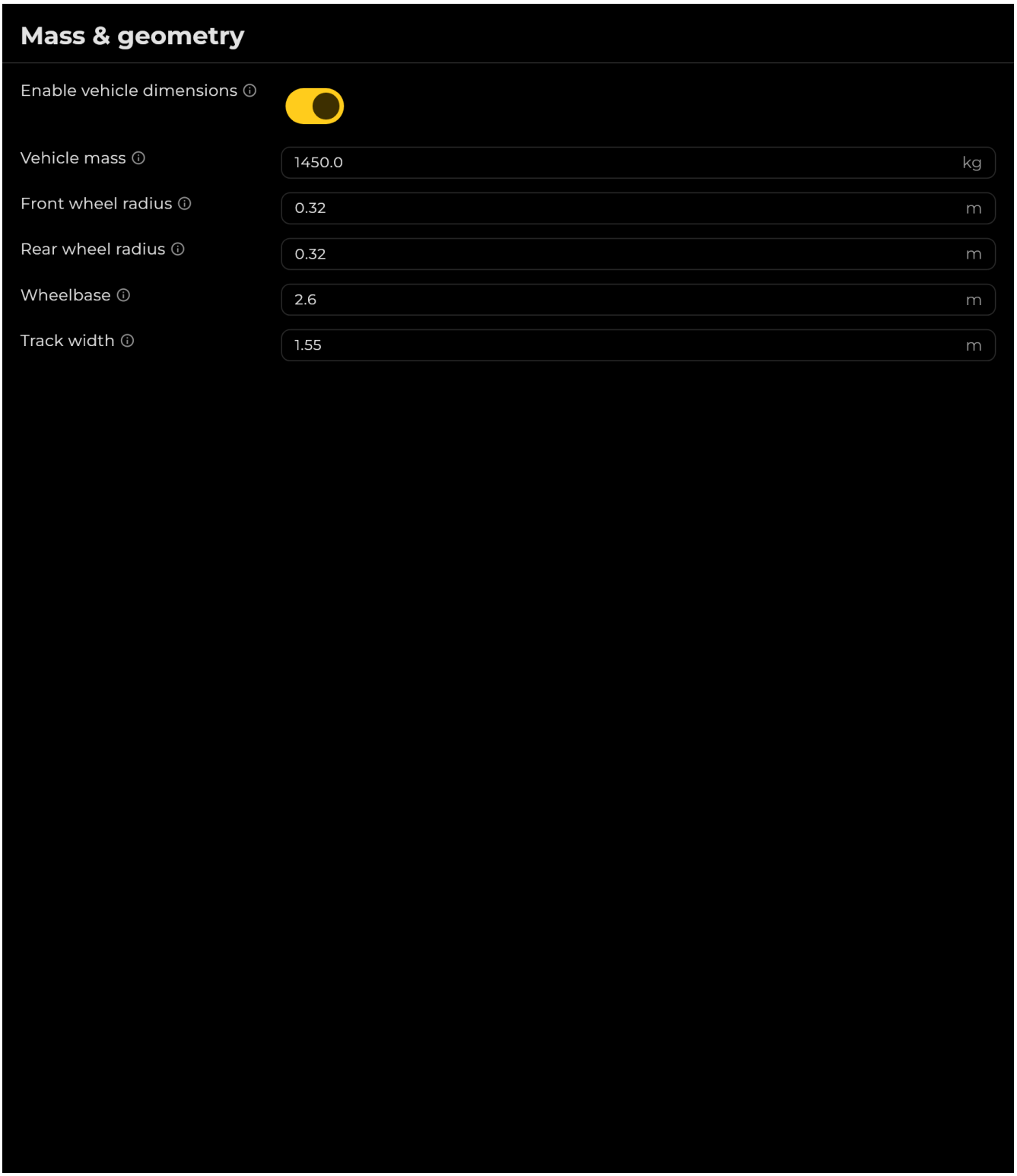


Settings

- **Front motors** — How many drive motors are fitted to the front axle.
- **Rear motors** — How many drive motors are fitted to the rear axle.

Mass & geometry

The vehicle's mass and key dimensions. These feed the grade, traction, and stability estimates.



Settings

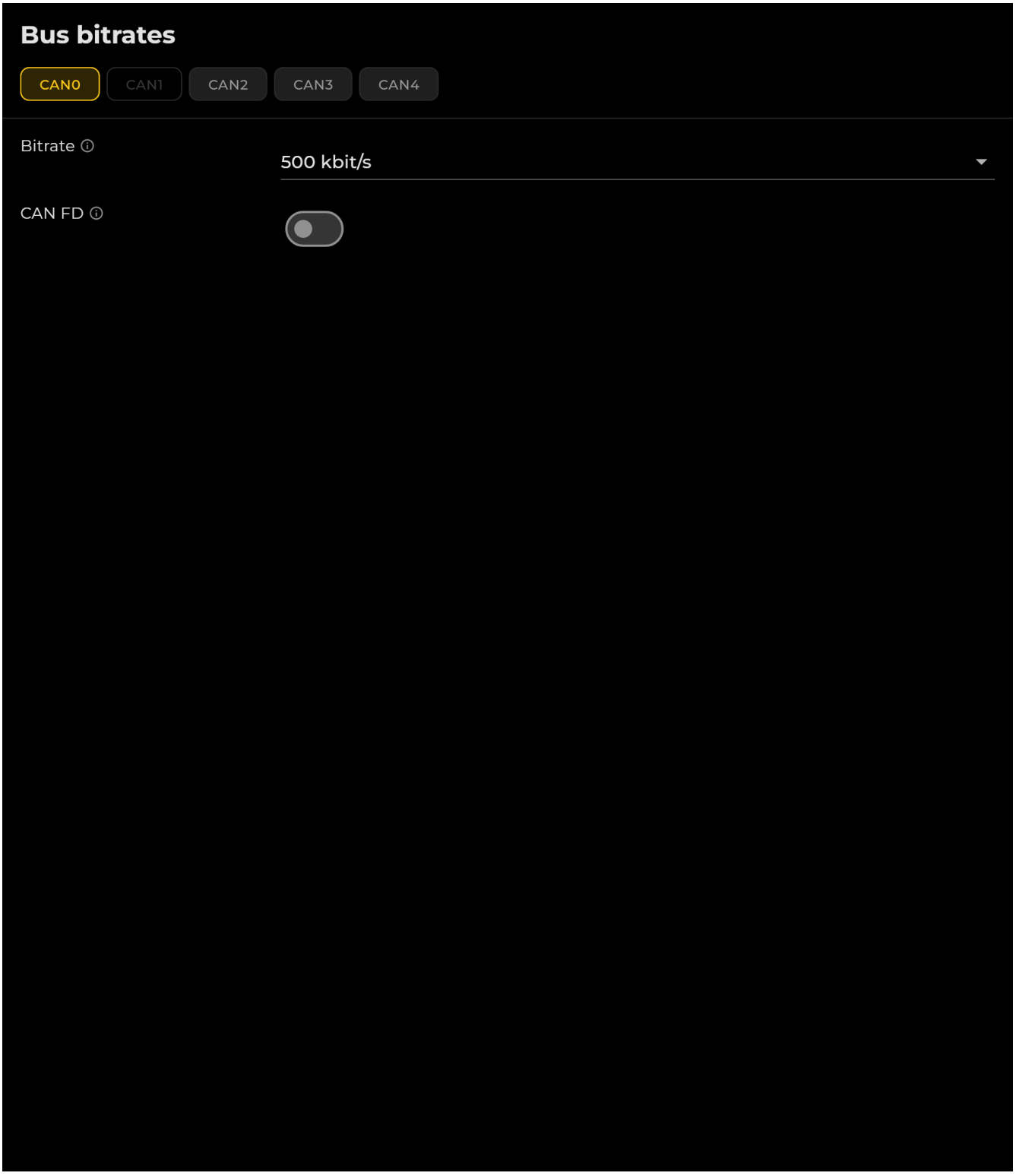
- **Enable vehicle dimensions** — Turn this on to enter this vehicle's mass and dimensions. While it is on, every dimension below must be filled in with a real, non-zero value before the configuration can be flashed, so a half-entered set can't ship by mistake. Leave it off for a vehicle that doesn't rely on the dimension-based grade, traction, and stability features — the values are then ignored and the fields are hidden.
- **Vehicle mass** (kg) — The vehicle's total mass, used by the dynamics and grade estimators.
- **Front wheel radius** (m) — Rolling radius of the front wheels.
- **Rear wheel radius** (m) — Rolling radius of the rear wheels. May differ from the front.
- **Wheelbase** (m) — Distance between the front and rear axles.
- **Track width** (m) — Distance between the left and right wheels on an axle.

CAN Buses

The CAN buses the controller uses to talk to motors, the battery, body modules, and diagnostic tools — plus the message databases and the script access rules for each bus.

Bus bitrates

The data rate of each CAN bus. Every device on a bus must run at the same rate.



Settings

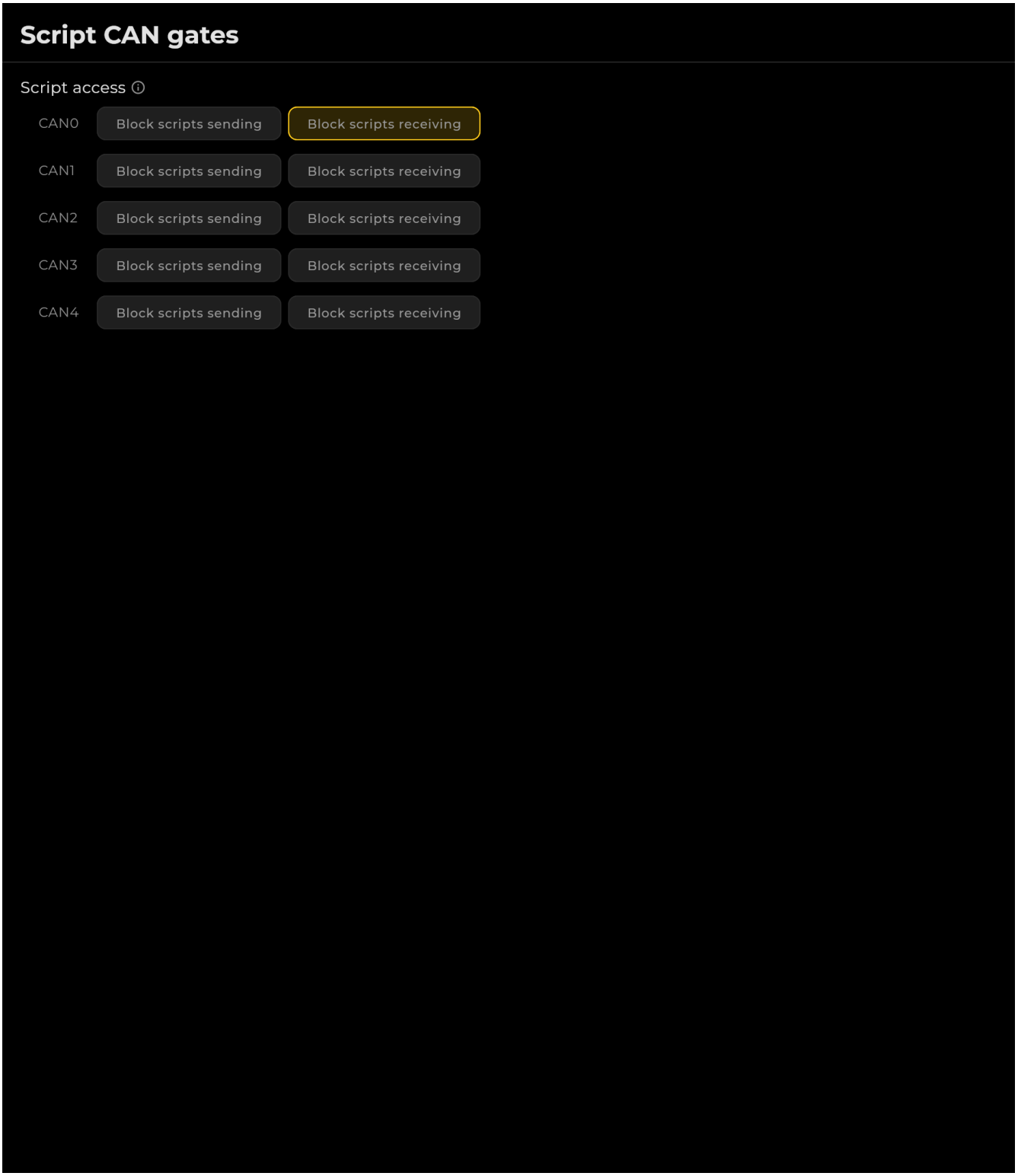
- **Bitrate** — The data rate this bus runs at. Every device on the bus must match.
- **CAN FD** — Enable for buses running CAN FD (flexible data rate) devices.

Outbound messages

Messages the controller sends on a CAN bus on a schedule or on demand. Each message is defined once here — its id, rate, and signals — and scripts and recipes reach it by name to fill in values, send it, or start and stop it. The copy icon on a row starts a new message from a copy of that one, named 'Name - Copy'.

Script CAN gates

Whether installer scripts are allowed to send and receive on each bus.



Settings

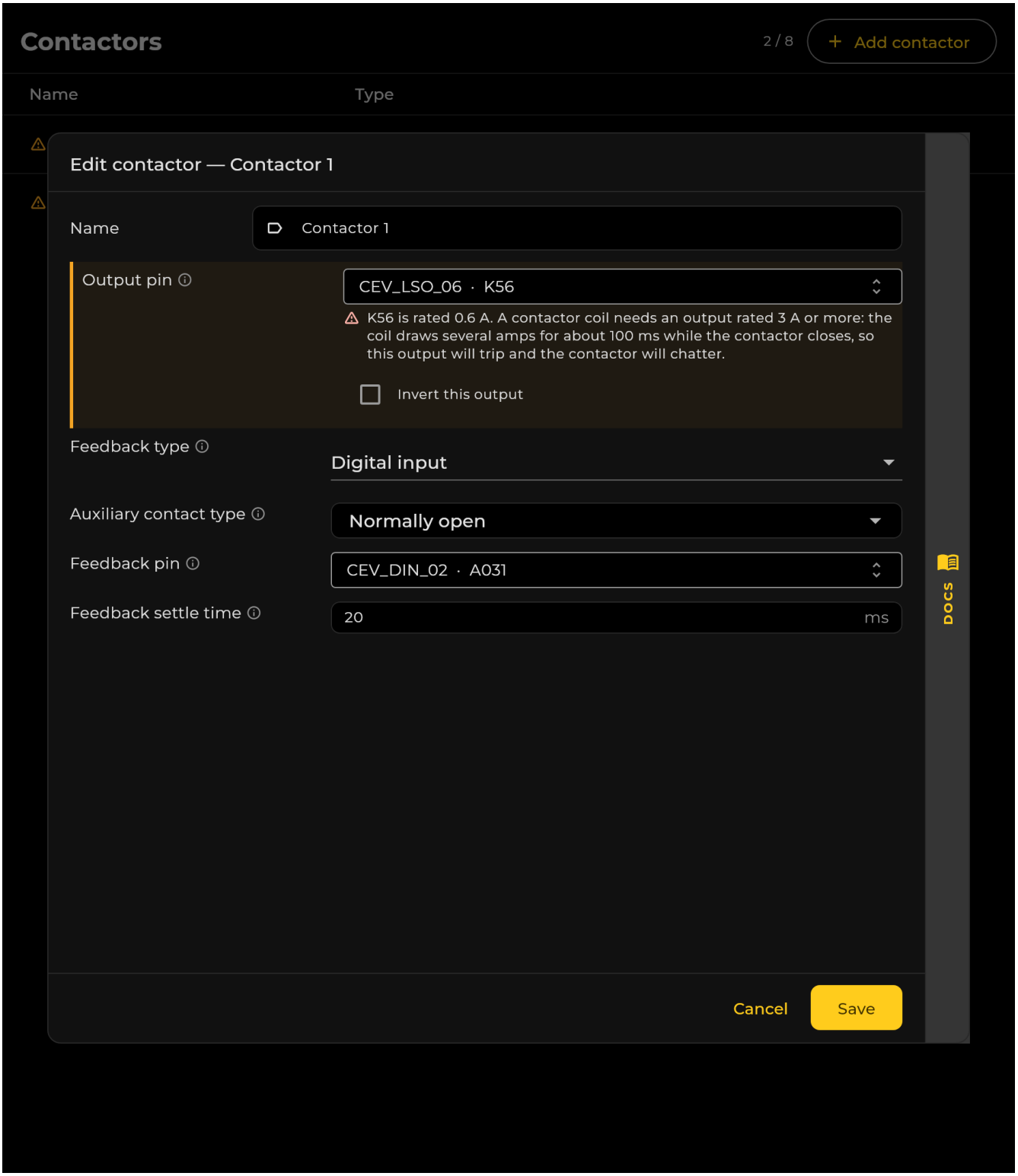
- **Script access** — Controls whether installer scripts may send or receive on each bus. By default a script may do both; tick a box to deny that direction.

HV Power

The high-voltage system: the contactors that connect the battery, the precharge sequence that brings the bus up safely, and the battery management system.

Contactors

The high-voltage contactors (relays) fitted to this vehicle — how each coil is driven and its optional feedback contact.



A high-voltage contactor is a heavy-duty relay that connects or isolates the traction battery from the rest of the high-voltage system. Each contactor has a coil: energize the coil and the contactor closes (connects); de-energize it and the contactor springs open (isolates). This Contactors system is the low-level "coil driver" for every HV relay on the vehicle. For each contactor it does exactly one job: drive an output pin to open or close the coil, and — if you wire up the optional feedback contact — watch a separate input that

mirrors the relay's real position, compare what was commanded against what actually happened, and log a fault if they disagree. It does **not** decide *when* to open or close anything, and it never opens a contactor on its own. That sequencing — precharge, going live, shutting down, and reacting to a fault — belongs to the high-voltage bring-up controller and the vehicle's state machine. This layer only obeys commands and reports what it sees.

There are eight contactor slots. Each one you fill describes a single physical relay: which pin drives its coil, and (optionally) which pin reads its feedback contact. Empty slots are simply skipped. To add another contactor with the same settings, press the copy icon on its row. The copy opens as a new contactor named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

Every contactor you configure needs a valid output pin for its coil, set under **Output pin**. If that pin isn't a real output, the whole slot is rejected at startup — it is never armed, never driven, and a configuration fault is logged so you can find it. Feedback is optional: if you set **Feedback type** to use a feedback contact, you must also give it a valid digital input under **Feedback pin**, or the slot is likewise rejected.

Pin rating. Every output pin has a continuous current rating, and a contactor coil draws far more than its holding current while it closes. A typical HV contactor pulls about 3.6 A for around 100 ms, then drops to a fraction of an amp. An output rated below that trips during the pull-in, retries, and the contactor chatters instead of closing, with short-to-ground and over-temperature faults on the pin. Use an output rated 3 A or more for a coil. If you pick a weaker output, a warning appears under **Output pin** in the contactor editor, and the issues list repeats it before a flash.

The coil and feedback pins of every configured contactor are claimed as safety-owned hardware. They are deliberately kept off the list of pins that installer scripts can drive — no script can toggle an HV relay.

How it works

At power-on, every valid contactor is driven to its **open (de-energized, isolated) safe state** before anything else happens. Nothing closes until the bring-up controller asks for it.

When a close or open command comes in, the controller switches the coil output on to close the contactor and off to open it. There is nothing to choose here: a coil is energized or it is not, and both output families — high-side and low-side — energize their load when the controller switches them on. High-side or low-side only decides *which end* of the coil is switched, and that follows from the output you picked.

The one case that differs is an **inverting driver** between the controller and the coil — an active-low relay board, an opto-isolated driver, or a line held up by a pull-up resistor. On that wiring, "on" has to leave the pin low. Tick **Invert this output**, shown directly under the output pin, and the controller flips it for you. Leave it clear for a coil wired straight to the output, which is the usual case. Getting it backwards means the relay does the opposite of what's commanded — and at power-on it would be driven closed instead of open, which is unsafe.

If **Feedback type** is set to none, that's the end of it: the contactor is driven but never checked, and its feedback always reads "unknown." No weld or stuck-open detection is possible without a feedback contact wired.

If **Feedback type** uses a feedback contact, the system reads the **Feedback pin** 100 times a second (every 10 ms) and converts what it reads into a true contact state using **Auxiliary contact type**. Say whether the auxiliary contact is **normally open** — open while the contactor is open, closing along with the main contacts, which is the usual kind — or **normally closed**. The controller already knows how the input pin is biased, so that is all it needs from you.

A real contactor takes tens of milliseconds to physically move after the coil is switched. Judging the feedback during that travel would produce false alarms, so right after any command change the feedback is held as "unknown" and not judged until the **Feedback settle time (ms)** has elapsed:

```
if (time since last command change) < Feedback settle time (ms):
    feedback = unknown  (not judged, no fault)
```

Once the settle window passes, the measured contact state is compared against what was commanded:

```
commanded open,  contact open    → OK
commanded close, contact closed  → OK
commanded open,  contact CLOSED  → WELDED      (relay stuck shut)
commanded close, contact OPEN    → STUCK OPEN  (relay failed to close)
```

A **welded** result means the contacts have fused and the relay won't open — a serious HV fault. A **stuck-open** result means the relay was told to close but didn't. Either one is logged the moment it is first detected; it is not re-logged every cycle while it persists, and if the contactor returns to agreeing with its command the internal state quietly clears. Effective detection time after a command is up to the **Feedback settle time (ms)** plus one 10 ms check.

The bring-up controller uses this feedback as a safety witness. Before it energizes any coil, it checks each contactor it cares about; if any reads **welded**, it blocks HV bring-up entirely and requests a safe state — it will not energize a system that already has a stuck-closed relay. The live contactor state (commanded position and feedback result for each relay) is also reported on the live telemetry the Configurator dashboard reads.

A note on how the coil is switched (generic output, high-side, or low-side): the controller works this out from the output pin you choose, so there is nothing to set. The actual electrical behavior of the coil output, and whether smart-switch diagnostics (open-load / short detection) are available, are determined by the physical pin you chose under **Output pin**, not by this setting. A plain output pin simply has no extra diagnostics. Choosing an output pin is what puts the contactor in the vehicle; use Remove on the contactor list to take one out.

Tuning notes

The only value you normally tune is **Feedback settle time (ms)**. Set it longer than the slowest your contactor takes to fully seat or release — typically a few tens of milliseconds, per the relay's datasheet. Too short and the system may judge the contact mid-travel and report a false **stuck-open** or **weld** while the relay is still moving. Too long and a genuine weld or stuck-open takes that much longer to be caught. Start a little above the manufacturer's stated operate/release time and lengthen it only if you see spurious feedback faults during normal open/close cycles.

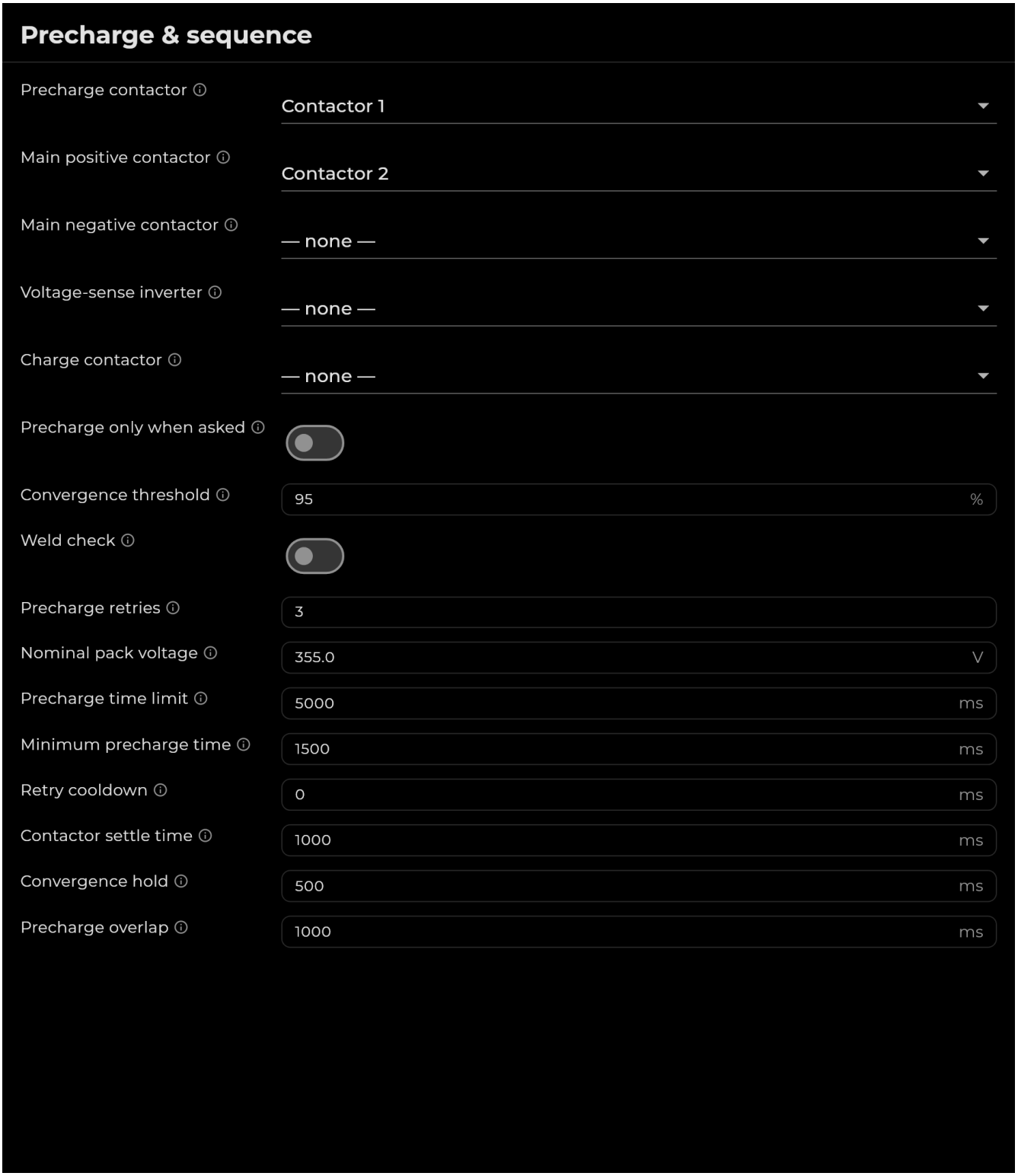
If a contactor logs a **weld** or **stuck-open** that you don't believe is real, check **Auxiliary contact type** first — the wrong contact type makes a healthy relay look permanently wrong (a closed relay reported as open, or vice versa). Then confirm **Invert this output** matches your driver wiring, since a wrongly inverted coil makes the relay's real position the opposite of what is commanded — and leaves it energized whenever the vehicle thinks it is open. Weld and stuck-open detection are only as trustworthy as a correctly wired and correctly polarized feedback contact; with no feedback contact wired, there is no detection at all.

Settings

- **Output pin** — The output that switches this contactor's coil. Choosing it is what puts the contactor in the vehicle — high-side or low-side follows from the output you pick. Use an output rated 3 A or more: a contactor coil draws several amps while it closes, and a weaker output trips and chatters. The editor warns under a pin that is too weak.
- **Feedback type** — Whether an auxiliary contact reports the contactor's real state. Choose a digital input if your contactor has an aux contact; otherwise leave it as none.
- **Auxiliary contact type** — Whether the contactor's auxiliary contact is closed or open when the contactor itself is open. Most auxiliary contacts are normally open — they close along with the main contacts.
- **Feedback pin** — The digital input wired to the contactor's auxiliary contact.
- **Feedback settle time** (ms) — How long to wait before trusting the feedback contact after a command.

Precharge & sequence

How the high-voltage bus is brought up safely: which contactors to use, the convergence target, and the timing of the precharge sequence.



Precharge is how the vehicle wakes up its high-voltage bus without slamming the full pack voltage onto the inverters' input capacitors. Connecting the pack directly would draw a huge inrush current that can weld a contactor shut or damage hardware. Instead the controller first brings the bus up gently through a precharge resistor, waits until the bus has charged to nearly pack voltage, and only then closes the main contactor across what is now an almost-zero voltage difference — so there is no inrush. This sequence

runs every time the vehicle is asked to bring up high voltage, and it runs in reverse (an orderly shutdown) when the vehicle leaves a powered state.

What it needs

The sequence drives a set of high-voltage contactors. Two are mandatory: the **Precharge contactor**, which puts the precharge resistor into the circuit, and the **Main positive contactor**, which connects the bus once it is charged. A third, the **Main negative contactor**, is optional — leave it unset for a single-main topology, or assign it for a two-contactor pack that switches both the positive and negative sides. The three must be distinct outputs; if a required contactor is missing, or two of them point at the same output, the controller refuses to bring up high voltage at all and reports a configuration fault rather than risk an unsafe sequence.

It also needs two voltage readings. The first is the **pack voltage** (the source it is charging toward): the live battery pack voltage is used when the battery is reporting a valid number, and the **Nominal pack voltage (V)** you enter is the fallback used only when no live pack voltage is available. The second is the **bus voltage** (how far the bus has charged so far), read from an inverter. **Voltage-sense inverter** selects which inverter provides that reading: leave it at none to use the most conservative (lowest) reading across all healthy inverters, or pin it to one specific inverter to trust only that one.

If the controller cannot get *both* a pack-voltage source and a bus-voltage reading, it falls back to an open-loop, time-only precharge (described below) and flags that it is running without voltage sensing.

How it works

When the vehicle requests high-voltage bring-up, the controller runs this sequence:

1. **Pre-existing weld gate.** Before energizing anything, it checks each contactor's feedback. If a contactor that should be open is already reporting closed, that contactor is welded shut — the controller stops immediately with a weld fault and never applies power.
2. **Active weld check (optional).** With **Weld check** enabled, the controller runs a three-step probe before precharging: it watches the bus voltage while it momentarily closes one main contactor at a time, confirming the bus stays dead each time it should be. At every step, if the bus voltage rises to or above the weld threshold while the bus is supposed to be unpowered, a contactor is welded and the sequence stops:

   ```
   weld detected if  bus voltage  >=  (Weld detect threshold / 100) x pack voltage
   ```

   Weld detect threshold (%) sets that sensitivity — a lower number trips on a smaller leak of voltage. The active weld check needs the **Main negative contactor** and a working bus-voltage reading to be meaningful; without a negative contactor it cannot run, so it disables itself and notes that the weld check was unavailable (bring-up still proceeds). An open-loop attempt (no voltage sensing) also skips it. The pre-existing weld gate in step 1 always runs regardless of this setting.
3. **Precharge.** The controller closes the **Main negative contactor** (if present) and the **Precharge contactor**, putting the resistor in circuit so the bus charges up through it. It then watches the bus climb toward pack voltage. Convergence is reached when:

```
bus voltage  >=  (Convergence threshold / 100) x pack voltage
```

Convergence threshold (%) is the target ratio and is held between 80 and 98 — values outside that band are pulled back in. To avoid going live on a noisy or marginal reading, convergence must hold *continuously* for the **Convergence hold (ms)** time; any dip resets that timer and the hold starts over. The controller goes live only when all of these are true:

```
convergence has held for at least  Convergence hold
AND  elapsed precharge time  >=  Minimum precharge time
AND  elapsed precharge time  <   Precharge time limit
```

Minimum precharge time (ms) is a floor — even once the bus has converged, the controller waits at least this long before closing the main. It starts at 1.5 seconds so that every bring-up runs the full sequence and *sounds* the same, which makes a healthy start something you can recognise by ear rather than by instrument. Setting it to 0 lets precharge finish as soon as the bus converges, and is also what enables the charged-bus fast path below. **Precharge time limit (ms)** is the abort ceiling, *not* a go-live trigger: if the bus has not converged by this time, the attempt is aborted.

4. **Go live.** The controller closes the **Main positive contactor** across the now near-zero voltage gap (no inrush). The **Precharge contactor** stays closed for the **Precharge overlap (ms)** so the main has physically pulled in before the resistor leaves the circuit, then opens. A contactor takes tens of milliseconds to pull in, so without the overlap the bus would float between the two. The bus is live as soon as the main closes, and the vehicle is cleared to move on to ready/drive. The duration of the precharge phase is recorded so you can read it back as a health indicator.
5. **Teardown.** When the vehicle leaves its powered states, the controller opens the **Main positive contactor** first (and the **Precharge contactor** with it, if the overlap is still running), waits the **Contactor settle time (ms)** for the contacts to physically seat, then opens the remaining contactor(s), and the bus returns to off.

What starts it. The controller starts this sequence on its own each time it powers up with the ignition. About a second after start-up, when the battery and the inverters had time to report, it asks the vehicle to go to Ready, and the steps above run. No recipe or script is needed for that. Turn on **Precharge only when asked** when a recipe or script decides when the vehicle goes live, for example one that waits for a brake press before it wakes the high-voltage system. With that setting on, nothing happens until a recipe asks for Ready. The automatic request is made once per power-up. If a recipe parks the vehicle again after a fault, it stays parked until the ignition is cycled or a recipe asks again. On a vehicle with no high-voltage contactors configured, nothing is asked for.

Charge contactor. Some vehicles put a dedicated contactor between an on-board charger and the battery, so the charger can be connected for charging and isolated the rest of the time. If yours does, assign it as the **Charge contactor** and set that charger's **Charge path** to use it. It is not part of the sequence above: it closes on its own while the vehicle is AC charging, the main contactors stay open, and the drive bus never comes up to charge. There is no precharge before it closes — a charger's output has nothing like the inrush an inverter's input capacitors do. Leave it unset unless a charger asks for it.

Retries. If a precharge attempt hits the **Precharge time limit (ms)** without converging, **Precharge retries** decides what happens next. With retries left, the controller opens all contactors, waits the **Retry cooldown (ms)** with everything open (resting the resistor so it does not overheat), then tries again. Total attempts equal **Precharge retries** plus one; once they are exhausted, it stops with a timeout fault.

Fast path for a charged bus. If the bus is already charged to the convergence target when bring-up starts (for example, immediately after a brief drop), and **Minimum precharge time (ms)** is set to 0, there is no voltage gap to bridge and therefore no inrush — the controller skips the resistor entirely and goes live at once.

Open-loop (time-only) mode. If there is no bus-voltage reading or no pack-voltage source, the controller cannot judge convergence, so it charges for the full **Precharge time limit (ms)** and then goes live purely on time. In this mode the convergence test, the charged-bus fast path, and the live under-voltage monitor are all unavailable, and the controller flags once that it is running without voltage sensing.

Live under-voltage watch. Once live, and only when it has valid voltage readings, the controller watches for the bus suddenly collapsing — an unexpected loss of high voltage. It faults and opens the contactors if:

```
bus voltage  <  (Convergence threshold / 2 / 100) x pack voltage
```

(about half the convergence ratio — for a **Convergence threshold (%)** of 95, roughly 47% of pack voltage).

Faults and cancellation. Any fault drops every contactor open immediately — that is always the safe fallback, and the controller will do it on its own authority. A high-voltage fault stays latched until the vehicle is fully keyed off; it does not clear by itself mid-cycle. Cancelling bring-up cleanly (the driver or another system aborts before the bus goes live) is *not* a fault — the contactors simply open and the controller returns to idle, ready to try again.

Contactor settle time note: this is the mechanical seat time used between the weld-check probe steps and during teardown. It is separate from each contactor's own feedback debounce, which is configured with that contactor.

When the **Voltage-sense inverter** is left at none and two or more inverters report bus voltage, the controller also cross-checks them: if their readings disagree by more than 12.5% of the highest reading, it flags a witness-disagreement warning. This is report-only — it does not change the sequence, which always uses the most conservative (lowest) reading.

Tuning notes

Set **Nominal pack voltage (V)** to your pack's true nominal voltage so the fallback is accurate when the battery is not yet reporting. Start **Convergence threshold (%)** around 90–95: higher means the main closes across a smaller voltage gap (gentler, but precharge takes longer), lower closes sooner with a bigger gap. **Convergence hold (ms)** should be long enough to ride out reading noise — a few tens of milliseconds is typical; raise it if you see premature go-live on a flickering reading. **Precharge overlap (ms)** only needs to cover your main contactor's pull-in time with margin; the default suits any common contactor, and values below 20 ms are raised to 20 ms. Size **Precharge time limit (ms)** comfortably longer than a healthy precharge actually takes for your pack and resistor; if it is too tight you will get nuisance timeouts

and retries. Leave **Minimum precharge time (ms)** at its default if you want every bring-up to take the same, audible path; set it to 0 if you would rather the vehicle go live as soon as the bus converges and get the charged-bus fast path with it. For **Weld detect threshold (%)**, lower is more sensitive to a leaking or partially-welded contactor; set it low enough to catch a real weld but above your bus's normal residual/ leakage so it does not false-trip. If precharge is occasionally slow on a cold or large pack, allow one or two **Precharge retries** with a **Retry cooldown (ms)** long enough to let the resistor cool between attempts. Leave the **Main negative contactor** assigned if you want the active **Weld check** to run, since it needs both mains to do its job.

An explicit Off or Accessory request cancels automatic precharge until the next controller boot. This includes a request during the first second, even if the vehicle is already Off. A later explicit Ready request can still start precharge.

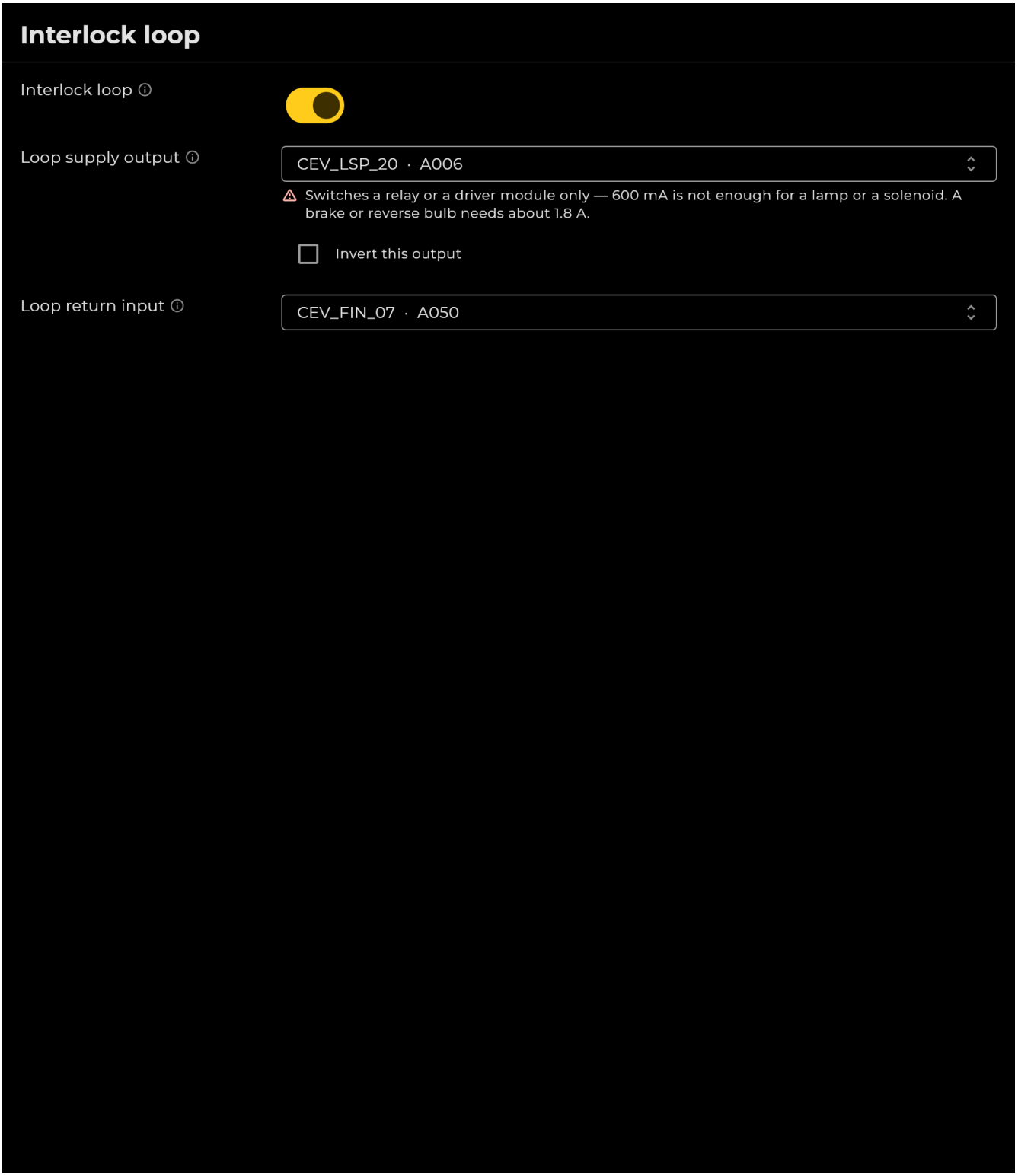
Settings

- **Precharge contactor** — The contactor that energizes the precharge resistor during bring-up.
- **Main positive contactor** — The main positive (battery+) contactor.
- **Main negative contactor** — The main negative (battery-) contactor, if fitted.
- **Voltage-sense inverter** — Which inverter's DC-link voltage witnesses precharge convergence. Leave unset to require every inverter to agree (the most conservative choice).
- **Charge contactor** — The contactor that connects an on-board charger to the battery, on a vehicle wired that way. Leave it unset unless a charger's "Charge path" is set to use one. It closes on its own for charging and the main contactors stay open, so the drive bus never comes up to charge.
- **Precharge only when asked** — Off, the default: the controller brings the high-voltage bus up on its own about a second after it powers up with the ignition, with no recipe or script involved. On: it waits until a recipe or script asks the vehicle to go Ready. An explicit Off or Accessory request cancels automatic precharge until the next controller boot. A later explicit Ready request can still start precharge.
- **Convergence threshold** (%) — Precharge is complete once the DC link reaches this percentage of the pack voltage.
- **Weld check** — Run an active contactor-weld pre-test before closing the main contactors.
- **Weld detect threshold** (%) — During the weld test, a welded contactor is declared if the DC link already sits above this percentage of pack voltage.
- **Precharge retries** — How many times to automatically retry precharge if it fails to complete.
- **Nominal pack voltage** (V) — The assumed pack voltage when no battery management system reports one.
- **Precharge time limit** (ms) — Abort precharge if it has not converged within this time.
- **Minimum precharge time** (ms) — An optional shortest precharge duration. Zero lets precharge finish as soon as it converges.
- **Retry cooldown** (ms) — The wait between precharge attempts, to let the precharge resistor cool.
- **Contactor settle time** (ms) — How long to wait for a contactor to mechanically close before relying on it.

- **Convergence hold** (ms) — The DC link must stay converged this long before the main contactor closes.
- **Precharge overlap** (ms) — How long the precharge contactor stays closed after the main positive contactor is told to close, in milliseconds. A contactor takes tens of milliseconds to physically pull in, so this keeps the precharge resistor in circuit until the main has seated instead of letting the bus float in between. Values below 20 ms are raised to 20 ms.

Interlock loop

The safety circuit that runs through every high-voltage connector and cover, so the vehicle drops high voltage when one of them is opened.



A high-voltage interlock loop is a small low-voltage circuit that runs in series through every high-voltage connector, service disconnect, and enclosure cover on the vehicle. Each of those parts carries a pair of tiny pilot contacts that make up before the high-voltage contacts do, and break before them. Thread all of those pilot contacts together, end to end, and you get one circuit that is closed only while every single high-

voltage joint on the vehicle is properly made up. Open any connector, lift any cover, pull the service disconnect, and the circuit breaks.

When this system sees that break, it raises a high-voltage fault, and the vehicle opens its main contactors. The bus goes dead. That is the whole point: somebody who has just opened a connector must not find live high voltage behind it.

This is a second line of defence, not the first one. The primary protections are the enclosures themselves, the insulation, the isolation monitoring, and the service disconnect. The interlock exists to catch the case where one of those has been opened while the vehicle is live. The strongest arrangement of all is one where the interlock loop physically breaks the contactor coil supply, so the contactors drop out through hardware with no software involved at all — and the VCU watches the same loop as an extra, independent set of eyes. If your battery pack or a separate interlock module already works that way, leave **Interlock loop** turned off here and let it report over CAN.

There are only three settings, and that is deliberate. Interlock loops have no agreed signalling standard — every manufacturer does it slightly differently, and the devices you will wire into the loop are almost all simple pass-throughs that carry whatever the loop's owner puts on them. So there is no correct pulse rate or tolerance for you to look up and type in. Everything except *whether there is a loop* and *which two pins it uses* is decided for you from the pins themselves.

Wiring it

The loop needs a supply and a return. Set **Loop supply output** to the output that feeds it, and **Loop return input** to the input the far end comes back to.

If something else already powers the loop, leave the supply output empty. The return is still watched, and the vehicle still drops high voltage when the loop breaks — it simply is not the thing energising it.

The supply and the return have to match each other. Every input pin rests at a level of its own when nothing is connected to it: some rest high, some rest low. That resting level is what makes a broken loop visible, because a healthy loop pulls the pin the *other* way. So an input that rests high needs a low-side output to pull it down, and an input that rests low needs a high-side output to pull it up. Pair them the wrong way round and the input reads the same either way — a cut loop would look perfectly healthy. The app checks this pairing and refuses to let you flash it, naming both pins and telling you which kind of output you need. The vehicle refuses the same combination on its own, and will not bring high voltage up until it is fixed.

Choosing the return input is worth a moment. Land the loop on a frequency input where you have one to spare. The vehicle then sends a pulse down the loop and checks that the same pulse comes back, instead of just checking that *something* comes back. A broken loop returns nothing either way — but a loop that has been shorted to ground, shorted to battery, or quietly bridged with a jumper wire across a connector still looks healthy to a simple check, and does not survive the pulse. You do not have to configure any of that; it happens automatically when the pins can support it.

Loop current and contact wetting. The loop carries only a few milliamps — around five, which sits at the bottom of the range production vehicles use. That is plenty to keep gold-plated pilot contacts clean, which is what nearly every interlock connector uses. It is not enough to reliably clean tin-plated contacts. If your

loop passes through many joints, or through anything with plain tin pilot pins, fit a resistor at the return end to raise the current. There is no setting for this; it is a resistor, chosen for your harness.

How it behaves

The vehicle waits a moment after waking before it judges the loop, so a healthy vehicle does not announce a fault every time it powers up. After that, the loop has to read faulty for about a tenth of a second before the vehicle acts — long enough to ride out the momentary interruption a connector makes over a bump, short enough that nobody is reaching into a live bus in the meantime.

Recovery is deliberately much slower: the loop has to read healthy again for a couple of seconds before the fault clears. A loop that keeps breaking and healing has a loose connector, not a fixed one, and a fault that flickers along with it tells a technician nothing.

What you will see when it trips

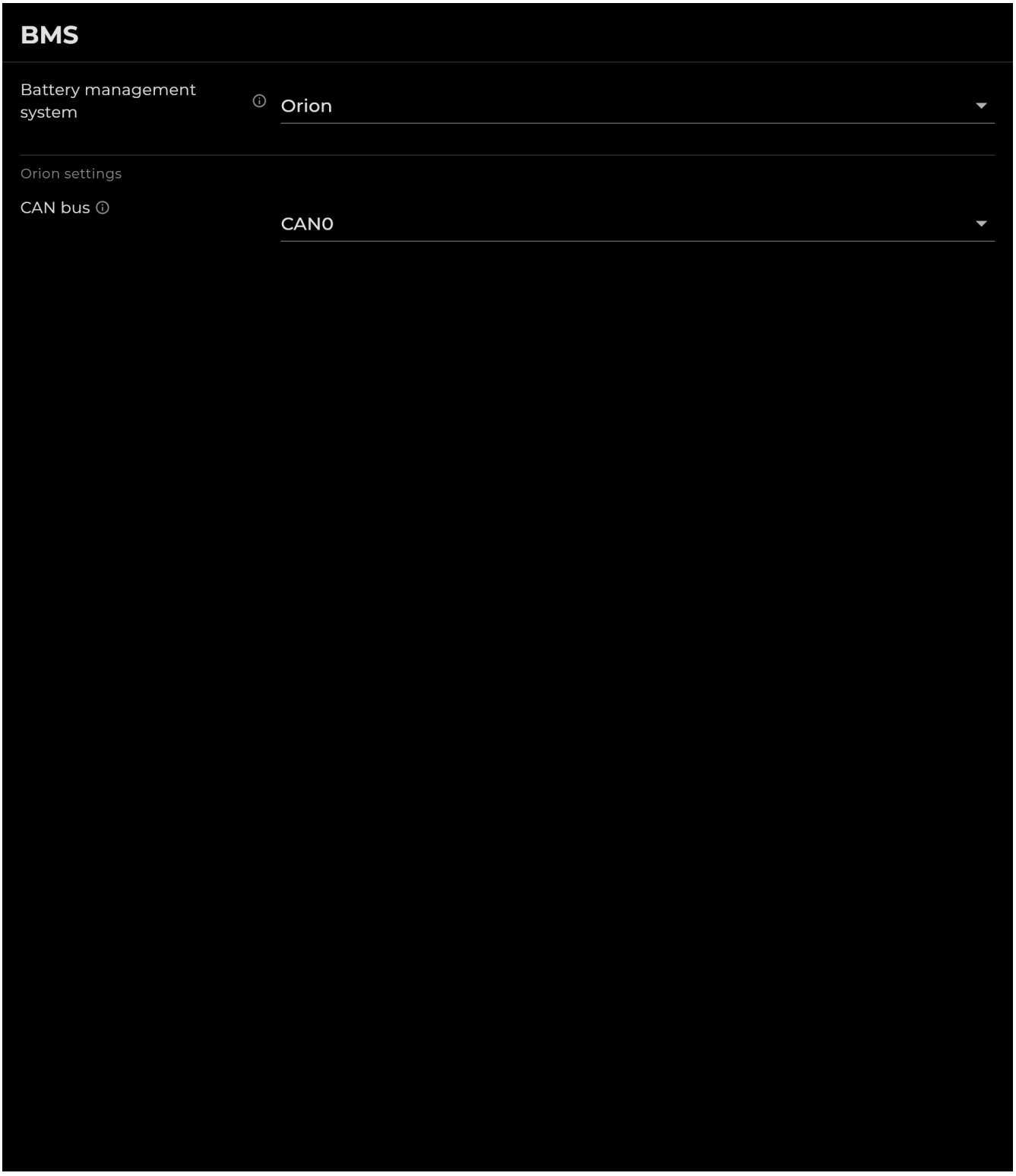
The vehicle opens its main contactors and will not bring high voltage back up while the loop still reads faulty. The fault log names which of two problems it found: a loop that has gone open, or a return signal that does not match what was sent. Those need different repairs — the first is a connector, the second is a short or something else driving the line. Fix the loop, and after the recovery delay the vehicle arms again.

Settings

- **Interlock loop** — Turn this on when the vehicle has a high-voltage interlock loop wired to the VCU. The interlock is a small low-voltage circuit that runs in series through every high-voltage connector, service disconnect, and cover. Open any of them and the circuit breaks, and the vehicle opens its main contactors so nobody meets a live bus at a part they just opened. Leave this off if the battery pack or a separate interlock module already owns the loop and reports it over the CAN bus.
- **Loop supply output** — The output that feeds the interlock loop. Leave it empty when something else already powers the loop and the VCU only watches the return; the loop is still monitored either way. The output has to match the input you choose below — a low-side output for an input that idles high, a high-side output for one that idles low — and the app tells you when the pair cannot work.
- **Loop return input** — The input the far end of the interlock loop comes back to. Choosing a frequency input is worth it where you have one: the vehicle then sends a pulse down the loop and checks the same pulse returns, which also catches a loop that has been shorted or bridged with a jumper wire, not just one that is broken. Everything else about the loop is set for you.

BMS

The battery management system fitted to this vehicle, if any.



Settings

- **Battery management system** — The battery management system fitted to this vehicle, if any.

Static settings

- **Discharge-limit curve** — Optional curve that lowers the discharge limit as pack voltage falls, to protect the pack from over-discharge near empty. Leave unset to use the fixed discharge limit at every voltage.
- **Charge-limit curve** — Optional curve that lowers the charge limit as pack voltage rises, to protect the pack from over-charge near full. Leave unset to use the fixed charge limit at every voltage.
- **State-of-charge curve** — Optional curve estimating the battery gauge from pack voltage. It is only a rough estimate — pack voltage sags under load — so leave it unset if you would rather show no gauge than an approximate one.
- **Pack temperature source** — Optional. Where pack temperature is read from, for display and logging only. Pick a temperature reading, or leave it as "none".

Orion / Thunderstruck settings

- **CAN bus** — Which CAN bus the battery management system communicates on.

Static settings

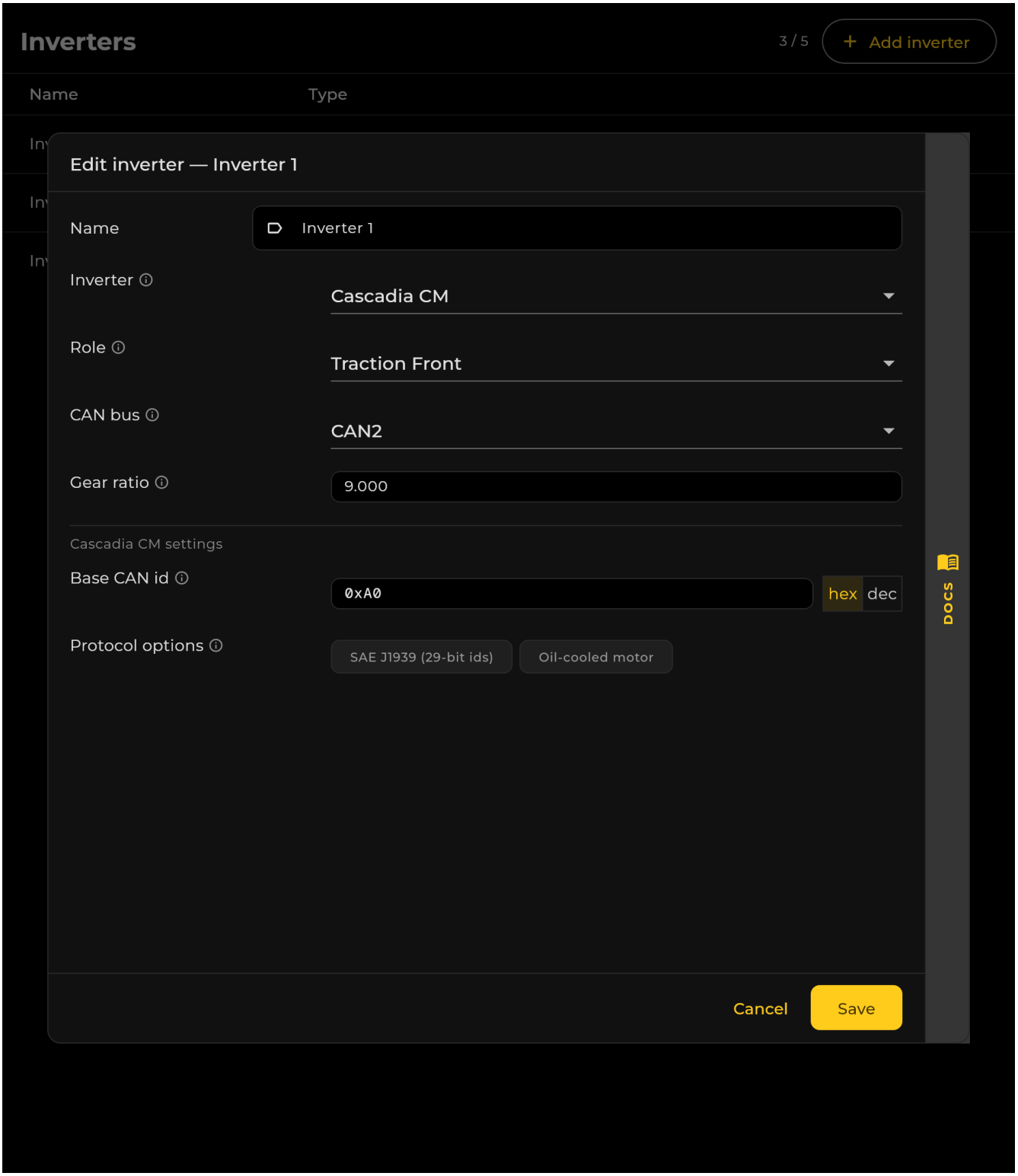
- **Discharge current limit** (A) — The most current the pack may deliver while driving. Used at every voltage unless a discharge-limit curve is set below. Set it to the pack's continuous discharge rating.
- **Charge current limit** (A) — The most current the pack may accept while charging or regenerating braking energy. Used at every voltage unless a charge-limit curve is set below. Zero disables charging and regen.

Drivetrain

The hardware that turns battery power into motion: the inverter driving each motor, and — on vehicles that keep one — the manual gearbox and its gear detection.

Inverters

The motor inverters fitted to this vehicle — up to five. For each, choose its make, the bus it talks on, its role (which wheels it drives), and its gearing.



This vehicle can carry up to five motor inverters, and each one is set up on its own. For every inverter you pick its make under **Inverter**, the bus it talks on under **CAN bus**, what it does for the vehicle under **Role**, and its gearing under **Gear ratio** — and, depending on the make, you give it an address so the controller can find it on the bus. The five inverter slots are fully independent: you can mix makes, buses, and roles freely, and any slot you are not using is simply left with its **Inverter** set to none. An empty slot is normal and

is ignored — it raises no fault. To add another inverter with the same settings, press the copy icon on its row. The copy opens as a new inverter named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

Each inverter you do configure needs a few things to come alive:

- A make chosen under **Inverter** (an H-Protocol Genus or Kendal drive, Cascadia CM, a CANopen CiA-402 drive, Steinmetz, or a Nissan Leaf inverter), and an address that matches how that make is wired on the bus. CANopen and H-Protocol Kendal drives are addressed by **Node address**; Cascadia drives are addressed by **Base CAN id** (plus the J1939 choice under **Protocol options**). H-Protocol Genus, Steinmetz, and the Nissan Leaf use fixed factory IDs and ignore both address fields. A Nissan Leaf instead needs its **Generation** chosen under **Protocol options** — Gen 1 (ZE0), Gen 2 (AZE0), or Gen 3 (ZE1) — to match the year of the inverter you fitted, plus a 160 kW option for a Gen 3 e+ high-power drive. The controller stands in for the Leaf's own vehicle and battery controllers, so a Leaf runs on its own CAN bus with nothing else from the donor car.
- A live **CAN bus** that actually carries the inverter's broadcast frames. Every make watches for fresh data and will mark an inverter unhealthy if its messages stop arriving (each make has its own freshness window, roughly a tenth of a second for the fast signals and up to a second for the slow ones). An inverter that goes quiet is still "present" but reports as faulted, and the safety check collapses that corner's torque to zero.
- For a **Role** of traction: a pedal demand, the 3D torque map and pedal curve, motor-speed feedback, and a correct **Gear ratio** and axle wheel radius (used for the traction-control speed estimate and wheel-speed reference). The vehicle must be in Drive before a traction inverter is allowed to make torque.
- For a **Role** of accessory: the accessory drive must be wired to its script-facing device so a script (or the controller) can command its speed.
- For a CANopen drive: the controller must be able to read the motor's rated torque from the drive at start-up. Until it does, the inverter is commanded to zero torque (it sits quiet and safe) rather than guessing.

How it works

The controller treats all five inverters the same way, no matter the make. It hands each one its commands and reads back its speed, torque, temperatures, DC voltage and current, available torque, and health — the make-specific wire details are handled underneath. That is why adding or swapping a make never changes how the rest of the vehicle behaves.

Picking a make and address. The **Inverter** make decides which address fields matter. The controller protects you from two inverters fighting over the same identity: if two inverters of the **same make** sit on the **same CAN bus** at the **same effective address**, they collide. The first one wins and the second is refused — it goes dark and reports a configuration fault. For CANopen drives the address is the **Node address**, so two CANopen drives with different node addresses happily share one bus. For Cascadia drives the address is the **Base CAN id** together with the J1939 setting; leaving **Base CAN id** at 0 uses the make's standard default address, so a blank field and the explicit default address count as the same address. Two fixed-ID

inverters of the same make (two H-Protocol Genus units, or two Steinmetz) can never share a bus — put them on different buses.

A Kendal drive has two more addresses. A Kendal drive listens for one exact message, and that message names both ends of the conversation — the drive and the controller. **Node address** is the drive's end. **Controller address** is this controller's end, and **Command address** is where commands are sent. Leave both at zero and the controller uses what a Turntide drive ships with, which is what almost every installation needs. Change them only if the drive was commissioned differently: Turntide build their drives to more than one address pair, and their CMC tool shows which one a given drive expects. A mismatch is silent in the worst way — the drive still broadcasts its status, so it looks alive and healthy, but it ignores every command and logs its own message timeout. If a Kendal drive reports present and healthy yet never makes torque, these two settings are the first place to look. **Command address** also has one special value: 255 sends commands to every drive on the bus rather than to this one by name, which a few Turntide configurations expect. Status messages are unaffected either way — they always follow **Node address**.

Protocol options is where you tell one make's driver which variant of that make you actually have. For Cascadia: whether it speaks 11-bit or SAE J1939 29-bit CAN (this also separates the address space, so an 11-bit and a 29-bit Cascadia never collide), and whether it is oil-cooled (which makes the controller expect a live oil-temperature reading). For CANopen: whether to listen for the drive's heartbeat for liveness, and whether to also send a maker-specific mode-select command at start-up (needed for Sevcon Gen4 as the master drive).

Role decides who drives the inverter and how. **Role** does three things at once. First it decides who commands the inverter: a traction role is driven by the torque arbiter, a generator role by the genset logic, and an accessory role by a script. Second it decides whether the inverter is commanded by torque or by speed: traction and generator inverters are torque-commanded; an accessory inverter is speed-commanded. Third, for traction it decides the left/right split — a centred front or rear role owns the whole axle's share, while a driver-side or passenger-side role drives just that corner. One consequence to know: if you set an accessory **Role** on a make that cannot accept a speed command (the Nissan Leaf inverter, which is traction-only), the Configurator flags it before you flash so you can fix it first, and as a backstop the controller also refuses that inverter at start-up with a configuration fault. Generators apply braking torque to harvest energy; they are torque-commanded, not run as a speed setpoint against the engine.

An accessory motor can be told how fast it may change speed. Two optional settings appear for an accessory role: **Acceleration limit** and **Deceleration limit**, each in rpm per second. With a limit set, a new speed request does not jump — the commanded speed walks to the requested value at no more than that rate, whichever way the motor is turning, and each direction has its own limit (leave one at zero and that direction changes instantly). This protects a shock-sensitive load — a hydraulic pump, a belt drive — no matter what commands the motor: a script, a control recipe, or a hand throttle all get the same ramp. Two things always cut through it instantly: disabling the motor for safety, and any safety action — the limits shape normal speed changes only. If the motor is already turning when a first command arrives, the ramp starts from its actual speed, so a coasting motor is picked up smoothly rather than dragged to zero and back up.

Gear ratio shapes the math, not the torque command. The torque the controller sends an inverter is already motor-shaft torque, so **Gear ratio** is **not** multiplied into the torque command — the number on the pedal map is what the inverter is asked for. **Gear ratio** is instead used wherever the controller has to translate between the motor and the wheel: the wheel torque it reports, the road-speed estimate for traction control, the regen blend, and (on a manual-gearbox build) the overall reduction. Enter it as the reduction from motor to wheel — for example a 9:1 final drive is a gear ratio of 9. Leaving it at 0 is treated as direct drive, 1:1.

```
wheel torque (as reported) = motor torque × gear ratio
wheel speed                = motor speed ÷ gear ratio
```

On a manual-gearbox vehicle, **Gear ratio** is the final-drive (differential) reduction only — the engaged gearbox ratio supplies the rest:

```
total reduction = gearbox ratio (engaged gear) × gear ratio
```

A protective speed ceiling always applies. Whenever the controller commands traction torque it also hands the inverter a fixed protective speed ceiling (20000 rpm) so the drive will back off torque before it overspeeds. This is a fixed safety ceiling and is **not** your **Max speed (rpm)** or **Overspeed governor (rpm)** setting. As a final guard, any speed limit sent to an inverter is capped so it can never wrap around to a tiny or negative value:

```
speed limit sent to inverter = min(requested limit, 32767 rpm)
```

Accessory drive settings written at start-up (CANopen accessory drives). Three ramp-and-cap settings take effect only on a CANopen inverter whose **Role** is accessory, and they are written to the drive once at start-up. **Max speed (rpm)** caps how fast the accessory is allowed to spin. **Acceleration (rpm/s)** sets how quickly it ramps up to a new target speed, and **Deceleration (rpm/s)** how quickly it ramps down. For all three, leaving the value at 0 means "leave the drive's own saved default alone" — the controller writes nothing.

How an accessory drive holds the speed you ask for. A CANopen drive has a speed mode of its own: it is given the target and regulates to it. An H-Protocol drive has no speed command, so the controller reaches the same result a different way — it asks for the full torque allowance and caps the speed in the direction you commanded, and the drive accelerates until it meets that cap and then holds there. Commanding zero stops the motor on either family. One difference is worth knowing on an H-Protocol drive: a speed cap stops the drive pushing, it does not brake, so the motor slows only as fast as its load slows it. On those drives the **Deceleration limit** shapes the command, and the actual slow-down is whatever the load gives.

Accessory torque allowance (Nm) is different from the three settings above: it is a live cap re-sent on every control cycle, not a one-time start-up write, and it applies to an accessory drive of any make. It is how much torque the drive may draw to reach and hold the speed it was asked for — a ceiling, not the torque applied, because the drive takes only what the load needs. Set it above the torque the load needs at full speed, or the motor will never get there; the motor's continuous rating is a safe place to start. Leaving it at 0 means no extra cap — the drive's built-in 100%-of-rated limit governs.

A rising load can have a curve instead of one number. Some loads need far more torque at speed than at rest — a pump, a fan, a compressor, where the torque needed climbs steeply as the shaft speeds up. For those, **Torque allowance curve** replaces the single number: you draw torque against commanded speed, and the controller reads the curve at whatever speed was asked for. The curve is a ceiling at every point, so it must stay above the torque the load actually needs at that speed, or the motor stalls short of the target. The curve editor's flat presets give you one level across the whole speed range in a click — the same thing as the single allowance, and a good place to start before you shape it. Leave the curve unset to keep the single allowance.

Overspeed governor (traction and generator). **Overspeed governor (rpm)** is the matching one-time start-up setting for torque-commanded inverters (traction and generator) on a CANopen drive. It is written to the drive's maximum-motor-speed limit, and from then on the **drive itself** enforces that ceiling autonomously — the controller cannot tighten it on the fly. Leaving it at 0 leaves the drive's own default. Two behaviours to know: if the drive does not support this limit, the controller skips the write so the inverter still boots, but raises a configuration fault so you can see the cap is not being enforced; and if a configured limit elsewhere asks for a tighter ceiling than this governor, the controller flags a configuration fault rather than silently failing to honour it. This per-inverter governor is separate from the vehicle-wide overspeed taper (set elsewhere in the vehicle's drive limits), which gently rolls each corner's forward torque off to zero as that motor approaches the configured top speed, and forces a corner's forward torque to zero if its speed reading is missing.

Tuning notes

Start by getting **Inverter**, **CAN bus**, **Role**, and the address (**Node address** — plus **Controller address** and **Command address** on a Kendal drive — or **Base CAN id** plus **Protocol options**) right and confirm each inverter reports present and healthy before touching anything else — a wrong address or a make/variant mismatch shows up as an inverter that never comes online or that reports faulted. If an inverter you expect is dark, check first for an address collision with another inverter of the same make on the same bus.

Set **Gear ratio** to the true motor-to-wheel reduction (final drive only on a manual build). It will not change pedal feel or how much torque the motor makes, but getting it wrong throws off reported wheel torque, traction control, and regen blending, so it is worth measuring rather than guessing.

For an accessory drive, tune **Max speed (rpm)** to the safe top speed of whatever it spins, then use **Acceleration (rpm/s)** and **Deceleration (rpm/s)** to make its motion as smooth or as snappy as the application wants — start gentle and tighten up. Set **Accessory torque allowance (Nm)** to protect the driven equipment; because it is enforced every cycle you can trust it as a hard cap. Start at the motor's continuous rating and bring it down until the motor still reaches every speed you need under real load. If the load is a pump or a fan, do that at full speed first, then move to a **Torque allowance curve** and trim the low-speed end, where far less torque is needed.

For traction and generator drives, set **Overspeed governor (rpm)** to the motor's safe maximum and verify the drive accepted it — if you see a configuration fault, the drive does not support being told its overspeed limit and you must rely on the drive's own internal protection and the vehicle-wide speed taper instead.

Kendal inverters have separate Drive torque limit and Brake torque limit settings. These motor-shaft limits are sent to the inverter; they do not request torque. Use a positive drive limit and a negative brake limit. Zero permits no torque for that limit. New projects start at the protocol limits of +2047.9375 Nm and -2048 Nm. Existing projects must set these new fields before flashing an updated configuration.

Settings

- **Inverter** — The motor inverter fitted to this slot. Leave unset for an unused slot.
- **Role** — What this inverter drives — a front or rear traction motor (or a single corner of an axle), a generator, or an accessory motor.
- **CAN bus** — Which CAN bus this inverter communicates on.
- **Gear ratio** — The total reduction between the motor shaft and the wheel (for example 9 means 9:1). On a manual-gearbox vehicle this is the final-drive reduction only. A drive unit with a built-in reduction fills this in for you when you pick it — a Nissan Leaf sets 8.19:1 — so change it only if your driveline differs (a swapped final drive, an added gearbox).

H-Protocol (Genus) / H-Protocol (Kendal/Gen6) / CANopen CiA-402 settings

- **Node address** — The network node address of this drive, for address-based protocols. A Turntide Gen6 ships with address 114 and listens only for commands sent to that address, so leave 114 unless the drive was commissioned with a different one; the drive's own tool shows its current address.

Cascadia CM / CANopen CiA-402 settings

- **Base CAN id** — The base message id this inverter uses. Leave at zero to use the vendor's default addressing; set it to run more than one identical inverter on a bus.

Cascadia CM / Nissan Leaf settings

- **Protocol options** — Protocol variants for this inverter family. Match them to your hardware. For a Nissan Leaf, pick the generation that matches the inverter you fitted (the car's model year), and tick 160 kW only for a Gen 3 "e+" high-power inverter.

CANopen CiA-402 settings — Settings specific to CANopen CiA-402 drives, used for accessory and generator motors run in speed mode.

- **Max speed** (rpm) — Speed-mode velocity ceiling. Leave at zero to keep the drive's own default.
- **Acceleration** (rpm/s) — Speed-mode acceleration ramp. Zero keeps the drive's own default.
- **Deceleration** (rpm/s) — Speed-mode deceleration ramp. Zero keeps the drive's own default.

Accessory settings

- **Accessory torque allowance** (Nm) — How much torque the accessory motor may draw to reach and hold its commanded speed. It is a ceiling, not the torque applied — the drive takes only what the load needs, up to this figure. Set it above the torque the load needs at full speed, or the motor never reaches the speed asked for. The motor's continuous rating is a safe starting point. Zero uses the drive's own rated default.

CANopen CiA-402 settings — Settings specific to CANopen CiA-402 drives, used for accessory and generator motors run in speed mode.

- **Overspeed governor** (rpm) — Protective maximum motor speed. Zero keeps the drive's own default.

Accessory settings

- **Acceleration limit** (rpm/s) — Fastest the accessory motor is allowed to speed up, in rpm per second. When a script or control asks for a higher speed, the command walks up to it at this rate instead of jumping. Applies whichever direction the motor turns. Zero means no limit — the new speed applies immediately.
- **Deceleration limit** (rpm/s) — Fastest the accessory motor is allowed to slow down, in rpm per second. When a lower speed (or a stop) is commanded, the command walks down at this rate instead of cutting instantly. Zero means no limit — the motor is commanded to the lower speed immediately. Disabling the motor for safety always acts instantly, whatever this is set to.
- **Torque allowance curve** — An optional curve that sets the torque allowance from the commanded speed, for a load whose torque rises with speed — a pump, a fan, a compressor. When set it replaces the single allowance above. The curve is read at the speed asked for, so it must stay above the torque the load needs at that speed, or the motor never gets there. Leave it unset to use one flat allowance at every speed.

H-Protocol (Kendal/Gen6) settings — Settings specific to Kendal v3 (Gen6) H-Protocol inverters.

- **Controller address** — The address this VCU uses when it talks to the drive. Leave it at zero unless the drive was commissioned to expect a different one.

The drive listens for one exact message id, and that id contains both addresses — the drive's and the controller's. If this does not match what the drive was set up with, the drive ignores every command and reports a message timeout, which looks like a dead controller. Turntide ship their drives on more than one address pair, so check the settings in their CMC tool if the drive does not respond.

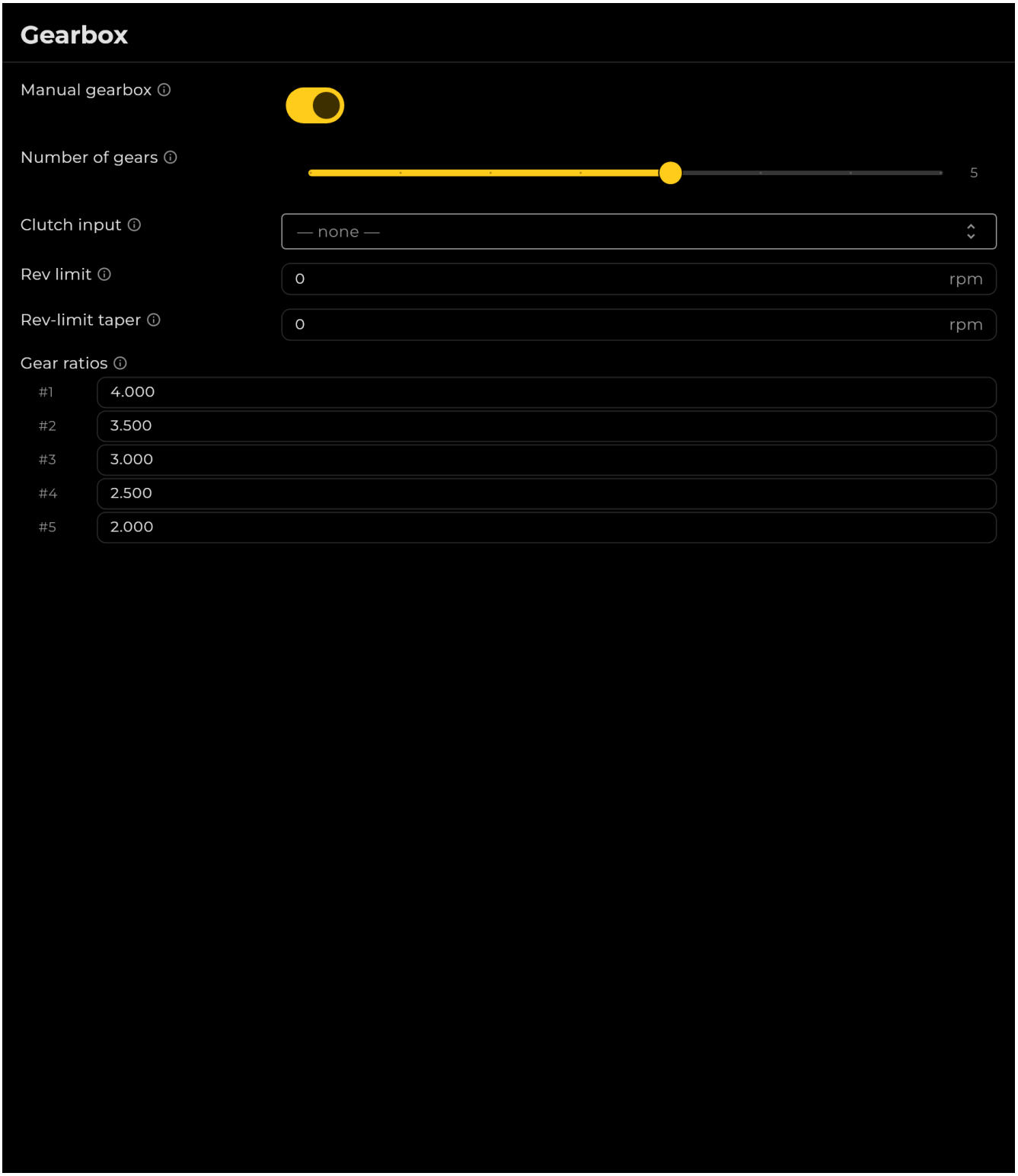
- **Command address** — The address this VCU sends commands to. Leave it at zero to send them to the node address above, which is what almost every installation uses.

Set it to 255 only if the drive was commissioned to accept broadcast commands. This does not change where status messages come from — those always follow the node address.

- **Drive torque limit** (Nm) — Maximum drive torque sent to this inverter, at the motor shaft. The protocol maximum is 2047.9375 Nm. Zero permits no drive torque. This is a limit, not a torque request. The pedal and safety monitor still determine the torque requested. Set this to suit the motor and inverter.
- **Brake torque limit** (Nm) — Braking torque limit sent to this inverter, at the motor shaft. Use a negative value, down to -2048 Nm. Zero permits no braking torque. This is a limit, not a braking request. Set its magnitude to suit the motor and inverter.

Gearbox

The manual gearbox: enable it, set the number of gears and their ratios, the clutch input, and the rev limit. The torque map is shared with the rest of the drivetrain.



The gearbox build mode lets you keep a real manual transmission — a donor gearbox with a clutch pedal and a shift lever — in front of a single electric drive motor. Turn it on with **Manual gearbox** and the controller switches the whole drivetrain over to a single-motor, single-axle behavior: the electric motor makes torque through the original gearbox, the driver works the clutch and shifts gears the way they always did, and the motor obeys a rev limit so it behaves like an engine. The amount of torque the motor

makes for a given pedal position comes from the same torque map the rest of the drivetrain uses — the gearbox mode does not get its own separate map.

What it needs

The gearbox mode only drives when the vehicle is actually in Drive, the accelerator pedal reads valid, and a direction (forward or reverse) is selected; in any other situation the motor command stays at zero. It needs the shared torque map loaded (if the map fails its checks, the controller falls back to a simple straight-line throttle-to-torque relationship), and it needs the motor's maximum torque rating to set the ceiling.

To know how fast the motor is spinning — which it needs for the rev limit — at least one drive motor must be reporting valid speed. If all motor-speed reporting is lost, the controller assumes a very high motor rpm (a deliberately conservative guess): that lands at the low-torque end of the map and tends to make the rev limiter cut, so a speed-signal failure makes the vehicle weaker and safer, never stronger.

The clutch feature is optional. It needs a clutch switch wired to a digital input and assigned under **Clutch input**. The switch must close when the clutch pedal is pressed (pedal down = clutch disengaged = driveline disconnected). If **Clutch input** is left unset, the controller behaves as though the clutch is always engaged.

How it works

While driving, the controller runs this sequence every control cycle. It is worth reading in order, because the rev limit, the clutch, and the direction sign are all applied at specific points.

First it works out the motor speed it will use, then it reads the clutch and merges the demand from the pedal, cruise, and hill-hold:

```
demand = the largest of: pedal %, cruise % (only if cruise is active), hill-hold demand
```

Hill-hold only contributes while the clutch is engaged — holding the car against a disconnected driveline is pointless, so an open clutch drops hill-hold to zero. Cruise is likewise cancelled when the clutch is pressed.

That demand is run through the shared torque map to get a motor torque:

```
motor torque = torque map( motor rpm , demand )      (capped at the motor's max torque
rating)
```

This is the same map, with the same pedal shaping and the same torque ceiling, that the non-gearbox drivetrain uses. The map is gear-agnostic: it does not know or care which gear is selected, so the pedal feels the same in every gear and the gear ratios you enter do not change it.

If the controller can confirm the engaged gear from a real gear sensor, an optional per-gear torque cap is applied next. This cap can only ever reduce torque, never raise it, and it only acts on a sensor-confirmed gear — an inferred or unknown gear never caps torque. If a previously healthy gear sensor drops out, the controller assumes the lowest (most restrictive) gear's cap and logs a gear-sensor fault, so a detection problem can only ever under-torque the vehicle. (The per-gear cap value and the gear-detection method are set in the related gear-detection block, not in the six gearbox settings here.)

Reverse then flips the sign of the torque so the motor drives the car backward. If the clutch is pressed, the motor torque is forced to zero at this point — there is no path to the wheels, so the motor makes nothing.

Finally the rev limit is applied to the signed command, which caps both forward and reverse:

```
if motor rpm >= Rev limit:                      motor torque = 0        (hard cut)
else if motor rpm > (Rev limit - Rev-limit taper):
        motor torque = motor torque x (Rev limit - motor rpm) / Rev-limit taper
else:                                            motor torque unchanged
```

So **Rev limit (rpm)** is the ceiling the motor will not cross, and **Rev-limit taper (rpm)** is how wide a band below that ceiling the torque eases off over. At the bottom of the taper band the motor still makes full torque; right at the limit it makes none; in between it falls off in a straight line. Setting **Rev limit (rpm)** to 0 disables the rev limiter entirely. Setting **Rev-limit taper (rpm)** to 0 means full torque holds right up to the limit and then cuts to zero abruptly at it.

Every result then passes through the independent torque-safety check, which has the final say on the motor command, and a traction-control overlay can still trim it further. Neither of those can increase torque.

A few of the six settings shape what the controller counts and how it reasons about gears. **Number of gears** tells it how many of the **Gear ratios** entries to use (up to eight; anything above eight is clamped to eight). **Gear ratios** are entered as the gearbox reduction for each gear — for example a 9:1 first gear. Importantly, **Gear ratios** do not pick the torque map. They are used for two things: (1) inferring the current gear by comparing motor speed against road speed, and (2) estimating torque at the wheels for the vehicle-dynamics features, by combining the selected gear's ratio with the electric drive's own fixed final-drive ratio.

Tuning notes

Set **Manual gearbox** on first; that is what swaps the vehicle into single-motor gearbox behavior and activates the clutch cut, the rev limiter, and gear detection. Enter **Number of gears** and the matching **Gear ratios** to describe the donor transmission accurately — these are what let the controller infer the gear and report wheel torque correctly, even though they never change pedal feel.

Tune **Rev limit (rpm)** to whatever red line you want the motor to respect — typically a safe motor speed, or a value that mimics an engine's red line for driver feel. Then set **Rev-limit taper (rpm)** for how the limit feels: a wider taper gives a soft, progressive "bounce off the limiter" that is easy to live with; a narrow taper (or zero) gives a hard, abrupt cut. Start with a generous taper and tighten it only if the driver wants a sharper limiter.

Assign **Clutch input** if a clutch switch is wired. With it set, pressing the clutch instantly drops motor torque to zero and cancels cruise and hill-hold, which prevents any lurch when the driver declutches to shift. Confirm the switch polarity on the bench: pedal pressed must read as disengaged. If the clutch feels backward (torque cutting when the pedal is out, or driving when the pedal is in), the input is inverted and must be corrected before the vehicle is driven.

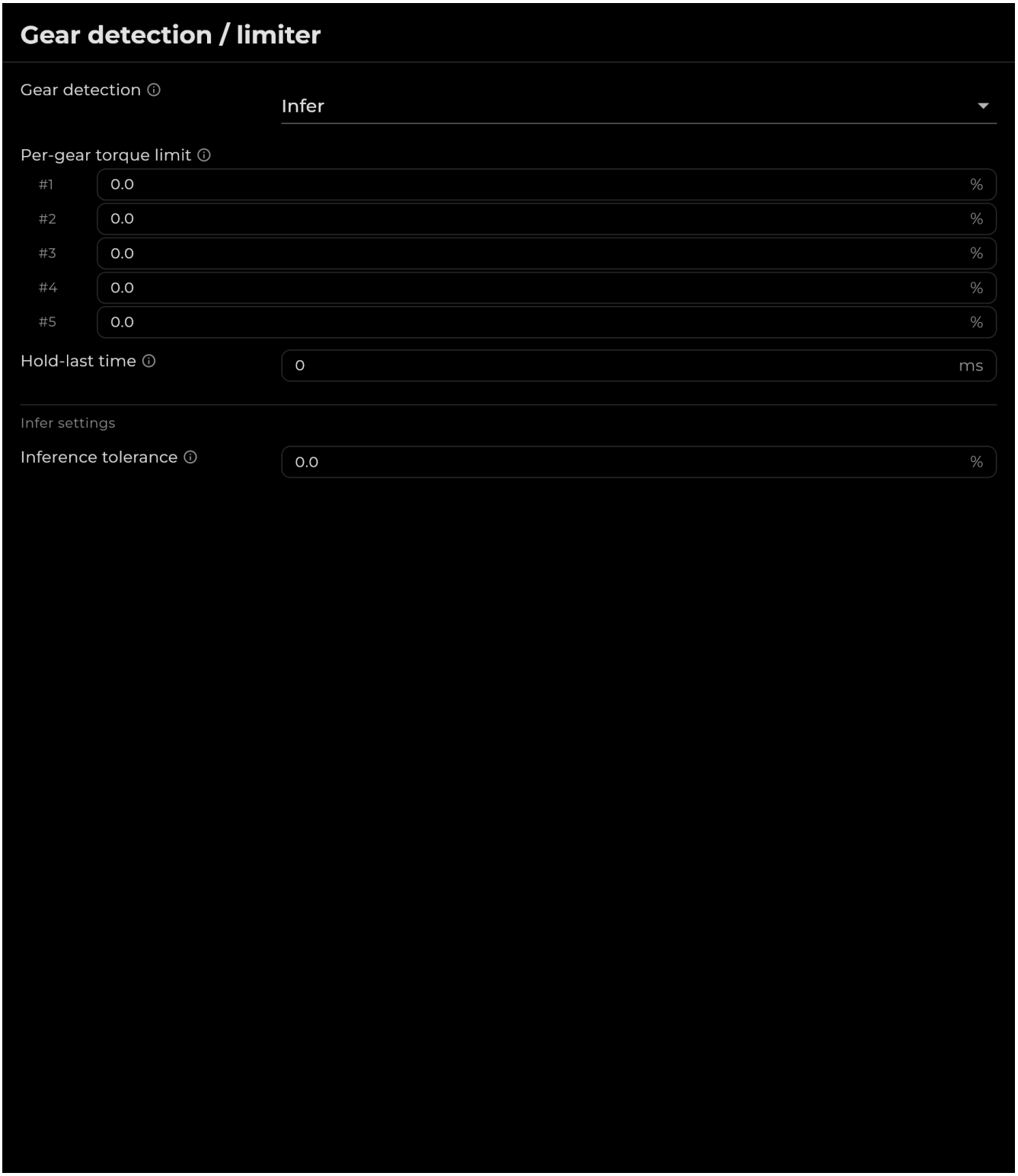
Settings

- **Manual gearbox** — Use this for a vehicle that keeps its manual transmission and clutch.

- **Number of gears** — How many forward gears the gearbox has.
- **Clutch input** — The input that signals the clutch is pressed, which cuts torque.
- **Rev limit** (rpm) — The motor speed at which torque is cut to protect the driveline. Zero means no limit.
- **Rev-limit taper** (rpm) — How far below the rev limit torque begins to taper off.
- **Gear ratios** — The ratio of each gear, used for the dashboard readout and gear detection.

Gear detection / limiter

How the engaged gear is detected, and an optional per-gear torque limit that protects the driveline in low gears.



Gear detection figures out which gear a manual gearbox is currently in, and an optional per-gear torque limit uses that answer to hold back motor torque in the lower gears so the e-motor can't overload a retained donor driveline. This system only exists on a manual-gearbox conversion — a build where the original clutch and gearbox are kept in front of the electric motor. On any other build the whole system is inert: it always reports "unknown / neutral" and the torque limit never engages.

The detection is split cleanly in two. One half *decides what gear you're in* (and how confident it is). The other half — the per-gear limit inside the torque path — *acts on that answer* by trimming torque. Knowing this split is the key to the safe-fallback behavior below: a detection problem can only ever make the vehicle produce **less** torque, never more.

What it needs

You must be on a manual-gearbox conversion build for any of this to do anything. You also tell the system how many gears the box has (up to 8); the per-gear settings are then read one slot per gear, first gear in the first slot.

Beyond that, the requirements depend on which detection method you pick under **Gear detection**:

- A voltage-based gear sensor (a shifter potentiometer) needs the **Gear sensor** pin wired, and a voltage window per gear set with **Gear window — low (V)** and **Gear window — high (V)**. Optionally a second redundant track on **Gear sensor (second)**.
- A set of discrete switches — one wire per gear — needs a **Gear input** pin assigned for each gear.
- A gear value sent over the CAN bus needs a **Gear signal** picked per gear, plus a value band on **Gear value — low** and **Gear value — high**.
- Inference (computing the gear from how fast the motor spins versus how fast the vehicle is actually rolling) needs no gear sensor at all, but it needs three other things to be trustworthy: an independent road-speed reference from the wheel-speed sensors (it deliberately uses the wheel sensors, preferring the non-driven axle, and never the motor-derived speedometer, which would make the math circular), live motor-rpm from the inverter, and a configured rolling radius for the driven axle's tyres. It also needs each gear's gear ratio entered. If a clutch-position switch is wired, it uses that too.

How it works

The gear is recomputed on every control cycle — about 100 times a second — just before the torque arbiter runs, so the limiter always acts on a fresh reading. The method is whatever you chose under **Gear detection**. There are five choices, and they fall into two groups that behave very differently with respect to torque.

Methods that can limit torque (sensor-backed): voltage sensor, discrete switches, CAN value. These produce a gear reading the limiter is allowed to trust, and they all pass through the same "ride through brief dropouts" logic described below.

A **voltage sensor** reads the shifter pot in volts. If you also configured **Gear sensor (second)**, the two tracks are first compared: if they disagree by more than **Max sensor disagreement (V)**, the reading is thrown out as invalid (this catches a single shorted or broken pot wire); otherwise the two are averaged. The resulting voltage is then matched against each gear's window:

```
gear matches if  Gear window — low  ≤  sensor volts  ≤  Gear window — high
```

The first gear whose window contains the voltage wins. A gear whose window is left at low = 0 and high = 0 counts as "not configured" and never matches.

Discrete switches read one wire per gear. A gear is reported only when **exactly one Gear input** line is on. If no line is on, or two or more are on at once, the read is treated as invalid — this is how a between-gears or miswired state is rejected.

Each line is read the way you describe it. **Gear switch type** says whether that gear's contacts close or open when the gear is selected; most gear-position switches close on the gear they detect, but a normally-closed switch is read the other way round. **Gear switch debounce (ms)** is how long the line must hold steady before it counts. Contacts bounce as they make and break, and a bounce that briefly lands on the wrong line looks exactly like a clean single-line read — so without a debounce time the wrong gear can be reported for an instant, and its torque limit applied with it. A few tens of milliseconds is enough for most switches; 0 turns it off.

A **CAN value** reads a gear number or shifter code off the bus. For each gear in turn, the decoded value is checked against that gear's band:

```
gear matches if  Gear value — low  ≤  decoded value  ≤  Gear value — high
```

The first gear that matches wins. For a sender that emits an exact code per gear, set **Gear value — low** equal to **Gear value — high**. For a swept signal (a pot reported over CAN) use a wider band. If the signal is stale or missing, that gear is skipped — which simply looks like a lost sensor and is handled the same way.

Riding through dropouts (all three sensor-backed methods). A clean read updates the reported gear immediately. If a read comes back invalid, the system does **not** drop the gear right away — it holds the last good gear for up to **Hold-last time (ms)**, so a momentary glitch or a brief between-gears blip doesn't cause a torque hiccup. Only if the invalid reads continue past **Hold-last time (ms)** does the system declare the gear lost, mark the reading not-valid, and log a gear-sensor fault. The fault clears automatically as soon as a valid read returns.

The safe fallback. When a sensor-backed method has lost the gear (valid = false), the limiter does **not** pass torque through untouched. Instead it assumes the worst case — the lowest, most torque-restrictive gear — by applying the smallest per-gear cap you've configured. So a sensor failure can only ever under-torque the vehicle, never over-torque it.

Methods that never limit torque: Off, and Inference. Choosing **Off** under **Gear detection** disables detection entirely; the gear stays "unknown" and the per-gear limit never engages — use this for a gear-agnostic manual build. **Inference** is special: it computes the gear and shows it on the dash and uses it for consistency checks, but it is deliberately treated as too soft to gate torque, so an inferred gear **never** engages the per-gear limit.

Inference works from the ratio between motor speed and true road speed. On each cycle:

```
wheel rpm   = road speed ÷ 3.6 ÷ (2 × π × rolling radius) × 60
implied ratio = motor rpm ÷ wheel rpm
```

The implied ratio is then compared against every gear's configured gear ratio, and the closest gear is chosen — but only if it lands close enough:

```
tolerance band = gear ratio × Inference tolerance
accept the gear only if  | implied ratio − gear ratio |  ≤  tolerance band
otherwise report unknown (neutral / clutch slipping)
```

Each gear's tolerance band is sized from its own ratio, so the spacing scales correctly from low gears to high. Once a gear is chosen, a little stickiness keeps it: the held gear is retained as long as the implied ratio stays inside that same band, which stops the readout flickering between two adjacent gears. Inference only runs when the conditions make it meaningful — above 3.0 km/h, with valid motor speed, a sensible rolling radius, and the clutch engaged (if a clutch switch is wired, a pressed clutch forces "unknown"). Below those it simply reports unknown.

Cross-check (advisory only). Whenever a real sensor produces a valid gear, the system independently infers the gear in the background and compares. If the two disagree continuously for more than half a second, it logs an advisory note that the sensor and the inferred gear don't match. This is informational only — it points at a misconfigured window, a mis-entered ratio, or a failing sensor — and it never changes torque.

The torque limit itself. When a sensor-backed gear is known, the per-gear limit trims the motor torque the engine map asked for:

```
limited torque = motor torque × (Per-gear torque limit ÷ 100)
```

It is reduce-only and never changes direction: a limit of 60% means "give this gear at most 60% of mapped torque"; 0% means "no cut" (the gear is left wide open); and 100% or anything above also means no reduction. The cut is applied early in the torque path, before regen blending, the direction sign, the clutch cut, and the rev limiter, so everything downstream sees the already-limited number. This is what protects a donor gearbox and driveline from full e-motor torque multiplied through a deep first-gear ratio.

Tuning notes

Pick the detection method first. If you have a real gear sensor — a shifter pot (voltage), per-gear switches, or a gear value on CAN — prefer it, because only a sensor-backed method can actually arm the per-gear torque limit. Reserve **Inference** for builds with no gear sensor where you just want a dash readout and the background consistency check; remember it will not protect the driveline.

For a voltage sensor, set each gear's **Gear window — low (V)** / **Gear window — high (V)** with a little margin around the measured rest voltage of each gear, and leave gaps between windows so adjacent gears never overlap. If you fit a redundant second track on **Gear sensor (second)**, set **Max sensor disagreement (V)** just wide enough to tolerate normal track-to-track spread but tight enough to catch a failed wire. For per-gear switches, double-check that exactly one **Gear input** asserts in each gear and none in neutral, set **Gear switch type** to match how each switch is wired, and give **Gear switch debounce (ms)** enough time to cover the contact bounce you can measure — long enough to reject a bounce, short enough that a real shift still registers promptly. For a CAN sender, use **Gear value — low** = **Gear value — high** for an exact gear code, or a band for a swept value.

Set **Hold-last time (ms)** to ride through the longest expected momentary dropout (typically a few hundred milliseconds) without masking a genuine sensor failure — too long and a real fault is hidden, too short and normal shift transients trip the fail-safe.

Set **Per-gear torque limit (%)** per gear from the weakest link in the donor driveline: the lower the gear (the deeper the ratio), the more torque is multiplied at the wheels, so the lower gears usually want the firmest caps. Leave higher gears at 0% (no cut) if they don't need protection. Note the smallest cap you configure also becomes the assumed value if the gear is ever lost, so make your lowest-gear cap genuinely safe.

If you are using **Inference** (or watching its cross-check), tune **Inference tolerance (%)** last: larger values make the inferred gear lock on more easily and resist flicker but blur the line between adjacent gears; smaller values are sharper but will more often read "unknown" during clutch slip or near a shift. If the cross-check keeps flagging a disagreement on a vehicle you know is configured right, your entered gear ratios or rolling radius are the first things to re-check.

Settings

- **Gear detection** — How the engaged gear of a manual gearbox is determined.
- **Per-gear torque limit** (%) — A torque cap for each gear, to protect the driveline in low gears. Full means no cut.
- **Hold-last time** (ms) — How long to hold the last valid gear when the sensor reads out of range.

Infer settings

- **Inference tolerance** (%) — How closely the speed-to-rpm ratio must match a gear to call it engaged.

Analog settings

- **Gear sensor** — The analog input wired to the gear-position sensor.
- **Gear sensor (second)** — An optional second gear sensor for redundancy.
- **Max sensor disagreement** (V) — The largest allowed difference between two gear sensors before a fault.
- **Gear window — low** (V) — The lowest sensor voltage for each gear.
- **Gear window — high** (V) — The highest sensor voltage for each gear.

Digital settings

- **Gear input** — The input wired to this gear's position switch. May be a digital input or a frequency input read as a switch.
- **Gear switch debounce** (ms) — How long this gear's switch must hold steady before the gearbox is read as being in it, in milliseconds. Rejects contact bounce, which would otherwise select the wrong gear for an instant; 0 disables it. Maximum 2550 ms.
- **Gear switch type** — Whether this gear's switch contacts are closed or open when that gear is selected. Most gear-position switches are normally open — they close on the gear they detect. Pick normally closed only if yours is wired the other way.

CAN settings

- **Gear signal** — The CAN signal that indicates this gear is engaged.

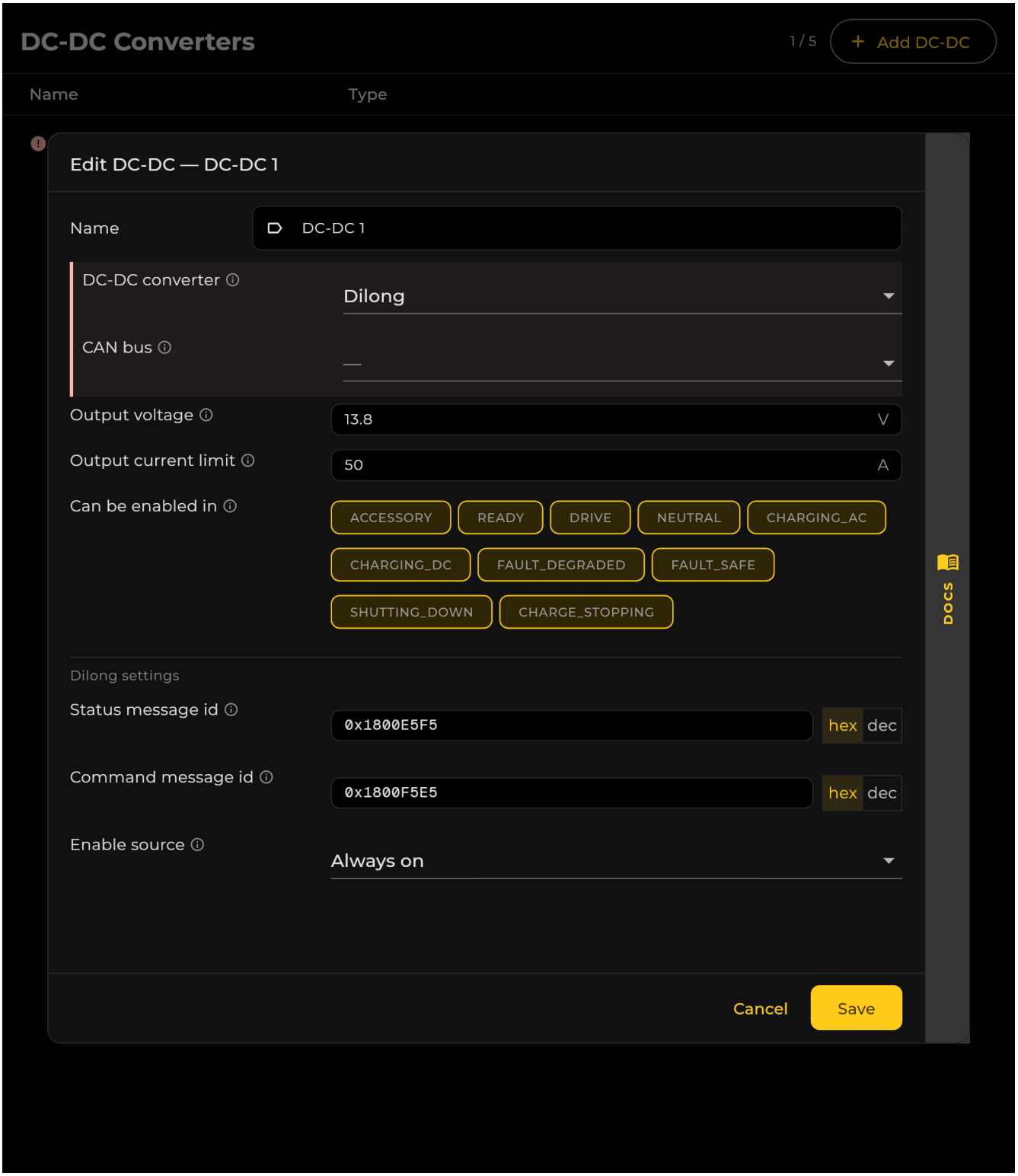
- **Gear value — low** — The lowest signal value that counts as this gear. Use the same low and high value for a fixed code.
- **Gear value — high** — The highest signal value that counts as this gear.

Charging & Power

The equipment that puts energy into the vehicle and keeps the low-voltage system alive: DC-DC converters, on-board chargers, and generator range extenders.

DC-DC Converters

The high-voltage-to-12V converters that keep the auxiliary battery charged.



A DC-DC converter takes power from the high-voltage pack and steps it down to the roughly 12 V auxiliary rail that runs the body electronics and keeps the auxiliary (12 V) battery topped up. It is the electric-vehicle equivalent of an alternator: it has nothing to do with motor torque, the pedal, or how power is split between drivetrains — its only job is to keep the low-voltage side of the vehicle alive whenever the vehicle is awake.

The VCU does not regulate the output itself; it simply tells the converter when to switch on and hands it two ceilings — a target voltage and a current limit — and the converter's own internal regulator does the rest.

The VCU supports more than one converter so you can run several units in parallel on a large vehicle. Each one is a **DC-DC converter** slot you configure independently. To add another converter with the same settings, press the copy icon on its row. The copy opens as a new converter named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

Each slot needs a converter vendor selected under **DC-DC converter** — at present the Dilong converter is the only supported unit. A slot left unconfigured is simply absent: no traffic is sent, nothing is reported, and it reads as "not present" rather than as a fault.

Each converter needs a **CAN bus** to live on — this is the high-voltage / vehicle bus that carries both the status the converter sends back and the command the VCU sends out. The VCU learns everything about the converter (its output voltage, output current, temperature, and any alarms) only from the status message the converter broadcasts about five times a second; the VCU reads no external sensors of its own. If that status message goes quiet for more than one second (about five missed updates), the converter is treated as faulted and its readings are dropped until it starts reporting again.

When two converters of the same make share one **CAN bus**, they must be told apart on the wire. Give each its own **Status message id** (the message the converter reports on) and its own **Command message id** (the message the VCU sends to it). Leave these at 0 to use the converter's factory-default addresses. If two same-make units on the same bus end up with the same status address, the second one is refused at startup, left switched off, and a configuration fault is logged — so always set distinct addresses when doubling up.

How it works

Ten times a second the VCU decides, for each converter, whether it should be on, and refreshes the command it is continuously sending (the command goes out about five times a second on its own). The on/off decision comes entirely from **Can be enabled in**: you pick which vehicle states the converter should run in, and it is commanded on only while the vehicle is actually in one of those states.

```
converter on = vehicle is in one of the Can be enabled in
               AND the Enable source says yes
```

The states you can choose from are every state the vehicle can be awake in — Accessory, Ready, Drive, Neutral, AC charging, DC charging, Fault-degraded, Fault-safe, Shutting-down and Charge-stopping. Off and Precharge are deliberately not offered: there is nothing to supply when the vehicle is off, and no high voltage bus to convert from during precharge. All of them are on by default, which is what you want — the 12 V rail runs the body electronics, the contactor coils and the controller itself, so it should stay up while the vehicle is faulted and while it is shutting down, not just while it is driving. Leaving **Can be enabled in** empty means the converter is never commanded on.

On top of the states, **Enable source** can require a signal before the converter runs at all. Leave it on *Always on* unless you have a specific reason: a converter that stops because a switch was miswired or a

CAN frame went quiet takes the 12 V system down with it. For that reason, if the vehicle is set to gate on an input and that input cannot be read, the converter keeps running — the opposite of how the charger behaves, and deliberately so. Even while commanded off, the converter keeps its command line active with the "on" bit cleared — it is told to stop, not abandoned.

While a converter is switched on, the VCU passes your two setpoints straight through on every cycle:

```
regulation voltage = Output voltage (V)
current ceiling    = Output current limit (A)
```

Output voltage (V) is the voltage you want the converter to hold on the 12 V rail — for example 14.0 V to keep the auxiliary battery charging. **Output current limit (A)** is the most current you will let it deliver, in whole amps. These are static targets; the converter's own loop holds the voltage and respects the current ceiling. The VCU does no ramping, no feedback, and no closed-loop control on top — it only sends on/off plus those two numbers. The current limit is generous by design; only an absurd value above about 6,553 A is rejected (clamped and flagged as a configuration fault), which exists purely to catch a typo, not to constrain any real converter.

The converter reports its health back through its status message, and the VCU sorts that into three levels:

- **OK** — reporting normally, no alarms.
- **Degraded** — a non-shutdown warning is present (such as low output voltage or an input-voltage warning). The output is still believed to be up, so this is informational.
- **Faulted** — either the status message has gone stale, or the converter has raised a shutdown-class protection alarm (over-temperature shutdown, output short, output over-voltage, output over-current, or an internal protection trip). A shutdown alarm means the converter has already cut its own output. Loss of communication and a protection shutdown are each logged the moment they happen.

There is one reporting safeguard worth knowing about. The reported output voltage is read as a normal 12 V-class reading. If the decoded value comes back implausibly high (above about 32.7 V), the VCU refuses to trust it: rather than show a wrong-but-plausible number, it marks the output voltage as unknown and logs a sensing-range fault, clearing it automatically once the reading returns to a sane range. A reading that stays stuck this way points at a wiring or communication problem — the wrong converter answering on the bus, or its status message being misread — rather than anything you set here.

Finally, the converter follows the vehicle's safety lifecycle. If the vehicle drops into its fault-safe state or begins shutting down, every converter is commanded fully off immediately — voltage and current setpoints zeroed — regardless of what **Can be enabled in** says. This happens alongside the rest of the high-voltage teardown.

Tuning notes

Set **Output voltage (V)** to your auxiliary-battery charging voltage (commonly around 14.0 V for a lead-acid 12 V battery; follow the battery maker's spec). Set **Output current limit (A)** to the converter's rated continuous output, or a little below if you want to cap the draw on the pack. There is no benefit to setting an artificially high limit — the converter will not exceed its own rating regardless.

Choose **Can be enabled in** to match when the vehicle needs 12 V power. Including Ready, Drive, Neutral, and both charging states is the usual choice and keeps the auxiliary battery charged whenever the vehicle is in use or on the charger. Leave Off and Shutting-down out so the converter powers down with the vehicle.

When running two same-make converters on one **CAN bus**, the only mandatory step beyond the setpoints is giving each unit a distinct **Status message id** and **Command message id**; if you skip this, the second converter will quietly refuse to come up and log a configuration fault. A single converter, or two converters on different buses, can keep the default addresses.

Settings

- **DC-DC converter** — The high-voltage-to-12V converter fitted to this slot, if any.
- **CAN bus** — Which CAN bus this converter communicates on.
- **Output voltage** (V) — The 12V-side output voltage setpoint.
- **Output current limit** (A) — The maximum 12V-side output current.
- **Can be enabled in** — The vehicle states in which this converter runs. A DC-DC converter powers the 12 volt system, so it normally runs in every state offered — including while faulted and while shutting down, since that is when the 12 volt loads (contactor coils, lights, the controller itself) matter most.

Dilong settings

- **Status message id** — Incoming status message id. Zero uses the vendor default.
- **Command message id** — Outgoing command message id. Zero uses the vendor default.
- **Enable source** — What turns the converter on, on top of the vehicle states above. "Always on" runs whenever the vehicle state allows — the usual setting. "Digital switch" and "CAN signal" gate it further, so the converter only runs when that input says so. "Script" hands the decision to a recipe or script; with it selected the converter stays off until something enables it, including if that automation never runs.

Digital switch settings

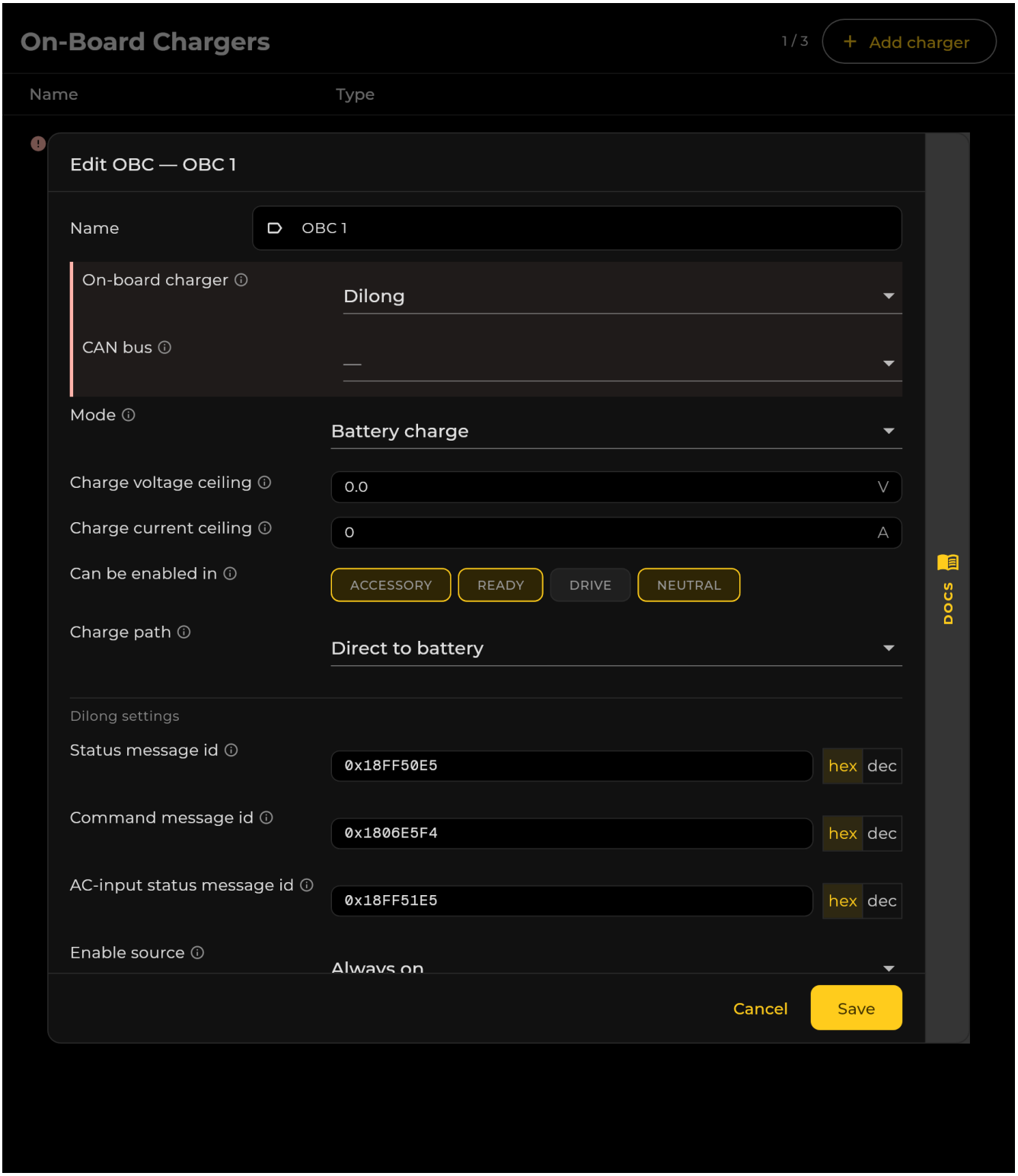
- **Enable switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.
- **Enable input** — The digital input wired to the enable switch.

CAN signal settings

- **Enable signal** — The CAN signal whose value enables the converter.
- **Enable threshold** — Enabled while the CAN signal reads at or above this value.

On-Board Chargers

The on-board chargers that charge the high-voltage battery from an AC supply.



An on-board charger turns AC mains power into DC and pushes it into the high-voltage traction pack. The VCU can run up to three on-board chargers at once, each as an independent slot with its own settings. The VCU does not regulate the charge itself — it only tells each charger when to start and stop, the highest voltage and current it is allowed to use, and whether to charge the pack or warm it. The charger hardware does the actual constant-current / constant-voltage work internally; the VCU is the supervisor that

commands it and watches its health. To add another charger with the same settings, press the copy icon on its row. The copy opens as a new charger named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

Each charger slot does nothing until you assign a charger to it with **On-board charger**. A slot left empty (no charger selected) is simply absent — it binds no driver, raises no faults, and reports nothing. Only the Dilong charger (Dilong OBC/DC-DC v1.8 protocol) is supported today; selecting it wires up that driver.

Every active slot needs a **CAN bus** to talk on — one of the high-voltage buses the charger's command and status frames travel over. If you run two chargers of the same make on the same bus, they must not share frame IDs: their **Status message id**, **Command message id**, and **AC-input status message id** have to be set to distinct values so the VCU can tell them apart. If a second same-make charger is placed on the same bus with the same status ID as an earlier one, that later slot is refused and left absent, and a configuration fault is raised. (Two chargers on two different buses, or two different makes, never collide.)

A charger runs whenever the vehicle is in the AC Charging state, and otherwise only while the vehicle is in a state you allow under **Can be enabled in** — normally Accessory, Ready or Neutral. You never tick AC Charging yourself: the charger takes the vehicle there on its own when it sees a cord, so that permission is granted automatically.

Where the charger's output is wired matters, and you tell the vehicle with **Charge path**. Left at *direct to battery* (the usual case) the charger feeds the battery directly, so it can charge with the vehicle asleep and the main contactors open. Turned on, the output lands on the vehicle's main high voltage bus instead, so the vehicle has to bring that bus up — close the contactors — before charging can start. Getting this wrong in the "off" direction means the charger pushes into a disconnected bus and nothing charges, so set it to match the loom. On top of that, **Enable source** can require a signal before it charges at all: leave it on *Always on* to charge whenever the state allows, or pick *Digital switch* or *CAN signal* to have it wait for a charge-plug sensor or an operator switch. If you choose one and then leave the input unset, the charger will not charge and the app will tell you so before you flash. If a configured enable input later becomes unreadable — a miswired switch, or a CAN signal that stops arriving — the charger stops charging and a fault is recorded. That is the deliberate direction for a charger: safer to stop than to push current on a signal nobody can read. The charger is purely a commanded peer: it does what your settings say, but only while the vehicle is already in one of the states you permit. Driving is offered for equipment that charges on the move, such as a vehicle with an on-board generator, and is off by default.

Finally, the charger supplies all of its own telemetry — output voltage and current, temperature, AC input voltage and current, and its alarm flags — over CAN. The VCU measures none of this; it reads what the charger reports. The charger physically only puts out power while it keeps receiving the VCU's command frame once per second, and it stops a few seconds after that frame goes silent (the charger's own watchdog).

How it works

Once per control cycle (about ten times a second) the VCU builds a fresh command for each running charger and sends it. The command carries four things: start-or-stop, the charge mode, the voltage ceiling, and the current ceiling.

The start-or-stop decision is the VCU's only real control law. There is no ramp, no hysteresis, and no SOC or temperature logic in the VCU layer — just a gate on vehicle state:

```
charge = ON   while the vehicle is in AC Charging (granted automatically)
charge = ON   while the live vehicle state is one you ticked in Can be enabled in
              AND the Enable source says yes
charge = OFF  in every other state
```

When the gate says ON, the VCU commands the charger to start; when it says OFF, it commands a stop. The charge mode comes straight from **Mode**: leave it on charge to charge the pack normally, or set it to heating to run the charger in battery-heating mode instead. The mode you pick is sent on every cycle.

The two ceilings are passed to the charger as the limits it must respect:

```
commanded voltage limit = Charge voltage ceiling (V)
commanded current limit  = Charge current ceiling (A), capped at 6553.5 A
```

Charge voltage ceiling (V) is the highest voltage the charger may drive into the pack — the target it regulates to during the constant-voltage phase near full charge. **Charge current ceiling (A)** caps the current during the constant-current bulk phase. Both are sent unchanged for the charger to honour; the VCU never closes the loop on them. The current ceiling is internally limited to 6553.5 A; setting a value that would exceed that limit clamps it and raises a one-time configuration fault, so a huge entry can never wrap around to a near-zero limit.

The very first command the VCU ever sends a charger is a deliberate, explicit stop (charge off, plain charge mode, zero volts, zero amps). This is a safety default: it guarantees a charger never sees a stray "start" before the VCU has decided what it actually wants.

While a charger runs, the VCU reads two broadcasts from it, each arriving once per second. The first (named by **Status message id**) carries the charger's output voltage and current, its temperature, and its status and alarm bits — including the charge-port electronic lock status and the S2 switch. The second (named by **AC-input status message id**) carries the AC line voltage and current coming in from the wall. Output and AC-input currents are signed: a positive current means power flowing into the pack (charging), a negative current means power flowing out. Temperature is reported with a fixed −40 °C offset:

```
charger temperature (°C) = reported temperature − 40
```

The VCU watches both broadcasts for freshness, and the two are timed separately:

```
a reading goes stale if its frame has not arrived for more than 5 seconds
(5 seconds = 5 missed once-per-second frames)
```

If the main status broadcast goes stale, the output voltage, current, temperature, and status readings are marked invalid and the charger is treated as out of communication. If only the AC-input broadcast goes stale, just the AC-input readings are dropped — the charger's health verdict is unaffected, because health

is read from the main status broadcast. This means a charger can keep reporting its DC output while its AC-input reporting drops out, and that alone is not a fault.

Health resolves to one of OK, Degraded, or Fault. Loss of communication on the main status broadcast is a Fault. A hardware, over-temperature, input-voltage, charge-port-socket-over-temperature, control-pilot, or charge-connector protection alarm is also a Fault — those mean the charger has already stopped its output. A communication-timeout flag or a charge-port-lock fault is reported as Degraded instead: visible on the dash, but not output-stopping. Two charger faults — a communication loss and a charger fault — are reported into the vehicle's central fault system as they come and go, where another subsystem can escalate them into a vehicle-wide safe or degraded response. An absent or unconfigured slot simply reports "absent" and raises nothing.

At boot, before any frame has arrived, a slot reads as faulted but starts with its communication latch already "lost", so the first frame it receives reports a recovery rather than spamming a boot-time failure.

Plugging in. The charger reports the AC line voltage at its own inlet, and that is what the vehicle uses to know a cord is connected — there is nothing extra to wire or configure. When a cord appears, the charger asks the vehicle to move into the AC Charging state, and the vehicle stays there until the cord is pulled. That has three consequences worth knowing.

First, a vehicle parked and switched off will wake up to charge. It steps up to Accessory (or Ready, if the charger's **Charge path** is through the drive bus) and then into AC Charging on its own, and when the cord comes out it returns to exactly where it started — back to off if it was off, so an overnight charge does not leave the vehicle awake draining its 12 V battery. If someone is sitting in the vehicle with accessories on, it stays awake for them instead.

Second, a full pack does not end the session. Charging stops when the battery says it is full, but the vehicle stays in AC Charging while the cord is still in the socket, so it can never be driven away still tethered. Pulling the cord is what ends the session, and the vehicle then winds down in its normal order.

Third, a brief interruption does not end the session either. A momentary mains dropout or a station glitch is ridden through; the cord has to read as genuinely gone for about five seconds before the session is closed.

If a cord is connected but the vehicle does not start charging within about thirty seconds, that is recorded as a diagnostic. It usually means the high voltage bus could not be brought up on a drive-bus-wired charger, or something else is holding the vehicle out of a charging state. A charger whose make cannot report its AC input simply never triggers any of this, and behaves exactly as it always did — charging whenever **Can be enabled in** allows.

Safe shutdown. Whenever the vehicle drops into its fault-safe or shutting-down state, the VCU immediately commands every charger to stop — charge off, zero volts, zero amps — regardless of what **Can be enabled in** allows, and it does this before the high-voltage contactors are opened. The charger is never relied upon to coast; it is told to stop first.

Tuning notes

Set **On-board charger** to the charger you actually fitted, pick the **CAN bus** it is wired to.

Set **Charge path** to match how the charger is actually wired. *Direct to battery* is the usual answer and lets the vehicle charge while asleep. Turn it on only if the output lands on the main high voltage bus; the app will then also require both main contactors to be configured, because the vehicle has to close them before it can charge.

For **Can be enabled in**, the defaults — Accessory, Ready and Neutral — suit a vehicle that charges while parked and plugged in. Leave Driving off unless the vehicle genuinely charges on the move. You do not need to tick anything for ordinary plug-in charging: AC Charging is granted on its own.

Set **Charge voltage ceiling (V)** to the pack's specified top-of-charge voltage and **Charge current ceiling (A)** to the lower of what the charger and the pack can sustain. These are hard limits the charger regulates to — keep them within the pack manufacturer's charge envelope. If you enter an implausibly large current and see a configuration fault, that is the clamp protecting you; lower it to a real value.

Use **Mode** to choose ordinary charging or battery heating. Leave it on charge for normal operation; use heating only when the pack needs to be warmed before or during charging in cold conditions, per the pack's guidance.

When you run more than one charger of the same make on a single **CAN bus**, the only extra work is giving each one its own **Command message id**, **Status message id**, and **AC-input status message id** so they don't talk over each other. Leaving any of these at the default uses the charger's protocol-default ID, which is fine for a single charger but will collide if two same-make chargers share a bus — set them distinct in that case.

Settings

- **On-board charger** — The on-board charger fitted to this slot, if any.
- **CAN bus** — Which CAN bus this charger communicates on.
- **Mode** — Whether this unit charges the battery or heats it.
- **Charge voltage ceiling** (V) — The maximum charging voltage handed to the charger.
- **Charge current ceiling** (A) — The maximum charging current handed to the charger.
- **Can be enabled in** — The vehicle states in which this charger runs, on top of charging itself. Once a cord is plugged in the vehicle moves into AC Charging on its own and the charger always runs there, so this list is about the other states. The usual set is parked — Accessory, Ready, Neutral. Driving is offered for equipment that charges on the move, such as a vehicle with an on-board generator, and is off by default.
- **Charge path** — Where this charger's high-voltage output is wired, which decides what has to close before it can charge.

"Direct to battery" is the usual wiring and the default: the output lands on the battery ahead of the main contactors, so nothing closes and a parked car charges with the drive bus dead.

"Through a charge contactor" adds a dedicated contactor in the charger's output. That one closes for charging, the main contactors stay open, and the drive bus still stays dead. Name it under Precharge & sequence.

"Through the drive bus" is for a charger wired to the main high-voltage bus instead of the battery. The vehicle brings the bus up, precharge and all, before charging starts.

Dilong settings

- **Status message id** — Incoming status message id. Zero uses the vendor default.
- **Command message id** — Outgoing command message id. Zero uses the vendor default.
- **AC-input status message id** — Incoming AC-input status message id. Zero uses the vendor default.
- **Enable source** — What turns the charger on, on top of the vehicle states above. "Always on" runs whenever the vehicle state allows — the usual setting. "Digital switch" and "CAN signal" gate it further, so the charger only charges when that input says so. "Script" hands the decision to a recipe or script; with it selected the charger stays off until something enables it, including if that automation never runs.

Digital switch settings

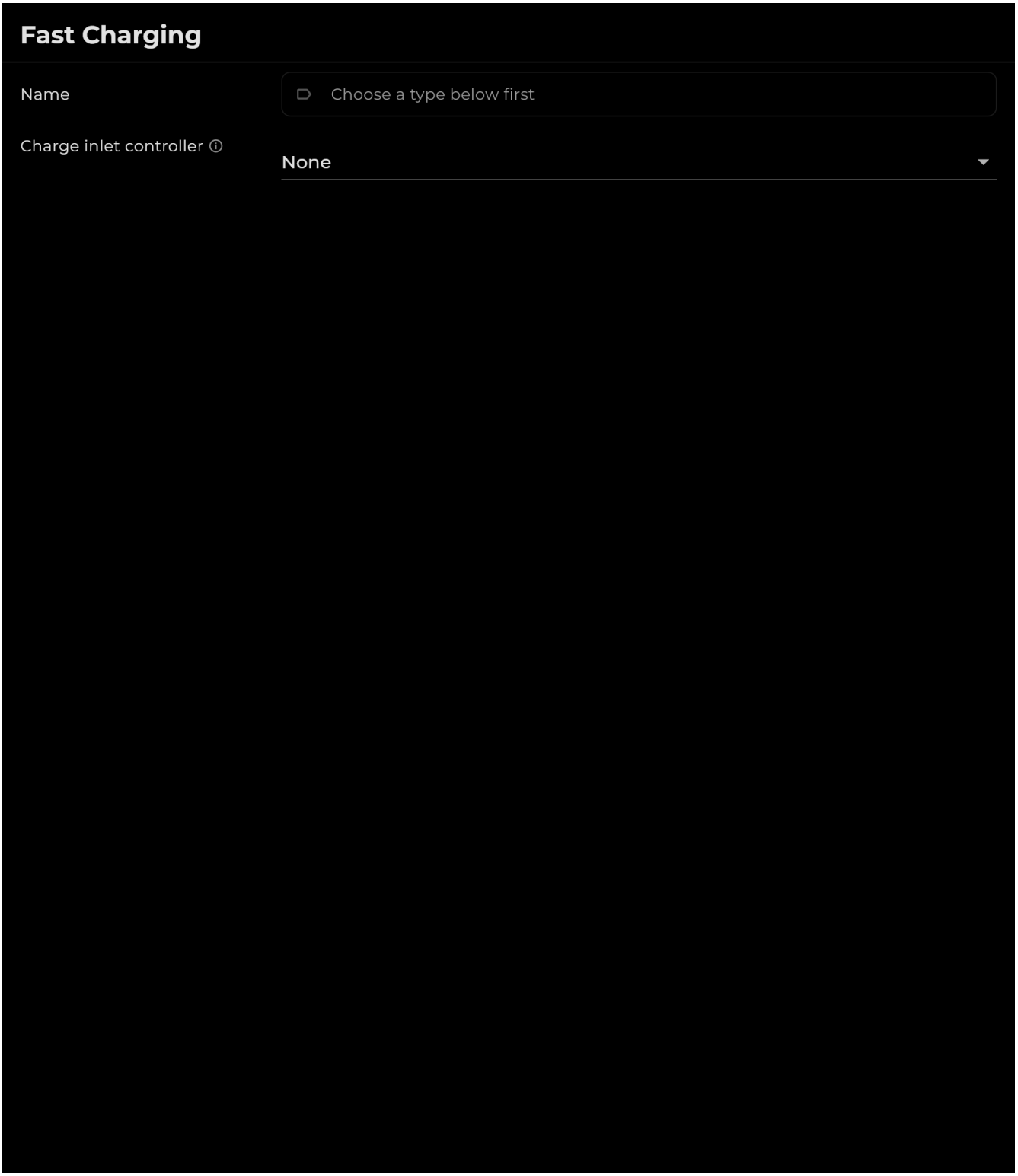
- **Enable switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.
- **Enable input** — The digital input wired to the enable switch.

CAN signal settings

- **Enable signal** — The CAN signal whose value enables the charger.
- **Enable threshold** — Enabled while the CAN signal reads at or above this value.

Fast Charging

The charge inlet controller that handles DC fast charging at a charging station.



Fast charging is what happens when the vehicle is plugged into a DC charging station — the tall roadside units that charge in minutes rather than hours. It works completely differently from AC charging. With AC, the on-board charger inside the vehicle does the work of turning mains power into something the battery can take. With DC, the charging station does that work itself and pushes high-voltage DC straight into the pack, which is why it is so much faster.

For that to happen safely, the vehicle and the station have to hold a real conversation: how much voltage and current the pack can take, whether the cable is safe to energise, when to lock the plug in, and when to stop. That conversation runs over the charging cable itself and is handled by a dedicated box — the **charge inlet controller**. The VCU cannot do this job itself: the control wires in the charge port are not reachable from the vehicle controller, so a separate controller is required for DC fast charging.

This whole section is optional. A vehicle with no DC charge port simply leaves **Charge inlet controller** set to None, and nothing here applies. Fitting one does not change AC charging, which is configured separately under On-Board Chargers.

What it needs

Choose the controller you actually fitted under **Charge inlet controller**, and the **CAN bus** it is wired to — this is the bus the controller and the VCU talk to each other on, not the charge cable.

Set the two ceilings, **Fast-charge voltage ceiling** and **Fast-charge current ceiling**. These are the most the vehicle will ever ask a charging station for. They are limits, not targets: the battery's own live limits still apply, and whichever is lower at any moment is what gets requested. Both must be set to a real value — a ceiling left at zero asks the station for no power at all, so charging would never start.

Charging may start in picks the vehicle states a charging session is allowed to begin in. Tick at least one, or this vehicle will never fast-charge. The choice is Accessory or Ready, and you can allow both. Accessory is the ordinary one: parked, switched on, but with the high voltage system not yet brought up. You do not need to say anything about the charging states themselves: once a session is under way the vehicle moves into a charging state on its own and stays permitted there. The controller is talked to in every state regardless, so the vehicle always notices a plug going in.

How it works

When a cable is plugged in, the controller notices the connection and works out what kind of charging the station offers. If the vehicle permits it, the controller negotiates the session, locks the plug into the port, and asks the station to bring its output up to match the pack's voltage before anything is connected through. Only once the two sides are close enough does power actually flow.

Throughout, the vehicle keeps telling the controller what the pack can take. It combines the battery's live limits with the ceilings you set here and passes on whichever is more conservative. It also reports the pack's present voltage, current and state of charge, which the station needs to charge safely.

Before a station will deliver power it runs its own insulation test on the cable, and it needs the vehicle to stand down its own insulation monitoring while that happens. **Allow isolation monitoring to be suspended** is how you grant that. It is off by default and deliberately so — with it off, the vehicle refuses to fast-charge and reports why, rather than quietly claiming a safety condition it cannot confirm. Turn it on once you have satisfied yourself the battery pack tolerates it.

Two safety behaviours are worth knowing. First, the plug stays locked until the charge port has genuinely gone dead — the vehicle will not release it while voltage is still present, and will not release it at all if it cannot read the port voltage. Refusing to unlock is the safe direction; every controller provides a manual release for the rare case where you need one.

Second, when a charging session ends — normally or because something went wrong — the order matters. The controller's own charging contactors must open before the vehicle's main battery contactors do; doing it the other way round can damage them. The vehicle waits for the controller to confirm it has opened, up to the **Shutdown wait** you set, and then opens anyway rather than waiting forever. Leave this at the controller manual's recommended value.

If a charging fault does occur, the vehicle stays driveable. A failed charging session should never strand a vehicle on a charger with a half-full battery, so charging faults degrade rather than shut the vehicle down. Genuine high-voltage safety problems — a broken interlock, or voltage present at an unlocked charge port — are treated as the serious faults they are.

Tuning notes

Set the two ceilings from the lowest of three things: what the pack is rated to accept, what the charge cable and inlet are rated for, and how hard you are willing to charge this particular vehicle. Charging below the pack's rating is a perfectly reasonable choice for battery longevity or for a lighter cable.

Leave **Allow isolation monitoring to be suspended** off until you have checked the battery pack's documentation. If DC charging never starts and the vehicle reports a charging configuration fault, this setting is the first thing to check.

For **Charging may start in**, Accessory alone is a sensible starting point — the vehicle can begin a session while parked. If the charge port feeds the vehicle's main high voltage bus rather than the battery directly — the **Needs the drive bus live** setting above — then Ready is required, not optional: a session can only start once that bus is already up, which is what Ready means.

Settings

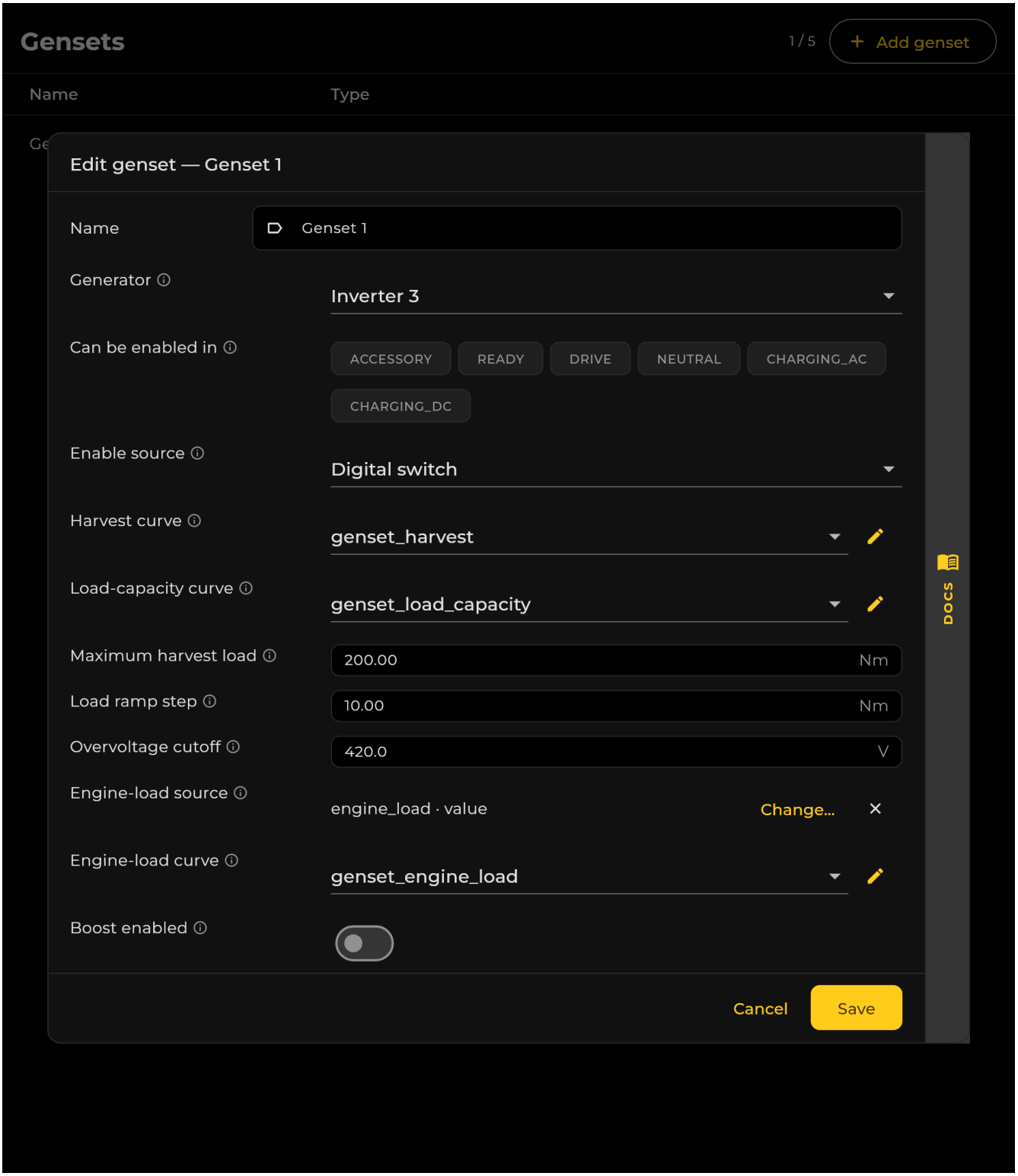
- **Charge inlet controller** — The fast-charge inlet controller fitted to this vehicle, if any. This is the box that talks to the charging station through the charge port and manages the port latch and its temperature sensors. Leave it as None on a vehicle without DC fast charging.
- **CAN bus** — Which CAN bus the controller communicates on.
- **Needs the drive bus live** — Turn this on when the charge port connects to the vehicle's main high voltage bus rather than straight to the battery. The vehicle then brings the bus up before a charging session may start.
- **Allow isolation monitoring to be suspended** — A DC charging station has to run its own insulation test before it will deliver power, and it needs the vehicle to stand down its own insulation monitoring while it does. Turn this on only once you have confirmed the battery pack tolerates that. Left off, the vehicle refuses to fast-charge and reports why, rather than claiming something it cannot verify.
- **Fast-charge voltage ceiling** (V) — The highest voltage this vehicle will ask a DC charging station for. The battery's own limit still applies — whichever is lower wins.
- **Fast-charge current ceiling** (A) — The highest current this vehicle will ask a DC charging station for. Use it to charge below what the battery could take — for a lighter charge cable, a smaller inlet, or to be gentler on the pack. The battery's own limit still applies; whichever is lower wins.
- **Shutdown wait** (ms) — How long to wait for the charge controller to open its own charging contactors before the vehicle opens the main contactors anyway. Opening the main contactors first can damage

the charging contactors, so the vehicle waits — but not forever. Only raise this if the controller's manual asks for it.

- **Charging may start in** — The vehicle states a fast-charge session is allowed to begin in. Pick at least one, or this vehicle will never fast-charge. Once a session is under way the vehicle moves itself into a charging state and stays permitted there, so there is nothing further to allow. The controller is talked to in every state regardless, so a plug going in is always noticed.

Gensets

Engine-driven generators that charge the high-voltage battery on the move (series-hybrid range extenders). For each, choose the generator inverter it loads, how the engine is controlled, and when it may run.



A genset is an engine-driven generator that charges the high-voltage battery while the vehicle is moving — a range extender for a series-hybrid build. The engine never turns the road wheels. Instead it spins a generator, and the VCU uses the **Generator** inverter as a brake on that generator's shaft: the harder it brakes, the more mechanical power it pulls off the engine and turns into electrical power that flows into the high-voltage pack. So a genset only ever *takes* power from a spinning shaft to charge the battery; it can

never add power to the drivetrain. You can configure more than one genset, each with its own engine and its own generator. To add another genset with the same settings, press the copy icon on its row. The copy opens as a new genset named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

A genset only harvests when every one of these is true:

- The high-voltage bus is live. This is a hard gate — no high voltage, no harvest, ever.
- A **Generator** is selected and resolves to a generator-role inverter slot. The genset reads the engine's speed and the DC-link voltage *from this inverter* — the engine itself has no speed sensor, so the generator inverter is the only source of shaft rpm and DC-link voltage in the whole system. If the **Generator** doesn't resolve, or points at a traction (road-drive) motor instead of a generator, the genset shuts itself off entirely, commands nothing, and logs a configuration fault. This refusal is deliberate: it guarantees harvest braking can never be sent into a wheel motor.
- A live battery management system (BMS) that reports both a valid state of charge (SOC) and a valid charge-current limit. Harvest is required to see the battery — if either reading goes stale, commanded load drops to zero and a fault is logged.
- The current vehicle state is one you allowed under **Can be enabled in**. Outside that set, the genset is held off.
- The configured enable is asserted (see below).

For the VCU to actually start and run the engine, **Engine control** must be set to the controllable (generic) mode and you must wire up at least a run/ignition output, and normally a throttle output. If **Engine control** is set to the uncontrollable mode, the operator owns the engine — the VCU never cranks, runs, or throttles it; it just harvests whenever the engine is spinning on its own.

How it works

Every cycle (about ten times a second), each genset resolves one enable decision, decides whether the engine should run, manages cranking and throttle, then works out how hard to brake the generator.

Enable. **Enable source** picks where the on/off command comes from: always-on (enabled whenever state-allowed), a digital input, a CAN signal, or a script. For a digital input it reads the pin named by **Enable input**, and **Enable switch type** says whether that switch is normally open or normally closed. For a CAN signal it reads **Enable signal** and treats the genset as enabled when that value is at or above **Enable threshold**; an unreadable signal counts as off. The script source is for installer logic and is only honored on a genset whose **Enable source** is set to script. The resolved enable, together with the **Can be enabled in** gate, hard-gates everything: if either is false, the engine is commanded off and harvest stops.

When the engine runs. If **Auto start/stop on battery level** is off, a controllable engine simply runs whenever it is enabled and state-allowed. Turn it on to make the genset charge-sustaining: the engine starts when SOC falls to or below **Start below (%)** and stops when SOC rises to or above **Stop above (%)**. Between those two numbers the genset holds whatever it was doing, so it won't chatter on and off around one threshold. Set **Stop above (%)** comfortably higher than **Start below (%)** to get a clean deadband. If SOC is enabled-but-unreadable, the engine is held off — the genset will not charge blind.

Cranking and governing the engine. With **Engine control** in controllable mode, the VCU runs a simple start sequence driven entirely by the shaft speed it reads back from the generator inverter:

- From stopped, if the shaft is already turning at or above **Catch speed (rpm)**, the engine is taken to be already running and adopted as-is — no starter is engaged, so there's never a starter clash with a live engine.
- Otherwise it asserts the run/ignition output and engages the starter (if one is configured), then cranks. If the shaft reaches **Catch speed (rpm)** it's caught and goes to running. If **Crank time (ms)** elapses without a catch, it tries again, up to **Crank retries** extra attempts. If those are exhausted it stops, drops ignition, and logs a crank-failed fault that stays latched until the run request is dropped.
- While running, if the shaft falls below the stall threshold it re-cranks from scratch. The stall threshold is **Stall speed (rpm)**, or, if that's left at zero, half of **Catch speed (rpm)**:

```
stall threshold = Stall speed, or Catch speed ÷ 2 if Stall speed is 0
```

While the engine is running, throttle is set by a proportional governor; in every other state throttle is zero. The governor pushes throttle up the further the engine is below its target:

```
speed error = Governor target speed − shaft rpm
throttle = Idle throttle + speed error × Governor gain ÷ 100
clamp throttle to between Idle throttle and Maximum throttle
```

Governor gain (%) is how much extra throttle is added per 100 rpm of under-speed. **Idle throttle (%)** is the floor — when the engine is at or above target, throttle is pulled back toward idle but never below it — and **Maximum throttle (%)** is the ceiling.

Output wiring. Three engine outputs are configured the same way, each with a channel type, output pin, CAN signal, and carrier frequency. If a relay board or driver stage between the controller and the load turns it on when the pin is driven low, tick **Invert this output** under that pin — the tick belongs to the pin, so it applies whatever is wired there. The run/ignition line uses **Run/ignition channel**, **Run output** (the pin), **Run signal** (CAN, reserved for future use), and **Run PWM frequency (Hz)** if ever driven as a PWM line — it is normally a plain on/off output, held on while cranking or running and dropped when stopped or faulted. The starter uses **Starter channel**, **Starter output**, **Starter signal**, and **Starter PWM frequency (Hz)**; if **Starter channel** is set to none, the genset does a passive crank — it holds ignition but never drives a starter, for pull-start or self-cranking engines. The throttle uses **Throttle channel**, **Throttle output**, **Throttle signal**, and **Throttle PWM frequency (Hz)**; throttle is normally a PWM output, and if **Throttle output** or **Throttle PWM frequency (Hz)** is left at zero no throttle signal is sent.

How hard it brakes the generator (the harvest load). When harvest is allowed, the genset works out a target braking torque and then trims it under several ceilings. The starting target comes from **Maximum harvest load (Nm)** shaped by the **Harvest curve**, a table that maps SOC to a fraction of that maximum:

```
harvest target = Maximum harvest load × Harvest-curve fraction at this SOC
```

With no **Harvest curve**, the target is the full **Maximum harvest load (Nm)** at any SOC. That target is then capped by the smallest of these ceilings:

- The **Load-capacity curve**, a table mapping shaft rpm to the fraction of maximum the engine can hold at that speed — shape it from the engine's torque curve. Its low-rpm region is the anti-lug protection: low rpm yields a low fraction, so the load is light at low engine speed and can't bog the engine down:

```
load-capacity limit = Maximum harvest load × Load-capacity-curve fraction at this shaft rpm
```

- The battery's charge-current limit, converted from electrical power into a shaft torque. Pulling more battery charge current at a given DC-link voltage is more electrical power, and more power at a given shaft speed is more braking torque:

```
shaft speed (rad/s) = shaft rpm × 3.1416 ÷ 30
charge power limit (W) = DC-link voltage (V) × battery charge-current limit (A)
charge limit (Nm) = charge power limit (W) ÷ shaft speed (rad/s)
```

Below about 50 rpm this ceiling is ignored (it would divide by a near-zero speed); the load-capacity curve and crank logic own that low-speed regime. If the charge-current limit is invalid, this ceiling forces zero load.

- The optional **Engine-load curve**, a table mapping the engine's own percent-load to a fraction of maximum, so harvest eases off as the engine nears its own capacity. This needs an **Engine-load source** to supply the percent-load reading; with no source or no curve, this ceiling simply doesn't apply:

```
engine-load limit = Maximum harvest load × Engine-load-curve fraction at this engine %load
```

The genset takes the smallest of those ceilings and then the smaller of that and the target:

```
ceiling = smallest of (charge limit, load-capacity limit, engine-load limit)
commanded load = smaller of (harvest target, ceiling)
```

Finally, **Load ramp step (Nm)** limits how fast the commanded load may *rise* each cycle, easing load onto the shaft instead of slamming it on:

```
if commanded load > last cycle's load + Load ramp step:
    commanded load = last cycle's load + Load ramp step
```

Decreases are never rate-limited — shedding load is always the safe direction, so the genset can back off instantly. With **Load ramp step (Nm)** at zero there's no ramp at all.

Safety cutoffs. Two cutoffs act immediately, with no ramp. **Overvoltage cutoff (V)** is an independent overcharge backstop: if the DC-link voltage reaches it, load is forced to zero and latched off, and a fault is raised, until harvest is disabled once to clear the latch. Set it at zero to disable the backstop. Separately, if the battery readings go stale while harvest is wanted, load drops straight to zero and a degrade-class fault is logged, clearing automatically when the battery returns. The genset reports running/enabled/faulted status plus live shaft rpm, harvested power and current, and the boost state (whether more power is wanted, and the live governed speed) to the dashboard for read-only monitoring.

Boost — more power in a time of need. Boost raises a controlled engine from its efficient governor speed to **Boost engine speed (rpm)** — a louder, thirstier operating point that harvests more power (the same braking torque at more shaft speed is more electrical power). The **Boost Enabled** switch turns the feature on — off, every other boost setting is ignored. **Boost sources** then picks what may request it, and any enabled source can only *raise* the operating point, never lower it: the battery-level source asserts whenever SOC is at or below **Boost below (%)**; the battery-drain source watches net battery current and asserts when the pack drains at or above **Drain to enter boost (A)** for **Drain persistence (ms)**, releasing once drain falls to or below **Drain to leave boost (A)** for the same persistence; and the script source lets installer logic or a Recipe request it. The governed speed *walks* between the two operating points at **Boost walk rate (rpm/s)** — while the drain sits between the two thresholds the speed holds where it settled, so the engine finds the speed whose harvest covers the load instead of bouncing between two notes. Boost never weakens a limit: the battery's charge-current limit and the overvoltage backstop apply at all times. On a genset whose engine the vehicle does not control there is no speed to raise — boost is then just the "more power wanted" indication, which a script or Recipe can turn into an operator lamp.

Tuning notes

Start with the run conditions: pick the vehicle states under **Can be enabled in** where range-extending is appropriate, set **Enable source**, and decide whether to charge-sustain with **Auto start/stop on battery level**. For charge-sustaining, set **Start below (%)** to where you want the engine to kick in and **Stop above (%)** a healthy margin above it so it runs long enough to be worth starting.

Tune the engine before tuning the harvest. Set **Catch speed (rpm)** just above a confirmed firing speed and **Stall speed (rpm)** below normal idle so a real stall is caught but normal speed dips aren't. Give **Crank time (ms)** enough to catch a cold engine and a couple of **Crank retries**. Then tune the governor: set **Governor target speed (rpm)** to the engine's best generating speed, **Idle throttle (%)** to a stable no-load idle, **Maximum throttle (%)** to a safe upper limit, and raise **Governor gain (%)** until the engine holds its target under load without hunting — back it off if speed oscillates.

Tune the harvest last, because it leans on the engine being well-behaved. Set **Maximum harvest load (Nm)** to the most braking torque the engine can take, then shape it with the **Harvest curve** (charge harder when the pack is low, taper as it fills). Use the **Load-capacity curve** to tell the genset how much load the engine can hold at each speed — shape it from the engine's torque curve; its low-speed region keeps load light until the engine is up to speed, your main protection against bogging the engine on load pickup. Add the **Engine-load curve** with an **Engine-load source** only if your engine reports its own load and you want harvest to back off as it nears capacity. Use **Load ramp step (Nm)** to soften how quickly load comes on; lower it if load pickup is rough. Finally, set **Overvoltage cutoff (V)** as a hard backstop above the highest DC-link voltage you ever expect to see in normal charging — it should never fire in healthy operation, only catch a genuine overcharge.

If the build needs more charging in a pinch — a trailer, a long grade, a winch — enable boost last: turn on **Boost Enabled**, then set **Boost engine speed (rpm)** to the engine's higher power point (at or above the governor target; the Configurator checks this), pick **Boost sources**, and start with a generous **Drain persistence (ms)** (a second or two), **Drain to enter boost (A)** above any brief accessory surge, and **Drain**

to leave boost (A) well below it so the speed settles instead of hunting. A moderate **Boost walk rate (rpm/s)** (tens of rpm per second) keeps the change civil in the cabin.

All fitted generators share one battery charging allowance. Regenerative current from drive and auxiliary motors has priority. If one generator needs less than its share, the others can use the remaining capacity. Missing current feedback from a fitted drive or auxiliary motor stops generation. Pump or drive consumption does not increase the allowance.

Generation commands can decrease every 10 ms. Engine control and load increases operate every 100 ms. The physical generator responds after communication and inverter delays, so these command intervals do not establish permitted battery transients.

Settings

- **Generator** — The inverter, in Generator role, whose shaft this engine spins. The genset loads this inverter to harvest power into the battery.
- **Can be enabled in** — Which vehicle states the genset may run and harvest in. Charge-at-rest is "Ready"; charge-while-driving is "Drive".
- **Enable source** — What turns the genset on. "Always on" runs whenever the vehicle state allows; "Digital switch" / "CAN signal" gate on an operator control; "Script" lets an installer script request it.
- **Harvest curve** — The calibration curve mapping battery charge to how hard to harvest — harder when low, easing off as the pack fills. Pick an existing curve or create one.
- **Load-capacity curve** — The calibration curve mapping engine shaft speed to the load the engine can hold at that speed — shape it from the engine's torque curve. Its low-speed region is the anti-lug protection: load backs off as the engine slows, so it is never stalled. Leave unset for none.
- **Maximum harvest load** (Nm) — The largest braking load applied to the generator at full harvest.
- **Load ramp step** (Nm) — How much the harvest load may rise per update, easing load onto the engine so it is never slammed. Larger is faster.
- **Overvoltage cutoff** (V) — An independent safety backstop: if the DC-link voltage reaches this, harvest load drops to zero at once, regardless of the battery limits.
- **Engine-load source** — Optional. Where the engine's %load reading comes from, fed through the engine-load curve above. Pick a source and the quantity to read, or leave it as “none” if the engine reports no load.
- **Engine-load curve** — The calibration curve mapping the engine's reported %load to a load ceiling, so we never overwork the engine. Used only with the engine-load input below. Leave unset for none.
- **Boost enabled** — Let the genset raise itself to a boost operating point — less fuel- efficient, higher-power — for a time of need (low battery, a trailer, a long hill). Off, every boost setting below is ignored. On an engine the vehicle does not control, boost is only the "more power wanted" indication.
- **Boost sources** — What may request boost. Any enabled source can only raise the operating point, never lower it. At least one must be ticked while boost is on.
- **Boost below** (%) — The battery level at or below which boost is requested, regardless of the drain reading. Used by the "Battery level" source.

- **Boost engine speed** (rpm) — The engine speed governed to at full boost. Must be at or above the normal governor target; the engine speed walks smoothly between the two as demand comes and goes. Only a controlled engine has a speed to govern.
- **Boost walk rate** (rpm/s) — How fast the governed engine speed moves between the normal and boost points. Zero steps immediately.
- **Drain to enter boost** (A) — Net battery discharge at or above which the "Battery drain" source asserts boost — the vehicle is using more than the genset harvests.
- **Drain to leave boost** (A) — Net battery discharge at or below which the "Battery drain" source releases boost. Must be below the enter threshold; between the two the engine speed holds where it settled.
- **Drain persistence** (ms) — How long the drain must persist past a threshold before boost engages or releases, so a momentary surge does not rev the engine. Zero reacts immediately.
- **Engine control** — How the engine is controlled. "None" is an engine we cannot command (we only apply generator load to harvest); "Generic" drives start/stop and an optional throttle through the channels below.
- **Auto start/stop on battery level** — Start the engine when the battery falls to a low level and stop it once charged, instead of running whenever enabled. Only applies to a controlled engine.
- **Start below** (%) — Start the engine when the battery charge falls to this level.
- **Stop above** (%) — Stop the engine once the battery charge reaches this level.

Digital switch settings

- **Enable switch type** — Whether the enable switch's contacts are closed or open at rest. Most switches are normally open — they close when operated.
- **Enable input** — The digital input wired to the genset run switch.

CAN signal settings

- **Enable signal** — The CAN signal whose value enables the genset.
- **Enable threshold** — The genset is enabled when the CAN signal reads at or above this value.
- **Run/ignition channel** — How the run/ignition line is driven while the engine should run.

Digital settings

- **Run output** — The output that holds the engine's run/ignition line on.

CAN settings

- **Run signal** — The CAN signal that commands the engine to run.

PWM settings

- **Run PWM frequency** (Hz) — Carrier frequency when the run line is a PWM output.
- **Starter channel** — How the starter is engaged during cranking.

Digital settings

- **Starter output** — The output that engages the starter motor while cranking.

CAN settings

- **Starter signal** — The CAN signal that engages the starter.

PWM settings

- **Starter PWM frequency** (Hz) — Carrier frequency when the starter line is a PWM output.
- **Throttle channel** — How the engine throttle (governor command) is driven. Leave "None" for a fixed-speed engine that governs itself.

PWM settings

- **Throttle output** — The PWM output that sets the engine throttle.

CAN settings

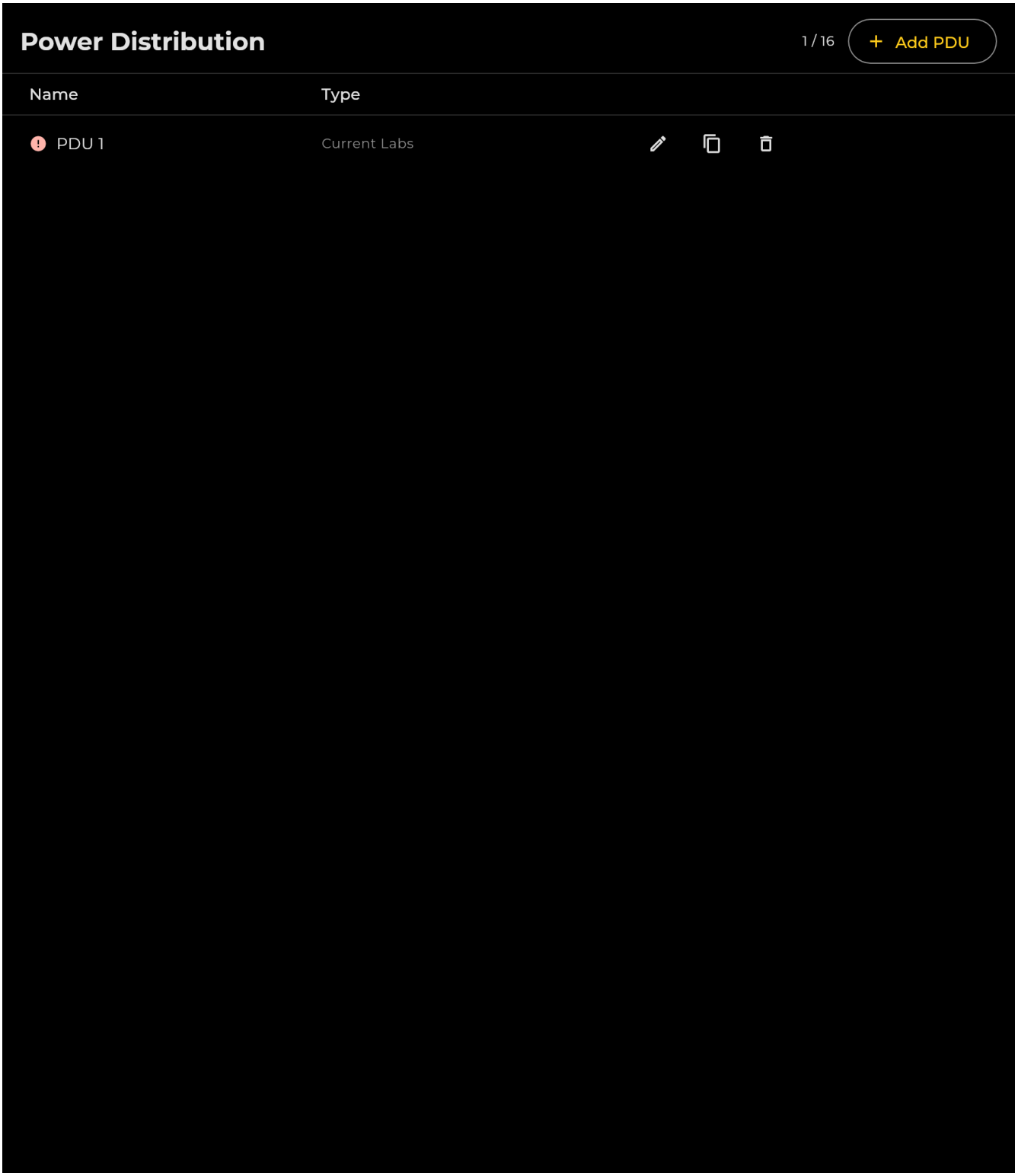
- **Throttle signal** — The CAN signal that sets the engine throttle.

PWM settings

- **Throttle PWM frequency** (Hz) — Carrier frequency of the throttle PWM output.
- **Catch speed** (rpm) — Shaft speed at or above which the engine is considered started (caught).
- **Stall speed** (rpm) — A running engine below this speed is treated as stalled. 0 = half the catch speed.
- **Crank time** (ms) — How long each cranking attempt lasts before retrying or giving up.
- **Crank retries** — Extra cranking attempts after the first before reporting a no-start.
- **Governor target speed** (rpm) — The engine speed the throttle governor aims to hold while running.
- **Governor gain** (%) — How strongly the throttle reacts to a speed error. Higher is more aggressive.
- **Idle throttle** (%) — The throttle applied at the governor's lower limit.
- **Maximum throttle** (%) — The highest throttle the governor may command.

Power Distribution

CAN power distribution units whose switched channels feed lights, pumps, and accessories.



A power distribution unit (PDU) is a box of electronically switched, fused-by-software outputs — a modern replacement for a bank of relays and fuses. Each unit connects to the vehicle over CAN and provides eight switched channels that feed loads like work lights, pumps, compressors, and other accessories. The vehicle keeps each unit's command messages flowing continuously; the unit turns everything off by itself if those commands ever stop, so a wiring or communication failure fails safe. To add another unit with the

same settings, press the copy icon on its row. The copy opens as a new unit named 'Name - Copy'. Its pins start empty, because a pin does one job.

What it needs

Assign a unit to a slot with **Power distribution unit** — the Current Labs PDU is the supported unit today. A slot left empty is simply absent: nothing is sent, nothing is reported. Pick the **CAN bus** the unit is wired to and set **Unit address** to the address configured on the unit's DIP switches (1 to 8). Two units on one bus must have different addresses.

Then set **Channel current limits** — one limit per channel, in amperes. A channel's limit is a wiring protection, like choosing a fuse: set it at or below what the channel's wiring can carry. Channels 1, 4, 5 and 8 are rated 20 A; channels 2, 3, 6 and 7 are rated 10 A. A limit of 0 marks the channel unused — it is held off at all times and cannot be switched on from anywhere.

How you drive the channels

Channels are not switched from this form. Each channel you give a limit automatically appears as a **PDU Channel** device in the Device registry (Pins & Devices) — named after its unit, ready to rename to what the wire feeds — and from there it behaves exactly like any output: recipes and scripts turn it on and off, dim it with a duty cycle, and read back its measured current and its condition (open circuit, current limit exceeded, or short circuit). All the ready-made recipes that drive an output — a thermostat, a light switch, a dimmer, a beeper — work on a PDU channel unchanged.

A script can also lower a channel's current limit at runtime — for example to derate a circuit that runs hot — but it can never raise it above the value you set here, and if the script system restarts, every channel returns to the state configured on this form: off, with its full configured limit.

What the vehicle watches

Each unit reports every channel's measured current and condition several times a second. If a configured unit stops reporting for about a second, it is marked offline and a diagnostic event is recorded — the unit itself has already switched its channels off by then. A channel reporting an open circuit, an exceeded current limit, or a short circuit records a diagnostic event too. These events are informational: if a channel matters enough to warrant a warning lamp or a stronger reaction on a particular vehicle, wire that up with a warning light trigger or a recipe reading the channel's condition.

Tuning notes

Set each used channel's limit to the lowest value that reliably carries the load — the unit cuts the channel the moment its limit is exceeded, which is what protects the wiring. Leave unused channels at 0. If you dim a channel with a duty cycle, size the limit for the peak current, not the average.

Settings

- **Power distribution unit** — The CAN power distribution unit fitted to this slot, if any. Its switched channels then appear as PDU Channel devices you can drive from recipes and scripts like any output.
- **CAN bus** — Which CAN bus this unit communicates on.

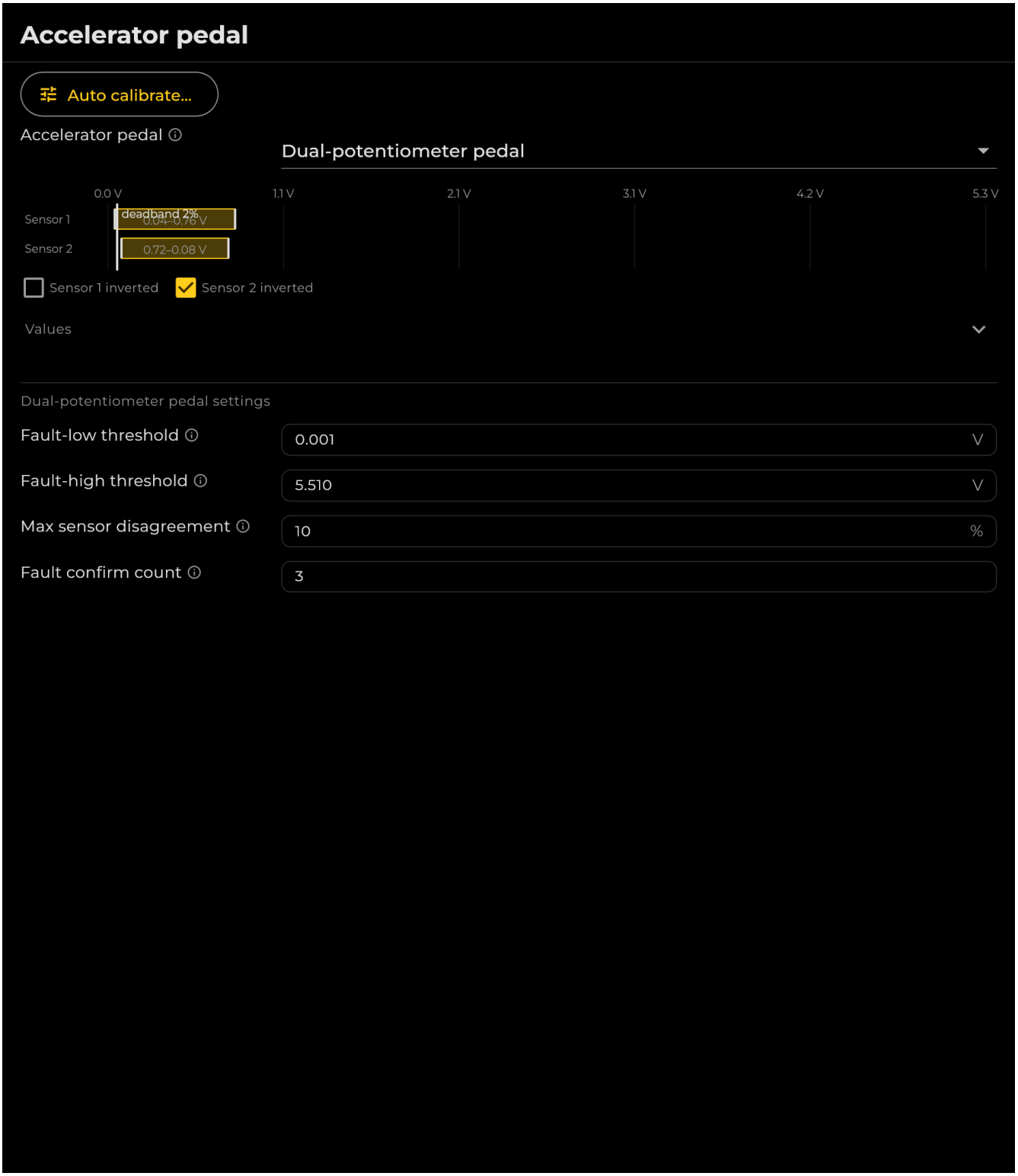
- **Unit address** — The unit's address on the bus, 1 to 8, as set by its DIP switches. Two units on one bus must have different addresses.
- **Channel current limits** (A) — Each channel's current limit, in amperes — a wiring protection, like a fuse rating. A limit of 0 marks the channel unused; it is held off at all times. Scripts can lower a limit at runtime but never raise it above this value. Channels 1, 4, 5 and 8 are rated 20 A; channels 2, 3, 6 and 7 are rated 10 A.

Pedal & Brake

How the accelerator and brake pedals are sensed and calibrated.

Accelerator pedal

Calibrate the accelerator pedal: the resting and full-travel reading of each sensor, the deadband, and the safety thresholds. Drag the bands on the graph to set the sensor ranges and the deadband.



The accelerator pedal calibration tells the controller how to read the driver's foot. The pedal carries two independent position sensors, and this screen sets, for each one, the voltage it produces with the pedal released and the voltage it produces at full travel. From those endpoints the controller turns each sensor's live voltage into a travel percentage (0% = foot off, 100% = floored), cross-checks the two sensors against each other, watches for broken or shorted wiring, and produces a single trustworthy **Driver Demand** the

rest of the drivetrain follows. The pedal reading is read fresh 100 times a second. This system only watches and reports: if anything looks wrong it commands zero pedal demand, but it never opens contactors or changes the vehicle's drive mode on its own — the torque arbiter is what acts on the demand it produces.

What it needs

The pedal uses two redundant accelerator-position sensors, Sensor 1 and Sensor 2, each a dual-track ASIL-rated unit with its own 5 V supply and its own ground. Both are wired to the VCU's accelerator-pedal inputs. You must give the controller a valid rest and full-travel voltage for at least Sensor 1, plus the electrical fault window and the safety thresholds described below.

The wiring style is inferred from the calibration — there is no separate "pedal type" picklist:

- **Dual-sensor (the normal, redundant setup):** both Sensor 1 and Sensor 2 have a real rest-to-full span. The two are cross-checked every reading. This is the only configuration that carries the redundant-sensor ASIL claim.
- **Single-sensor:** Sensor 2 is ignored entirely, however its boxes happen to be filled in. The pedal still works and is still range-checked, but there is no second sensor to cross-check against, so the dual-channel safety claim is forfeited. The controller never invents a second reading to cover the gap.
- **No pedal:** the build has no accelerator at all (a pump or generator rig). The pedal is reported as unconfigured and raises no fault.

Which of the three you have is something you **declare**, with the **Accelerator pedal** choice at the top of this screen — it is never guessed from whether the calibration boxes happen to be filled in. So a vehicle with no pedal says so outright, and a two-sensor pedal whose second sensor was never calibrated is a plain error you get told about, instead of quietly dropping to single-sensor and giving up the redundancy.

A sensor's full-travel voltage is allowed to be *lower* than its rest voltage — that is a normal "inverted" sensor whose voltage falls as the pedal is pressed, and it is fully supported. Many two-sensor pedals are built exactly this way, with the second sensor running opposite the first. Tick **inverted** under the graph for that sensor and its two readings swap over; you can also just type them in that order. The only thing that is not allowed is a rest voltage exactly equal to the full-travel voltage (no span), which is treated as a broken calibration.

How it works

Each sensor's voltage is converted to a travel percentage using its two endpoints:

```
Sensor 1 travel % = (Sensor 1 reading - Sensor 1 at rest) / (Sensor 1 at full travel -
Sensor 1 at rest) x 100
Sensor 2 travel % = (Sensor 2 reading - Sensor 2 at rest) / (Sensor 2 at full travel -
Sensor 2 at rest) x 100
```

Each result is clamped to the 0%–100% range, so a reading slightly past either endpoint just pins at 0% or 100%.

Before trusting a reading, the controller checks it against an absolute electrical window that applies to **both** sensors. This is the open-circuit / short-circuit guard — a healthy pedal sensor never sits at the very bottom or very top of its electrical range:

```
if reading < Fault-low threshold or reading > Fault-high threshold  ->  range fault
```

Set **Fault-low threshold (V)** just below the lowest voltage a healthy sensor ever produces, and **Fault-high threshold (V)** just above the highest. A wire that shorts to ground falls below the low threshold; a wire that breaks or shorts to supply rises above the high threshold. The low threshold must be below the high threshold, or the whole calibration is rejected as invalid.

In dual-sensor mode the controller then compares the two travel percentages and requires them to agree:

```
allowed disagreement = the smaller of ( Max sensor disagreement, 20% )
if | Sensor 1 travel % - Sensor 2 travel % | > allowed disagreement  ->  disagreement fault
```

Note the hard ceiling: **Max sensor disagreement (%)** can only *tighten* this check below 20%. Entering a value above 20 has no effect — the firmware silently uses 20%. This is a safety limit you are allowed to make stricter but never looser.

When both sensors pass, the controller takes the **lower** of the two as the driver demand:

```
Driver Demand = the lower of ( Sensor 1 travel %, Sensor 2 travel % )
```

Choosing the lower reading is deliberate and conservative: one sensor reading high can never inflate the demand, only a genuine press on both can raise it.

Finally the deadband floor is applied so a pedal that rests slightly off zero still commands nothing:

```
if Driver Demand < Deadband  ->  Driver Demand = 0
```

In single-sensor mode the cross-check step is skipped — only Sensor 1 is read, range-checked, and used directly, then the deadband floor is applied the same way.

Fault confirmation, immediate cut, and release

There are two separate timings, and the difference matters:

- **Torque is cut instantly.** The moment a single reading looks bad — out of the electrical window, or the two sensors disagreeing — driver demand is forced to zero on that very reading. There is no waiting.
- **The recorded fault waits for confirmation.** A logged pedal fault is only latched after **Fault confirm count** consecutive bad readings, where each reading is 10 milliseconds apart. So Fault confirm count of 5 means a fault must persist for 5 readings (about 50 ms) before it is recorded as a confirmed fault.

```
fault is latched after Fault confirm count consecutive bad readings (one reading every 10
ms)
```

This separation lets a genuine momentary glitch cut torque safely without spamming a recorded fault, while a real wiring failure is confirmed and logged. **Fault confirm count** of 0 is invalid — it would defeat the debounce entirely — so the firmware floors it to 1 and flags the calibration as invalid; set it to at least 1.

Once a fault is latched, it does **not** clear the instant the signal looks healthy again. The driver must lift off first:

```
a latched fault clears only when  pedal travel <= 10%  AND  Fault confirm count consecutive
good readings
```

This release interlock means a fault that clears while the pedal is still pressed will not snap torque back mid-press. The driver has to bring the pedal back to roughly rest, and the signal has to read clean for the same confirm count, before the pedal will command torque again.

Downstream, the validated driver demand is the primary input to the torque path: it is only honored while the vehicle is in DRIVE, it is blended (highest-wins) with cruise and hill-hold requests, and it is then shaped by the pedal-feel curve and the torque map before becoming motor torque. An independent torque monitor separately re-reads the same two pedal sensors as a watchdog against unintended torque.

Tuning notes

Set the endpoints from the pedal itself. With the pedal fully released, read each sensor's resting voltage and enter it as **Sensor 1 at rest (V)** and **Sensor 2 at rest (V)**; with the pedal floored, read and enter **Sensor 1 at full travel (V)** and **Sensor 2 at full travel (V)**. Leave a little margin so a healthy pedal reaches a true 0% at rest and a true 100% floored without pinning early. Dragging the bands on the graph sets these endpoints visually.

Set **Fault-low threshold (V)** and **Fault-high threshold (V)** as a snug electrical window around the full sensor range — wide enough that a healthy sensor at rest or floored never trips them, tight enough that a real open or short is caught. A common starting point is a few tenths of a volt outside the rest and full-travel voltages of both sensors.

Start **Max sensor disagreement (%)** modestly (well under the 20% ceiling) — the two sensors track each other closely on a healthy pedal, so a tight value catches a drifting or failing sensor early. If you get nuisance disagreement faults on a known-good pedal, loosen it slightly (up to but never beyond 20%), but first confirm both sensors' endpoints are calibrated accurately, since mismatched endpoints are the usual cause of false disagreement.

Set **Deadband (%)** just above the worst-case off-rest reading so the vehicle never creeps with the driver's foot off, but no higher — every percent of deadband is travel the driver presses through before any torque appears.

Keep **Fault confirm count** small. One to a few readings (10–50 ms) is plenty to reject electrical noise without delaying the recorded fault. Remember it does not slow the safety response — torque is already cut on the first bad reading — it only governs when the fault is logged and how long a clean signal is required before re-arming after a lift-off.

Auto calibration. Select Auto calibrate… to measure rest and full-travel voltages over three guided cycles. It requires a live VCU, a stationary vehicle, and drive disabled. Hold each endpoint steady for one second and follow the progress prompts. Review the measured endpoints and inversion before applying them to the project. Electrical fault thresholds, disagreement, and deadband are unchanged. Save and flash the configuration separately. An unstable or incomplete capture cannot be applied.

Settings

- **Accelerator pedal** — What kind of accelerator pedal this installation has. A dual-potentiometer pedal has two sensors that are checked against each other, which is what a road vehicle should use. A single-potentiometer pedal has one sensor and no such cross-check. Equipment with no driver's throttle at all — a pump, a generator set — has none.

Single-potentiometer pedal / Dual-potentiometer pedal settings

- **Sensor 1 at rest** (V) — Sensor 1 reading with the pedal fully released. Use the live capture to set it.
- **Sensor 1 at full travel** (V) — Sensor 1 reading with the pedal fully pressed.

Dual-potentiometer pedal settings

- **Sensor 2 at rest** (V) — Sensor 2 reading with the pedal fully released.
- **Sensor 2 at full travel** (V) — Sensor 2 reading with the pedal fully pressed.

Single-potentiometer pedal / Dual-potentiometer pedal settings

- **Fault-low threshold** (V) — A pedal sensor reading below this is treated as a wiring fault (open or short).
- **Fault-high threshold** (V) — A pedal sensor reading above this is treated as a wiring fault (open or short).

Dual-potentiometer pedal settings

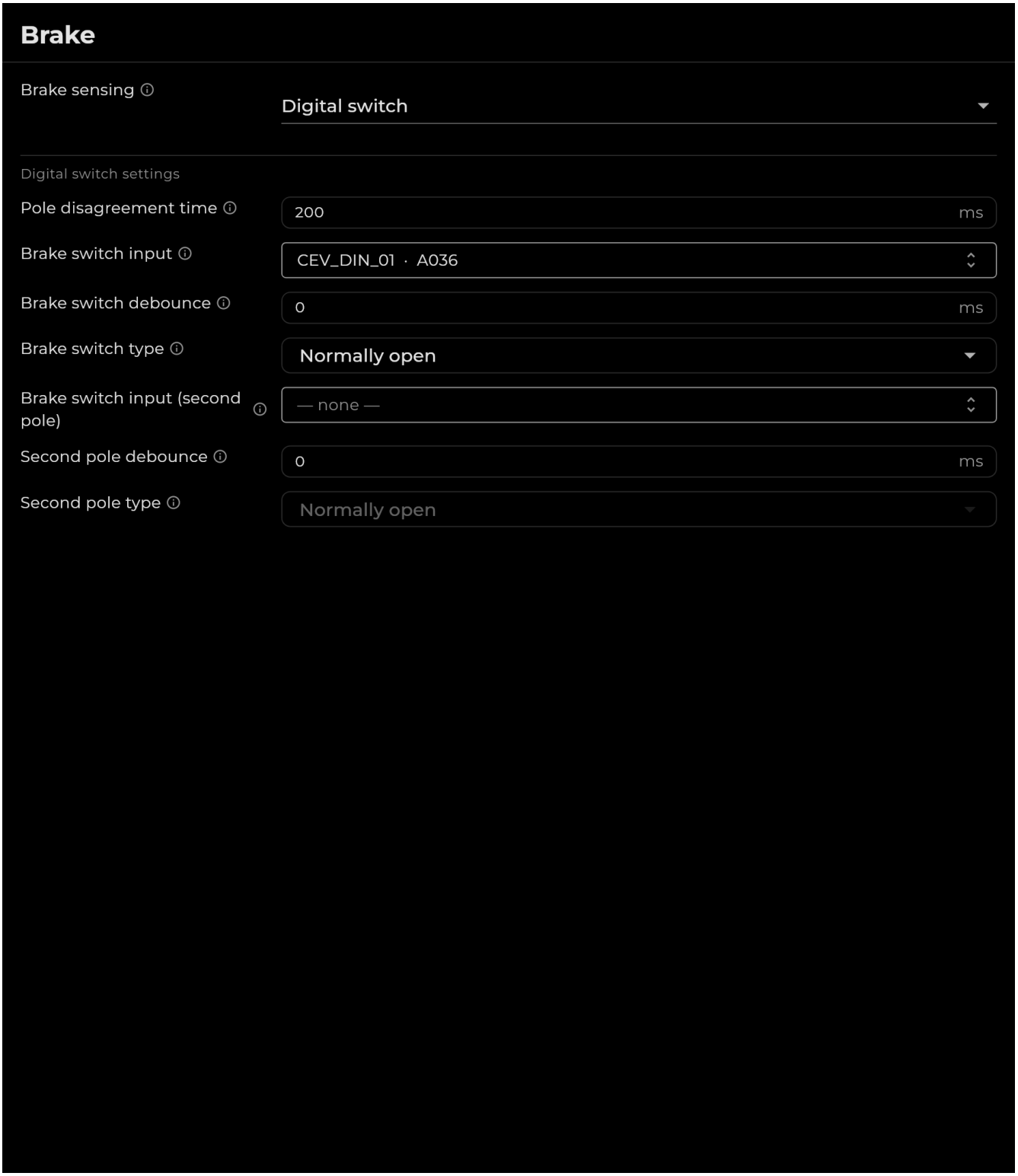
- **Max sensor disagreement** (%) — The largest allowed difference between the two pedal sensors before a fault is raised. This is a safety check — keep it tight.

Single-potentiometer pedal / Dual-potentiometer pedal settings

- **Deadband** (%) — Pedal travel below this is treated as no demand.
- **Fault confirm count** — How many bad readings in a row must occur before a pedal fault is raised.

Brake

How brake application is sensed — a switch or an analog sensor — and its calibration.



The brake module is how the VCU knows the driver is on the brake. It is a sensing stage only — it never commands torque itself — but almost everything that should back off when you brake depends on it: cruise control cancels, regen blending modulates, and the vehicle refuses to come out of PARK until it sees a confirmed brake press. Because so much hangs on it, the brake is a **required** input: if it is left unconfigured the vehicle will not finish its safe startup and will not arm. You tell it how the brake is wired and calibrated,

and it publishes a clean, validated brake position (0–100%) plus a simple pressed / not-pressed verdict every control cycle.

What it needs

You must pick a sensing method under **Brake sensing**. There are two:

- A **switch** — a simple on/off brake-pedal switch. In this mode you assign the **Brake switch input** — which may be a plain digital input or a frequency-input pin used as a switch — and say whether the switch is **normally open** (open at rest, closing when you press the pedal, which nearly all brake switches are) or **normally closed**. The controller already knows how the input pin is biased, so that is all it needs. A **debounce** time rejects contact bounce.

Many brake switches have **two circuits in one body**: one pole closes when the pedal is pressed while the other opens. If yours does, wire the second one too and assign it as the **Brake switch input (second pole)**, declared normally closed. The controller then cross-checks the two, so a failed switch is caught instead of quietly reading "not braking" — which is the failure that matters, because a brake the vehicle cannot see is one that will not cancel cruise or blend regen.

The two poles never change at exactly the same instant: one breaks before the other makes, so they disagree for a moment on every press. **Pole disagreement time** is how long they may contradict each other before that counts as a fault — 200 ms suits a mechanical switch. Below that the last agreed reading stands. Past it the brake reads as not pressed and logs a fault, because which pole is telling the truth is exactly what cannot be known.

In switch mode there is no position, no calibration, and no electrical-range checking. The brake reads as either fully released (0%) or fully pressed (100%).

- An **analog sensor** — a pressure or position sensor that gives a rising voltage as the brake is applied. This mode needs the **Brake sensor** assigned, plus its calibration (**Sensor at rest (V)**, **Sensor at full press (V)**) and its electrical fault window (**Fault-low threshold (V)**, **Fault-high threshold (V)**). Optionally you can add a second, redundant sensor with **Brake sensor (second)** for cross-checking.

If **Brake sensing** is left unset, or the analog calibration is inconsistent (for example **Sensor at rest (V)** equal to **Sensor at full press (V)**, or **Fault-low threshold (V)** not below **Fault-high threshold (V)**, or a required sensor left unassigned), the brake is treated as not configured. It reports a configuration fault and the vehicle stays in its safe startup state. Fix the configuration and re-check before flashing.

How it works

Every control cycle (about every 10 ms) the module reads the brake, decides whether the reading is trustworthy, and publishes a position and a press verdict.

Switch mode. The switch is read directly: closed means pressed (100%), open means released (0%). There is nothing to calibrate and no electrical fault checking — a switch is either made or not.

Analog mode — turning volts into position. The sensor voltage is mapped to a brake position using your two calibration points:

```
brake position % = (sensor volts - Sensor at rest (V)) / (Sensor at full press (V) - Sensor
at rest (V)) x 100
```

The result is clamped to the 0–100% range, so a reading slightly below rest or above full press simply pins at 0% or 100% rather than running off the ends.

Analog mode — electrical sanity (open/short detection). Before trusting the reading, each sensor voltage is checked against its fault window:

```
if sensor volts < Fault-low threshold (V):  electrical fault
if sensor volts > Fault-high threshold (V): electrical fault
```

These thresholds sit just outside the legitimate signal band — below **Sensor at rest (V)** and above **Sensor at full press (V)** — so a disconnected, shorted, or out-of-band sensor is caught instead of being read as a real brake position.

Two-sensor cross-check. If you assign **Brake sensor (second)**, both sensors are read and both must pass their electrical window. Then the two positions are compared:

```
disagreement % = | sensor position - second sensor position |
allowed disagreement = min(Max sensor disagreement (%), 10%)
if disagreement % > allowed disagreement: correlation fault
```

Note the hard 10% ceiling: the firmware will tighten **Max sensor disagreement (%)** if you set it higher than 10%, but it will never loosen past 10%. A smaller value you set is honored. When both sensors agree, the published brake position is the **higher** of the two readings, and a separate, stricter "both sensors agree it's pressed" check uses the **lower** of the two:

```
brake position    = higher of (sensor position, second sensor position)
strict-press check uses lower of (sensor position, second sensor position)
```

Deciding "pressed". The brake counts as pressed once position reaches your threshold:

```
pressed if brake position >= Press threshold (%)
```

There are two press verdicts. The normal one — used by cruise-cancel and regen blending — uses the higher (more eager) reading, because there an extra "yes, braking" is harmless (the driver expects to slow down). The strict one — used by the leave-PARK guard — uses the lower reading, so one sensor stuck high can never on its own release PARK. In switch mode position is only ever 0% or 100%, so **Press threshold (%)** has no practical effect.

Fault confirmation and recovery. A single bad reading does not trip anything; faults are filtered with **Fault confirm count**:

```
time to set a fault   = Fault confirm count x 10 ms of consecutive bad readings
time to clear a fault = Fault confirm count x 10 ms of consecutive good readings
```

A fault (electrical-range or sensor disagreement) only latches after that many bad readings in a row, and once latched it stays set until the same number of good readings in a row clears it. Setting **Fault confirm**

count to 0 is treated as 1 — meaning the filter is effectively switched off and a single sample sets or clears the fault — and the configuration is flagged. While any fault is latched, and during the confirm window on the way in or out, the brake publishes its safe fallback: position 0%, not pressed, marked invalid. An invalid brake always reads as released, so consumers fail to the safe side (cruise cancels, regen stands down, PARK stays put).

Tuning notes

Set **Sensor at rest (V)** and **Sensor at full press (V)** to the actual measured voltages at a fully released and fully applied pedal — these two points define the whole 0–100% scale, so measure them on the vehicle rather than trusting datasheet nominals. Place **Fault-low threshold (V)** and **Fault-high threshold (V)** just outside that real signal band: tight enough to catch a disconnected or shorted sensor, but with enough margin that normal pedal travel never clips them. **Press threshold (%)** sets how far the pedal must move before the system calls it "braking" — set it above the sensor's resting noise so a relaxed foot doesn't read as a press, but low enough that a light, deliberate brake still cancels cruise and blends regen. For a two-sensor install, start **Max sensor disagreement (%)** tight (a few percent) and only loosen it if matched, healthy sensors nuisance-trip — remembering 10% is the hard ceiling. Raise **Fault confirm count** if electrical noise causes brief false faults, but keep it modest: the value directly sets how long a real fault takes to be caught and how long a genuine recovery takes to be trusted (each count is one 10 ms cycle).

Settings

- **Brake sensing** — How brake application is sensed — a switch or an analog sensor.

Digital switch settings

- **Pole disagreement time** (ms) — How long the two poles of a two-circuit brake switch may contradict each other before a fault is raised. They always disagree for an instant as the pedal moves, because one pole breaks before the other makes, so this must be longer than that. 200 ms suits a mechanical switch.

Analog settings

- **Brake sensor** — The analog input wired to the brake-pressure sensor.
- **Brake sensor (second)** — An optional second analog brake sensor for redundancy.
- **Sensor at rest** (V) — The brake sensor reading with the pedal released.
- **Sensor at full press** (V) — The brake sensor reading with the pedal fully pressed.
- **Fault-low threshold** (V) — A reading below this is treated as a wiring fault.
- **Fault-high threshold** (V) — A reading above this is treated as a wiring fault.
- **Max sensor disagreement** (%) — The largest allowed difference between two brake sensors before a fault.
- **Fault confirm count** — How many bad readings in a row must occur before a brake fault is raised.
- **Press threshold** (%) — Sensor reading above which the brake counts as applied.

Digital switch settings

- **Brake switch input** — The input wired to the brake switch. May be a digital input or a frequency input read as a switch.
- **Brake switch debounce** (ms) — How long the brake switch must hold steady before its state changes, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.
- **Brake switch type** — Whether the brake switch's contacts are closed or open when the pedal is at rest. Almost every brake switch is normally open — it closes when you press the pedal. Pick normally closed only if yours is wired the other way.
- **Brake switch input (second pole)** — Optional. The second pole of a two-circuit brake switch — the one that OPENS when the pedal is pressed, while the first pole closes. Wiring both lets the controller cross-check them, so a failed switch is caught instead of silently reading "not braking". Leave it unset for a single-pole switch.
- **Second pole debounce** (ms) — How long the second pole must hold steady before its state changes, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.
- **Second pole type** — The second pole of a two-circuit brake switch is the opposite of the first: if pole one closes when you press the pedal, this one opens. Leave it as normally closed unless your switch is unusual.

Torque Arbiter

How pedal input becomes wheel torque: the base map and pedal feel, the drive-direction selector, and the optional features (cruise, hill assist, traction control, vectoring, regen) layered on top — all bounded by an independent safety monitor.

The torque arbiter is the part of the VCU that decides how much torque each motor makes, one hundred times a second. It reads what the driver asks for, adds what the driving features ask for, shapes the result with your calibration, applies the limits, splits the total across the motors, and then lets an independent safety guard have the last word before the commands go to the inverters. One firmware serves every build: one motor or four, a fixed reduction or a retained manual gearbox, and any of the supported inverter families. The arbiter as a system is ASIL C, the highest safety integrity level this controller targets, and the two steps that decide direction and permitted torque carry that grade.

This overview explains the path from the pedal to the wheels, the order of the limits, and what each form under *Torque Arbiter* owns. The forms that follow describe every setting.

From the pedal to the wheels. Every 10 milliseconds the arbiter runs the same chain.

1. Read everything: the accelerator pedal on its two redundant channels, the brake, the shifter, the cruise setting, road speed, motor speed, the battery limits, and the road grade.
2. Decide who is asking. The pedal is the usual source, but cruise, hill hold, creep and hill descent can ask for torque too. The arbiter takes the largest accelerate request and, separately, the largest slow-down request. The driver's foot wins at once when it asks for more than any feature.

3. Pick the direction: Park, Neutral, Forward or Reverse. The arbiter owns this decision and its interlocks, and it turns the request into a positive torque for forward drive or a negative torque for regen and reverse.
4. Shape the request. A pedal-feel curve maps pedal travel to a throttle demand. Then the torque map turns throttle demand and motor speed into the total motor-shaft torque the build makes. Half pedal does not mean half torque. It means whatever the map says, and the map is where the feel of the vehicle lives.
5. Apply the everyday limits. Traction control cuts torque on a spinning corner. On a manual gearbox the rev limiter tapers torque to zero near redline. The speed taper rolls each motor off as it nears its configured speed limit.
6. Split the total across the motors, evenly by default. A build with front and rear motors can bias the split by measured acceleration, and torque vectoring can move torque between the two sides of an axle. These splits move torque. They never add to the total.
7. Give each motor its share. The map is already in motor torque, so each motor's command is its slice of the total. The gear ratio is not part of this step. It serves the speedometer, slip detection and the dashboard.
8. Let the safety guard decide. Before a command leaves, an independent check clamps it to a physical envelope: what each inverter reports it can deliver, the speed cap, an independent pedal check, and a cut when the battery goes silent.
9. Send it. Each inverter receives its command over CAN in its own protocol.

Steps 3 and 8 are the ASIL C steps. Everything between them is the feature path, rich in calibration but always bounded by what step 8 permits.

Direction and the shifter. The arbiter owns the direction state, with the safety interlocks built in. All motor torque is zero in Park and in Neutral. Forward turns the pedal into positive torque. Reverse turns it into negative torque, is capped at a low road speed, 5 km/h by default, and cannot be entered above that speed. The brake must be pressed to leave Park.

The *Direction (P/R/N/D)* form selects how the shifter is wired: four digital lines, one or two analog voltages sorted into windows, a CAN signal from a donor module, or an installer script that asks for a direction on every tick. Whatever the source, the same rules decide whether a request is granted. Two optional gates sit on top: brake required to change into Drive or Reverse, and a road-speed limit on changes into Drive. Neither gate ever blocks Neutral or Park. A refused change leaves the vehicle in the direction it already had. It never drops to Neutral on its own, and the refusal is logged, never silent.

Push buttons behave differently from a lever. A lever stays where you put it. A button springs back, so the vehicle remembers the last press, starts every key-on in Neutral, and ignores a button that is already pressed when the vehicle wakes. A button held for more than five seconds is reported as jammed.

Two drivetrain shapes. The same arbiter serves two drivetrains, and both use the same torque map. The dividing line is whether the ratio between the motor and the wheels changes while driving.

A fixed-reduction drivetrain is the common EV: one or more motors, each with its own fixed reduction. The map is total motor torque against motor speed, with throttle as the third axis, so one calibration serves a

single-motor, twin-motor or four-motor build. Torque vectoring and the front and rear bias are available when the motors make them meaningful. Regen is the negative half of the map, so lifting off the pedal slows the vehicle without any extra setting.

A manual-gearbox conversion keeps a driver-shifted transmission. It uses the same map, because the map reads motor speed and does not need to know the gear, and it wraps the map in manual-only stages. The clutch switch is the master gate: with the clutch pressed, torque is cut to zero, cruise cancels, hill hold releases, and regen blending stops. The rev limiter tapers torque to zero near redline. An optional per-gear torque cap reduces torque to protect a donor gearbox from low-gear multiplication. It only ever reduces, and it engages only with a real gear-position sensor. Gear inference from motor speed and wheel speed drives the dashboard only, never the torque cap. The *Gearbox* and *Gear detection / limiter* forms under *Drivetrain* hold these settings.

The features on top. Each feature is a demand source or a limiter added to the basic chain, and each can be switched on its own.

- Cruise holds road speed. It asks for torque on its own, and the driver overrides it at once by pressing past it. The brake cancels it, a shifter move out of Drive cancels it, and on a manual gearbox the clutch cancels it. Resume returns to the last set speed. Engagement is bumpless.
- Hill hold applies torque against the slope at near standstill, only uphill in the selected direction, so the vehicle does not roll back while the driver moves from the brake to the pedal.
- Creep crawls the vehicle at walking pace when both pedals are released, like an automatic, and settles at the crawl speed. Any brake press cancels it. Creep and one-pedal driving to a stop are opposite feels, so a build gets one or the other.
- Regen blending adds regen when the driver presses the brake, on top of the closed-throttle regen the map already gives. It stands down when the battery cannot accept charge and below a minimum motor speed.
- Drive profiles let the driver swap the pedal curve and the torque map while driving, with a short crossfade. A profile carries no safety claim. The safety guard never reads which profile is active, so a profile can only make the vehicle tamer.
- Traction control reduces torque on a spinning driven wheel against a wheel-speed reference. It picks its method from the build: the driven axle against the non-driven axle on a two-wheel-drive build, and motor speed against the ratio on an all-wheel-drive build.
- Torque vectoring biases left and right on an axle with two motors, in proportion to yaw rate.
- Hill descent applies regen to hold a speed on a downhill grade, on fixed-reduction builds.

A feature that needs a road-state signal reads the dynamics snapshot: yaw rate, acceleration and wheel speeds. When a required sensor is missing or stale, the feature stands down for that tick. It does not invent a value, and it does not fight the other features. A bad comfort sensor costs you the feature, not the vehicle.

Cruise and hill hold ask for torque without a foot on the pedal, so each declares a bounded authority, a calibrated ceiling on what it can ask for. The safety guard enforces that ceiling, so a wrong cruise gain cannot ask for more than its cap.

The installer script can switch every one of these features off and back on while driving, for an off-road button or a battery-care mode. The script can only remove a feature the configuration enabled, never add one. If the script stops, every feature returns to its configured state.

The order of the limits. Several layers stack between the pedal and the motor, and each one can only reduce torque:

1. What is asked for: pedal, cruise, hill hold, hill descent.
2. The calibration shape: pedal feel and the torque map.
3. The rev limit, on a manual gearbox.
4. Traction control.
5. Distribution: front and rear bias, then vectoring.
6. Each motor's available torque, as the inverter reports it after its own thermal and voltage derating.
7. The speed cap.
8. The driver-authority check, from an independent pedal read.
9. The safety cutoff.

The last two are the safety guard. They run apart from the rest of the chain, with their own sensor reads, so a fault in the feature path cannot fool them.

The safety guard. Two parallel paths watch the pedal. The feature path computes the torque the vehicle wants. The safety guard, a deliberately simple and independent calculation, computes the maximum torque the vehicle is allowed. The motor gets the smaller of the two, every tick. The ASIL C claim for the torque path lives in the guard and only in the guard. That is what lets cruise, vectoring and traction be tuned at ASIL B without weakening the claim.

The guard reads the two pedal channels itself and runs its own plausibility checks. It uses its own permitted-torque curve, calibrated above the torque map so it never clips normal driving. The Configurator refuses to sign a configuration whose torque map exceeds the guard's curve anywhere. The guard never asks an inverter for more than the inverter reports it can deliver. It tapers torque to zero above the maximum drive speed. It cuts propulsion when a configured battery management system goes silent, because an unmonitored pack must not keep delivering power. It honours the emergency cutoff input, which is energize-to-run: a lost signal, a broken wire or a dead relay trips it, torque goes to zero at once, and the vehicle is asked into fault-safe.

The guard is organized as two axle guards, front and rear, each with its own per-corner ceilings and an equal share of the driver-authority cap. When the feature path stays inside the envelope, the guard is invisible. When the feature path asks for more, the guard clamps it. When the disagreement persists, the guard logs a fault, reduces power, and escalates to fault-safe if the condition does not clear. The *Safety monitor* form holds the permitted-torque curve, the maximum drive speed and the cutoff input.

What each form owns.

- *Drive Profiles*: the pedal-feel curve and the torque map, one pair per profile, the selector, the default profile and the blend time.
- *Direction (P/R/N/D)*: the shifter source, its wiring, the two shift gates and the reverse speed cap.

- *Cruise*: the gains, the maximum demand and the declared authority.
- *Hill hold / descent*: the standstill and grade thresholds, the hold demand and the descent target speed.
- *Creep*: the crawl speed, the crawl effort and the engagement smoothness.
- *Traction control*: the slip threshold, the cut depth and the low-speed floor.
- *Torque vectoring*: the yaw gain and the maximum transfer per corner.
- *Distribution*: the front and rear bias curve and the left and right bias curve.
- *Regen blending*: the regen map against motor speed and brake travel, the minimum motor speed and the authority.
- *Brake light*: when the lamp lights on regen.
- *Safety monitor*: the permitted-torque curve, the maximum drive speed and the emergency cutoff.

The brake and the pedal themselves are configured under *Pedal & Brake*. The gearbox, the per-gear ratios and the gear-aware limiter are under *Drivetrain*.

When something goes wrong. The arbiter never opens a contactor or disables drive on its own. It asks the vehicle state machine, which owns every mode decision. What the driver sees falls into four behaviours. Normal driving: the feature path and the guard agree. Reduced power: a motor faulted, the guard caught a sustained over-ask, or a pedal channel disagreed, and the vehicle keeps moving at reduced torque. Drive disabled at speed: the disagreement persisted, the vehicle stops answering the pedal and coasts, and the operator must restart and clear. Immediate torque cut: the emergency cutoff tripped, every motor goes to zero and the contactors open shortly after.

A pedal fault forces the driver demand to zero and nothing more. The brake still works and the vehicle coasts. Every fault is logged as a diagnostic event and is visible in the Faults screen.

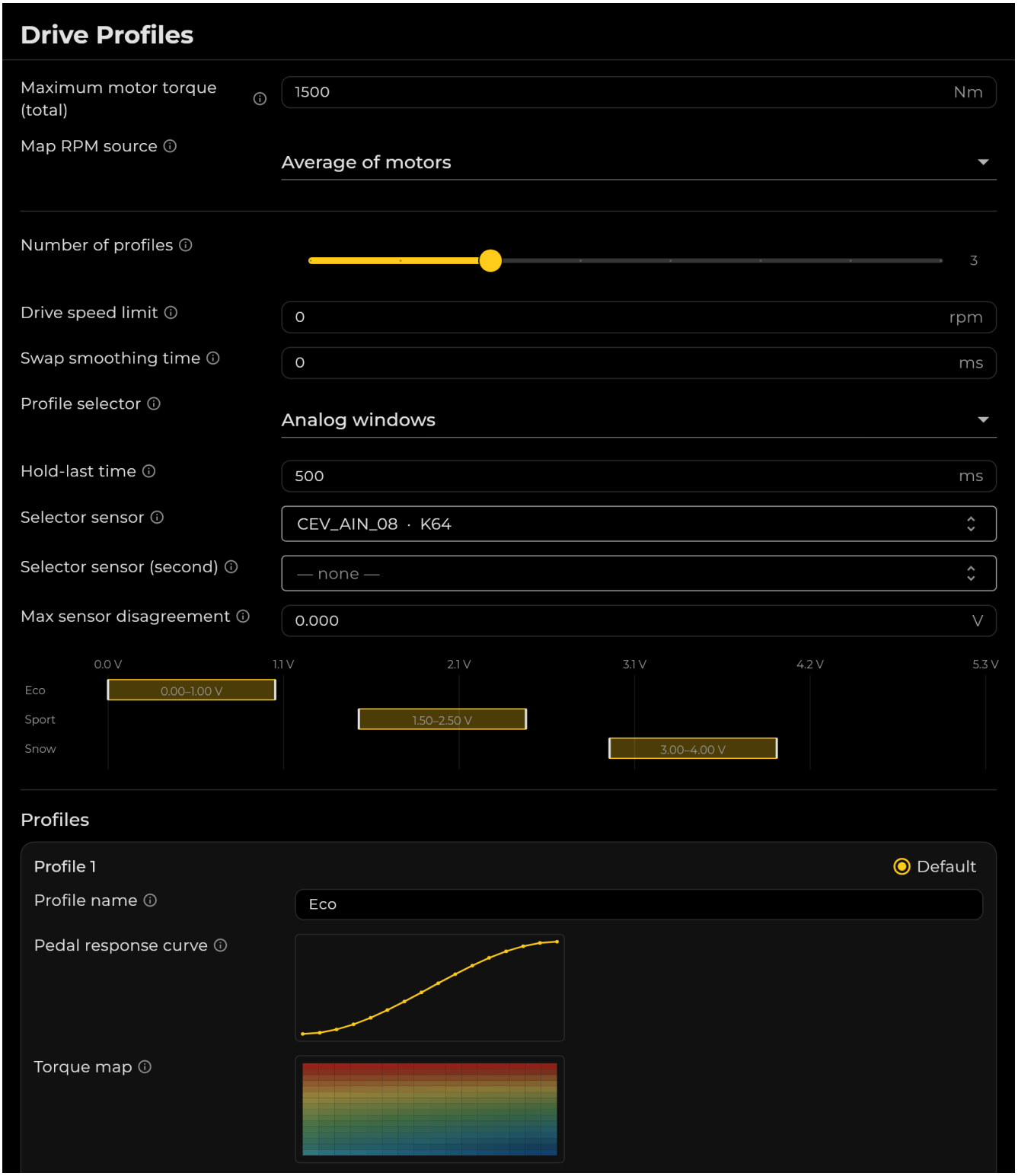
Brake blending adds to lift-off braking in every drive profile. Its authority limits only the added torque before distribution. The final inverter limits still apply to the combined command.

Automatic Drive selection. Digital, Analog and CAN shifters enter Drive when the direction interlocks accept D or R, the accelerator input is valid, and the vehicle is Ready with the high-voltage bus live. A direction accepted during precharge takes effect after Ready. None (forward-only) also enters Drive automatically. Park, Neutral or an invalid direction removes drive permission; the high-voltage bus remains live. Faults, charging and shutdown retain priority. Script is the only source that needs an explicit vehicle-state request to enter Drive.

Pedal calibration. Set the sensor rest and full-travel voltages and electrical fault limits once on the Accelerator pedal page. The safety monitor uses those settings, reads the sensors itself, and applies its own fixed disagreement check. Its permitted-torque curve remains separate from the drive profiles. Verify the measured pedal travel because an incorrect sensor endpoint affects both paths.

Drive Profiles

The core torque response: the maximum motor torque, the drive profiles (each a pedal-feel curve and torque map the driver can switch between), the profile selector, and how demand combines with speed.



The torque map defines driver demand in total motor-shaft Nm across all traction motors. Positive values request drive. Negative values request regen. Direction remains separate. For example, a 20 Nm reverse demand is R with +20 Nm. With two equally loaded motors, 20 Nm total gives each motor 10 Nm before limits.

The pedal response curve maps pedal travel to shaped demand. The torque map then uses shaped demand and motor-speed magnitude to find total torque. You can select average motor speed, the highest speed, or the front axle as the map input. Each table holds its edge value outside its calibrated range.

If the pedal curve or torque map is invalid, the firmware uses a linear drive-demand fallback. The shared maximum torque limits the map output. The independent safety monitor uses its own pedal calibration and physical motor limits.

Each profile has a pedal curve, a torque map, a regen speed-fade curve, and separate regen application and release rates. The negative map region defines lift-off strength. There is no separate lift-off multiplier.

The shared brake curve requests total regen from brake input. The stronger of lift-off and brake demand wins. For example, −30 Nm and −50 Nm select −50 Nm before fade and safety limits.

The profile fade uses vehicle-speed magnitude and outputs 0 to 100 percent of combined regen demand. Start the curve at zero speed and zero percent. Select km/h or mph in the curve editor. Both directions use the same curve. Invalid speed disables regen and reports that regen is unavailable. Regen stops at standstill. Holding remains a separate function.

Application and release rates use total motor-shaft Nm per second. Drive waits until regen releases to zero. Safety limits and reduced fade apply immediately. A profile change continues from the current torque with the new profile fade and rates. Positive drive transitions use the profile swap time.

The arbiter converts demand into physical shaft torque. Drive uses the selected direction. Regen opposes each motor's actual shaft motion, including rollback. Commands and measurements retain physical shaft signs. A negative measured value alone does not identify regen.

Set the shared torque limit first. Define drive and lift-off demand in each torque map. Then calibrate the shared brake response, profile fades, and rates. The defaults start fade at zero and reach 100 percent at 5 km/h. The default application rate is 100 Nm/s. The default release rate is 200 Nm/s. Adapt these values to the vehicle.

Settings

- **Maximum motor torque (total)** (Nm) — The total motor-shaft torque delivered at full throttle, summed across all drive motors. Sets the overall drive strength.
- **Map RPM source** — When the drive motors spin at different speeds, which motor speed reads the torque map — the average of the drive motors, the highest, or the front axle's.
- **Profile name** — A short name for this drive profile, shown on the dashboard (for example Eco, Sport, or Snow).
- **Pedal response curve** — Shapes how pedal travel maps to demand for this profile — softer off idle, or more linear.
- **Torque map** — How motor speed and throttle combine into total motor-shaft torque for this profile. Its off-throttle (negative) region sets the coasting/regen feel and its high-speed edge sets this profile's top speed.
- **Regen speed fade** — Vehicle-speed magnitude sets the percentage of combined regen demand. Start at zero speed and zero percent. Holding is separate.

- **Regen application rate** (Nm/s) — The maximum increase in regen magnitude per second, measured as total motor-shaft torque across all traction motors.
- **Regen release rate** (Nm/s) — The maximum decrease in regen magnitude per second. Drive waits for zero regen. Safety limits and reduced speed fade apply immediately.
- **Profile selector** — How the driver picks which drive profile is active. None means a single fixed profile.
- **Number of profiles** — How many drive profiles this vehicle offers, from 1 to 8.
- **Default profile** — The profile the vehicle starts in, and the one it falls back to if the selector is lost.
- **Swap smoothing time** (ms) — The drive-torque transition time when the profile changes, in milliseconds. Regen uses the new profile application and release rates. Zero uses the built-in default.

Digital lines / Analog windows / CAN value settings

- **Hold-last time** (ms) — How long to hold the last valid profile when the selector reads out of range, before falling back to the default.

Digital lines settings

- **Profile input** — The input that selects this profile. May be a digital input or a frequency input read as a switch.
- **Input debounce** (ms) — How long the line must hold steady before the change is accepted, in milliseconds. Rejects contact bounce; 0 disables it.
- **Switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.

Analog windows settings

- **Selector sensor** — The analog input wired to the profile selector knob.
- **Selector sensor (second)** — An optional second selector sensor for redundancy.
- **Max sensor disagreement** (V) — The largest allowed difference between two selector sensors before the reading is rejected.
- **Profile window — low** (V) — The lowest selector voltage for this profile.
- **Profile window — high** (V) — The highest selector voltage for this profile.

CAN value settings

- **Profile signal** — The CAN signal that indicates this profile is selected.
- **Profile value — low** — The lowest signal value that counts as this profile. Use the same low and high value for a fixed code.
- **Profile value — high** — The highest signal value that counts as this profile.

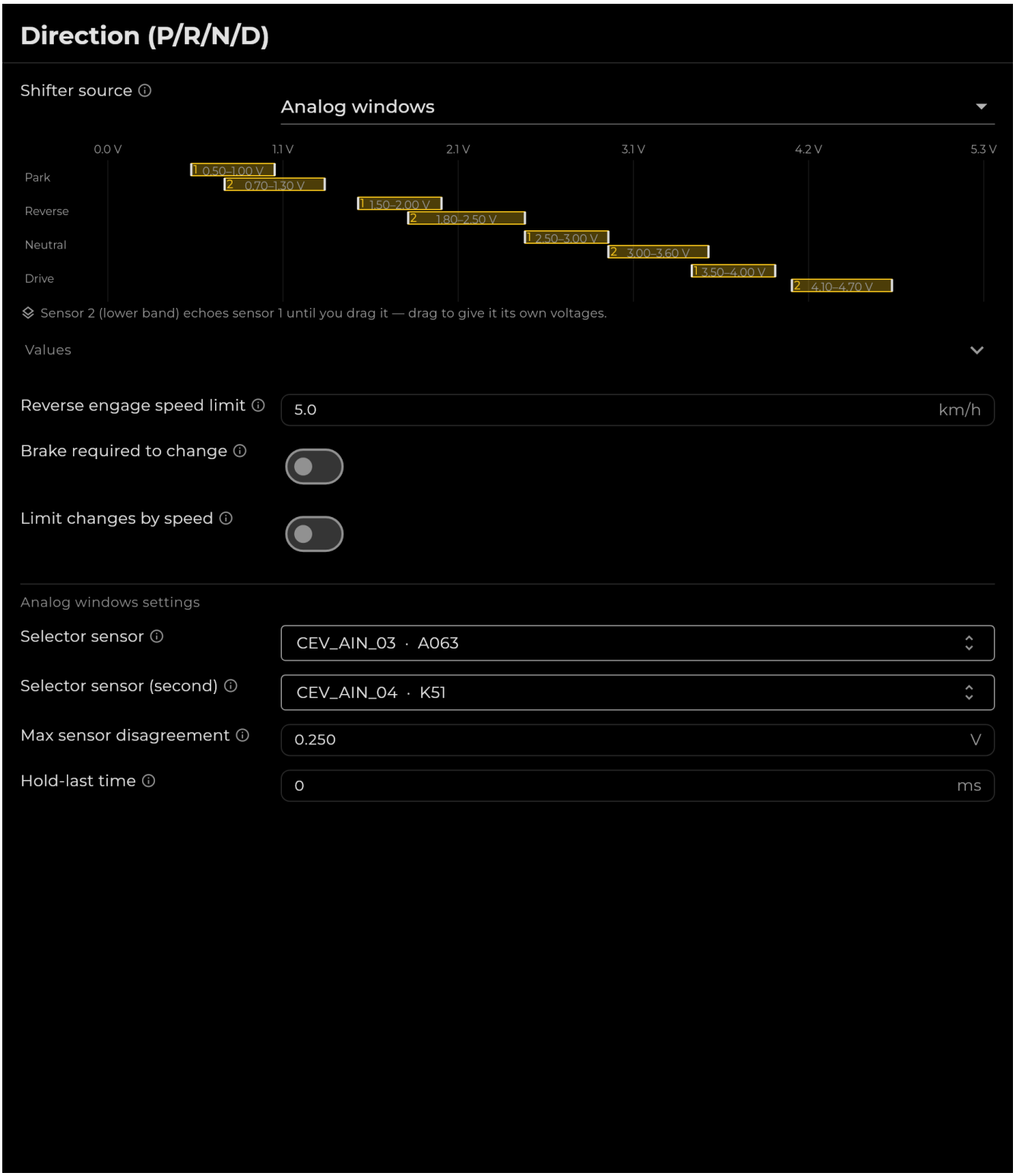
Script settings

- **Script request timeout** (ms) — How long a script's profile request stays in effect before the vehicle falls back to the default, in milliseconds. 0 uses the built-in default.
- **Script request rate limit** (ms) — The shortest time allowed between a script's profile requests, in milliseconds. 0 uses the built-in default.

- **Drive speed limit** (rpm) — Drive torque is tapered off as each motor's shaft approaches this speed, in RPM. Zero means no limit.

Direction (P/R/N/D)

How the driver's Park / Reverse / Neutral / Drive selection reaches the vehicle.



The Direction system reads the driver's Park / Reverse / Neutral / Drive selection and turns it into the gear the vehicle is actually engaged in. That engaged direction does two jobs: it decides whether the vehicle is allowed to make drive torque at all, and — in Reverse — it flips the sign of that torque so the wheels turn the other way. Park and Neutral always mean zero drive torque; Drive and Reverse are the only two positions that let the pedal produce propulsion. The system is deliberately cautious: anything it cannot read cleanly is treated as Neutral, and the two most safety-relevant moves — pulling out of Park and dropping into Reverse — are guarded.

What it needs

You first tell the vehicle how the driver's selection arrives, using **Shifter source**. There are five choices, and each one needs different wiring or signals:

- *Forward-only* — there is no selector at all; the vehicle is permanently in Drive. Nothing else in this section applies.
- *Digital* — four separate on/off lines, one per position. You assign the **Park input**, **Reverse input**, **Neutral input**, and **Drive input** lines. Any line you leave unassigned is simply treated as absent (for example, a vehicle with no Park line). Each of these lines may be wired either to a plain on/off switch input or to a frequency-input pin used as a switch, and for each you say whether the switch is **normally open** (open at rest, closing when that position is selected — what a P/R/N/D selector's contacts almost always are) or **normally closed**; the controller already knows how the input pin is biased. Each also carries a **debounce** time that the line must hold steady before the change is accepted — a short debounce rejects contact bounce and electrical noise.
- *Analog* — a single voltage from the selector, read on the **Selector sensor** input, sorted into one of four voltage bands. Optionally a second, redundant sensor on **Selector sensor (second)** for cross-checking.
- *CAN bus* — the selector position comes from a vendor module over the CAN bus. You point each position at the right **Signal** and give it a valid range.
- *Script* — the installer script asks for the position, on every tick. There is no wiring at all; the script is the selector. The script must keep asking, or its request expires (see below).

The two safety guards also need their own inputs to be healthy. The Reverse speed guard needs a trusted road-speed signal, and the leave-Park guard needs a working brake. If road speed is missing, Reverse is refused; if the brake reads as not-pressed, the vehicle will not leave Park. Drive torque only ever flows when the vehicle is in its Drive state and the pedal is reading valid — Park and Neutral hold torque at zero regardless of pedal.

How it works

The selection is read fresh every 10 ms. How it is decoded depends entirely on **Shifter source**.

Digital. The system counts how many of the four position lines are switched on. Exactly one on wins and selects that position. Zero on, or two or more on at once, is treated as ambiguous and drops immediately to Neutral — there is no grace period for digital lines, because a shorted or broken harness must not be guessed at.

```
exactly one of {Park, Reverse, Neutral, Drive} on  ->  that position
none on, or two or more on                          ->  Neutral (immediately)
```

Analog. The **Selector sensor** voltage is checked against four voltage bands, in a fixed priority order: Park first, then Neutral, then Reverse, then Drive. The first band the voltage falls inside wins. Each band is set by a low/high pair — **Park window — low (V)** / **Park window — high (V)**, **Neutral window — low (V)** / **Neutral window — high (V)**, **Reverse window — low (V)** / **Reverse window — high (V)**, and **Drive window — low (V)** / **Drive window — high (V)**. A band with either end left at 0 is treated as switched off and never matches.

```
position = first band, in order Park, Neutral, Reverse, Drive,
           where  low <= selector voltage <= high
no band matches  ->  invalid
```

If you fit a second selector on **Selector sensor (second)**, the system runs a two-way agreement check before trusting the reading. The two sensors must land in the *same* position, and their voltages must be within **Max sensor disagreement (V)** of each other. If either check fails, the sample is rejected as invalid. (Set **Max sensor disagreement (V)** to 0 to skip the voltage-gap check while still requiring the two sensors to agree on the position.) Note the engaged position is always taken from the primary **Selector sensor**; the second sensor only votes yes/no, it does not get averaged in.

```
dual-sensor accepted only if:
  position(sensor 2) == position(sensor 1)
  AND  |sensor 1 voltage - sensor 2 voltage| <= Max sensor disagreement (V)
```

CAN bus. Each position points at a **Signal** coming from a vendor module, with a valid band set by **Value — low** and **Value — high**. The four positions are checked in order Park, Neutral, Reverse, Drive, and the first one whose signal currently reads inside its band wins. A signal that is stale, timed out, or out of range simply does not match — so a dropout looks the same as "no position selected."

```
position = first of Park, Neutral, Reverse, Drive
           whose Signal reads between Value — low and Value — high
nothing matches (incl. stale signal)  ->  invalid
```

Script. The installer script asks for a position, and only the four real positions are accepted. Two settings keep it honest.

Command timeout (ms) is the longest silence allowed. The vehicle holds the last accepted position only for this long. If the script stops asking, the position drops to Neutral at once and a fault is logged, until the script asks again. So the script must ask on every tick — once at start-up is not enough. This is deliberate: a script that stops running, or gets stuck, must never leave a vehicle in Drive.

Minimum command spacing (ms) sets how close together requests may be. A request that arrives sooner is refused, and the script is told so; the position it asked for last still stands. Repeating the same position on every tick is normal and costs nothing.

Two mistakes are reported back to the script as errors instead of being refused quietly: asking for a position on a vehicle whose **Shifter source** is not *Script* (there, the physical selector owns the shifter), and asking for something that is not one of the four positions.

```
request accepted only if at least Minimum command spacing (ms) since the last accepted one
position valid only while less than Command timeout (ms) since the last accepted request
otherwise  ->  Neutral
```

Hold-last (Analog and CAN bus only). A momentary glitch should not slam the vehicle into Neutral. So when an Analog or CAN-bus reading goes invalid, the system keeps reporting the last good position for a short grace window — **Hold-last time (ms)**. If a good reading comes back inside that window, nothing happened. If the grace window runs out, the position drops to Neutral and a fault is logged. (There are two separate **Hold-last time (ms)** values — one for the Analog source, one for the CAN-bus source — and each applies to its own selector.) Digital and Script sources have no grace window at all and drop to Neutral on the very next read.

```
while invalid for less than Hold-last time (ms):  keep last good position
once invalid past Hold-last time (ms):            Neutral + log a fault
a good reading at any point:                      resume normally, clear the fault
```

Momentary buttons. Turn on **Momentary buttons** if your shifter is push-buttons that only make contact while pressed, rather than a lever or rotary that stays where you put it. The controller keeps an accepted selection after the button is released. If the brake or speed interlock refuses a button press, that request is discarded. Hold the brake and press the direction button again once the speed condition is met. Pressing the brake later, even while the old button is still held, does not complete the refused shift. All four inputs must be the same kind.

Two things follow from that, and both are deliberate. A button cannot be read when the vehicle wakes up — there is no lever position to look at — so the vehicle starts in **Neutral with nothing selected** and will not move until the driver presses a position. And if two buttons are pressed at once, the controller keeps the selection it already had rather than dropping to Neutral, which at road speed would be far worse than ignoring a fumbled press. A button that reads as held for more than five seconds is treated as jammed and logged; the selection is untouched and the other buttons keep working.

The two safety guards. Once a position has been decoded, two interlocks gate whether the vehicle is allowed to *change into* it. They only ever gate a transition — a direction already engaged is never yanked away.

The **Reverse engage speed limit (km/h)** guard refuses a change into Reverse unless road speed is known *and* at or below that limit. If speed is unknown or above the limit, the Reverse request is dropped (not queued) — the driver simply re-selects Reverse once the vehicle is slow enough. The fail-safe assumption is that an unreadable speed means "too fast."

```
allow change into Reverse only if road speed is valid
                              AND road speed <= Reverse engage speed limit (km/h)
```

Brake required to change extends that idea to every change into Drive or Reverse: the brake pedal must be pressed. It starts turned on when a brake is configured, and off when there is none — with no brake wired the vehicle could never change gear at all. **Limit changes by speed** adds a **Change speed limit (km/h)** on moving into Drive; Reverse keeps its own tighter limit above and is not governed twice. Both refuse the change when road speed cannot be trusted, for the same reason the Reverse guard does.

Neither guard ever blocks **Neutral or Park**. Those stay selectable with no brake and at any speed — refusing a driver Neutral is the one thing these guards must never do. And a refused change simply holds the direction you are already in; it never drops you into Neutral.

The leave-Park guard refuses any move out of Park unless the brake is firmly pressed — and it uses a strict brake reading so that a single stuck brake sensor cannot fake a press and let the vehicle creep out of Park on its own.

```
allow leaving Park only if the brake is firmly pressed
```

Whenever the decoded source is reporting invalid (held out past its grace window, ambiguous lines, an expired script request), the engaged direction is forced to Neutral. And once engaged, the direction signs the torque: in Reverse the whole composed motor torque is negated after shaping, limiting, and regen blending, so the wheels turn backward; in Park or Neutral the drive command is held at zero.

Tuning notes

For an Analog selector, set the four voltage bands so they do not overlap and leave a clear gap between them — a voltage that lands between two bands reads as invalid, which is the safe outcome but will pull you to Neutral if the bands are too tight around the real selector voltages. Leave any unused band's low/high at 0 so it stays switched off. If you fit a second selector, start **Max sensor disagreement (V)** generously and tighten it once you have watched the real spread between the two sensors; too tight and ordinary sensor tolerance will keep rejecting good selections.

Size each **Hold-last time (ms)** to ride out the worst real glitch you expect on that selector (a connector wiggle, a brief CAN dropout) without being so long that a genuine loss of the selector takes seconds to show up as Neutral. A few hundred milliseconds is typical.

Set **Change speed limit (km/h)** higher than the reverse limit — engaging Reverse while rolling forward is the dangerous one, which is why it keeps its own number. Set **Reverse engage speed limit (km/h)** to a low walking-pace value — Reverse is meant to be selected at a stop or near-crawl. Remember the guard is fail-safe: if it ever refuses Reverse unexpectedly, suspect the road-speed signal before raising the limit.

For a Script selector, keep **Command timeout (ms)** comfortably longer than the script's own update interval so ordinary timing jitter never trips an expiry, and set **Minimum command spacing (ms)** to match how fast the script is allowed to change position. The defaults — a 1-second timeout and 100 ms spacing — suit a script that asks every 100 ms.

Both direction-change speed limits use speed magnitude. Reverse travel does not bypass a forward-selection limit. Neutral remains available above the limit.

Automatic Drive selection. Digital, Analog and CAN shifters enter Drive when the direction interlocks accept D or R, the accelerator input is valid, and the vehicle is Ready with the high-voltage bus live. A direction accepted during precharge takes effect after Ready. None (forward-only) also enters Drive automatically. Park, Neutral or an invalid direction removes drive permission; the high-voltage bus remains live. Faults, charging and shutdown retain priority. Script is the only source that needs an explicit vehicle-state request to enter Drive.

Settings

- **Shifter source** — How the driver's Park/Reverse/Neutral/Drive choice reaches the vehicle.
- **Reverse engage speed limit** (km/h) — Reverse may only engage below this speed.
- **Brake required to change** — Require the brake pedal to be pressed before the vehicle will move into Drive or Reverse. Neutral and Park can always be selected without it. This needs a brake to be configured under Brake sensing.
- **Limit changes by speed** — Refuse to move into Drive above a road speed. Neutral and Park can always be selected. Reverse has its own, tighter limit below.
- **Change speed limit** (km/h) — Drive may only be selected below this speed. Set it higher than the reverse limit — engaging reverse while rolling forward is the dangerous one, and it keeps its own number.

Digital P/R/N/D settings

- **Switch action** — What kind of switch the shifter uses. **Maintained** stays where you put it — a lever, a rotary, a toggle — so the controller can simply read which position is selected. **Momentary** springs back when released, like a push-button, so the controller remembers the last one pressed until another is pressed.

Momentary buttons cannot be read at power-on: there is no position to look at. So the vehicle starts in Neutral with nothing selected until the driver presses a position. All four inputs must be the same kind.

- **Park input** — The input that signals Park. May be a digital input or a frequency input read as a switch.
- **Park debounce** (ms) — How long the Park line must hold steady before the change is accepted, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.
- **Park switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.
- **Reverse input** — The input that signals Reverse. May be a digital input or a frequency input read as a switch.
- **Reverse debounce** (ms) — How long the Reverse line must hold steady before the change is accepted, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.
- **Reverse switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.
- **Neutral input** — The input that signals Neutral. May be a digital input or a frequency input read as a switch.
- **Neutral debounce** (ms) — How long the Neutral line must hold steady before the change is accepted, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.

- **Neutral switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.
- **Drive input** — The input that signals Drive. May be a digital input or a frequency input read as a switch.
- **Drive debounce** (ms) — How long the Drive line must hold steady before the change is accepted, in milliseconds. Rejects contact bounce; 0 disables it. Maximum 2550 ms.
- **Drive switch type** — Whether this switch's contacts are closed or open at rest. Most switches are normally open — they close when operated. Pick normally closed only if yours is wired the other way.

Analog windows settings

- **Selector sensor** — The analog input wired to the shifter position sensor.
- **Selector sensor (second)** — An optional second selector sensor for redundancy.
- **Max sensor disagreement** (V) — The largest allowed difference between the two selector sensors before a fault.
- **Park window — low** (V) — The lowest sensor voltage that still reads as Park.
- **Park window — high** (V) — The highest sensor voltage that still reads as Park.
- **Neutral window — low** (V) — The lowest sensor voltage that still reads as Neutral.
- **Neutral window — high** (V) — The highest sensor voltage that still reads as Neutral.
- **Reverse window — low** (V) — The lowest sensor voltage that still reads as Reverse.
- **Reverse window — high** (V) — The highest sensor voltage that still reads as Reverse.
- **Drive window — low** (V) — The lowest sensor voltage that still reads as Drive.
- **Drive window — high** (V) — The highest sensor voltage that still reads as Drive.
- **Hold-last time** (ms) — How long to hold the last valid selection when the sensor reads out of range before falling back to Neutral.

CAN shifter settings

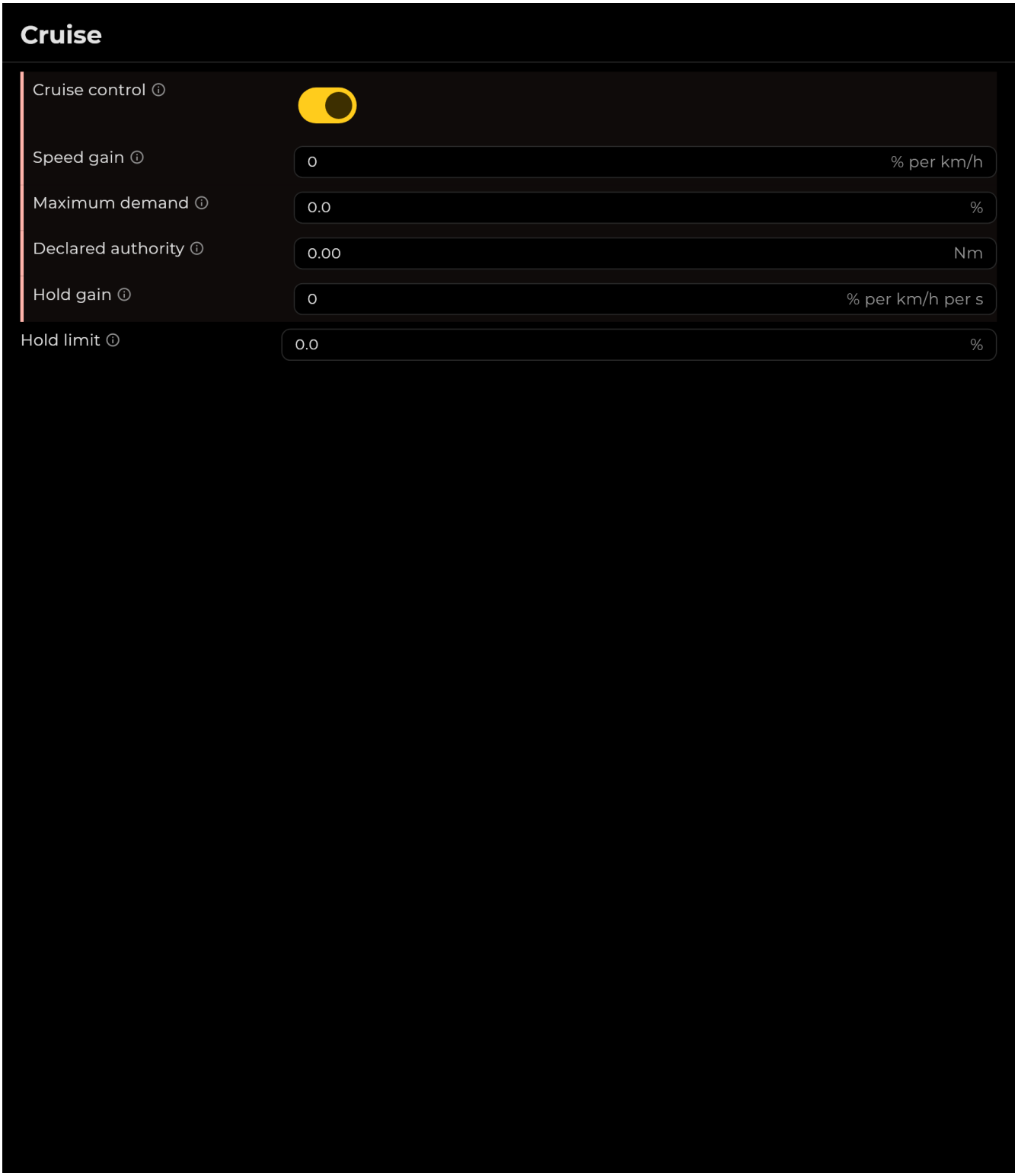
- **Signal** — The CAN signal that indicates this position.
- **Value — low** — The lowest signal value that selects this position. Use the same low and high value for a fixed code.
- **Value — high** — The highest signal value that selects this position.
- **Hold-last time** (ms) — How long to hold the last valid selection on a lost signal before falling back to Neutral.

Script settings

- **Command timeout** (ms) — If a script stops asking for a direction for this long, the vehicle falls back to Neutral and logs a fault. Keep it comfortably longer than the script's own update interval.
- **Minimum command spacing** (ms) — The shortest time allowed between two accepted direction requests. A request that arrives sooner is refused, and the script is told so.

Cruise

Cruise control: holds a set speed, blended with the pedal and cancelled by the brake.



Cruise control holds a road speed you set. While it is engaged it works out how much throttle is needed to keep that speed and feeds that in as a throttle demand, exactly as if a steady foot were on the pedal. It is blended with the real pedal so that whichever is asking for more torque wins — press harder than cruise and you accelerate normally; lift off and the vehicle settles back to the set speed. Touching the brake cancels it instantly. Cruise only ever *adds* drive torque; it never applies braking or regen to slow you down,

so on a downhill it will not hold you back below the set speed — that job belongs to the separate hill-descent feature, not to cruise.

What it needs

Cruise is a speed controller, so it needs a valid, fresh road-speed reading. That speed comes from the drive motors' reported wheel speed. If the speed signal is missing or goes stale, cruise refuses to engage, and if it is already engaged it cancels itself — it will never try to hold a speed it cannot measure.

To engage, the vehicle must be in Drive and the requested set speed must be sensible (a request above 300 km/h is rejected as a fault). Cruise is turned on, cancelled, and resumed from the driver's controls (steering-wheel buttons, dash, or an installer script) — there is no automatic engagement.

On a manual-transmission build cruise also watches the clutch: pressing the clutch cancels cruise, so it cannot keep building throttle while the driveline is open and then lurch when the clutch comes back.

How it works

The **Cruise control** setting is the master switch. With it off, cruise does nothing — it never folds into the throttle demand and never reserves any torque. With it on, the controller runs whenever cruise is engaged.

Cruise runs as two parts. About ten times a second it recalculates how much throttle to ask for. A faster check runs about a hundred times a second, but that fast check only looks for reasons to cancel (brake pressed, left Drive, speed lost) — it never changes the throttle figure, it just lets the rest of the system keep using the last value.

The throttle figure is built from the speed error — how far below the set speed you are:

```
speed error = set speed - measured speed
```

A positive error means you are going too slow and need more torque. Two terms are added together. The first reacts to the error right now; the second slowly builds up to erase any steady error so the exact speed is held on a grade or into a headwind.

```
fast term = speed error x Speed gain
hold term = hold term + (speed error x Hold gain, every 0.1 s)
cruise demand = fast term + hold term
```

Speed gain (% per km/h) sets how hard cruise reacts to being off-speed at this instant. Raise it for a stiffer, quicker-recovering hold; too high and the throttle demand swings noticeably. **Hold gain (% per km/h per s)** sets how quickly the slow build-up erases a standing error. Raise it to settle onto the exact set speed faster on hills; too high and the speed overshoots and hunts up and down.

The result is then capped:

```
if cruise demand > Maximum demand:  cruise demand = Maximum demand
if cruise demand < 0:               cruise demand = 0
```

Maximum demand (%) is the ceiling on how much throttle cruise may ask for on its own, as a percentage of full pedal. It is the only upper limit on cruise's request. Left at 0 it is treated as 100%. The lower limit is fixed at 0 — this is what keeps cruise from ever commanding negative (braking) torque.

To stop the slow build-up term from running away, it is held in check two ways. First, it stops growing whenever the throttle demand is already pinned at a limit, or whenever your foot is already asking for more than cruise — so it never winds up while it is being overridden or clamped. Second, **Hold limit (%)** puts a hard ceiling on how large the build-up term alone may grow:

```
if hold term > Hold limit:   hold term = Hold limit
if hold term < -Hold limit:  hold term = -Hold limit
```

Left at 0, this hard ceiling is off and the build-up is bounded only by the anti-windup logic above and the overall cap. A tighter **Hold limit** limits how much standing throttle the build-up can store, which limits overshoot and any lurch when cruise re-engages.

Smooth take-over. When cruise engages it does not snap to a new throttle figure. It looks at how much pedal you were holding at that moment and starts its calculation so the very first cruise demand matches the throttle you already had. The result is no torque step as cruise takes over — the hand-off is seamless.

Blending with the pedal. The cruise demand is merged with the real pedal (and with hill-hold, if active) by taking whichever is largest:

```
throttle demand = max(pedal, cruise, hill-hold)
```

Because it is a maximum, cruise acts as a torque *floor*: lifting your foot cannot drop the vehicle below the cruise demand, but you can always press past it to go faster. That merged throttle demand then runs through the rest of the torque path — the motor torque map, the gear limiter, regen blending, the rev limiter, and the final torque-safety clamp — exactly like any other throttle request.

Reserving torque (Declared authority). The torque-safety monitor normally permits more torque the more pedal it sees, so at zero pedal it would permit almost nothing — which would strangle a foot-off cruise hold down to nothing. To prevent that, while cruise is engaged it tells the monitor how much torque it is allowed to command, via **Declared authority (Nm)**. The monitor then permits at least that much regardless of pedal position:

```
permitted torque = max(pedal-based permitted torque, Declared authority)
```

Set **Declared authority** to the real maximum torque you want cruise to be able to command. It does not scale the throttle demand or make cruise stronger — it *authorizes* the torque so a lifted-pedal cruise hold survives the safety clamp. Set it too low and cruise will feel weak or unable to hold speed up a grade with your foot off; set it no higher than the torque cruise should ever be trusted to apply.

Cancel and resume. The brake cancels cruise two independent ways for safety. Leaving Drive cancels it, losing the speed signal cancels it, and on a manual build pressing the clutch cancels it. Resume re-engages at the last speed you were holding (re-running the same speed-valid and sensible-speed checks and the same smooth take-over); if there is no stored previous setting, Resume does nothing.

Tuning notes

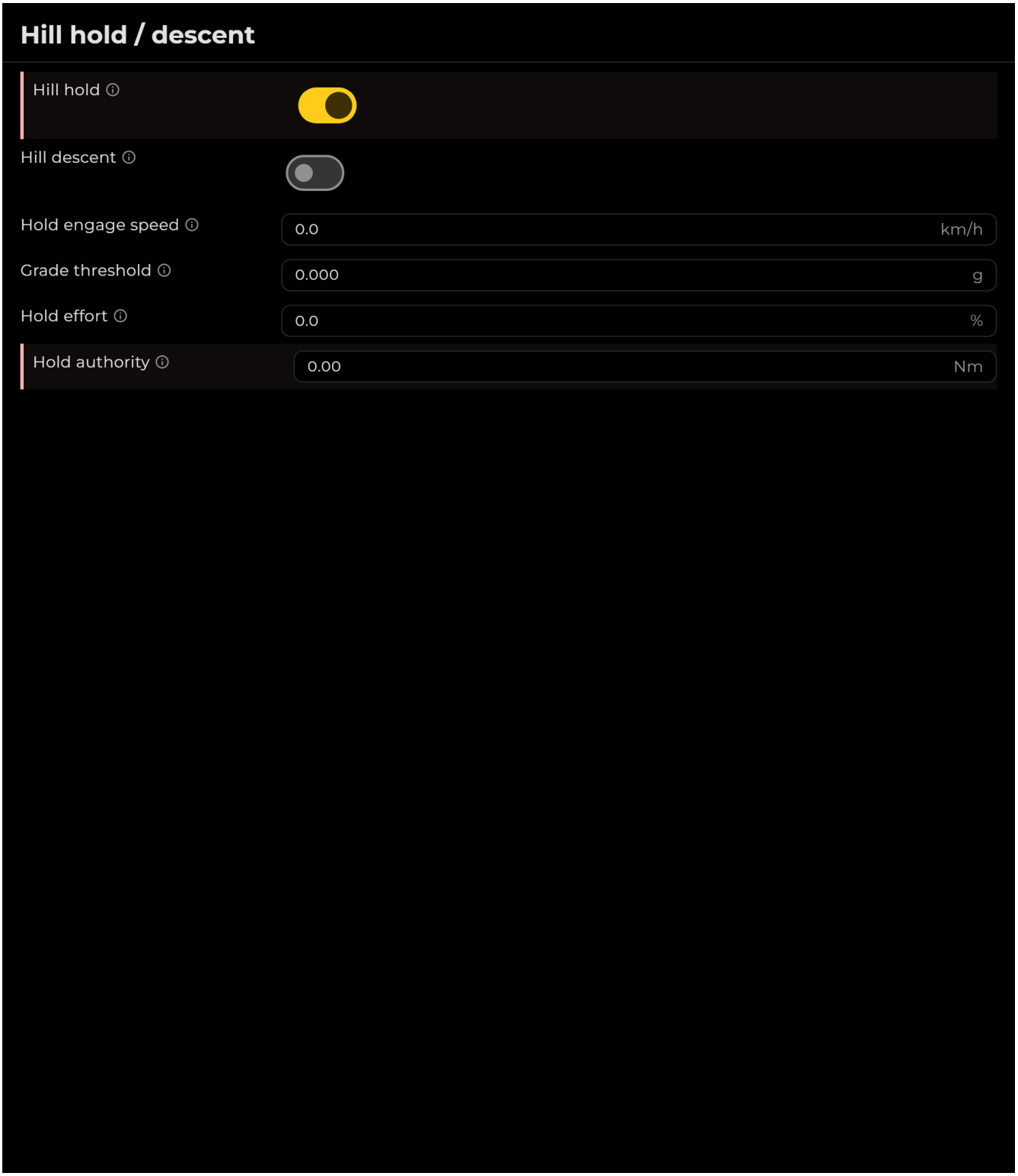
Start with **Speed gain**. Drive a steady road and raise it until cruise holds speed crisply without the throttle visibly pulsing; back off if the vehicle feels nervous or surges. Then add **Hold gain** to clean up the small steady offset that **Speed gain** alone leaves on grades — raise it just until the vehicle settles onto the exact set speed within a few seconds, and no further, or it will overshoot and hunt. Use **Hold limit** to cap how much standing throttle the build-up can hoard, which limits overshoot and any re-engage lurch. Set **Maximum demand** to the most throttle you are comfortable letting cruise request on its own. Finally set **Declared authority** to the real torque ceiling cruise is allowed to apply — high enough that it can hold speed up your steepest expected grade with the pedal released, but no higher than you want the system to ever command unattended. If cruise holds well with your foot lightly on the pedal but sags the moment you lift off, **Declared authority** is the setting to raise.

Settings

- **Cruise control** — Holds a set speed, blended with the pedal and cancelled by the brake.
- **Speed gain** (% per km/h) — How hard cruise pushes to correct a speed error. Higher reacts faster but can feel surgy.
- **Maximum demand** (%) — A cap on how much acceleration cruise may request.
- **Declared authority** (Nm) — The torque cruise is permitted to add, enforced by the independent safety check.
- **Hold gain** (% per km/h per s) — How firmly cruise holds speed on a long grade once settled.
- **Hold limit** (%) — A limit on the accumulated hold effort, to prevent overshoot.

Hill hold / descent

Hill assist: briefly holds the vehicle against rollback on a grade, and caps downhill speed with regenerative braking.



Hill assist is two separate helpers that share one job: keeping the vehicle composed on a slope. **Hill hold** briefly holds the vehicle against rolling backward when you stop on a grade and lift off the pedal. **Hill descent** caps your downhill speed by using regen to brake, so the vehicle settles toward a target speed on a long downgrade instead of running away. The two work independently — you can enable either, both, or neither — and both are recomputed continuously while you drive.

What it needs

Both helpers need a trusted vehicle speed and a trusted grade estimate (a measurement of the slope you're on, read as a longitudinal acceleration in g). The grade estimate comes from whichever Dynamics source you've configured — a direct accelerometer, an estimate inferred from drive torque, or an estimate inferred from suspension travel. If the speed signal isn't trusted, or the grade estimate is missing or stale, both helpers quietly stand down and contribute nothing. That is deliberate: hill hold will never push the vehicle forward when it can't trust how fast it's going, and hill descent will never command regen braking blind — your normal friction brakes remain fully available either way.

Hill hold additionally only acts near a stop: it needs the vehicle to be in Drive, with a valid pedal and a gear direction engaged (forward or reverse), and on a manual-gearbox build it needs the clutch closed (holding motor torque against a disengaged drivetrain does nothing, so it's switched off while the clutch is open).

Hill descent needs the fixed-reduction (non-manual) drivetrain and a regen-capable motor, because it works by commanding negative (regen) torque. On manual-gearbox builds, hill descent does not run at all, regardless of its setting.

How it works — hill hold

While you're stopped or creeping on a slope with your foot off the pedal, hill hold checks two gates each control cycle. First, speed: it only engages below the **Hold engage speed (km/h)** — above that, it's no longer "near standstill" and it disengages. Second, grade: it measures the steepness as an absolute value and only engages once that exceeds the **Grade threshold (g)**. On flatter ground than the threshold, there's no rollback risk, so it stays off.

When both gates pass, hill hold asserts a fixed anti-rollback effort:

```
hold demand = Hold effort (%)
```

This is a constant effort, not a closed loop. It is fed through the same pedal-shaping and torque map as a real pedal press, so **Hold effort (%)** becomes a steady amount of motor torque that resists rollback. Because it reads grade as an absolute value, it engages on both uphill-facing and downhill-facing stops, as long as the slope is steep enough.

Hill hold is blended into the pedal path by taking the larger of the two, never the sum:

```
applied demand = the larger of (pedal, hill hold)
```

So the moment you press the pedal past the hold effort, your foot takes over smoothly — there's no lurch and no stacking. That is the intended hand-off.

There's one more piece. While you're holding with your foot off the pedal, the independent torque-safety monitor would normally expect near-zero torque (your pedal is at zero) and could trim the hold away. To prevent that, hill hold declares a permitted-torque floor to the monitor:

```
permitted torque = the larger of (pedal-based limit, Hold authority (Nm))
```

Hold authority (Nm) must be set at least as high as the torque your **Hold effort (%)** actually produces on the steepest grade you expect to hold on. If it's set too low, the safety monitor will trim the hold and the vehicle can creep backward.

How it works — hill descent

Hill descent watches your speed against the **Descent target speed (km/h)**. At or below that target it does nothing. Once you're over the target on a downgrade, it applies regen braking in proportion to how far over you are:

```
overspeed = vehicle speed − Descent target speed (km/h)
regen demand = overspeed × Descent strength
```

The harder you overshoot the target, the more regen it asks for — a steady proportional pull back toward the target, with no overshoot-and-hunt. That regen demand is then capped so it can never exceed the limit you set:

```
if regen demand > Descent limit (%):  regen demand = Descent limit (%)
```

The capped regen is converted to negative motor torque through the same torque map as the drive direction and subtracted from any drive torque, so the result is net braking that holds your downhill speed near the target. Note that hill descent does not look at how steep the grade is or whether you're at a stop — it acts purely on being over the target speed (with the trusted-speed and trusted-grade preconditions met).

All of this still passes through every downstream limit — the per-gear torque cap, traction control, the torque-safety monitor, and the rev limiter — so neither helper can ever push the vehicle past a safe limit.

Tuning notes

For hill hold, set **Hill hold** on, then pick the **Grade threshold (g)** so the hold arms on the mildest slope where you actually notice rollback and stays quiet on flat ground. Set **Hold engage speed (km/h)** to the creep speed below which you want the hold to be allowed — leave it low for a crisp "only when basically stopped" feel, raise it if you want the hold to persist through slow creep. Then raise **Hold effort (%)** until the vehicle reliably sits still on the steepest grade you expect to stop on. Remember this is a constant effort, not a true zero-rollback regulator: if the effort is too low for the slope, the vehicle will still creep backward — raise the effort rather than expecting it to self-correct. Whenever you raise **Hold effort (%)**, raise **Hold authority (Nm)** to match (a little above the torque that effort produces), or the safety monitor will trim the hold.

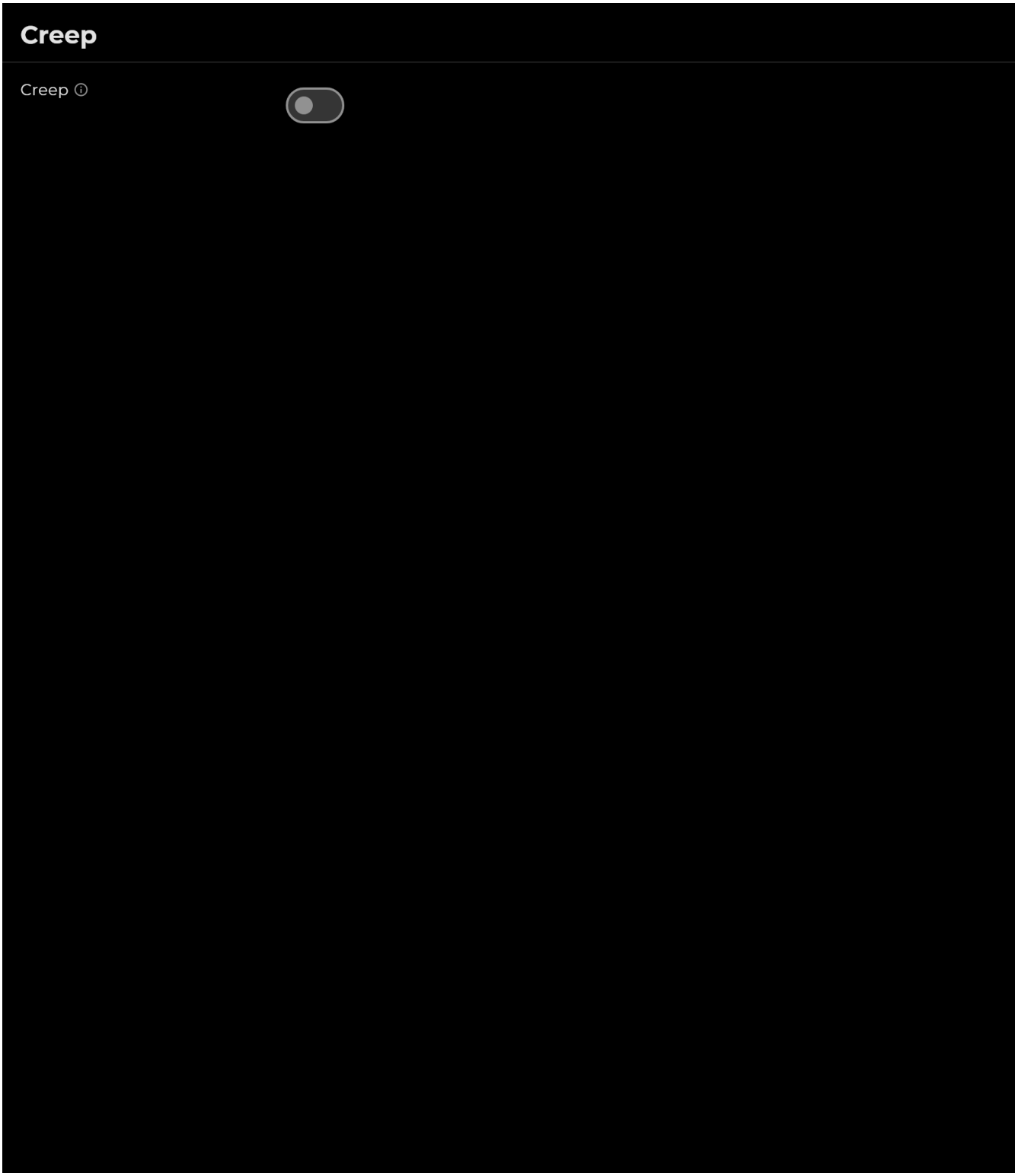
For hill descent, set **Hill descent** on and choose the **Descent target speed (km/h)** you want the vehicle to settle to on a long downgrade — lower targets brake earlier and harder. Tune **Descent strength** for feel: higher values pull back to target more firmly and quickly, lower values feel softer and let speed drift a little further over target before correcting. Finally use **Descent limit (%)** to cap the worst-case regen, so even a steep, fast descent can't ask for more braking torque than you want. A good order is to set the limit generously, tune the strength and target for the feel you want, then bring the limit down to bound the maximum.

Settings

- **Hill hold** — Briefly holds the vehicle against rollback when stopped on a grade.
- **Hill descent** — Caps downhill speed using regenerative braking.
- **Hold engage speed** (km/h) — Hill hold becomes active below this speed.
- **Grade threshold** (g) — How steep the road must be before hill assist activates.
- **Hold effort** (%) — How much torque hill hold applies to resist rollback.
- **Hold authority** (Nm) — The torque hill hold is permitted to use, enforced by the independent safety check.
- **Descent target speed** (km/h) — The downhill speed hill descent tries to maintain.
- **Descent strength** — How hard regen is applied per unit of overspeed on a descent.
- **Descent limit** (%) — A cap on the regen used to hold downhill speed.

Creep

Creep: makes the vehicle crawl at a slow walking pace when you lift off the brake, like an automatic.



Creep makes the vehicle inch along at a slow walking pace when you lift off the brake with the accelerator released -- the familiar behaviour of an automatic that eases forward at a stop light when you come off the brake. It is off by default and is turned on with the **Creep** master switch. It is a comfort feature for stop-and-go traffic, parking, and hill starts; it is not required to drive.

What it needs

Creep needs a trusted vehicle speed, because it eases off as the vehicle approaches the crawl speed. If the speed reading is missing or stale, creep stands down and contributes nothing -- it will never push the vehicle forward when it cannot tell how fast it is going. Your normal brakes are always available. Creep only acts in Drive, with a gear direction engaged (forward or reverse), and only with the brake released -- any brake touch cancels it. On a manual-gearbox build it also needs the clutch closed, so a clutchless one-gear conversion (driven like an automatic) gets full creep while a clutch press switches it off.

How it works

With the vehicle in Drive, your foot off both pedals, and a slope no steeper than the motor can pull, creep applies a small, steady effort that eases the vehicle forward and then tapers away as the vehicle speeds up:

```
crawl effort = Crawl effort x (1 - vehicle speed / Crawl speed)
```

At a standstill it applies the full **Crawl effort**; as the vehicle approaches the **Crawl speed** the effort falls to zero, so the vehicle settles at the crawl speed and cannot overshoot. On a gentle uphill it crawls more slowly; on a steep enough hill it simply holds or stalls, exactly as an automatic does -- creep is not a speed controller and will not force a fixed speed up a grade. In reverse it behaves the same way, and the reverse crawl is additionally held under the vehicle's reverse speed limit.

Creep is blended with the pedal by taking whichever is larger, never the sum:

```
applied demand = the larger of (pedal, creep)
```

So creep acts as a gentle floor: lifting off lets the vehicle crawl, and the moment you press past the crawl effort your foot takes over smoothly. Touching the brake cancels creep to zero at once, so it never fights the brakes -- lift off the brake and it crawls again.

To keep lift-off smooth, creep does not snap on. It eases in over the **Engagement smoothness** time when you come off the brake, so the vehicle moves off gently instead of lurching. Set it to zero for instant engagement.

Because creep asks for torque with your foot off the pedal, it tells the independent torque-safety monitor how much torque it is allowed to command, via **Crawl authority** -- set a little above what the **Crawl effort** actually produces -- so the safety check permits the crawl instead of trimming it away.

Creep and one-pedal driving. Creep and true one-pedal driving are opposite feels and cannot both be had at once. With creep on, lifting off brings the vehicle to a slow crawl and you use the brake to stop and hold; with creep off, lifting off lets the vehicle coast and, if regen is set up for it, slow toward a stop. Choose creep for automatic-like stop-and-go comfort, or leave it off for one-pedal driving.

Tuning notes

Set **Creep** on, then choose the **Crawl speed** you want the vehicle to settle to on level ground -- a slow walking pace is typical. Raise **Crawl effort** until the vehicle pulls away willingly from a stop and holds a gentle crawl on the mild grades you expect; too high feels pushy when you are trying to hold on the brake, too low and it will not move off on a slight rise. Whenever you raise **Crawl effort**, raise **Crawl authority** to

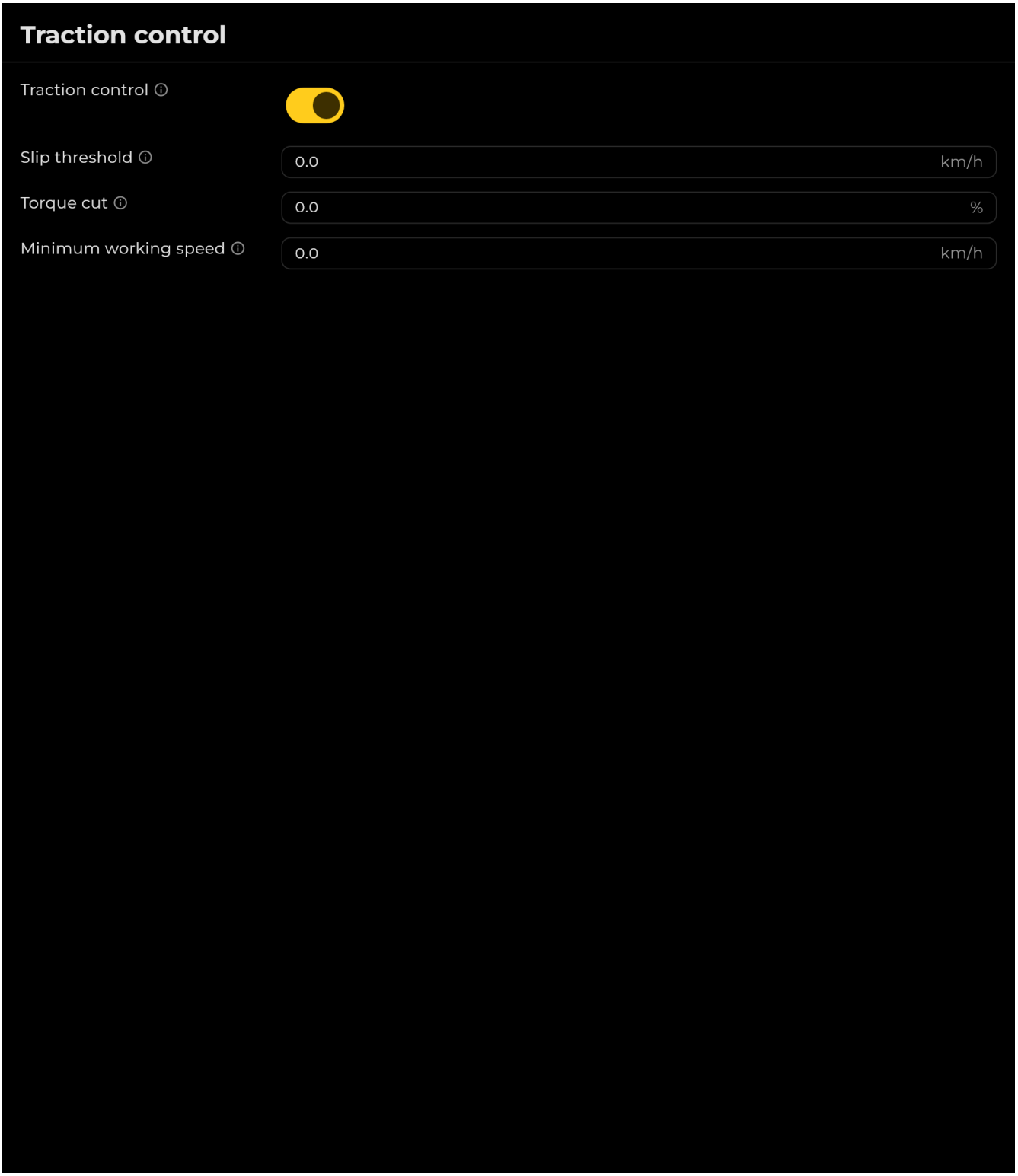
match (a little above the torque that effort produces), or the safety monitor will trim the crawl. Finally set **Engagement smoothness** for a lift-off that eases in without lurching -- longer is gentler, zero is instant.

Settings

- **Creep** — Makes the vehicle crawl at a slow walking pace when you lift off the brake with the accelerator released, like an automatic. Leave it off for one-pedal driving that coasts to a stop — you can't have both, since creep holds the vehicle moving where one-pedal would bring it to rest.
- **Crawl speed** (km/h) — The speed the vehicle crawls to on level ground. Creep eases off as this speed is reached, so it settles here and never overshoots; it crawls slower up a hill and stops on a steep one. In reverse the crawl is also held under the reverse speed limit.
- **Crawl effort** (%) — How much accelerator-equivalent effort creep applies from a standstill. Higher pulls away more briskly; too high feels pushy when you are trying to hold on the brake.
- **Crawl authority** (Nm) — The torque creep is permitted to use, enforced by the independent safety check. Set it a little above what the crawl effort actually needs.
- **Engagement smoothness** (ms) — How gently creep builds up when you lift off the brake, so it eases into the crawl instead of lurching. Zero engages instantly.

Traction control

Reduces drive torque when the driven wheels spin faster than the road. Needs a wheel-speed source.



Traction control reduces drive torque when a driven wheel spins faster than the road underneath it. It only ever *reduces* torque — it never adds any — and it runs as the very last step in the torque path, after the pedal, the front/rear split, and torque vectoring have all had their say. So whatever torque those stages ask for, traction control is the final gate that trims it back when a wheel breaks traction. It is off by default and is armed with the **Traction control** master switch.

What it needs

Traction control needs a wheel-speed source — a per-corner measurement of how fast each wheel (front-left, front-right, rear-left, rear-right) is actually turning, taken either from wheel-speed messages on the CAN bus or from frequency/Hall pickups configured with their pulses-per-revolution and tire size. You set this up under the vehicle's wheel-speed source; traction control will not arm without one.

How much it needs depends on how the vehicle is driven, and it picks the right method automatically from the drive-motor layout:

- **One driven axle** (a single-motor RWD or FWD build, or any conversion that drives only the front or only the rear): it compares the driven wheels against the non-driven wheels, which are just rolling with the road and make an honest road-speed reference. For this method it needs at least one working wheel-speed signal on **both** the driven axle and the non-driven axle.
- **Both axles driven** (all-wheel drive): there are no free-rolling wheels to use as a reference, so each driven wheel is judged on its own. The controller predicts what road speed each wheel's motor *implies* — from the motor rpm, the gear ratio, and the tire size — and compares that against that same corner's measured wheel speed. For this method each driven corner needs a valid motor-speed reading, a configured gear ratio, and a non-zero wheel/tire radius for its axle.

In all cases, if a needed signal is missing or stale, traction control quietly stands down for that moment rather than guessing — a dropped wheel-speed signal will never strand torque or trigger a phantom cut. It simply does nothing until good data returns.

How it works

On every control cycle while driving, traction control measures how much faster a driven wheel is turning than the road, and if that overspeed is large enough, it scales that wheel's torque down.

For a vehicle with **one driven axle**, it averages the driven wheels and the non-driven (reference) wheels, then takes the difference:

```
driven speed    = average speed of the driven wheels      (km/h)
reference speed = average speed of the non-driven wheels   (km/h)
slip            = driven speed - reference speed            (km/h)
```

It compares speed *magnitudes*, so it works identically forward and in reverse. Two gates decide whether anything happens:

```
if reference speed < Minimum working speed:   do nothing this cycle
if slip <= Slip threshold:                     do nothing this cycle
otherwise:                                      cut the driven wheels
```

The first gate is the **Minimum working speed**: at a crawl the percentage difference between wheels is too noisy to trust, so below this road speed traction control stands down (set it to 0 to remove the floor entirely). The second gate is the **Slip threshold**: the driven wheels must be spinning faster than the reference by more than this margin before any torque is cut. When both gates pass, every driven wheel's torque is scaled down by the same proportion.

For a vehicle with **both axles driven**, each driven wheel is judged on its own:

```
predicted wheel speed = motor rpm, through the gear ratio and tire size, as km/h
slip                  = predicted wheel speed - that wheel's measured speed   (km/h)
cut that wheel when slip > Slip threshold
```

If a particular wheel's measured speed isn't available that cycle, that wheel is simply left alone while the others are still checked. (The **Minimum working speed** floor does not apply to this all-wheel-drive method.)

In both methods the actual reduction is a single proportional scale:

```
new torque = torque x Torque cut / 100
```

Note carefully what **Torque cut** means here: it is the share of torque the wheel is allowed to *keep* once it's slipping, not the share removed. At 100% the wheel keeps all its torque (no real cut at all); at 50% its torque is halved; at 20% it keeps only a fifth (an 80% reduction); at 0% the slipping wheel's torque goes to zero. So a **smaller** number cuts **harder**. Whichever direction the wheel was driving, the cut preserves it — reducing the magnitude only.

The cut is recomputed from scratch every cycle. There is no ramp, no memory, and no build-up: while a wheel is slipping it gets the flat proportional trim, and the instant slip falls back to or below the **Slip threshold**, full torque is restored on the very next cycle. The live dashboard shows a traction-control indicator while torque is actively being cut.

Because traction control only ever scales torque down, it can never push a wheel past its own independent torque-safety limit — it sits underneath that hard limit as a softer grip-and-comfort layer.

Tuning notes

Start with the **Slip threshold**: lower it for earlier, more aggressive intervention (the wheels are allowed less overspeed before torque is trimmed), raise it to tolerate more wheelspin before stepping in. If the vehicle feels like it's surging then catching on a slick launch, the threshold is too high; if it feels like it's being held back on dry pavement during normal cornering, it's too low.

Set **Torque cut** to control how deep each intervention goes, remembering that the number is the torque the wheel *keeps*: 100% is no cut (useful only to confirm the rest of the setup before committing to a real reduction), something around 40–60% is a moderate trim, and small numbers approaching 0% are an aggressive near-total cut for very low-grip surfaces. If traction control feels too abrupt — torque dropping out hard the moment a wheel slips — raise this value so more torque is retained. If wheels keep spinning through the intervention, lower it.

Because the cut has no hysteresis, a vehicle held right at the **Slip threshold** can flutter the intervention on and off; if you see that chatter, widen the **Slip threshold** a little so the system isn't living exactly on the trigger point.

On a single-driven-axle build, use **Minimum working speed** to keep traction control from reacting to wheel-speed noise at very low speed — raise it if you see spurious low-speed interventions, lower it

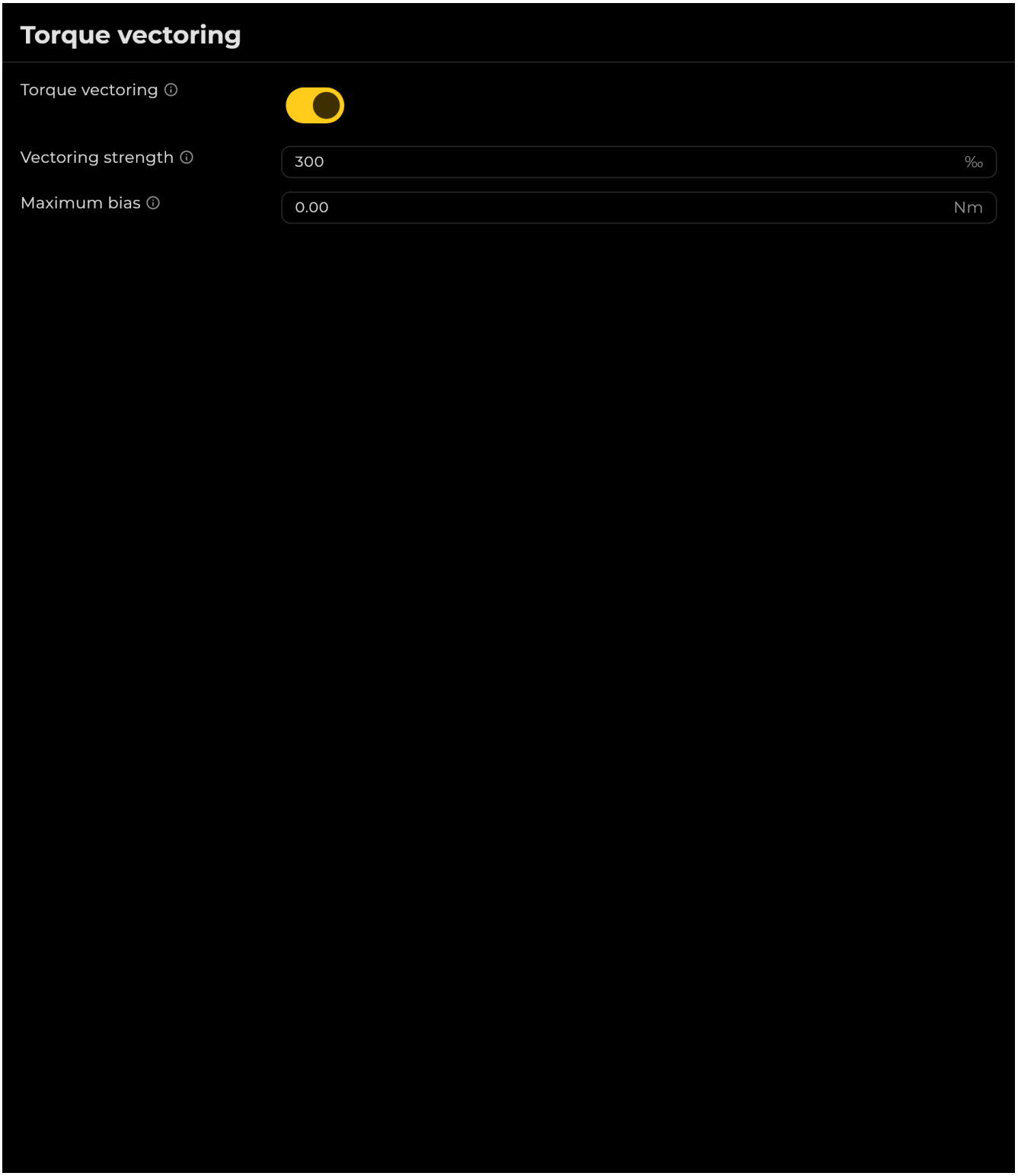
(toward 0) if you need traction control active right down to a crawl. This setting has no effect on all-wheel-drive builds, which judge each wheel from its motor speed instead.

Settings

- **Traction control** — Reduces drive torque when the driven wheels spin faster than the road.
- **Slip threshold** (km/h) — How much faster the driven wheels may spin before torque is cut.
- **Torque cut** (%) — How strongly torque is reduced once wheelspin is detected.
- **Minimum working speed** (km/h) — Below this speed the slip measurement is too noisy, so traction control stands down. Zero means no floor.

Torque vectoring

Biases torque side-to-side on a two-motor axle to help the vehicle turn. Needs a yaw-rate source.



Torque vectoring sends slightly different torque to the left and right wheels of a two-motor axle so the vehicle rotates more willingly into a turn. It does not change how much torque the axle makes in total — it only moves some of that torque from one side to the other, so the pedal feel stays the same.

What it needs

Torque vectoring works on an axle that has its own left and right drive motors (a driver-side motor and a passenger-side motor on the same axle). On a single-motor axle there is nothing to bias, so it has no effect there. It also needs a yaw-rate signal — a measurement of how quickly the vehicle is turning, in degrees per second. If no yaw-rate source is configured, or its signal is missing or stale, torque vectoring quietly stands down and the axle is driven evenly. You choose a yaw-rate source under Dynamics → Producers.

How it works

On every control cycle while driving, the controller reads the yaw rate and turns it into a torque shift:

```
torque shift = yaw rate × strength
```

The shift is then capped so it can never exceed the maximum bias you set:

```
if torque shift >  maximum bias:  torque shift =  maximum bias
if torque shift < -maximum bias:  torque shift = -maximum bias
```

Finally the shift is applied equally and oppositely across the axle: one wheel's torque goes up by the shift and the other wheel's goes down by the same amount. Because the two changes cancel, the axle's total torque — and therefore the pedal feel — is unchanged; only the side-to-side balance moves. In a corner this puts more torque on the outer wheel and less on the inner wheel, which helps turn the vehicle. The harder and faster the vehicle is turning, the larger the shift, up to the cap.

Each wheel's result is still bounded by the independent torque-safety check, so vectoring can never push a wheel past its own torque limit; if a wheel is already near its limit, the shift it receives is trimmed to fit.

Tuning notes

Tune the strength first with the maximum bias set generously, then lower the maximum bias to clamp the worst case. If the vehicle darts or feels unstable when you lift off or brake mid-corner, lower the strength. If it pushes wide in steady cornering, raise it. Because the axle total never changes, vectoring is safe to tune independently of the overall torque map and the front-to-rear split.

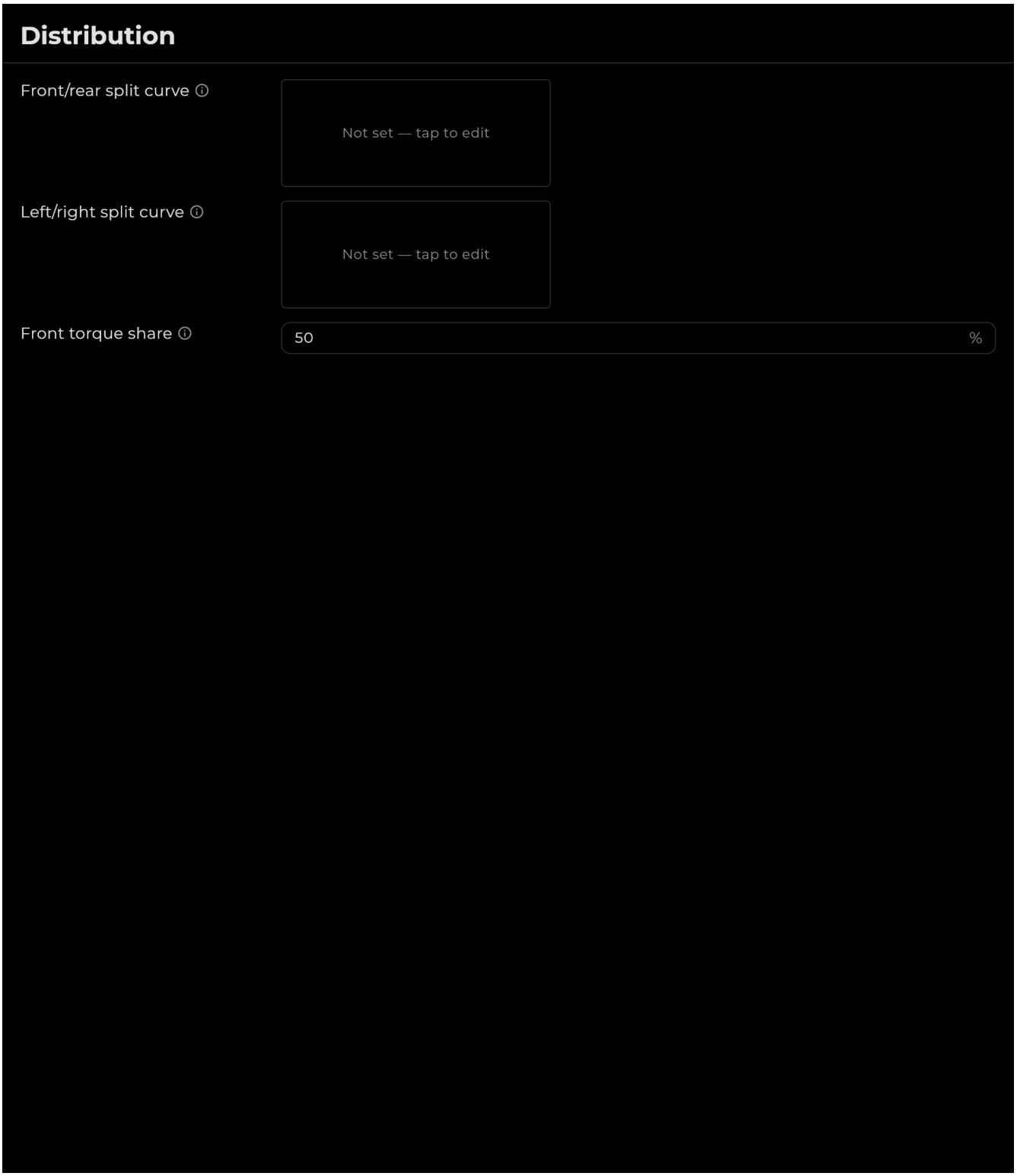
If either motor reaches a limit, the transfer shrinks so torque that cannot be transferred stays at the other motor. The combined propulsion demand must remain within the driver-authority limit in either motor direction. New safety limits can still reduce total torque.

Settings

- **Torque vectoring** — Biases torque side-to-side on a two-motor axle to help the vehicle turn.
- **Vectoring strength** (‰) — How aggressively torque is shifted toward the outside wheel in a corner. Think of this as how eager the vehicle is to rotate: it sets how much the left/right torque split responds to the rate of turn. A gentle setting is felt only in hard corners; a stronger one sharpens turn-in everywhere but can feel twitchy on a slippery surface. Start gentle and raise it until the vehicle turns the way you want without feeling unsettled when you lift off mid-corner. The Maximum bias setting caps the result no matter how high you set this.
- **Maximum bias** (Nm) — A cap on how much torque may be shifted between the two wheels.

Distribution

How total torque is split front-to-rear and side-to-side as the vehicle accelerates and corners.



Distribution is the stage that takes the single total torque figure the pedal map has already produced for this moment — the total torque at the motor shafts, in Nm, with the correct sign for whichever direction you're

driving — and decides how much of it each drive motor makes. It splits that total front-to-rear first, then side-to-side on each axle, before torque vectoring and traction control run. It never creates or loses torque: every split is share-conserving, so the pieces always add back up to the same total the pedal asked for. On a manual-gearbox build there is only one motor and one axle, so distribution does nothing — the whole total goes to that motor.

What it needs

Distribution only commands torque when the vehicle is actually driving: the vehicle state is Drive, the pedal is reading valid, and a forward or reverse direction is engaged. Outside that, every motor command is zero.

It needs at least one traction motor with a vendor configured. Slots set up as a generator or an accessory drive are skipped — they never receive a share of drive torque. Each traction motor's position (front or rear axle, and driver-side, passenger-side, or centered) comes from how you've assigned its role.

To split anything other than evenly, distribution also needs live measurements:

- For the front-to-rear split it needs a valid **longitudinal acceleration** signal — how hard the vehicle is accelerating or decelerating along its length, in g (positive accelerating, negative braking). If that signal is missing or stale, the front-to-rear split falls back to an even 50/50.
- For the side-to-side split it needs a valid **lateral acceleration** signal — how hard the vehicle is cornering, in g. If that signal is missing or stale, the side-to-side split falls back to even.

Both signals come from your configured dynamics sources (an accelerometer / IMU, or an inferred estimate). And each split needs its curve to be usable — see below.

How it works

Every control cycle while driving, distribution runs in two passes.

First it works out the front axle's share. It reads the longitudinal acceleration and looks it up on the **Front/rear split curve**, whose output is the front axle's percentage of the total torque (0 to 100):

```
front percent = Front/rear split curve looked up at longitudinal acceleration (g)
front torque  = total torque x front percent / 100
rear torque   = total torque - front torque
```

Writing the rear share as "total minus front" is what keeps the two sides adding up exactly to the total. If only one axle has motors, that axle gets 100% and the other gets nothing — the curve isn't consulted. If the longitudinal signal is invalid, or the curve is empty or unusable, the front percent is simply 50.

Then, on each axle independently, it splits that axle's share between the two sides using the **Left/right split curve**, whose output is the driver-side percentage (0 to 100) read against lateral acceleration:

```
driver percent   = Left/right split curve looked up at lateral acceleration (g)
driver-side share = axle torque x driver percent / 100
passenger share   = axle torque - driver-side share
```

The same curve and the same lateral-g reading are used on both axles. This side-to-side split only applies on an axle that has exactly one driver-side motor and one passenger-side motor. On an axle with a single

centered motor, or only one side fitted, that motor takes the whole axle share and the curve is ignored. If the lateral signal is invalid, or the curve is empty or unusable, the driver percent is 50 (even).

Both curves read flat outside their calibrated range: an acceleration below the first point on a curve uses the first point's value, and above the last point uses the last point's value — they never extrapolate past their ends. A curve left empty is a deliberate "stay even" setting and raises no fault. A curve that is filled in but malformed (points out of order, or too many of them) also reverts to even, but flags a one-time information notice at startup so you know the table didn't take.

If a side of an axle has more than one motor (for example two motors on the driver side), that side's share is divided evenly among them.

Finally, distribution checks every motor's command against the independent torque-safety monitor and trims if needed — but it trims the whole axle together, not one motor in isolation:

```
for each motor: allowed fraction = (torque the monitor permits) / (torque requested)
axle trim       = the smallest allowed fraction among that axle's motors
each motor command = its share x axle trim
```

Applying the same trim to every motor on the axle is deliberate: it keeps your front-to-rear and side-to-side ratios intact even when one motor is the limiting factor. A single motor that's near its limit pulls the whole axle down in proportion rather than collapsing on its own and distorting the balance you tuned. If the monitor allows a motor zero torque (for example a faulted or absent inverter), that motor drags its axle's commands to zero and the vehicle drops into a degraded, reduced-power state. If the overall safety cutoff has tripped, every motor command becomes zero.

Because the pedal map already produced torque measured at the motor shaft, these percentages divide motor-shaft torque directly — there is no gear-ratio conversion anywhere in distribution.

A note on **Front torque share (%)**: despite the name, this setting does **not** drive the front-to-rear split above. The dynamic front-to-rear ratio comes entirely from the **Front/rear split curve**. **Front torque share (%)** is a separate, fixed front-to-rear ceiling used only by the independent ASIL-C torque-safety monitor. The monitor takes the total drive torque it will permit at any moment and reserves that fraction for the front axle and the remainder for the rear:

```
front axle ceiling = permitted total x Front torque share / 100
rear axle ceiling  = permitted total x (100 - Front torque share) / 100
```

The monitor never looks at the acceleration signals or the split curves — it is a deliberately independent safety layer. So you have two front-to-rear authorities that don't talk to each other: the live **Front/rear split curve** that shapes feel, and the static **Front torque share (%)** that sets the safety ceiling the trim step above ultimately binds against. If only one axle has traction motors, that axle gets the full permitted budget regardless of this setting. A setting of 0 (all of the safety budget allowed to the rear) or 100 (all to the front) are both valid.

Tuning notes

Tune the **Front/rear split curve** by shape: with weight transferring rearward under acceleration you'll typically send more torque rearward at higher positive g, and shift it forward under braking-side (negative)

g. Leave a curve empty to get a clean 50/50 — that's a supported choice, not a fault. If you fill a curve in and the vehicle drives evenly anyway, check that the table flagged no startup notice and that the relevant acceleration sensor is configured and live; without a valid signal the split silently stays even.

Tune the **Left/right split curve** as a cornering bias against lateral g; remember it applies to both axles with the same curve, and only on axles that actually have a left and a right motor. This lateral-g bias is the base side-to-side balance; torque vectoring (driven by how fast the vehicle is turning) layers on top of it and adds to it — it does not replace it.

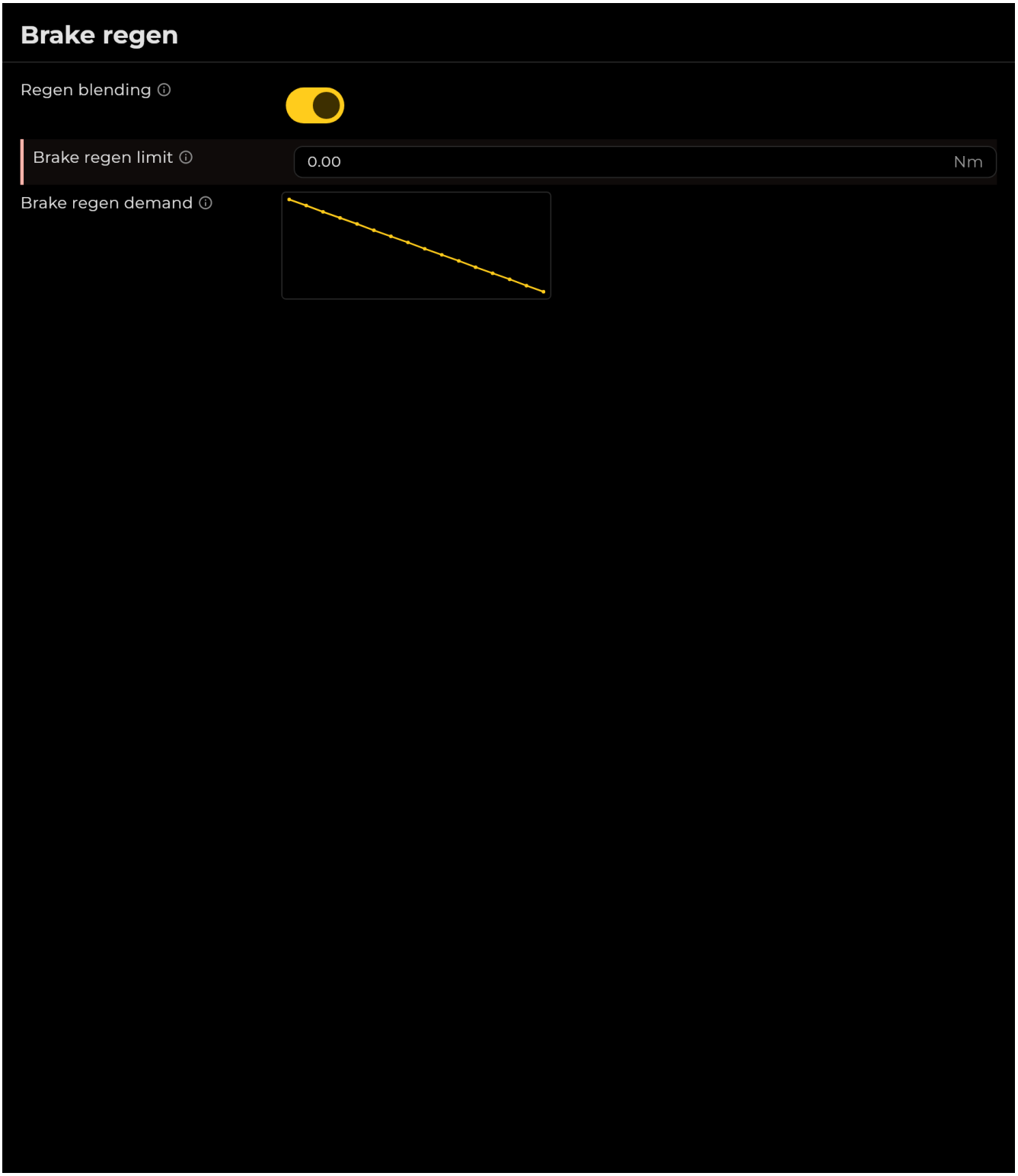
Set **Front torque share (%)** to match the front-to-rear capability you want the safety layer to enforce — think of it as the guardrail, not the everyday balance. Keep it consistent with the general intent of your **Front/rear split curve**: if the curve routinely sends, say, most torque to the rear but the safety share reserves most of the budget to the front, the safety ceiling will be the thing that limits the rear axle in hard acceleration. Because every split is share-conserving and the pedal total never changes, you can tune the front-to-rear and side-to-side balance without re-tuning the overall torque map.

Settings

- **Front/rear split curve** — How drive torque is split front-to-rear as the vehicle accelerates or brakes.
- **Left/right split curve** — How an axle's torque is biased side-to-side during cornering.
- **Front torque share** (%) — The share of total torque the independent safety check attributes to the front axle. Set conservatively for your drivetrain.

Brake regen

The shared brake-demand curve requests total regenerative braking.



Brake input requests negative total motor-shaft torque. A switch uses the 100 percent endpoint. An analog pedal uses its measured input percentage.

The VCU selects the stronger of brake demand and lift-off regen. It does not add them. Each drive profile supplies a speed-fade curve and separate application and release rates.

The fade uses vehicle-speed magnitude and scales the combined request from zero to 100 percent. Regen stops at standstill or when speed is invalid. Holding remains separate.

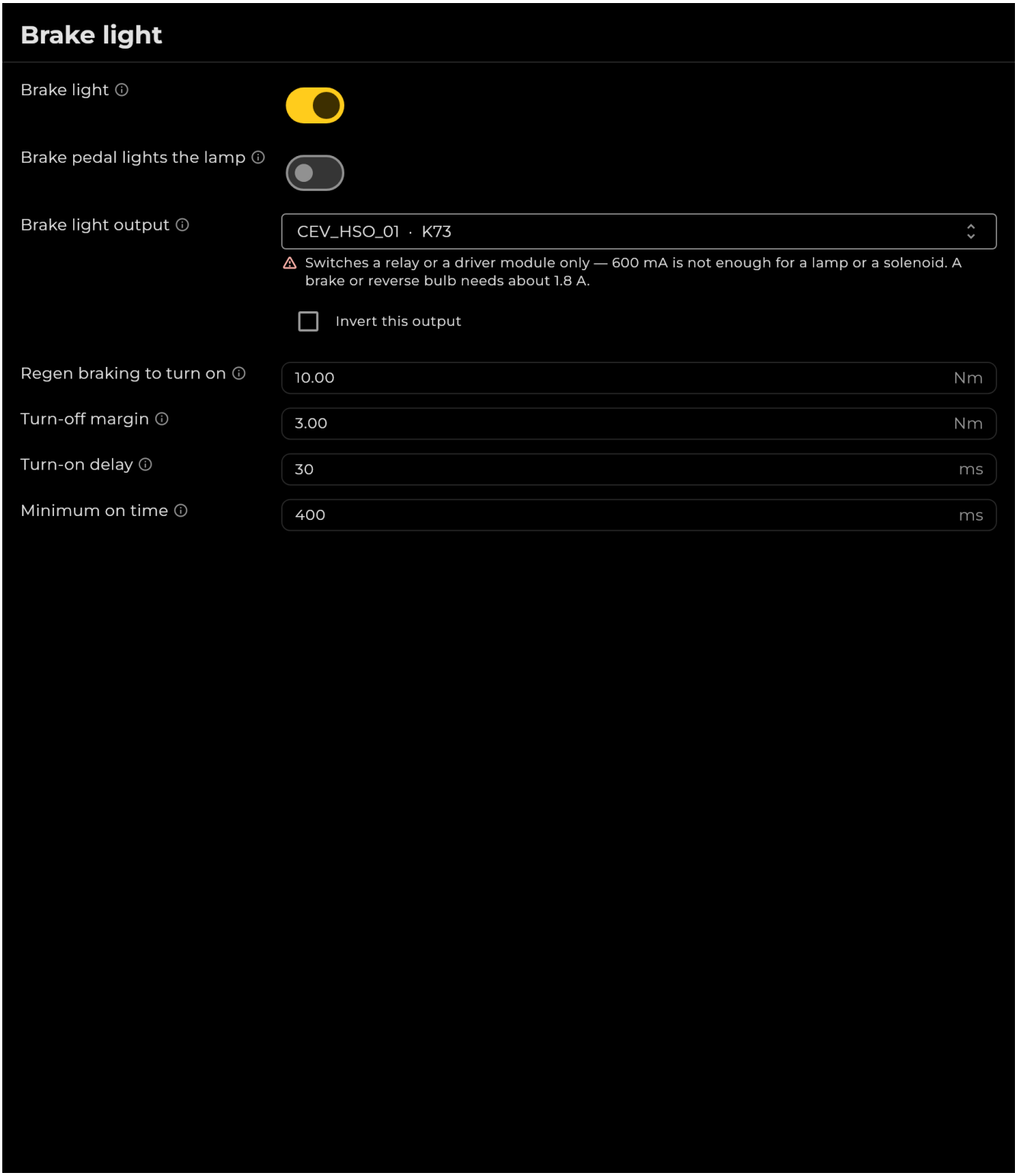
Regen opposes actual motor motion in forward, reverse, and rollback. The brake regen limit caps the brake request. Shared safety limits apply immediately.

Settings

- **Regen blending** — Applies regenerative braking when the brake pedal is pressed.
- **Brake regen limit** (Nm) — The maximum total motor-shaft braking torque permitted for the brake pedal. Enter a positive magnitude.
- **Brake regen demand** — Brake input requests negative total motor-shaft torque. A brake switch uses the 100 percent endpoint. The stronger regen request wins.

Brake light

Turns the brake lights on when the vehicle slows itself under regen braking or while holding on a hill.



The brake light turns on the vehicle's brake lamps whenever the VCU is slowing the vehicle by itself — without the driver pressing a friction-brake pedal — so the traffic behind sees the same red light it would see during normal braking. There are three situations that count as "the vehicle is slowing itself": the motors are pushing back against your direction of travel (regen braking) when you lift off the throttle, hill-hold is keeping the vehicle still on a grade, and hill-descent is using regen to cap your speed going

downhill. The controller never measures brake-pedal pressure — there is no brake-pressure sensor in this path. It infers braking from how much torque the drive motors are actually commanding, read straight from the drivetrain. On a vehicle whose brake lights are driven by the VCU alone, **Brake pedal lights the lamp** adds a fourth situation: the brake pedal is pressed, as reported by the vehicle's brake input. It only reads that torque; it never changes it, so turning the brake light on or off can never affect how the vehicle drives.

What it needs

You must turn the feature on with **Brake light** and assign a real output for the lamp with **Brake light output** — a high-side or low-side digital output that drives the brake-lamp relay coil (or the lamp directly). With **Brake light** off, the feature is completely inert: it claims no output, drives nothing, and reports no fault. **Brake pedal lights the lamp** is off by default: in a stock harness the mechanical brake switch is wired to the lights itself, and the two paths must not double up. Turn it on only when the VCU output is the sole thing driving the brake lights, and make sure a brake input is configured, because the pedal is read from that input. If you turn it on but leave **Brake light output** unassigned, the controller cannot do its job and flags a configuration fault instead of silently doing nothing — so check that the output is set whenever the feature is enabled. All faults this feature can raise (a missing output, or a wiring/short fault detected on the lamp output) are indication-only: they light a diagnostic but never change torque, never open contactors, and never affect the safety state of the vehicle.

The regen, hill-hold, and hill-descent conditions all come from the drivetrain controller, so this feature only does something while the vehicle is actually driving. In Park or Neutral — or any state where the vehicle is not driving — there is no commanded torque to read, so the lamp stays off.

How it works

On every control cycle while driving (about every 10 ms), the controller looks at how much torque the front and rear motors are commanding, adds them together, and works out how much of that is braking — torque pushing back against the direction you are travelling:

```
regen amount = how hard the motors are pushing against your direction of travel, in Nm
```

This is direction-aware. Going forward, braking torque is what slows you down; in reverse, the geometry flips but the controller accounts for it, so the brake light works correctly when backing up too. Either way the result is a braking amount in Nm that is zero or positive — pure acceleration (driving the vehicle faster) never counts as braking.

The regen condition uses a turn-on threshold with a margin underneath it, so light coasting doesn't make the lamp flicker on and off:

```
turn-off level = Regen braking to turn on - Turn-off margin    (never below 0 Nm)

regen condition turns ON   when regen amount is at least Regen braking to turn on
regen condition turns OFF  when regen amount falls to turn-off level or below
in between those two values, the condition holds whatever it was
```

So the lamp's regen trigger arms once braking reaches **Regen braking to turn on**, and disarms only after braking eases back down past **Turn-off margin** below that point. The gap between the two is a deadband

that keeps the lamp steady while you feather the throttle. (If you ever set **Turn-off margin** as large as or larger than **Regen braking to turn on**, the turn-off level simply clamps at 0 Nm — the regen trigger then releases only when braking reaches zero, which is harmless but very sticky.)

The regen condition, the hill-hold condition, the hill-descent condition, and (when **Brake pedal lights the lamp** is on) the brake pedal are then combined — if any one of them is active, the lamp wants to be on. The pedal counts only while the brake input reads valid, so a faulted brake signal never lights the lamp:

```
lamp wanted = regen condition OR hill-hold active OR hill-descent active OR (brake pedal
pressed, when enabled)
```

Torque vectoring (shuffling torque side to side in a corner) and traction-control torque cuts also happen in the drivetrain, but they are deliberately not brake-light conditions — neither one means the vehicle is slowing for the driver behind to see.

Finally, that combined "lamp wanted" signal is timed before it actually drives the output, with a fast turn-on and a slow turn-off:

```
lamp turns ON   after "lamp wanted" stays true for Turn-on delay
lamp turns OFF  after "lamp wanted" stays false for Minimum on time
```

Turn-on delay keeps a brief flicker of regen from blipping the lamp; keep it short so engagement feels immediate. **Minimum on time** does double duty — it is both the turn-off delay and a minimum time the lamp stays lit once it comes on. After braking ends, the lamp holds on for this long, which keeps it steady through the natural ripple in regen torque as you ease off the throttle on a coast, instead of strobing every time the braking amount wobbles across the threshold.

Tuning notes

Start with **Regen braking to turn on**. Set it to the amount of regen braking that should clearly read as "slowing down" to following traffic — low enough that meaningful lift-off deceleration lights the lamp, high enough that a trickle of regen while cruising does not. If the lamp comes on too eagerly during gentle coasting, raise it; if it fails to light when the driver clearly feels the vehicle slowing, lower it.

Then set **Turn-off margin** for stability. A larger margin widens the deadband, so the lamp stays committed through throttle modulation and won't chatter on a wavering coast; too small and the lamp can blink near the threshold. A few Nm is usually enough.

Use **Turn-on delay** to reject brief blips — keep it short (it is meant to be snappy) so the brake light appears promptly when the driver lifts off. Use **Minimum on time** to set how long the lamp lingers after deceleration ends and to smooth out regen ripple; longer values give a steadier, more visible lamp at the cost of it staying lit a little after you're back on the throttle. If the lamp ever strobes during a long trailing-throttle coast, raise **Minimum on time** first, then widen **Turn-off margin**.

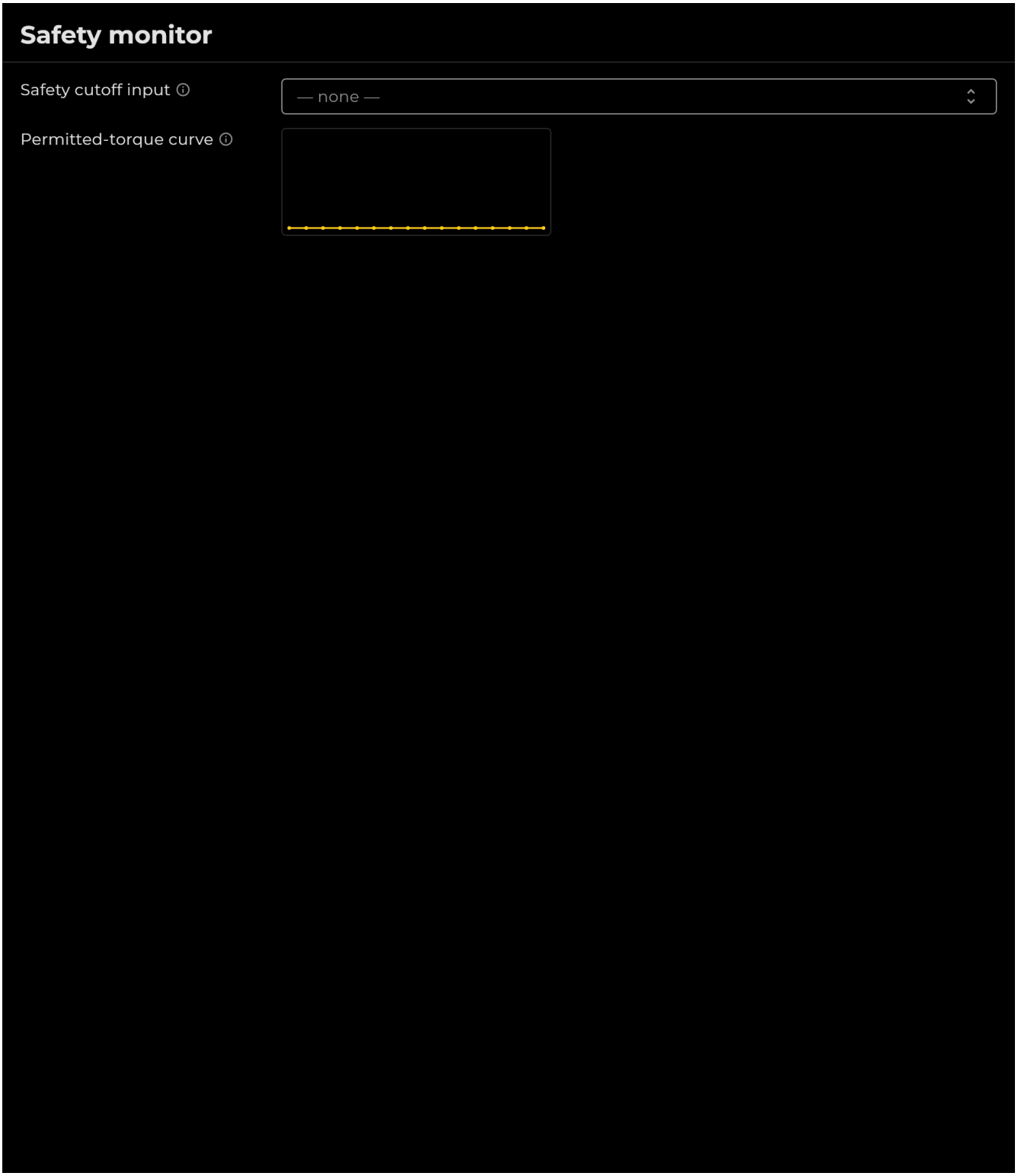
The triggers — regen, hill-hold, hill-descent, and the pedal when enabled — share these same timing settings and the regen threshold/margin; hill-hold, hill-descent, and the pedal always light the lamp whenever they're active, with no separate threshold to tune.

Settings

- **Brake light** — Turns the brake lights on when the vehicle slows itself — under regen braking or while holding on a hill. Wire the mechanical brake switch to the lights separately, or turn on Brake pedal lights the lamp below to let the VCU drive them alone.
- **Brake pedal lights the lamp** — Also turns the brake lights on while the brake pedal is pressed, for a vehicle whose brake lights are driven by the VCU alone. Leave off when the mechanical brake switch is wired to the lights in the harness, or the two paths double up.
- **Brake light output** — The output that switches the brake lights, driving the lamps directly or through a relay.
- **Regen braking to turn on** (Nm) — How much regenerative braking effort is needed before the brake lights come on.
- **Turn-off margin** (Nm) — How far below the turn-on level regen must fall before the lights go back off, so they don't flicker while coasting.
- **Turn-on delay** (ms) — How quickly the lights respond once braking begins.
- **Minimum on time** (ms) — How long the lights stay on after braking eases, keeping them steady through light, varying regen.

Safety monitor

The independent safety check that bounds how much torque the vehicle may ever produce. Set these conservatively for your drivetrain.



The safety monitor is an independent watchdog that sits at the very end of the torque path and puts a hard ceiling on how much torque the vehicle is ever allowed to make. It does not decide how much torque to produce — the pedal, cruise, hill-hold and the distribution logic do that. The monitor's only job is to re-check, on its own, what is physically and legally allowed at each wheel right now, and to clamp every torque command down to that limit before it can reach a motor. It checks raw pedal readings itself against a

separate permitted-torque curve. The sensor calibration is shared with the pedal control, so an incorrect sensor endpoint can affect both paths. Set its limits conservatively for your drivetrain: this is the last line of defense, not a place to chase performance.

What it needs

The monitor leans on a few inputs, and it fails toward zero torque whenever any of them is missing or implausible:

- A wired **Safety cutoff input** if you want the hard kill (see below). Leave it unassigned to disable that layer.
- Its own pedal readings, converted using the sensor endpoints and electrical fault limits set on the Accelerator pedal page. The monitor applies its own immediate range checks and fixed 20% channel disagreement limit.
- Live data from each inverter: how much forward and regen torque that motor currently has available, its shaft speed, and a health/validity flag. If an inverter reports absent, faulted, or "torque not valid," that corner is held at zero.
- The battery pack's health status. Keeping pack current within its limits is the inverter's job, not the monitor's; the monitor only watches pack health, and if a configured pack reports a fault (for example it has gone offline) it drops propulsion torque to zero rather than draw from a pack it can no longer trust.
- The vehicle must be in Drive. Outside Drive, every corner's ceiling is zero.

The monitor tracks the front axle and the rear axle separately, and it knows which motors exist on your build. A generator motor, if fitted, still passes through the cutoff and the per-wheel physical limit, but it is left out of the per-axle pedal cap and the disagreement watchdog.

How it works

Every control cycle the monitor runs four independent guard layers. A command has to survive all of them.

1. Safety cutoff (system-wide hard kill). The **Safety cutoff input** is wired energize-to-run: the input must be held HIGH to allow torque. The moment it reads LOW it trips.

```
tripped = Safety cutoff input is assigned AND input reads LOW
```

On a trip the monitor immediately commands the vehicle into its fault-safe state, logs a cutoff fault, and forces every motor — drive and generator alike — to 0 Nm until the input goes HIGH again. This overrides every other layer. If the input is left unassigned, the cutoff never trips.

2. Permitted-torque cap (whole-drivetrain pedal ceiling). The **Permitted-torque curve** maps the monitor's own pedal reading to a permitted *total* drive torque for the whole drivetrain. It is authored separately from the main torque map, so it acts as a true upper bound rather than a copy of the demand.

```
permitted total = Permitted-torque curve looked up at the monitor's own pedal position
```

If a foot-off source like cruise or hill-hold is active, the permitted total is raised to at least that source's authority, so an automatic hold isn't strangled by a zero-pedal lookup. The permitted total is then split across the axles by the front torque split you set:

```
front permitted = permitted total x front split % / 100
rear  permitted = permitted total x (100 - front split %) / 100
```

On a single-drive-axle build the present axle gets the entire permitted total and the absent axle gets nothing; a split of 0% or 100% is a legitimate "all to one axle" setting. Within a cycle the monitor keeps a running tally of torque already granted on each axle and hands out only what's left:

```
remaining on axle = max(0, axle permitted - torque already granted this cycle)
if a corner command exceeds remaining, it is trimmed to remaining
```

If the **Permitted-torque curve** is empty or invalid this whole cap is simply switched off and the per-wheel physical limits below still apply. If the monitor's pedal reading is bad — out of range, or its two channels disagree by more than 20% of pedal travel — the permitted total is forced to 0 (no drive torque) and a pedal-fault is logged until the reading is plausible again.

3. Per-wheel torque limits. Each motor has a positive and a negative shaft-torque bound. The inverter supplies these bounds when available. A healthy inverter without torque-availability data uses the configured maximum motor torque as a fallback. Outside Drive, or with an absent or faulted inverter, both bounds are zero.

The guard uses shaft rotation to distinguish propulsion from braking. A manual reverse gear can reverse the wheels while the motor still turns forward. Torque in the direction of motor rotation is propulsion. Torque against known rotation is braking. At standstill or with unknown speed, both signs require driver or machine authority.

Speed governor. An enabled governor reduces propulsion over the last 200 rpm before the speed limit. At the limit, propulsion is zero. Braking remains available against known rotation. If motor speed becomes unknown, both torque bounds become zero.

Battery fault. A configured battery system with a fault blocks propulsion in either shaft direction. Braking retains its inverter limit while shaft direction is known. At standstill or with unknown speed, both bounds are zero. An installation without a battery management system does not trigger this fault.

Regen-share cap. An active regen blender also limits negative shaft torque to an equal share of its total authority across the configured traction motors.

4. Disagreement watchdog (debounced escalation). Layers 2 and 3 flag a "capped" event whenever a command overshoots a freshly recomputed ceiling by more than 16 Nm — a deliberately generous tolerance so ordinary rounding and one-cycle timing skew never trip it, while a genuine runaway of tens or hundreds of Nm always does. The monitor counts consecutive capped cycles per axle:

```
capped this cycle  -> count up (one per cycle)
clean cycle        -> count resets to 0 and any latched fault clears
count reaches 5    -> log disagreement fault, request degraded (limp) state
count reaches 50   -> request fault-safe state instead
```

Whichever fault state is highest severity wins, so a cutoff trip or a fault on the other axle is never quietly downgraded by this layer.

Tuning notes

Set the **Permitted-torque curve** to sit a little below the drivetrain's true capability at every pedal position — it should never be higher than what the demand side can ask for. If you size it too high it stops being a meaningful limit; if you size it too low the vehicle will feel capped before it reaches its motors' real output. Use the front torque split to decide how that permitted total is shared front to rear; note this is a fixed split for the safety ceiling and is intentionally independent of any dynamic front/rear distribution you tune elsewhere, which keeps the two from fighting.

Calibrate the sensor endpoints once on the Accelerator pedal page and verify the full travel of each channel. Both the pedal control and safety monitor use these settings.

Wire the **Safety cutoff input** energize-to-run: the input is held HIGH in normal operation and any open circuit, broken wire, or pressed E-stop drops it LOW and kills torque. Confirm on the bench that releasing the cutoff drops every motor to 0 Nm and that restoring it recovers cleanly. Leave it unassigned only if your build genuinely has no external torque-kill — otherwise assign it.

The guard uses motor rotation to distinguish propulsion from braking in either direction. A manual reverse gear can reverse the vehicle while the motor still turns forward. At standstill or with unknown motor speed, both torque directions require driver or machine authority. With an enabled speed governor, loss of motor speed blocks both torque directions.

Pedal calibration. Set the sensor rest and full-travel voltages and electrical fault limits once on the Accelerator pedal page. The safety monitor uses those settings, reads the sensors itself, and applies its own fixed disagreement check. Its permitted-torque curve remains separate from the drive profiles. Verify the measured pedal travel because an incorrect sensor endpoint affects both paths.

Settings

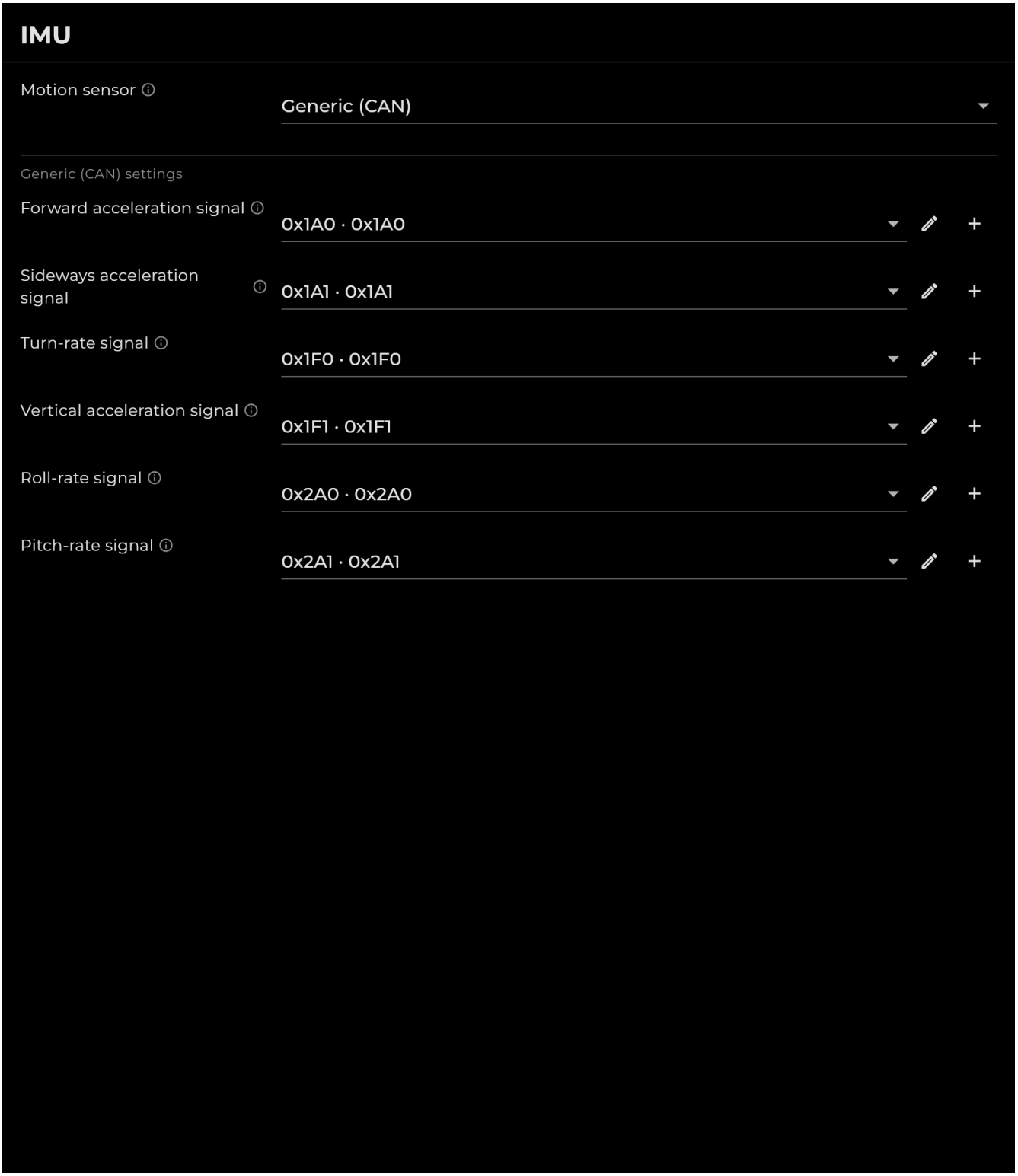
- **Safety cutoff input** — An input that must be energized for the vehicle to produce torque; losing it commands a safe stop. Leave unset if no such cutoff is wired.
- **Permitted-torque curve** — The independent safety check's own ceiling on total drive torque versus pedal demand.

Dynamics

The optional motion sensors — inertial, wheel-speed, steering, and suspension — and which of them provides each piece of vehicle-dynamics information.

IMU

An inertial sensor providing acceleration and turn rate, if fitted.



Settings

- **Motion sensor** — The inertial motion sensor fitted, chosen by name — a Bosch MM5.10 or its ruggedized MM5.10-R variant (both report acceleration and turn rate), the full six-axis Bosch MM7.10 (which also reports vertical acceleration, roll and pitch), or a generic sensor whose readings you map to CAN signals yourself.

Bosch MM5.10 / Bosch MM5.10-R / Bosch MM7.10 settings

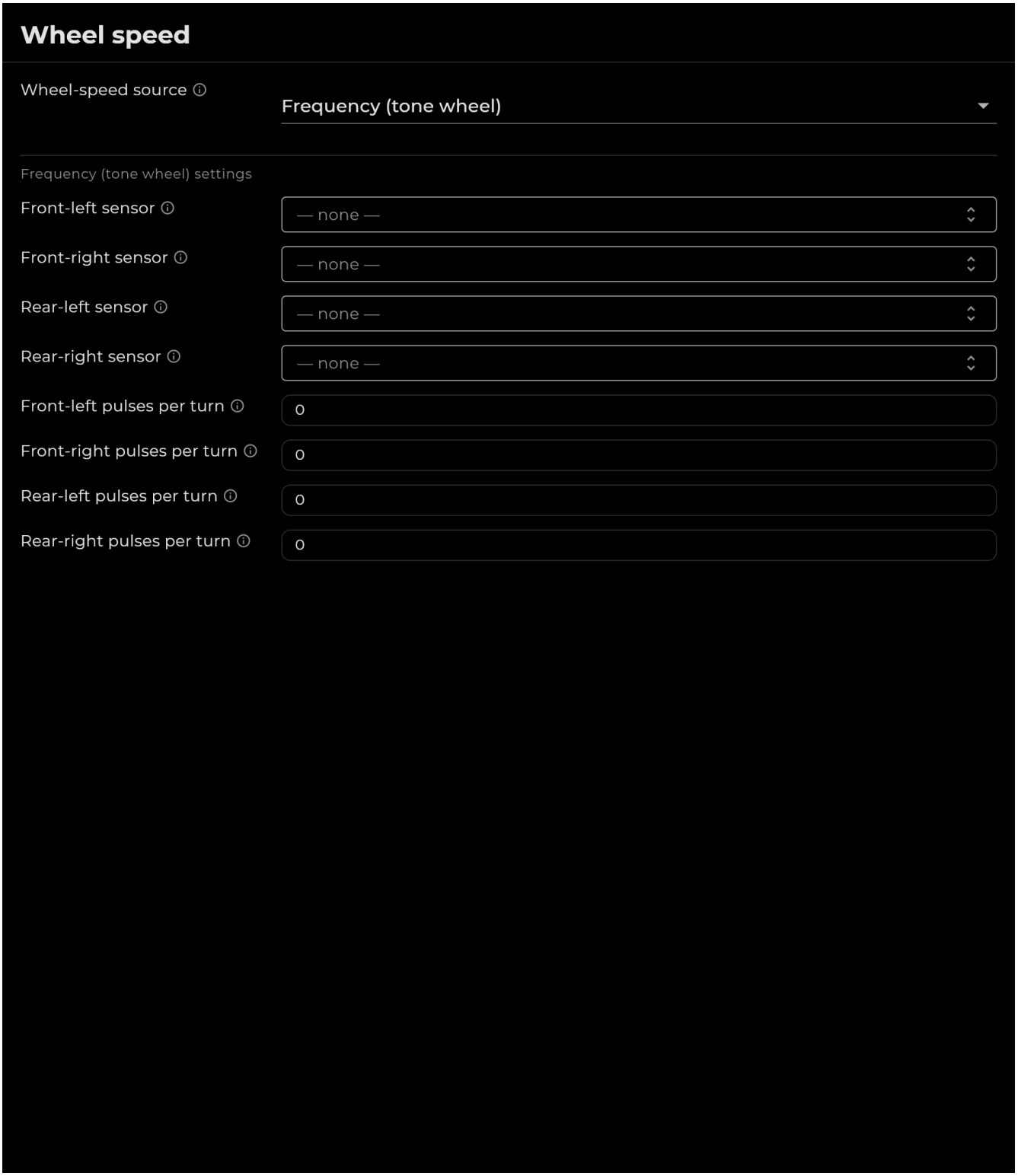
- **CAN bus** — Which CAN bus the motion sensor is wired to. The Bosch sensors report on their own fixed messages, so once the bus is set the vehicle reads them automatically — there is nothing to map.

Generic (CAN) settings

- **Forward acceleration signal** — The CAN signal carrying forward/backward acceleration.
- **Sideways acceleration signal** — The CAN signal carrying side-to-side acceleration.
- **Turn-rate signal** — The CAN signal carrying how fast the vehicle is turning.
- **Vertical acceleration signal** — The CAN signal carrying vertical (up/down) acceleration.
- **Roll-rate signal** — The CAN signal carrying how fast the vehicle is rolling (leaning side to side).
- **Pitch-rate signal** — The CAN signal carrying how fast the vehicle is pitching (nose up/down).

Wheel speed

How per-wheel road speed is measured — tone-wheel sensors or a CAN module.



Wheel speed measures how fast each corner of the vehicle is actually rolling on the road, reported separately for the front-left, front-right, rear-left, and rear-right wheels in km/h. This is *road* speed at each tire, which is different from the headline speedometer the driver sees — that number comes from the drive motor's rpm, not from this subsystem. Wheel speed instead feeds the vehicle's handling and dynamics features (traction control, stability, yaw estimation, and the corner-aware parts of the torque and regen

path). You pick how it is measured with **Wheel-speed source**: Off (no wheel-speed sensing), tone-wheel sensors wired to the VCU's frequency inputs, or a CAN module (an ABS/ESC unit that already broadcasts per-corner speed). When **Wheel-speed source** is Off, every corner simply reports "not measured" and nothing faults.

What it needs

Which settings matter depends entirely on the **Wheel-speed source** you choose.

If you choose tone-wheel sensors, each corner you want to use needs both of its settings filled in: the input it is wired to (**Front-left sensor**, **Front-right sensor**, **Rear-left sensor**, **Rear-right sensor**) and its tooth count (**Front-left pulses per turn**, **Front-right pulses per turn**, **Rear-left pulses per turn**, **Rear-right pulses per turn**). The rolling tire size comes from the wheel radii you enter under the vehicle's dimensions, defined once for the whole vehicle rather than per corner, so turn vehicle dimensions on before you rely on frequency wheel-speed sensors. If any one of those three is left at zero for a corner, that corner is treated as "not fitted" — it quietly reports no measurement and does **not** raise a fault. The sensor must also be delivering a live signal; if a wheel is stopped, disconnected, or the input has timed out, that corner reads zero and is marked not-valid.

If you choose a CAN module, each corner instead needs its broadcast speed identified: **Front-left speed signal**, **Front-right speed signal**, **Rear-left speed signal**, and **Rear-right speed signal**. Leaving a corner's signal unset marks that corner not fitted, with no fault. The CAN module must be on the bus and sending fresh frames; if its broadcast goes stale, the corner is marked not-valid.

You do not have to wire every corner. A three-corner setup, a single-axle setup, or a two-sensor bench build are all legitimate and will not fault — the corners you leave unconfigured simply report no measurement. Be aware that the handling features which rely on wheel speed (traction control and future stability features) stand themselves down when their corners read not-valid, so a partial or absent wheel-speed setup quietly removes those features rather than complaining.

How it works

With tone-wheel sensors, each corner has a toothed ring that passes a sensor as the wheel turns. The VCU counts how many teeth pass per second, then converts that to road speed using the tooth count and the tire size from your vehicle dimensions:

```
wheel speed (km/h) = (pulses per second / pulses per turn) x (tire circumference / 1000) x
3.6
```

Here **pulses per turn** is the tooth count, **tire circumference** is the rolling circumference in millimeters (the "/ 1000" turns that into meters of travel per wheel revolution), and the "x 3.6" converts meters-per-second into km/h. Two things follow from this. First, a higher tooth count gives finer resolution but does not change the computed speed — it is divided back out. Second, **tire circumference** scales the result directly: enter a circumference that is 5% too large and that corner will read 5% fast.

With a CAN module, the VCU does not do this math itself. It reads the speed each corner's broadcast already carries, in km/h, and uses it as-is. Out-of-range broadcast values are clamped to the highest (or lowest) value the system can hold rather than wrapping around.

Either way, only fresh, in-range readings count. The fail-safe rule is the same for both sources and is the important thing to understand: every consumer of wheel speed checks the per-corner "valid" flag, never the bare number. When a corner drops out — wheel stopped, sensor unplugged, CAN gone quiet — that corner's value is forced to zero and its valid flag is cleared, so a stale reading can never be mistaken for live data. A frozen, plausible-looking number is exactly what this design prevents.

Once the corners are measured, the dynamics system averages the corners that are currently valid to form a single road-speed estimate, and (when both front corners are valid) compares the front-left and front-right speeds to estimate how quickly the vehicle is turning:

```
road speed = average of the valid corner speeds
turn rate (deg/s) = (front-right speed - front-left speed) / track width
```

The turn-rate estimate uses the front pair only, and needs a sensible **track width** entered in the vehicle's dimensions for the result to mean anything. These dynamics values then feed the torque arbiter, regen blending, cruise, gear logic, vehicle state, and the live telemetry dashboard.

One limitation to know: wheel speed is always treated as forward motion. Tone-wheel sensors cannot tell direction (a tooth passing looks the same whether the wheel spins forward or back), and the system keeps every corner non-negative, so reverse roll is not represented in these numbers.

Tuning notes

The single most important entry is **tire circumference** for each corner — measure the actual rolling circumference of the fitted tire (roll the loaded tire one full revolution and measure the distance, in millimeters) rather than trusting a sidewall spec, because this value scales the reported speed one-for-one. Get **pulses per turn** exactly right from the tone-ring's tooth count; an off-by-one here throws the corner's speed off proportionally. After setup, drive at a known steady speed and confirm each corner reads close to the others and close to a trusted reference — a single corner reading high or low almost always means a wrong tire circumference or tooth count on that corner. If a corner refuses to read at all, check first that its sensor input, **pulses per turn**, and **tire circumference** are all non-zero, since any one of them left at zero disables that corner silently. For a CAN module, confirm each **... speed signal** points at the right broadcast and that the module is live; a corner that stays not-valid usually means the wrong signal was selected or the module is not transmitting.

Settings

- **Wheel-speed source** — How per-wheel road speed is measured, if at all.

Frequency (tone wheel) settings

- **Front-left sensor** — The frequency input wired to the front-left wheel sensor.
- **Front-right sensor** — The frequency input wired to the front-right wheel sensor.
- **Rear-left sensor** — The frequency input wired to the rear-left wheel sensor.
- **Rear-right sensor** — The frequency input wired to the rear-right wheel sensor.
- **Front-left pulses per turn** — How many sensor pulses occur per wheel revolution at the front left.
- **Front-right pulses per turn** — How many sensor pulses occur per wheel revolution at the front right.

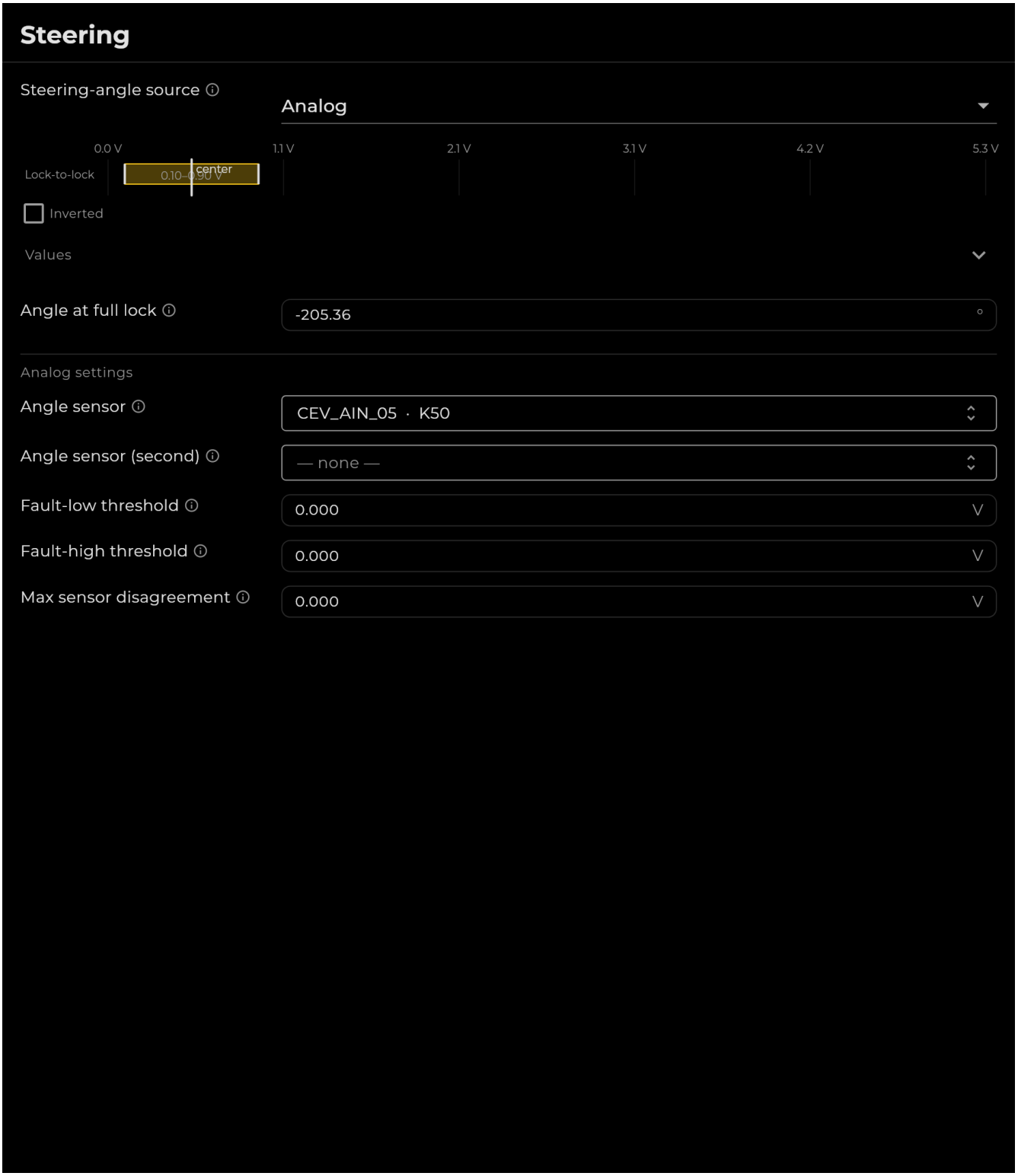
- **Rear-left pulses per turn** — How many sensor pulses occur per wheel revolution at the rear left.
- **Rear-right pulses per turn** — How many sensor pulses occur per wheel revolution at the rear right.

CAN settings

- **Front-left speed signal** — The CAN signal carrying front-left wheel speed. Define its scale and offset so it decodes to km/h.
- **Front-right speed signal** — The CAN signal carrying front-right wheel speed. Define its scale and offset so it decodes to km/h.
- **Rear-left speed signal** — The CAN signal carrying rear-left wheel speed. Define its scale and offset so it decodes to km/h.
- **Rear-right speed signal** — The CAN signal carrying rear-right wheel speed. Define its scale and offset so it decodes to km/h.

Steering

How the steering angle is measured, if at all.



Steering tells the controller which way the front road wheels are pointed and how far. It is a sensing function only — it never makes torque or moves anything by itself. Its single job is to publish a clean road-wheel steering angle, in degrees, that the vehicle-dynamics system can use. From there the controller can optionally estimate how fast the vehicle is turning (yaw rate, in deg/s) and how hard it is cornering (lateral

acceleration, in g), and those estimates can feed torque vectoring and the left-to-right torque split. If you do not need any of that, you can leave steering off and nothing else is affected.

What it needs

First, pick how the angle is measured with **Steering-angle source**. There are three choices:

- **None** — there is no steering sensor. The angle is simply reported as "not available" on every cycle, and this never raises a fault. Use this when the vehicle has no steering-angle sensing.
- **Analog** — a steering-angle potentiometer (or similar voltage sensor) wired to an analog input. The controller converts its voltage to an angle. This is the mode with the full calibration below.
- **CAN bus** — the angle arrives as a message on the CAN bus from another module that already knows the road-wheel angle.

For the **Analog** source you need a sensor wired to the input you select under **Angle sensor**, plus a valid three-point calibration: **Sensor at center (V)**, **Sensor at full left (V)**, and **Sensor at full right (V)** must all be different voltages, and **Angle at full lock (°)** must be greater than zero. If any of those is missing or contradictory, steering reports a configuration fault once at power-up and then reports "not available" on every cycle — it will never publish an angle until the calibration is fixed.

For the **CAN bus** source you need to point **Steering-angle signal** at the incoming CAN message that carries the road-wheel angle, and the source module must actually be transmitting it. If that signal is left unset, steering reports a loud configuration fault.

Whichever source you use, the published value must be the **road-wheel** angle (how far the tires are turned), not the steering-wheel angle. If your sensor measures the steering wheel, divide out the steering ratio during calibration — see the tuning notes.

Finally, for steering to actually influence anything, the dynamics system must be told to use the sensor (set the steering-angle source under Dynamics to the sensor, not an inferred value), and torque vectoring or the left-right split must be set to derive their inputs from steering. None of that happens automatically from this screen alone.

How it works

The controller reads steering every 10 ms, just before it assembles the rest of the vehicle-dynamics picture, so the angle is always fresh for that cycle.

Analog source. Every cycle the controller reads the sensor voltage on **Angle sensor**. If you have also set **Angle sensor (second)** to a real input, the system runs in dual-track mode: it reads both sensors, checks them against each other, and only then averages them. Leaving **Angle sensor (second)** unset runs a single sensor with no cross-check.

Two safety checks run before any angle is published:

Range check — if you have set **Fault-low threshold (V)** and/or **Fault-high threshold (V)**, each sensor reading must fall inside that window:

```
fault-low threshold  <=  sensor voltage  <=  fault-high threshold
```

A reading below the low threshold or above the high threshold means a broken, shorted, or disconnected sensor. Steering records a range fault and reports "not available" for that cycle. The fault is logged once when it first goes out of range and clears itself automatically when the reading comes back in range, so it does not spam. If you leave both thresholds at zero, the range check is simply switched off — useful on an un-calibrated bench build so it does not fault constantly.

Disagreement check (dual-track only) — the two sensors must read close enough together:

```
voltage difference = | sensor 1 voltage - sensor 2 voltage |
fault if  voltage difference > Max sensor disagreement (V)
```

If the two sensors disagree by more than **Max sensor disagreement (V)**, steering records a disagreement fault and reports "not available" — it will not trust a pair of sensors that do not agree. Like the range fault, it is logged once and self-clears on recovery. Note: if you are running dual-track, **Max sensor disagreement (V)** must be greater than zero. Leaving it at zero in dual-track mode is treated as a setup error (a zero tolerance can never pass two real, slightly-noisy sensors), so the angle is never published until you give it a real tolerance.

When the checks pass, the controller computes the angle. In dual-track it first averages the two sensor voltages. Then it measures how far the reading is from straight-ahead and scales that into degrees:

```
offset = sensor voltage - Sensor at center (V)

if turned right (offset >= 0):
    span = Sensor at full right (V) - Sensor at center (V)
else (turned left):
    span = Sensor at center (V) - Sensor at full left (V)

steering angle = offset / span x Angle at full lock (°)
```

In words: **Sensor at center (V)** is straight-ahead and reads as zero degrees. The voltage swing from center out to **Sensor at full right (V)** (or out to **Sensor at full left (V)**) is treated as the full-lock swing, and **Angle at full lock (°)** is the road-wheel angle that swing represents. The angle is positive to the right and negative to the left, and it is capped at roughly ±327 degrees (far beyond any real steering travel). If the calibration is degenerate — for example a full-lock voltage that lands on the same side as center, giving a zero or negative span — the controller quietly reports "not available" rather than publishing a wrong angle.

CAN bus source. The angle is read directly from the **Steering-angle signal** message, which the sending module has already converted to road-wheel degrees. There is no range or disagreement check here — instead, the controller trusts the message only while it is fresh. If the message stops arriving (a stale or silent signal), steering reports "not available." It does not run the analog calibration; the upstream module owns the conversion.

What the angle does downstream. Once steering publishes a valid angle and the dynamics system is set to use the sensor, the controller can infer turning behavior from it together with road speed and the vehicle's wheelbase:

```
yaw rate        = road speed x steering angle / wheelbase
lateral accel   = road speed x road speed x steering angle / wheelbase
```

Road speed comes from the wheel-speed sensors, so at least one fresh wheel-speed corner and a real wheelbase value are required for these estimates; without them they switch off. The yaw rate can then drive torque vectoring, and the lateral acceleration can drive the left-to-right torque split. Crucially, the moment steering goes "not available" — sensor fault, stale message, or source set to None — every estimate built on it drops out, and the torque path falls back to neutral, even behavior. The fail-safe is always "no angle," never a wrong angle.

Tuning notes

Calibrate the three voltage points by hand, on the vehicle, with the wheels physically positioned. Center the steering and read the sensor voltage into **Sensor at center (V)**. Turn full lock left, read that into **Sensor at full left (V)**; full lock right into **Sensor at full right (V)**. The normal wiring convention is that left reads a lower voltage than center and right reads a higher voltage. If the angle comes out backwards (reads right when you turn left), your sensor is wired with the opposite polarity — swap the left and right voltage entries (or re-wire) so the sign matches reality. The controller will faithfully report whatever polarity you give it, so verify it.

Set **Angle at full lock (°)** to the actual road-wheel angle at the lock positions you just calibrated — typically somewhere around 30 to 40 degrees for a passenger vehicle. If your sensor is on the steering wheel rather than the road wheels, divide the steering-wheel travel by the steering ratio first: for example, if the steering wheel turns 540 degrees from lock to lock for 35 degrees of road wheel, enter the road-wheel figure (35), not the steering-wheel figure. Everything downstream assumes road-wheel degrees.

For the protection thresholds, set **Fault-low threshold (V)** just below and **Fault-high threshold (V)** just above the full voltage range the sensor can legitimately produce across full travel, with a little margin for noise — tight enough to catch an open or shorted wire, loose enough not to trip on a healthy sensor at the ends of travel. If you genuinely want the range check off (bench testing), leave both at zero.

If you have fitted a second sensor for redundancy, set **Max sensor disagreement (V)** to the largest voltage gap you ever expect between two healthy sensors at the same wheel position — small enough to catch a failing channel, but with enough headroom that normal sensor-to-sensor tolerance and noise never trip it. Start a little generous, watch for nuisance faults, then tighten. Remember it must be non-zero whenever a second sensor is configured.

Settings

- **Steering-angle source** — How the steering angle is measured, if at all.
- **Angle at full lock** (°) — The actual road-wheel angle at full lock. It also bounds the reading — an angle beyond twice this value is treated as a sensor fault.

Analog settings

- **Angle sensor** — The analog input wired to the steering-angle sensor.
- **Angle sensor (second)** — An optional second steering sensor for redundancy.
- **Sensor at center** (V) — The sensor reading with the wheels pointed straight ahead.
- **Sensor at full left** (V) — The sensor reading at full left lock.
- **Sensor at full right** (V) — The sensor reading at full right lock.

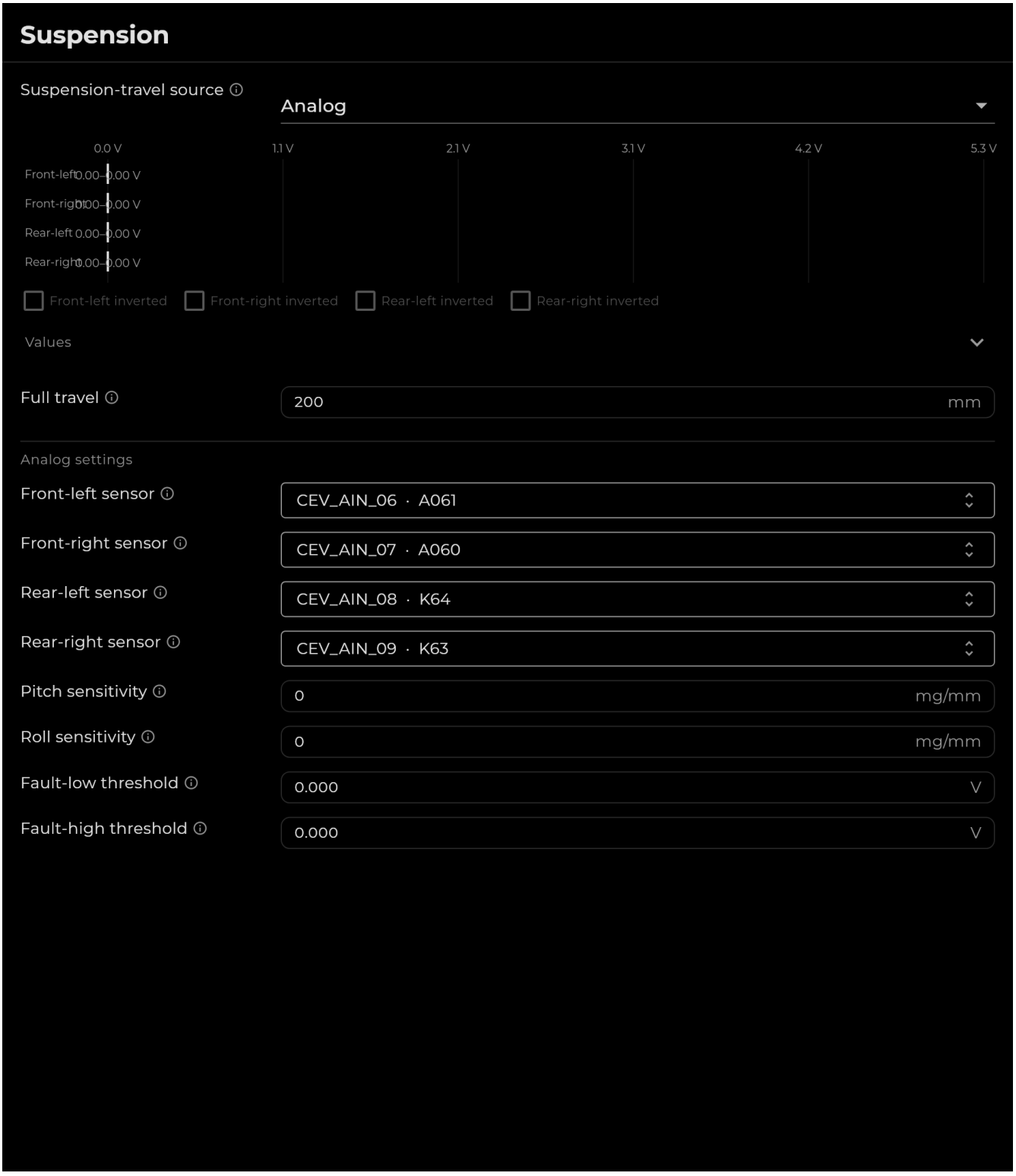
- **Fault-low threshold** (V) — A reading below this is treated as a wiring fault.
- **Fault-high threshold** (V) — A reading above this is treated as a wiring fault.
- **Max sensor disagreement** (V) — The largest allowed difference between two steering sensors before a fault.

CAN settings

- **Steering-angle signal** — The CAN signal carrying the road-wheel steering angle. Define its scale and offset so it decodes to degrees, reading 0 with the wheels straight ahead.

Suspension

Per-corner suspension travel sensing, if fitted.



Suspension travel sensing watches how far each corner of the vehicle is compressed, in millimetres, and turns the four readings into a picture of how the body is leaning. When the body squats at the back under acceleration, or rolls to one side in a corner, that weight transfer is a direct measurement of how hard the vehicle is accelerating forward or sideways. The controller can use that measurement instead of an inertial sensor to shape how torque is split front-to-rear and side-to-side, and to sense grade for the hill features.

This system produces no torque on its own — it is purely a sensor that feeds the dynamics picture. It is entirely optional; if it is not fitted, nothing about the vehicle changes.

What it needs

First, pick how the corners are measured with **Suspension-travel source**. There are three choices:

- *None* — no suspension sensing is fitted. The system is silent and inert, raises no faults, and never feeds the dynamics picture. Leave it here if the vehicle has no ride-height sensors.
- *Analog* — four ride-height sensors wired to analog inputs, each reporting a voltage that rises or falls with travel.
- *CAN* — four travel values delivered over the vehicle network from another module that already measures ride height.

For the **Analog** source, each corner needs its sensor assigned with **Front-left sensor**, **Front-right sensor**, **Rear-left sensor**, and **Rear-right sensor**, plus a two-point calibration per corner: the voltage at design ride height (**Front-left at rest**, **Front-right at rest**, **Rear-left at rest**, **Rear-right at rest**) and the voltage at full compression (**Front-left at full bump**, **Front-right at full bump**, **Rear-left at full bump**, **Rear-right at full bump**). You also set **Full travel (mm)**, the physical stroke in millimetres between rest and full bump. A corner is quietly ignored (no fault) if its sensor is left unassigned, if its *at rest* equals its *at full bump* (no usable span), or if **Full travel (mm)** is zero. If **Full travel (mm)** is zero, or if no corner is usable at all, the vehicle raises a one-time suspension configuration fault.

For the **CAN** source, each corner points at the network value carrying its travel with **Front-left travel signal**, **Front-right travel signal**, **Rear-left travel signal**, and **Rear-right travel signal**. A corner left unset is ignored; if a value goes stale or drops out, that corner is treated as missing. If no corner references a value at all, the vehicle raises the same one-time configuration fault. The CAN source does not use the analog calibration — only the two sensitivity settings below apply to it.

Finally, for any of this to actually affect the vehicle, the dynamics picture must be told to *use* suspension. Under Dynamics → Producers, the forward/back (longitudinal) source and/or the side-to-side (lateral) source must be set to infer from suspension; otherwise the travel is measured but ignored. And the inferred acceleration only changes behavior if the front/rear and left/right torque-split curves are authored and the hill features are turned on.

How it works

Travel is read fresh on every control cycle, just before torque is decided, so there is no lag. Positive millimetres always mean compression (the corner loaded, the body moving down onto the wheel); negative means the corner extending.

On the **Analog** source, each corner is turned into millimetres by a straight-line scale between your two calibration points:

```
corner travel (mm) = (sensor voltage - at rest) / (at full bump - at rest) x Full travel
(mm)
```

So a corner sitting exactly at design ride height reads 0 mm, and a corner at full compression reads the full **Full travel (mm)** value. If you set *at full bump* to a lower voltage than *at rest*, that is allowed — it simply flips which voltage direction means compression, so make sure your two points stay consistent with "more compression = more positive" or the lean directions will read backwards.

Each analog corner can also be range-checked against **Fault-low threshold (V)** and **Fault-high threshold (V)**. The check is armed if either threshold is non-zero:

```
corner is trusted only if  Fault-low threshold (V) <= sensor voltage <= Fault-high threshold
(V)
```

A reading below the low threshold or above the high threshold is treated as an open or shorted sensor: that corner is marked untrustworthy and a single shared suspension range fault is raised. The fault is one band shared across all four corners — it sets when any corner first goes out of range and clears only once every corner is back in range. Setting both thresholds to zero disarms the check entirely (no plausibility gating).

On the **CAN** source, each corner's travel arrives already in millimetres from the network, with the same positive-means-compression convention, and the network value's freshness is watched automatically — a value that stops updating makes that corner missing.

Once all four corners are in millimetres, the lean is computed. Forward/back lean (pitch) compares the rear pair to the front pair:

```
rear average  = (Rear-left  + Rear-right) / 2
front average = (Front-left + Front-right) / 2
forward/back acceleration = (rear average - front average) x Pitch sensitivity (mg/mm)
```

Under forward acceleration the rear squats, so the rear average exceeds the front average and the result is positive (forward). **Pitch sensitivity (mg/mm)** is the gain that converts millimetres of front-to-rear difference into acceleration; its output is in thousandths of a g (1000 = 1 g).

Side-to-side lean (roll) compares the left pair to the right pair:

```
left average  = (Front-left  + Rear-left)  / 2
right average = (Front-right + Rear-right) / 2
sideways acceleration = (left average - right average) x Roll sensitivity (mg/mm)
```

In a right-hand turn weight transfers to the left, so the left average exceeds the right average and the result is positive. **Roll sensitivity (mg/mm)** is the gain for this conversion, again in thousandths of a g. Both sensitivity settings apply to both the Analog and CAN sources.

Both inferences require **all four corners valid in the same cycle**. If even one corner is unassigned, out of range, or stale, the relevant lean value goes invalid for that cycle — partial coverage gives nothing.

When suspension is the chosen source, the forward/back acceleration does three things: it sets the front-to-rear torque split (via the front/rear curve), and it acts as the grade signal that enables hill-hold and hill-descent near a standstill. The sideways acceleration sets the left/right torque bias (via the left/right curve), which adds on top of any yaw-based torque vectoring without changing an axle's total.

The important safety property: an invalid lean value never adds torque. If any corner drops out, the front/rear split falls back to an even 50/50, the left/right split falls back to even, and hill-hold and hill-descent stand down. A suspension fault can only remove torque-shaping authority — it can never raise torque.

Tuning notes

Set the **Analog** calibration with the vehicle at its normal design ride height: read each corner and enter that voltage as the *at rest* value, then cycle each corner to full compression and enter the *at full bump* voltage. Enter the real physical stroke as **Full travel (mm)**. Confirm each corner reads near 0 mm at rest and climbs positive as you push the corner down — if a corner reads negative under compression, swap that corner's *at rest* and *at full bump*, or check the wiring.

Tune **Pitch sensitivity (mg/mm)** and **Roll sensitivity (mg/mm)** against a known reference: drive a steady acceleration or a steady corner and adjust the gain until the inferred acceleration matches reality. Too low and the front/rear and left/right splits barely move; too high and they over-react to small body motions. Because these gains feed the split curves and the hill grade signal, get them honest before tuning those curves.

Use **Fault-low threshold (V)** and **Fault-high threshold (V)** to bracket the real working voltage range of your sensors with a small margin, so a disconnected or shorted sensor is caught. Set them just outside the lowest and highest voltages you see across full travel. If you do not want plausibility gating during early bring-up, leave both at zero — but a fielded vehicle should have them set, because a bad sensor that is not caught will quietly collapse the lean values and drop the vehicle back to even splits with the hill features disabled.

Settings

- **Suspension-travel source** — How per-corner suspension travel is measured, if at all.
- **Full travel** (mm) — The suspension travel between rest and full compression. It also bounds the reading — travel beyond twice this value is treated as a sensor fault.

Analog settings

- **Front-left sensor** — The analog input wired to the front-left ride-height sensor.
- **Front-right sensor** — The analog input wired to the front-right ride-height sensor.
- **Rear-left sensor** — The analog input wired to the rear-left ride-height sensor.
- **Rear-right sensor** — The analog input wired to the rear-right ride-height sensor.
- **Front-left at rest** — The front-left sensor reading at design ride height.
- **Front-right at rest** — The front-right sensor reading at design ride height.
- **Rear-left at rest** — The rear-left sensor reading at design ride height.
- **Rear-right at rest** — The rear-right sensor reading at design ride height.
- **Front-left at full bump** — The front-left sensor reading at full compression.
- **Front-right at full bump** — The front-right sensor reading at full compression.
- **Rear-left at full bump** — The rear-left sensor reading at full compression.
- **Rear-right at full bump** — The rear-right sensor reading at full compression.

- **Pitch sensitivity** (mg/mm) — How much front-to-rear travel difference corresponds to acceleration. Vehicle-specific.
- **Roll sensitivity** (mg/mm) — How much side-to-side travel difference corresponds to cornering force. Vehicle-specific.
- **Fault-low threshold** (V) — A reading below this on any corner is treated as a wiring fault.
- **Fault-high threshold** (V) — A reading above this on any corner is treated as a wiring fault.

CAN settings

- **Front-left travel signal** — The CAN signal carrying front-left suspension travel. Define its scale and offset so it decodes to millimetres, reading 0 at design ride height.
- **Front-right travel signal** — The CAN signal carrying front-right suspension travel. Define its scale and offset so it decodes to millimetres, reading 0 at design ride height.
- **Rear-left travel signal** — The CAN signal carrying rear-left suspension travel. Define its scale and offset so it decodes to millimetres, reading 0 at design ride height.
- **Rear-right travel signal** — The CAN signal carrying rear-right suspension travel. Define its scale and offset so it decodes to millimetres, reading 0 at design ride height.

Producers

Which source provides each piece of motion data. Point each one at a sensor you've configured above.



Producers is where you tell the vehicle *where each piece of motion data comes from*. The controller keeps a single live picture of how the vehicle is moving — how hard it is accelerating forward, how hard it is being pushed sideways, how fast it is turning, where the steering is pointed, how fast each of the four wheels is spinning, and — when a six-axis motion sensor is fitted — how hard it is accelerating up and down and how fast it is rolling and pitching. Producers does not measure any of that itself; it just decides, for each

quantity, which configured source fills it in. For every quantity you pick a source from a short menu, and each source is one of two kinds: a **direct read** from a real sensor, or a **derived estimate** the controller computes from other motion data it already has plus the vehicle geometry you've entered.

Whatever you choose here is what the comfort and dynamics features actually "see": traction control, torque vectoring and front/rear torque distribution, hill assist, gear logic, scripting, and the live dashboard all read this one shared picture. The safety monitor deliberately does **not** read it — a comfort-grade motion source that drops out can cost you a feature, but it can never raise a torque limit or override a safety decision.

What it needs

Each quantity is only filled in if its chosen source actually succeeds on that control cycle. If it doesn't, that quantity is marked *unknown* — never zero — and any feature that depends on it cleanly stands down rather than acting on a wrong value.

A **direct** source just needs its sensor present and reporting a good reading that cycle:

- **Forward acceleration from**, **Sideways acceleration from**, and **Turn rate from** can each be set to the motion sensor (the IMU), which reports forward g, sideways g, and turn rate in deg/s directly. **Vertical acceleration from**, **Roll rate from**, and **Pitch rate from** read the matching channels of that same motion sensor when the fitted sensor provides them: the five-axis Bosch MM5.10 and its ruggedized MM5.10-R variant report forward and sideways acceleration plus roll and turn rate, but not vertical acceleration or pitch; the six-axis Bosch MM7.10 (or a generic CAN sensor wired for all six channels) reports every axis. The vehicle uses whatever axes the chosen sensor provides, and any axis the sensor cannot measure is simply left unknown.
- **Steering angle from** has only two choices: the steering-angle sensor, or nothing. It reports steering in degrees, positive to the right and negative to the left.
- **Wheel speed from** is set per corner (front-left, front-right, rear-left, rear-right). Each corner can read the wheel-speed sensor for that corner, or be left off.

A **derived** source needs no dedicated sensor for that quantity, but it does need the geometry and the upstream data it builds on. The relevant vehicle numbers — **vehicle mass**, **front and rear wheel radius**, **wheelbase**, **track width**, and the suspension gains — must be entered, or the estimate that depends on them produces nothing. Out of the box none of this is configured: every Producer defaults to *off* and every geometry value defaults to zero, so a brand-new vehicle produces no motion data at all until you pick sources and fill in the geometry.

How it works

Once per control cycle the controller rebuilds the motion picture from scratch, running the sources in a fixed order so that a derived estimate can use a value that was filled in earlier the same cycle. The order is:

```
1. wheel speeds (all four corners)
2. steering angle
3. forward acceleration
4. turn rate
5. sideways acceleration
```

This order is the reason only certain estimate chains work. An estimate can only build on a quantity that comes *before* it in this list. Wheel speeds and steering are produced first, so anything may build on them; turn rate is produced before sideways acceleration, so sideways acceleration may build on turn rate — but not the other way around. If you point a derived source at a quantity that is produced later in the list, that quantity isn't ready yet and the estimate silently produces nothing.

Several estimates need a single road speed. The controller gets it by averaging only the wheel-speed corners that are working that cycle; if no corner is working, road speed is treated as zero and the estimate is skipped rather than run on a false speed.

Forward acceleration (**Forward acceleration from**) can be:

- the motion sensor's forward channel, read straight through, or
- estimated from drive torque — from how much wheel torque was commanded on the previous cycle (so it carries a tiny lag of about a hundredth of a second and can briefly misread right at a standstill or just after a sharp pedal change):

```
forward acceleration (g) = wheel torque (Nm) / wheel radius (m) / vehicle mass (kg) / 9.81
```

This needs **vehicle mass** above zero and at least one **wheel radius** entered; the radius used is the front/rear average when both are set, otherwise whichever one is set.

- estimated from suspension squat — the vehicle pitches under acceleration and braking, so the difference between rear and front ride height tracks forward g:

```
forward acceleration = (average rear travel - average front travel) x forward-from-
suspension strength
```

This needs all four suspension corners reporting and the forward-from-suspension strength entered.

Sideways acceleration (**Sideways acceleration from**) can be:

- the motion sensor's sideways channel, read straight through, or
- estimated from steering and speed (a simple cornering model):

```
sideways acceleration (g) = road speed (m/s)^2 x steering angle (deg) / wheelbase (m) /
57.3 / 9.81
```

- estimated from turn rate and speed:

```
sideways acceleration (g) = turn rate (deg/s) / 57.3 x road speed (m/s) / 9.81
```

(Because turn rate is produced before sideways acceleration, an *estimated* turn rate still feeds this on the same cycle.)

- estimated from suspension roll — in a turn the vehicle leans, so the left-to-right ride-height difference tracks sideways g:

```
sideways acceleration = (average left travel - average right travel) x sideways-from-
suspension strength
```

(In a right-hand turn the body rolls onto the left wheels, compressing them more.) This needs all four suspension corners reporting.

The two steering-and-speed estimates above need the steering angle valid, at least one working wheel for road speed, and **wheelbase** entered.

Turn rate (**Turn rate from**) can be:

- the motion sensor's turn-rate channel, read straight through, or
- estimated from the front wheel-speed difference (the outer wheel runs faster than the inner one in a turn):

```
turn rate (deg/s) = (right-front speed (m/s) - left-front speed (m/s)) / track width (m) x
57.3
```

This needs both front corners working and **track width** entered.

- estimated from steering and speed:

```
turn rate (deg/s) = road speed (m/s) x steering angle (deg) / wheelbase (m)
```

This needs the steering angle valid, at least one working wheel, and **wheelbase** entered.

Steering angle (**Steering angle from**) is either the steering sensor or nothing — there is no derived option. Because steering feeds the steering-based estimates of both sideways acceleration and turn rate, leaving **Steering angle from** set to nothing also disables those estimates.

Wheel speed (**Wheel speed from**) is chosen independently for each corner. Each corner is either the wheel-speed sensor for that corner, or off. There is also an inverter-derived option listed, but it is reserved and not yet implemented — selecting it behaves exactly like off and leaves that corner unknown. Corner wheel speeds feed traction control's slip reference and the road-speed average that nearly every other estimate relies on, so the wheel-speed corners are worth getting right first.

All acceleration estimates are limited to a sensible range, so an extreme input is capped rather than allowed to roll over into a wrong-signed value.

Tuning notes

Configure from the bottom up, in the order the controller uses. Get the wheel-speed corners working first — they underpin road speed and most of the estimates. Then steering. Only then the acceleration and turn-rate quantities. If a feature that depends on motion data isn't engaging, the most common cause is a derived source missing one of its inputs: a geometry value still at zero (mass, wheel radius, wheelbase, or track width), a sensor not reporting, or an estimate pointed at a quantity that is produced later in the cycle than itself.

Prefer a real sensor where you have one — the motion sensor for forward/sideways/turn, and the wheel-speed and steering sensors for their quantities — and reserve the derived estimates for filling gaps where a

sensor isn't fitted. The suspension-based estimates are only as good as their configured strengths; set those by comparing the estimate against a known-good reading (a steady cornering or braking manoeuvre) and adjusting the strength until they agree. Remember the torque-based forward estimate carries a small lag and can twitch at a standstill, so it's a coarse stand-in, not a substitute for a real forward-g sensor. Whatever you pick, a source that goes unavailable simply switches off the feature that needs it — it never feeds a stale or zero value into the controls.

Settings

- **Forward acceleration from** — Which source provides forward/backward acceleration.
- **Sideways acceleration from** — Which source provides side-to-side acceleration.
- **Turn rate from** — Which source provides how fast the vehicle is turning.
- **Steering angle from** — Which source provides the steering angle.
- **Vertical acceleration from** — Which source provides vertical (up/down) acceleration.
- **Roll rate from** — Which source provides how fast the vehicle is rolling (leaning side to side).
- **Pitch rate from** — Which source provides how fast the vehicle is pitching (nose up/down).
- **Wheel speed from** — Which source provides each wheel's speed.

Tools

Cross-cutting authoring tools: the calibration curves and tables referenced throughout the configuration, and the installer scripts that add custom behavior.

Recipes

No-code automations: each recipe is a set of if/then rules — a trigger over the vehicle's inputs and readings, and the actions taken while it holds or the moment it fires. Recipes run on the vehicle alongside any hand-written scripts. The copy icon on a row starts a new recipe from a copy of that one, named 'Name - Copy'.

Diagnostics

Live diagnostic views read from a connected vehicle.

Worked Examples

Complete vehicles, followed through the journeys they make — switching on, driving, charging, and failing safely. Each example names only the settings that cause its sequence; everything else is configured as described in Part 2.

How the vehicle changes state

The vehicle is always in exactly one **state** — `OFF`, `READY`, `DRIVE`, `CHARGING_AC` and the rest. The state decides what the vehicle will do: the pedal produces torque only in `DRIVE`, the high-voltage bus is up only in the live states, and every fitted converter and charger checks the state before it runs. The examples that follow walk the common journeys through it.

Three different things move the vehicle between states, and telling them apart is most of what this chapter teaches.

A recipe asks. A rule in a recipe performs the *Request vehicle state* action. This is how a vehicle responds to its key switch, its start button, its dash controls — and it is the only way the vehicle ever reaches `DRIVE`. A request is an *ask*, not a command: it can be rerouted, delayed, or refused.

A built-in system asks. Some requests come from the vehicle itself, with no recipe involved and nothing for you to configure. A charger that sees a cord asks for a charging state. Anything that detects a serious fault asks for `FAULT_SAFE`. The high-voltage controller asks for `READY` once it has the bus up. You cannot write a rule for these and you do not need to.

The vehicle also asks for `READY` itself, about a second after it powers up with the ignition. It is why a project needs no rule to go live. If a rule must decide the moment instead, turn on **Precharge only when asked** under Precharge & sequence.

When Bosch enters **PostDrive** after ignition is removed, the vehicle starts its controlled shutdown without a recipe. Monitor → Drive shows the Bosch controller state as **Run State**, separately from the vehicle state.

The vehicle moves on its own. A few changes have no requester at all. Once the high-voltage bus is fully down, `SHUTTING_DOWN` settles to whichever parked state was asked for. Once a charging session has been safely unwound, `CHARGE_STOPPING` moves on.

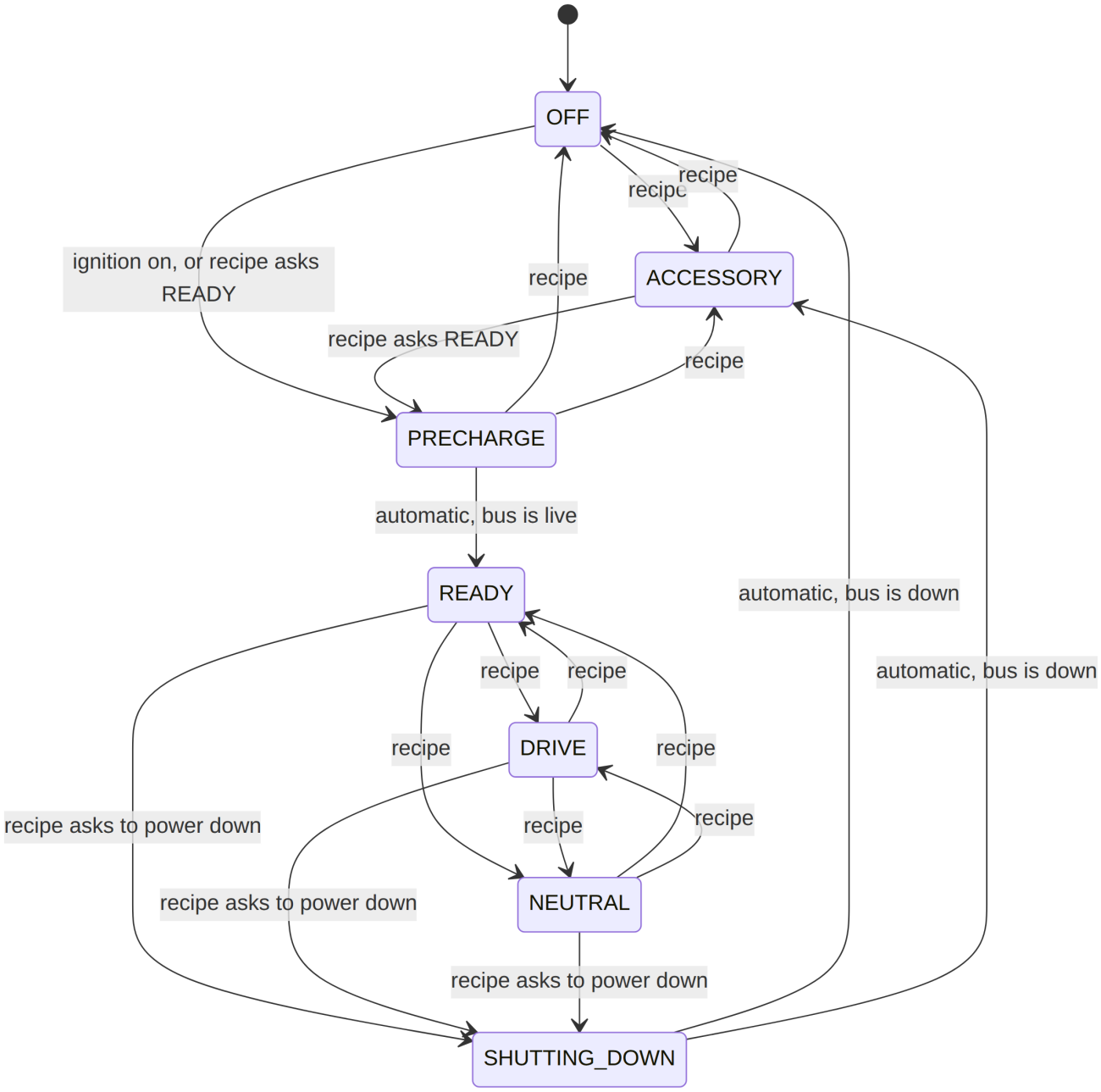
Being *allowed* in a state is not the same as being *started*

Every commanded high-voltage peer — a DC-DC converter, an on-board charger, a charge inlet controller, a generator set — carries a **Can be enabled in** setting: the list of states it is permitted to run in. This is permission, and only permission. Ticking a state never starts anything, and clearing one does not command a stop — it withdraws the permission.

So a DC-DC converter with `READY` ticked and its **Enable source** set to *Always on* comes up the moment the vehicle reaches `READY`, with no rule and no trigger anywhere in the project. That is not the recipe working; that is the converter running everywhere it is allowed to.

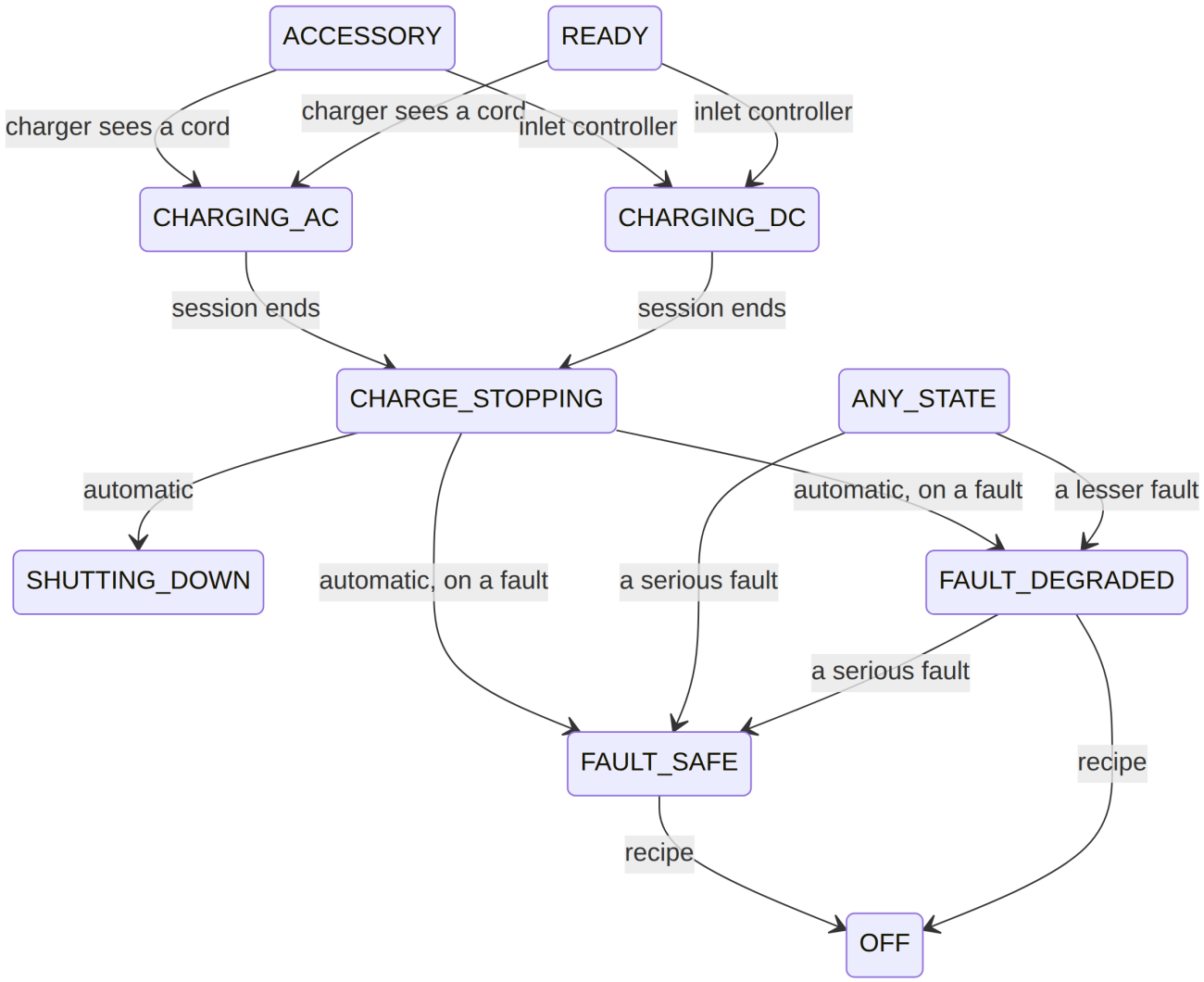
Driving

Switching on, driving, and switching off. This is the loop a vehicle spends almost all of its life in.



Charging, and going wrong

Charging is entered from a parked state and always leaves through `CHARGE_STOPPING`. The fault states are reached from anywhere at all.



`ANY_STATE` is not a state the vehicle can be in — it stands for *wherever the vehicle happens to be*. A serious fault is honoured from anywhere, including at speed, which is why those two arrows are drawn from a placeholder rather than from twelve separate boxes. The one exception is a live charging session, which always unwinds through `CHARGE_STOPPING` first so the charging contactors open in the right order — even when a fault is what ended it.

Key-on to rolling

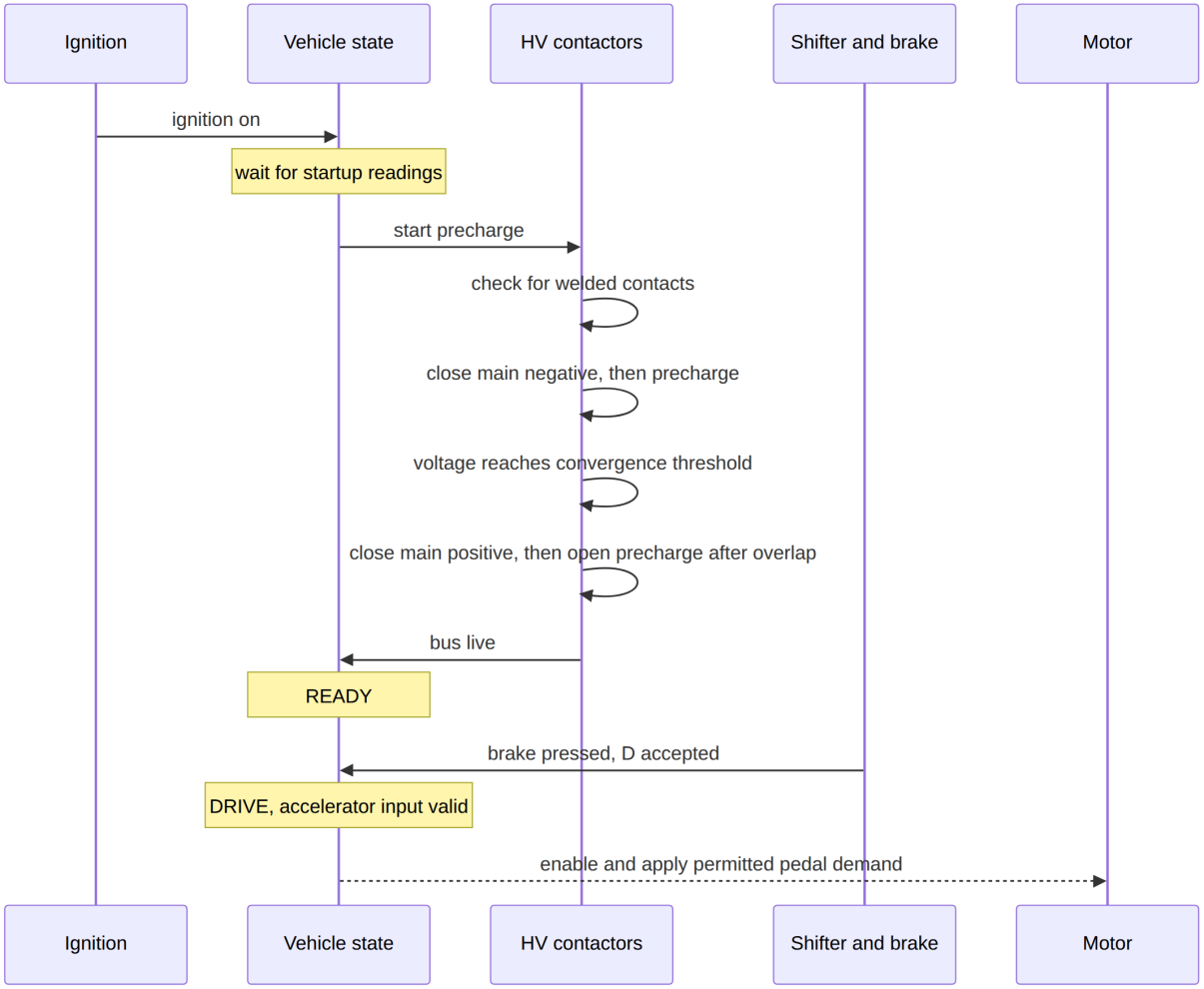
The vehicle. A single rear motor, one battery pack with a BMS, a DC-DC converter, a Digital P/R/N/D selector and a brake switch.

The settings that cause this.

Where	Setting
HV Power → Precharge	Automatic startup precharge enabled
HV Power → Precharge	**Convergence threshold** — the percentage of pack voltage that ends precharge

Where	Setting
Direction (P/R/N/D)	Digital selector, with the position inputs assigned
Direction (P/R/N/D)	Brake required for direction changes

What happens.



No script is needed. Digital, Analog and CAN selectors request Drive after the direction interlocks accept D or R. None (forward-only) also enters Drive automatically once Ready, with a valid accelerator input. A selection accepted during precharge takes effect after Ready without another shift.

Ready confirms that precharge completed. If precharge fails, the vehicle enters a fault state. Selecting D cannot bypass it. The independent torque checks remain active after Drive entry.

Script is an explicit choice. With the Script direction source, the script must request both the direction and the vehicle state. Automatic Drive entry does not apply. If precharge is configured to start on request, a recipe or script must also request Ready.

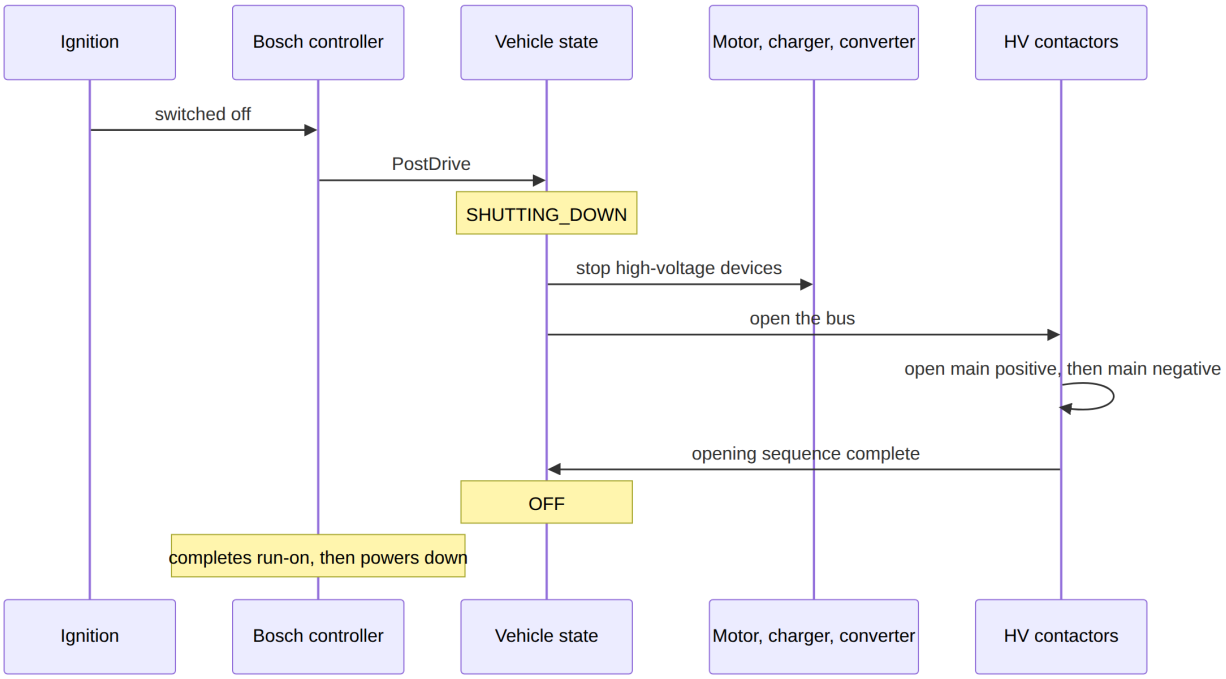
The DC-DC follows its configuration. With an Always on enable source and Ready and Drive in its permitted state list, it supplies the 12 volt system without a recipe.

Key-off and shutting down

The vehicle. The same vehicle as *Key-on to rolling*, now parked and switched off.

The settings that cause this. The ignition switch supplies the VCU ignition input. No recipe is required for this shutdown. The VCU reads the Bosch controller state and starts shutdown when that state becomes **PostDrive**. The Drive monitor shows **Run State** below the vehicle state.

What happens.



The same shutdown path applies from `READY`, `DRIVE`, and `NEUTRAL`. A charging session first completes its controlled stop. The vehicle stays in `SHUTTING_DOWN` until the contactor opening sequence is complete. The VCU can remain powered for several seconds after the contactors open.

Bosch can report **PreDrive** again during run-on. That change does not restart the vehicle. After shutdown, an explicit start request is accepted only when the high-voltage bus is down and Bosch is back in **Drive**. Automatic startup requires a new boot.

Separate accessory and stop controls

A recipe can still request `OFF` or `ACCESSORY` for a separate control. When the bus is live, the vehicle enters `SHUTTING_DOWN`, stops the high-voltage devices, completes contactor opening, and then reaches the requested state. Use `ACCESSORY` when the radio, lights, and dashboard must remain available. Bosch PostDrive takes priority over these requests and ends in `OFF`.

Ignition shutdown and recipe requests have different limits

A power-down request from a recipe or script is rejected above about 2 km/h, or when road speed cannot be confirmed. The refusal is reported as an error. The direct Bosch shutdown path is independent of that

recipe restriction: **removing ignition starts shutdown even while the vehicle is moving.** Stop the vehicle before switching it off.

To coast without torque while the ignition remains on, request `NEUTRAL` — see *Coasting in neutral*.

Accessory power without high voltage

The vehicle. A conversion that keeps the donor's original three-position key switch: OFF, ACC, RUN. In ACC the customer expects the radio, the windows and the dash to work with the high-voltage bus still down.

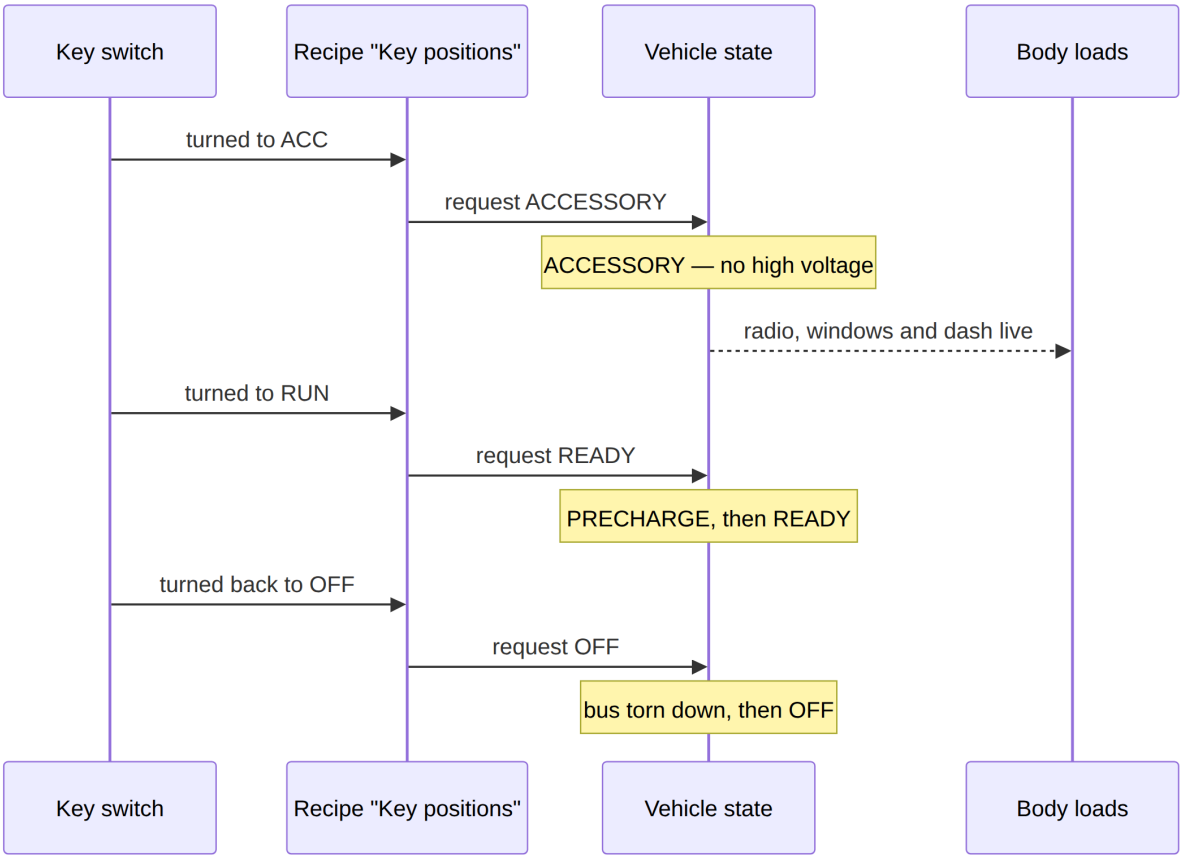
The settings that cause this.

Where	Setting
Devices	A **Digital Input** named `key_acc`, bound to the pin wired to the ACC position
Devices	A **Digital Input** named `key_run`, bound to the RUN position
Tools → Recipes	A recipe named **Key positions**, with the rules below

The recipe.

- **Accessory.** While `key_acc` is on **and** `key_run` is off, request vehicle state `ACCESSORY`.
- **Run.** When `key_run` turns on — rising edge — request vehicle state `READY`.
- **Off.** While `key_acc` is off **and** `key_run` is off, request vehicle state `OFF`.

What happens.



`ACCESSORY` is a genuinely parked state. The high-voltage bus is down and no torque is possible, but the vehicle is awake: devices read their inputs, recipes run, and the diagnostic connection works. It is the right state for anything that should work with the key in ACC.

Precharge is interruptible. If the driver turns the key straight back to OFF or ACC while the bus is still coming up, the request is honoured — precharge abandons and the vehicle parks. You do not have to guard your rules against someone being impatient with the key.

A charger may still act from here. `ACCESSORY` is one of the states a charging session can begin in — plugging in while the key is in ACC can move the vehicle to a charging state with no rule involved. See *Charging from a wall socket*.

An explicit Off or Accessory request cancels automatic precharge until the next controller boot. This includes a request during the first second, even if the vehicle is already Off. The Ready recipe can still start precharge later.

Coasting in neutral

The vehicle. The Digital-selector vehicle from *Key-on to rolling*.

Select Neutral. Moving the selector to N removes torque permission and moves the vehicle from Drive to Neutral. The high-voltage bus stays live. Park or an invalid direction also removes drive permission. A refused shift keeps the previously accepted direction.

Return to Drive. Press the brake and select D or R. Once the direction interlocks accept the selection, the vehicle enters Drive automatically. The same behavior applies to Analog and CAN selectors. None (forward-only) returns to Drive automatically when Ready or Neutral and the accelerator input is valid.

Script-controlled vehicles. To control these states explicitly, select the Script direction source. The script supplies direction requests and vehicle-state requests separately. A recipe can request Neutral from Ready or Drive, and Ready from Neutral or Drive. It can request Drive from Ready or Neutral when the installation's entry conditions are met. Script-controlled vehicles do not automatically return to Drive while D remains selected.

Neutral keeps the bus up. Ready, Drive and Neutral are all live states. Moving between them does not open the HV contactors. Ready is the state after bring-up; Neutral expresses a request for no drive torque.

Charging from a wall socket

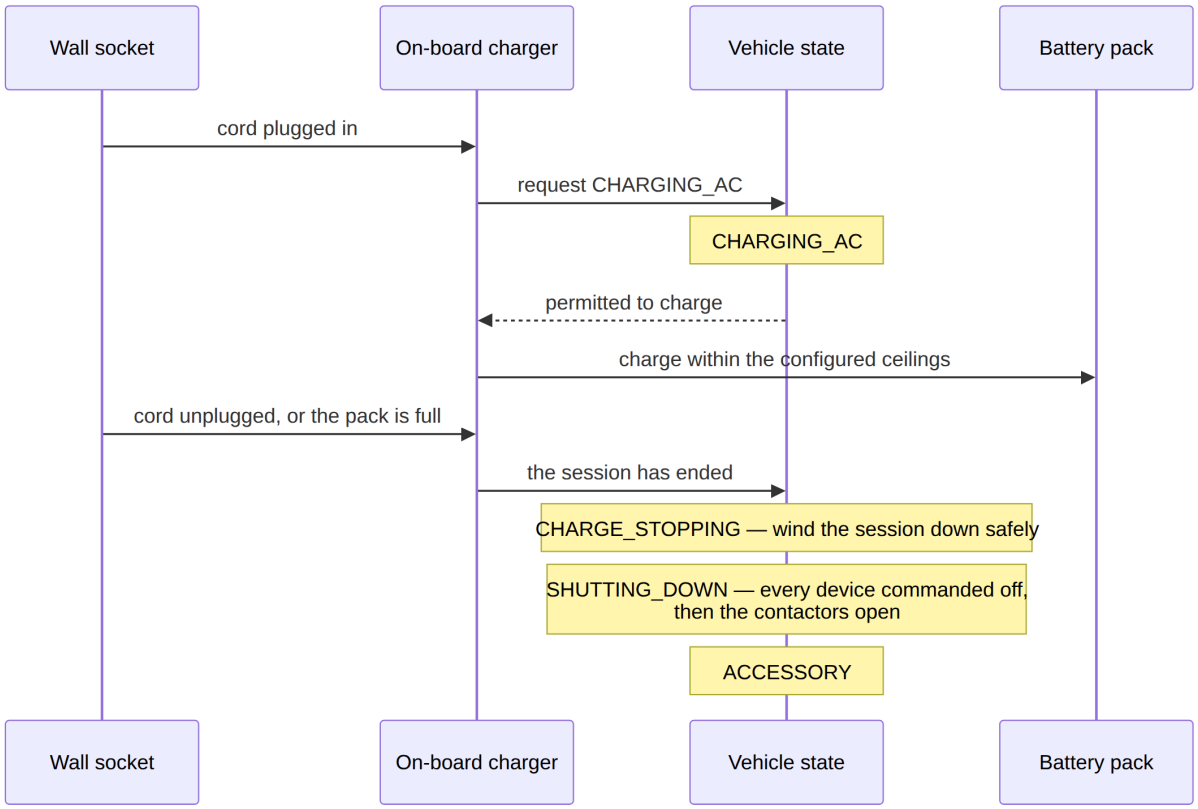
The vehicle. The same conversion, with an on-board charger fitted so it can be plugged into a wall socket overnight. The key is out of the ignition.

The settings that cause this.

Where	Setting
Charging & Power → On-board charger	**On-board charger** — the vendor fitted, and its **CAN bus**
Charging & Power → On-board charger	**Charge voltage ceiling** and **Charge current ceiling**
Charging & Power → On-board charger	**Charge path** — *Direct to battery* for a charger wired straight to the pack, *Through a charge contactor* for one behind its own contactor, *Through the drive bus* for one wired to the main bus
High voltage → Precharge & sequence	**AC-charge contactor** — only for the *Through a charge contactor* path
Tools → Recipes	*Nothing.*

The recipe. There isn't one. This entire journey happens with no rule anywhere in the project.

What happens.



Plugging in is not a trigger. The charger detects the cord itself and asks for the charging state on its own. There is nothing to wire, no device to create and no rule to write. This is the clearest example of the second kind of state request: the vehicle asking on its own behalf.

Where the charger's output lands decides where charging can start. Set **Charge path** to *Direct to battery* and the charger feeds the pack directly, so a session can start from `ACCESSORY` with the drive bus still down — the normal overnight case. *Through a charge contactor* also starts from `ACCESSORY`: one dedicated contactor closes, the main contactors stay open, and the drive bus stays dead. Name that contactor under **High voltage → Precharge & sequence**; without it the charger has no path to the pack at all, and the Configurator refuses the pairing. *Through the drive bus* puts the output on the main bus, which must therefore already be live, so charging starts from `READY` instead. Getting this wrong is the usual reason a charger appears to do nothing when plugged into a parked vehicle.

The charging states are not in the charger's Can be enabled in list, and that is correct. The charger is what asked to go to `CHARGING_AC`; requiring it to also be ticked there would revoke, on the very next moment, the permission the move was made on. Use that list to say where charging may *begin* — typically the parked states.

Charging never stops abruptly. Unplugging does not open the contactors immediately. The session unwinds through `CHARGE_STOPPING` first, then the normal teardown runs. That ordering matters much more on a fast charger — see *Fast charging and safe teardown*.

Fast charging and safe teardown

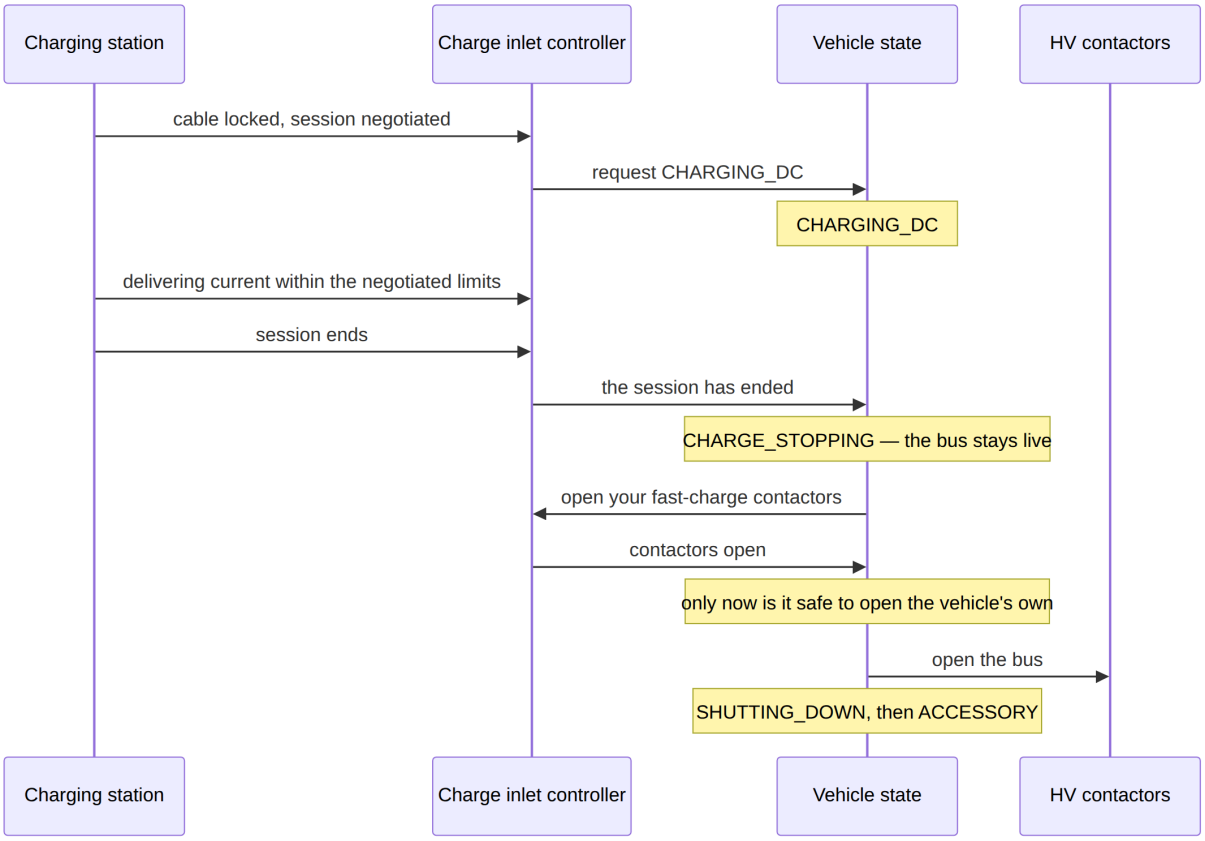
The vehicle. A conversion with a DC fast-charge inlet, driven by a charge inlet controller on its own CAN bus. During a fast-charge session the station — not the vehicle — supplies the current, at a rate the vehicle negotiates.

The settings that cause this.

Where	Setting
Charging & Power → Charge inlet	**Charge inlet controller** — the vendor fitted, and its **CAN bus**
Charging & Power → Charge inlet	**Fast-charge voltage ceiling** and **Fast-charge current ceiling** — your limits, on top of the pack's own
Charging & Power → Charge inlet	**Charging may start in** — the parked states a session may begin from
Charging & Power → Charge inlet	**Needs the drive bus live** — as with the on-board charger, where the inlet's output lands
Charging & Power → Charge inlet	**Shutdown wait** — how long to wait for the controller to open its own contactors
Tools → Recipes	*Nothing.*

The recipe. Again, none.

What happens.



`CHARGE_STOPPING` exists to get the order right. A fast charger is the highest-power path into the pack there is. Opening the vehicle's contactors while the station is still delivering current can weld them shut. So the vehicle enters `CHARGE_STOPPING`, keeps its own contactors *closed*, and asks the inlet controller to open its fast-charge contactors first. Only once that is confirmed does the normal teardown run.

The wait is bounded. If the controller never confirms — it has lost power, or gone quiet — the vehicle waits for **Shutdown wait** and then opens anyway, reporting a fault as it does. Waiting forever would be worse than the risk the ordering protects against. Leave **Shutdown wait** at its default unless a vendor tells you otherwise; setting it to zero does not skip the wait, it selects the default.

Even a fault waits its turn. This is the one situation where a serious fault does *not* go straight to `FAULT_SAFE`. A fault during a fast-charge session still routes through `CHARGE_STOPPING` first, so the contactors still open in the right order, and only then does the vehicle land in `FAULT_SAFE` — or in `FAULT_DEGRADED` for a lesser fault. The delay is bounded by the same **Shutdown wait**, so failing safe is postponed by at most that long, never abandoned.

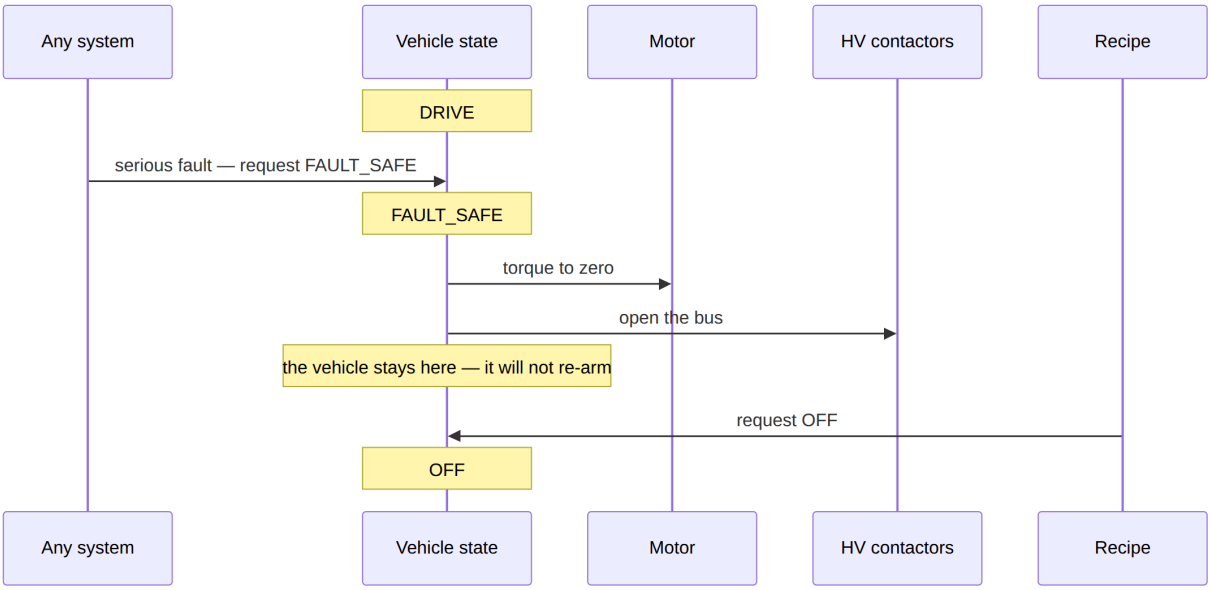
A fast-charge session cannot begin from `DRIVE`. Charging is entered from a parked state only. **Charging may start in** offers only the states where that makes sense.

When something goes wrong

The vehicle. Any of the vehicles above, driving normally, when something serious happens — a contactor reports itself welded, the accelerator pedal's two sensors stop agreeing, an inverter drops off its bus.

The settings that cause this. None. There is nothing to configure and no recipe to write. Every fitted system detects its own faults and asks for a safe state on its own.

What happens.



A fault is honoured from anywhere. Unlike every other move in this chapter, going to a fault state is not restricted by which state the vehicle is in. It happens from `DRIVE` at speed, from `READY` on the drive, from `PRECHARGE` mid-bring-up. The only case that is not immediate is a live charging session, which unwinds through `CHARGE_STOPPING` first so the charging contactors open in the right order — see *Fast charging and safe teardown*.

There are two fault states, and they are not interchangeable. `FAULT_DEGRADED` is the lesser of the two, used for conditions the vehicle can report and trap without treating as an emergency. `FAULT_SAFE` is the real one: torque to zero, bus open, no drive.

A fault never gets downgraded. Once the vehicle is in `FAULT_SAFE`, a later `FAULT_DEGRADED` will not pull it back up — that request is refused. Faults only ever escalate. So `FAULT_DEGRADED` can become `FAULT_SAFE`, never the reverse.

Getting out is a key cycle, not a reset. There is no "clear the fault and carry on" path, by design. From either fault state the vehicle accepts a request to go to `OFF`, and that is the way out — power the vehicle down, correct the cause, and start again. The ignition shutdown described in *Key-off and shutting down* requires no extra recipe: switch off, correct the cause, and start again.

What your recipes should and should not do about faults. A recipe can watch for a fault and react to it — light a lamp, sound a buzzer, send a CAN message to a dash. What a recipe must never be asked to do is *decide* the response to a fault. That decision has already been made by the system that raised it, and the vehicle has already acted on it by the time any rule runs.