

---

# Open Vehicles

Apr 06, 2021



---

## User Guides:

---

<b>1</b>	<b>Open-Vehicle-Monitoring-System-3 (OVMS3)</b>	<b>1</b>
<b>2</b>	<b>User Guide</b>	<b>9</b>
<b>3</b>	<b>Plugins</b>	<b>77</b>
<b>4</b>	<b>BMW i3 / i3s</b>	<b>101</b>
<b>5</b>	<b>DBC Based Vehicles</b>	<b>105</b>
<b>6</b>	<b>DEMO Vehicle</b>	<b>109</b>
<b>7</b>	<b>Fiat 500e</b>	<b>111</b>
<b>8</b>	<b>Hyundai Ioniq vFL</b>	<b>113</b>
<b>9</b>	<b>Kia e-Niro</b>	<b>115</b>
<b>10</b>	<b>Kia Soul EV</b>	<b>117</b>
<b>11</b>	<b>Maxus eDeliver3</b>	<b>123</b>
<b>12</b>	<b>Mercedes-Benz B250E W242</b>	<b>125</b>
<b>13</b>	<b>MG EV</b>	<b>127</b>
<b>14</b>	<b>Mitsubishi Trio</b>	<b>129</b>
<b>15</b>	<b>Nissan Leaf/e-NV200</b>	<b>135</b>
<b>16</b>	<b>OBDII Vehicles</b>	<b>139</b>
<b>17</b>	<b>Renault Twizy</b>	<b>141</b>
<b>18</b>	<b>Renault Zoe</b>	<b>165</b>
<b>19</b>	<b>Smart ED Gen.3</b>	<b>167</b>
<b>20</b>	<b>Smart ED/EQ Gen.4 (453)</b>	<b>171</b>

<b>21 Tesla Model 3</b>	<b>173</b>
<b>22 Tesla Model S</b>	<b>175</b>
<b>23 Tesla Roadster</b>	<b>177</b>
<b>24 Tracking Vehicles</b>	<b>183</b>
<b>25 VW e-Up</b>	<b>185</b>
<b>26 VW e-Up via OBD2</b>	<b>199</b>
<b>27 VW e-Up via Comfort CAN (T26A)</b>	<b>209</b>
<b>28 Command Line Interpreter</b>	<b>217</b>
<b>29 CAN Bus Data Logging</b>	<b>223</b>
<b>30 CRTD CAN Log Format</b>	<b>227</b>
<b>31 Web Framework &amp; Plugins</b>	<b>231</b>
<b>32 Scripting</b>	<b>311</b>
<b>33 CANopen</b>	<b>321</b>
<b>34 OVMS Server</b>	<b>331</b>
<b>35 OVMS Protocol v2</b>	<b>337</b>
<b>36 OVMS HTTP API</b>	<b>357</b>
<b>Index</b>	<b>367</b>

## Open-Vehicle-Monitoring-System-3 (OVMS3)



### 1.1 Introduction

The OVMS is an **all open source** vehicle remote monitoring, diagnosis and control system.

The system provides **live monitoring** of vehicle metrics like state of charge, temperatures, tyre pressures and diagnostic fault conditions. It will **alert** you about critical conditions and events like a charge abort, battery cell failure or potential theft. Depending on the vehicle integration it allows you to **take control** over the charge process, climate control, tuning parameters of the engine and more. OVMS developers are enthusiasts trying hard to provide as **detailed information** about the internals of a vehicle as possible.

While most new vehicles now include some kind of remote control system, very few allow deep insight, some will not work in all regions and none will give you **control over your personal data**. The OVMS fills that gap, and also enables you to add all these features to existing vehicles of any kind.

The OVMS can also be used for **fleet monitoring**. It allows a fleet manager to not only track the vehicle locations but also to monitor the vehicle's vitals, remotely check for fault conditions, offer services like automatic preheating for users and take active control in case of abusive use. As the system is open source and fully scriptable, it can easily integrate custom access and control systems.

For **developers and technicians**, the OVMS includes a range of CAN tools including multiple logging formats, a configurable OBD2 translator, a DBC decoder, a reverse engineering toolkit and a CANopen client. The module provides SSH access and WebSocket streaming and can stream and inject CAN frames via TCP. Both the module and the web frontend can be customized by plugins. The module has three builtin CAN buses and [can be extended by a fourth one](#).

The **OVMS base component** is a small and inexpensive hardware module that connects to the vehicle **OBD2** port. The standard kit includes a 3G modem to provide **GSM** connectivity and **GPS** and comes with a ready-to-use [Hologram.io](#) SIM card. The US kit has been **FCC certified**, the EU kit **CE certified**.

The module provides a **built-in Web App** user interface and remote control via native cellphone Apps available for **Android** and **iOS**. It integrates into home/process automation systems via **MQTT** and provides data logging to SD card and to a server.

## 1.2 Vehicle Support

- *Native Integration*
  - Chevrolet Volt / Opel Ampera
  - BMW i3 / i3s
  - Fiat 500e
  - Hyundai Ioniq vFL
  - Kia e-Niro / Hyundai Kona
  - Kia Soul EV
  - Maxus eDeliver 3
  - Mercedes-Benz B250E
  - MG ZS EV
  - Mitsubishi Trio (i-MiEV et al)
  - Nissan Leaf / e-NV200
  - Renault Twizy
  - Renault Zoe
  - Smart ED Gen.3
  - Smart ED/EQ Gen.4 (453)

- Tesla Model S
- Tesla Roadster
- Think City
- VW e-Up
- *General Support*
  - DBC File Based
  - GPS Tracking
  - OBD-II Standard
  - Zeva BMS

## 1.3 Links

- *User Resources*
  - User and Developer Guides
  - User Support Forum
  - Android App
  - iOS App
- *Distributors*
  - FastTech (global)
  - OpenEnergyMonitor (UK/Europe)
- *Servers*
  - Asia-Pacific
  - Germany/Europe
- *Developers*
  - Developer Guide
  - Developer Mailing List & Archive
  - Server Source
  - Android App Source
  - iOS App Source

## 1.4 Hardware



- **Module Schematics and PCB Layouts**

- **Base Module**

- Black injection-moulded plastic enclosure, approximately 99x73x29 mm excl. plugs
- ESP32 WROVER processor (16MB flash, 4MB SPI RAM, 520KB SRAM, dual core 160/240MHz Xtensa LX6 processor)
- WIFI 802.11 b/g/n
- Bluetooth v4.2 BR/EDR and BLE
- 3x CAN buses
- 1x Micro USB connector (for flash download and serial console)
- 1x Micro SD card slot
- 1x Internal expansion slot
- 8x EGPIO, 2x GPIO
- 1x GSM antenna connector
- 1x GPS antenna connector
- 1x DB9 vehicle connector
- 1x DB26 expansion connector

- **Modem Module**

- US edition is SIM5360A (Dual-Band UMTS/HSPA+ 850/1900MHz, Quad-Band GSM/GPRS/EDGE 850/900/1800/1900MHz)
- EU edition is SIM5360J(E) (Dual-Band UMTS/HSPA+ 900/2100MHz, Quad-Band GSM/GPRS/EDGE 850/900/1800/1900MHz)
- 3G (EV-DO/HSPA+) dual band modem
- Includes 2G (GSM/GPRS) and 2.5G (EDGE) quad band
- GPS/GNSS



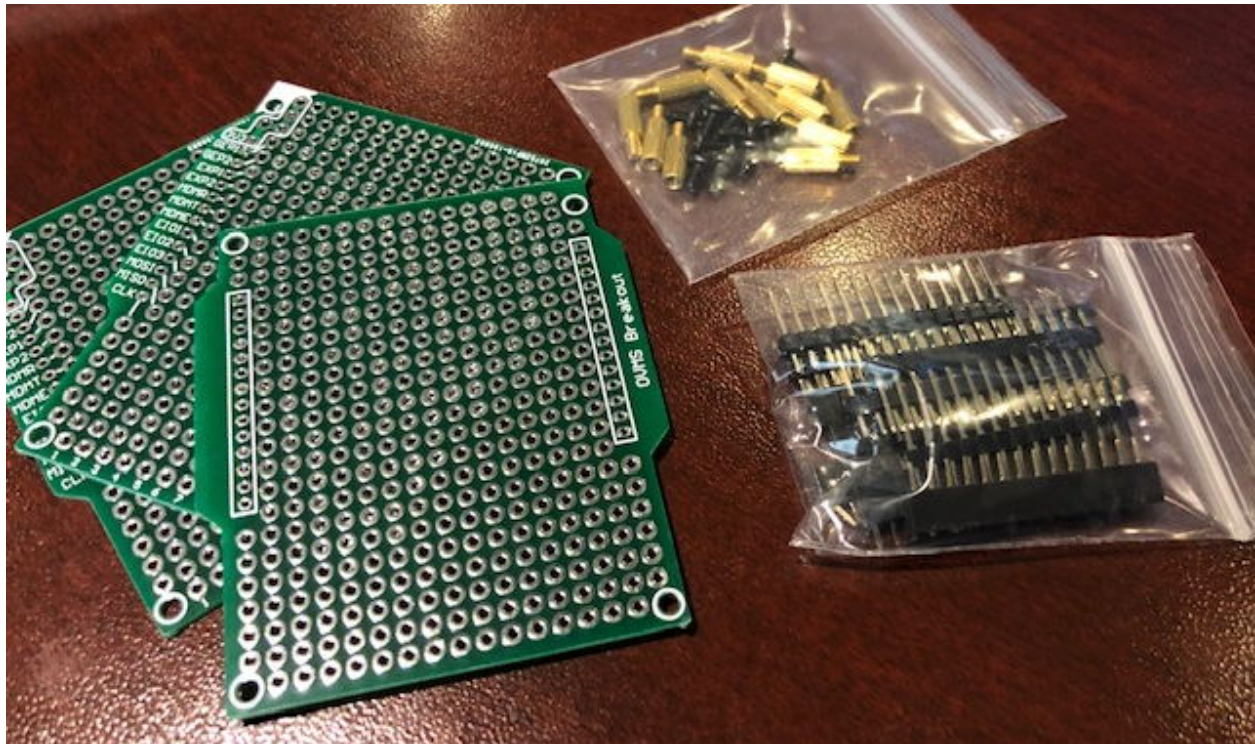
- Nano (4FF) SIM slot
- HOLOGRAM.IO nano sim included (can be exchanged if necessary)

### 1.4.1 Extensions

The external DB26 DIAG connector provides access to the three CAN buses and offers some free extension ports. The internal PCB expansion connector allows stacked additions of further modules and serves for routing GPIO ports to the external DIAG connector. See schematics for details.

A very nice first extension module has been developed by Marko Juhanne: [OVMS-SWCAN](#)

If you plan on developing a hardware extension or just want to do some custom adaptations, have a look at our prototyping PCB kit. It's available in packs of 3 PCBs and includes headers and mounting material:



If the kit isn't available at the distributors, please contact Mark Webb-Johnson [mark@webb-johnson.net](mailto:mark@webb-johnson.net).

## 1.5 Development and Contributions

**New developers are very welcome on any part of the system, and we will gladly provide any help needed to get started.**

The purpose of this project is to get the community of vehicle hackers and enthusiasts to be able to expand the project. We can't do it all, and there is so much to do. What we are doing is providing an affordable and flexible base that the community can work on and extend.

Everything is open, and APIs are public. Other car modules can talk to the server, and other Apps can show the status and control the car. This is a foundation that others hopefully will interface to and build upon.

**If you'd like to contribute, please accept our code of conduct:**

- Introduce yourself on the developer mailing list
- Be kind & polite
- Understand the framework concepts
- Ask if you need help
- Present your plans if in doubt
- Write decent code
- If you extend modules, stick to their code style
- Write brief but descriptive commit comments
- Add user level descriptions to the change history
- Provide documentation in the user guide
- Use pull requests to submit your code for inclusion

### **A note on pull requests:**

Pull requests shall focus on one specific issue / feature / vehicle at a time and shall only mix vehicle specific changes with framework changes if they depend on each other. If changes are not or only loosely related, split them into multiple PRs (just as you would do with commits).

Usage hint: create a branch for each pull request, include only those commits in that branch (by cherry-picking if necessary) that shall be included in the pull request. That way you can push further commits to that branch, Github will automatically add them to an open pull request.

## 1.6 Donations

The OVMS is a non-profit community project. Hardware production and service can normally be financed by sales, but some things (e.g. prototype development and certifications) need extra money. To help the project, you can make a donation on the OVMS website: <https://www.openvehicles.com/forum>

Please also consider supporting the vehicle developers directly. Check out their web sites and support addresses for their respective donation channels.

**Thank you!**

## 1.7 License

The project includes some third party libraries and components to which their respective licenses apply, see component sources for details.

The project itself is published under the MIT license:

Copyright (c) 2011-2020 Open Vehicles

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Software which uses other licenses will be annotated appropriately.



### 2.1 General Warnings



**Warning!**

OVMS is a hobbyist project, not a commercial product. It was designed by enthusiasts for enthusiasts. Installation and use of this module requires some technical knowledge, and if you don't have that we recommend you contact other users in your area to ask for assistance.



**Warning!**

The OVMS module is continuously powered by the car, even when the car is off. While the OVMS module uses extremely low power, it does continuously draw power from the car's battery, so it will contribute to 'vampire' power drains.

Do not allow your car battery to reach 0% SOC, and if it does, plug in and charge the car immediately. **Failure to do**

**this can result in unrecoverable failure of the car's battery.**

The module can monitor the main and 12V battery and send alert notifications if the SOC or voltage drops below a healthy level.

### 2.1.1 Average Power Usage

The power used by the module depends on the component activation. You can save power by disabling unused components. This can be done automatically by the Power Management module to avoid deep discharging the 12V battery, or you can use scripts to automate switching components off and on.

The base components need approximately these power levels continuously while powered on:

Component	Avg Power	12V Current
Base System	200 mW	17 mA
Wifi	330 mW	28 mA
Modem	170 mW	13 mA
GPS	230 mW	19 mA
<b>Total</b>	<b>930 mW</b>	<b>78 mA</b>

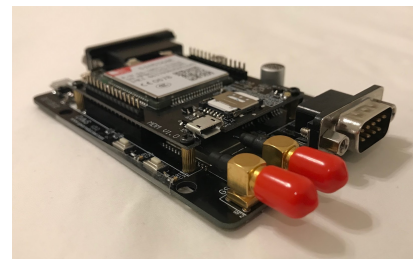
This adds up to:

- ~ 22 Wh or 2 Ah / day
- ~ 156 Wh or 13 Ah / week
- ~ 680 Wh or 57 Ah / month

Note that depending on the vehicle type, the module may also need to wake up the ECU periodically to retrieve the vehicle status. Check the vehicle specific documentation sections for hints on the power usage for this and options to avoid or reduce this.

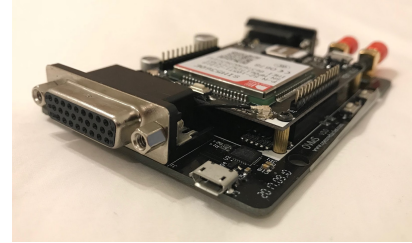
## 2.2 Components

### 2.2.1 The OVMS Module



The OVMS v3 module is housed in a plastic enclosure; held secure by four small screws. Once open, you can see the main OVMS v3 motherboard, and an optional modem board.

At one end of the module is the main DB9 connector you will use to connect to the vehicle, as well as GSM (cellular) and GPS (positioning) antenna connections.



At the other end of the module is the DA26 expansion connector, the USB diagnostic connector, and a Micro SD card slot.

**Warning:** The **USB port** is a fragile PCB socket without reinforcement. Handle it carefully, do not apply force when plugging or unplugging cables, to not bend the plug. If you need to use the port frequently, two dots of hot glue at the port sides on the PCB can help to avoid breaking it off.

If you open the module and take out the board, take care to guide the port carefully and correctly into its case hole when inserting the board into the casing again. Insert the board USB port side first.

If removing/installing optional expansion boards (such as used for cellular connectivity), please take care to ensure you secure screw down the expansion board using the four pillar posts provided. Also, please ensure that the cellular modem connections are correct (follow the printed table on the modem board to know which antenna is which).



### Warning!

The OVMS v3 enclosure is not waterproof, and the components can be damaged by water. Do not get the module wet, and do not connect it to your vehicle if it is wet.

## 2.2.2 Cellular Modem Option and GSM Antenna

The cellular modem option allows you to control your vehicle when out of wifi coverage range. The majority of OVMS users choose this option, and you will require it if you want to monitor your vehicle when away from home or office.

OVMS modules sold in USA and Europe are provided with a Hologram SIM card pre-installed. This low cost service allows you to get cellular connectivity simply. It also allows you to roam between countries without worry. For modules purchased from China, we recommend you purchase a Hologram SIM directly from the hologram.io store (also available on Amazon).

Depending on settings, verbosity towards the OVMS server, rhythm of GPS tracking, etc, OVMS v3 will use between 1 and 3 Megabytes per month of data (when using the v2 server protocol).

You do not have to use the Hologram service. If you use another cellular provider, the Sim Card format required is 4FF Nano. Micro Sim cards are hard to recut into the smaller format, so please be careful to not damage the socket; otherwise, ask your operator for a swap (some do it for free).

If you are using the cellular option, you should attach a suitable cellular antenna to the module, using the antenna connector labeled “GSM”.

### 2.2.3 GPS/GNSS Antenna

Some OVMS vehicles can read the GPS signals from the car communication networks directly, and do not require any additional hardware. For others, the OVMS v3 modem option also includes a GNSS/GPS satellite tracking receiver.

If you are using this option, you should connect a suitable active GPS antenna to the connector labelled “GPS”.

### 2.2.4 Vehicle Connection

The connection to the vehicle is by the DB9 connector labelled “VEHICLE”. This provides power to the OVMS module, as well as connection to the vehicle communication networks.

Different vehicles require different cables, so you should refer to the appropriate vehicle section of this user guide to determine which is correct for yours.

### 2.2.5 OVMS Server v2

The OVMS Server v2 protocol is a proprietary protocol used to communicate between the vehicle and an OVMS v2 server, as well as from that server to the cellphone apps. To provide compatibility with existing OVMS v2 cellphone apps and servers, OVMS v3 includes full support for the OVMS v2 protocol.

### 2.2.6 OVMS Server v3

The OVMS Server v3 protocol is MQTT. This is an industry standard protocol, and means an OVMS v3 module can communicate with any standard MQTT server. While this is the future of OVMS, support for this is experimental at the moment and production users should use OVMS Server v2 protocol.

### 2.2.7 Upgrading from OVMS v1/v2 to v3

The antenna and vehicle connectors for OVMS v3 are the same as for OVMS v2, and existing cables/antennas can generally be re-used for OVMS v3. Note, however, that the frequency ranges supported by individual 3G networks may be different than 2G, so may benefit from an antenna specifically designed for the 3G frequency ranges used

## 2.3 Installation

### 2.3.1 Pre-Installation Steps



#### **Warning!**

Prior to connecting the OVMS module to the vehicle, or computer via USB, if you have the GSM cellular option we recommend you connect a GSM antenna. GSM systems are designed to always operate with an antenna, and powering on one without could damage the equipment.



Prior to installation, please make sure you have the following available:

1. The OVMS v3 module in it's enclosure.
2. A micro-usb cable suitable for connecting to your computer.
3. A laptop or desktop computer (if necessary).
4. A cable suitable for connecting to your vehicle.
5. A GSM antenna (if you are using the cellular option).
6. A GPS antenna (if your vehicle type requires one).

You should also have ready access to this User Guide, and wifi connectivity to the Internet.

## 2.3.2 OVMS Module Installation

### Powering the module

If you intend to configure the module on your desk before connecting it to the vehicle, make sure your USB port delivers power (around 500mA, depending on modem and wifi activity). We recommend using a USB hub with a separate power supply or a direct port of your laptop / PC.

### OVMS Server account

If you want to use the OVMS App and/or server based telemetry services, you'll need an OVMS v2 server account. If you have not registered for an OVMS server account yet, you can do so before starting the wizard to avoid needing to switch networks in between. There are currently two public v2 OVMS servers:

1. Asia-Pacific: <https://www.openvehicles.com/>
2. Europe: <https://dexters-web.de/>

You will need to create a user account first. Within your user account you then need to create a vehicle account. You'll need to pick a unique vehicle ID for this, e.g. your vehicle license plate number.

### Initial Connection (Wifi and Browser)

From the factory, or after a factory reset, your OVMS module will be running an access point, with the following credentials:

SSID: OVMS Password: OVMSinit

As this is insecure, you should take care not to leave the module running unconfigured.

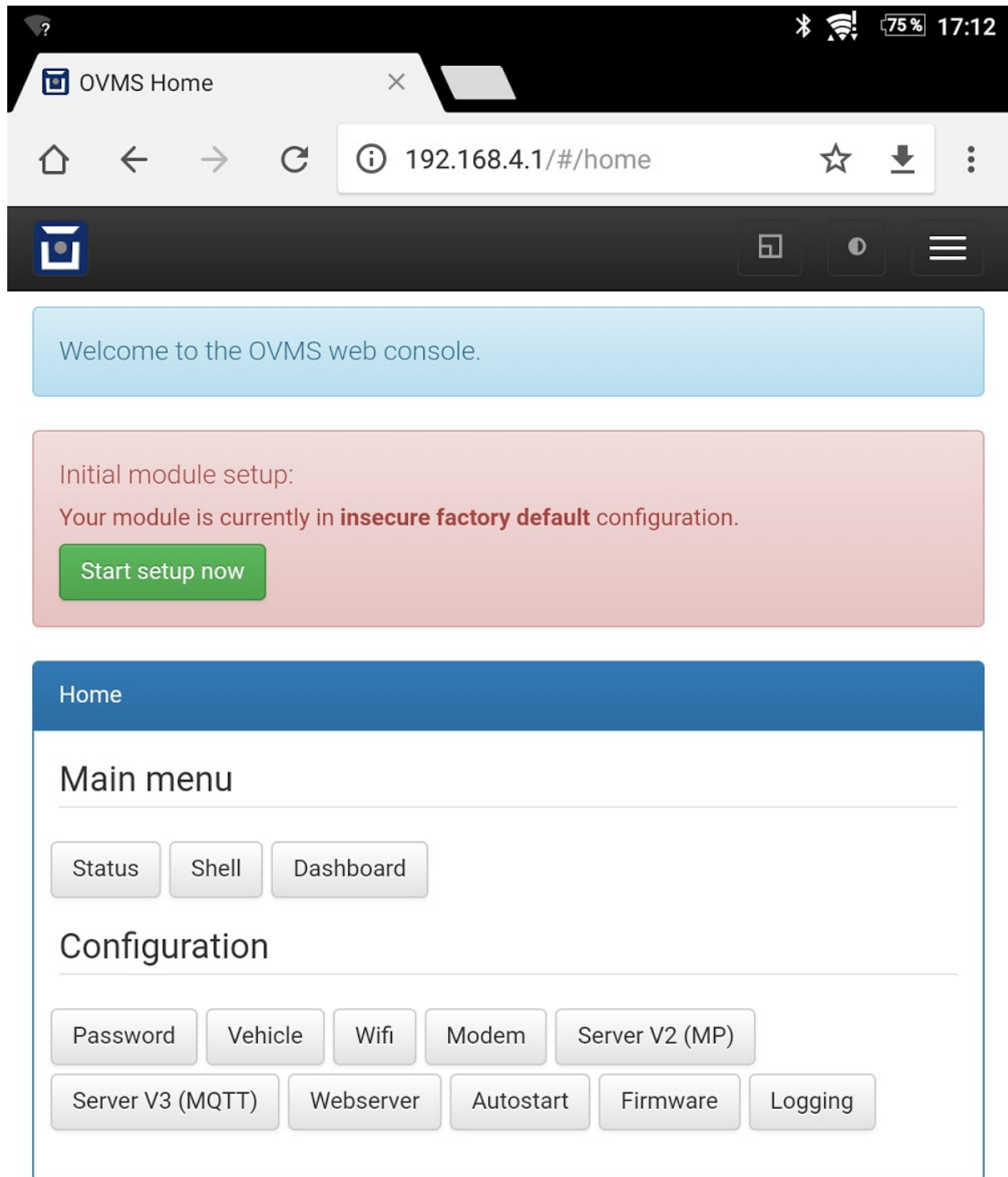
Using your laptop/tablet/phone, establish a wifi connection to the module. You should see an IP address in the range 192.168.4.x allocated, with a gateway at 192.168.4.1.

Note: Some smartphones (e.g Android) require mobile data to be switched off to use a WiFi connection without a internet connectivity.

Launch your web browser, and connect as follows:

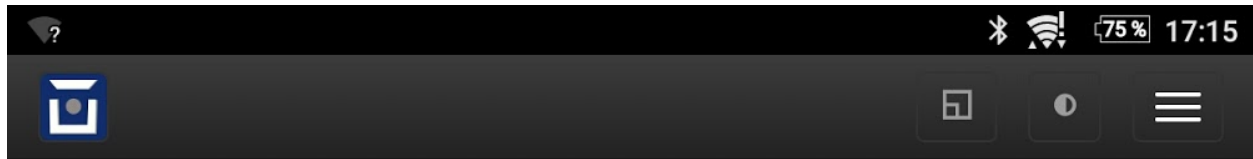
URL: <http://192.168.4.1/>

Once connected, you will be presented with a screen as follows:



## **Setup wizard**

The first thing to do is run the setup wizard. Click **Start setup now**. The wizard takes you through the initial setup in five simple steps, telling you what it is doing and what to expect for each step.



### Quick setup wizard

1. **Secure module access**
2. Connect module to internet (via Wifi)
3. Update to latest firmware version
4. Configure vehicle type and server
5. Configure modem (if equipped)

### Step 1/5: Secure Module

Your module may currently be accessed by anyone in Wifi range using the default password, so we first should do something about that.

This setup step will change both the Wifi access point and the module admin password, so the module is secured for both local and remote access.

#### Module ID:

Use ASCII letters, digits and '-'

Enter a unique personal module/vehicle ID, i.e. your vehicle license plate number.

If already registered at an OVMS server, use the vehicle ID as registered.

The ID will be used for the Wifi access point, check if it is free to use within your Wifi range.

#### Module password:

Enter a password, min. 8 characters

Inspiration: Q^oytxnVP#vX o^%;nrnyhnt. 2%x%h\*j\*=vE dc?n.&mbZObi  
fc3dmTzupv\_.

#### Repeat password:

Repeat password to verify entry

Proceed

Skip step

Abort setup

The wizard will need to reconfigure the module for the Wifi setup, read the notes and be prepared to reconnect to the module as necessary.

Note: we recommend not to use a password manager during the setup process. Some browsers, e.g. Chrome, will fill in the module ID as the username, which is wrong. The login username needs to be **admin**.

The wizard should be able to restore access after problems occurring in the process. As a last resort if it fails to recover at some point, you can always do a factory reset and start over again.

## Manual configuration

After finishing the wizard or if you prefer to do a manual setup, the configuration menus will provide single pages for each module function. These also contain advanced options for the features, so it's worth having a look.

## Vehicle Configuration

Go to Config / Vehicle:

The screenshot shows a web browser interface for vehicle configuration. The top navigation bar includes 'Status', 'Shell', and 'Config' (which is active). The main content area is titled 'Vehicle configuration'. It contains several form fields: 'Vehicle type' (a dropdown menu), 'Vehicle ID' (a text input field with a note: 'Use ASCII letters, digits and '-''), 'Vehicle name' (a text input field with a note: 'optional, the name of your car'), 'Time zone' (a text input field with a note: 'optional, default UTC'), and 'Distance units' (a dropdown menu currently set to 'Kilometers'). A 'Save' button is located at the bottom of the form.

You'll want to enter your vehicle type, Vehicle ID (the same as you registered on the OVMS server), and distance units. You can also optionally enter your timezone (see [https://www.gnu.org/software/libc/manual/html\\_node/TZ-Variable.html](https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html) for an article on GLIBC timezones for information on the format of this, a list of suitable zone strings can also be found here: <https://remotemonitoringsystems.ca/time-zone-abbreviations.php>).

## OVMS Server v2 Configuration

Go to Config / Server V2 to configure the connection to the OVMS v2 server you will be using:

The screenshot shows a mobile web browser interface for the 'Open Vehicles' application. The top status bar shows 'csl.', '2:48 PM', and '84%' battery. The browser address bar shows '192.168.4.1'. The application header has a menu icon, 'Status', 'Shell', 'Config', and a 'Logout' link. The main content area is titled 'Server V2 (MP) configuration' and contains the following fields and options:

- Host:** A text input field with the placeholder 'Enter host name or IP address'.
- Public OVMS V2 servers:** A list of two servers:
  - [api.openvehicles.com](#) Registration
  - [ovms.dexters-web.de](#) Registration
- Port:** A text input field with the placeholder 'optional, default: 6867'.
- Vehicle ID:** A text input field containing 'EV914'.
- Vehicle password:** A text input field with the placeholder 'empty = no change'.
- Note:** 'enter the password for the vehicle ID account, not your user account password'.
- Update intervals:** A section with two fields:
  - ...connected:** A text input field with the placeholder 'optional, in seconds, default: 60'.
  - ...idle:** A text input field with the placeholder 'optional, in seconds, default: 600'.
- Save:** A button at the bottom of the configuration area.

You should enter the server host ([api.openvehicles.com](#), or [ovms.dexters-web.de](#), usually), and vehicle password (aka *server password* - as entered on the server when you registered your vehicle). The Vehicle ID field should already be there, and the other parameters are optional.

### Auto Start Configuration

OVMS has a powerful scripting language that can be used for complex configurations, but to get started it is simplest to use the Auto Start system. You get to this from the web interface by clicking Config / Autostart.

Auto start configuration

☒ Enable auto start  
Note: if a crash or reboot occurs within 10 seconds after powering the module, this option will automatically be disabled and need to be re-enabled manually.

☐ Power on external 12V  
Enable to provide 12V to external devices connected to the module (i.e. ECU displays).

Wifi mode: Access point

... access point SSID: OVMS

... client mode SSID: Any known SSID (scan mode)

☐ Start modem  
Note: a vehicle module may start the modem as necessary, independantly of this option.

Vehicle type: --

Start OBD2ECU: --  
OBD2ECU translates OVMS to OBD2 metrics, i.e. to drive standard ECU displays

☐ Start server V2

☐ Start server V3

Save Save & reboot

You will usually want to click to **Enable auto start**, and **Start server v2**. The other fields should have been populated correctly automatically for you. If you are using the optional modem module, you should also click **Start modem** to enable the modem.

Once complete, you can **Save & reboot** to activate your new configuration.



### Warning!

Do not set the Wifi mode to **AP+Client** or **Client** before having configured your Wifi network.

If you have configured this manually, the Wifi network may not start automatically. Log in using a USB terminal and either do a factory reset (see Module Factory Reset) or (better) issue **enable** to enter secure mode, then issue **config set auto wifi.mode ap** and reboot.

## Networking Options

OVMS v3 has a number of networking options to choose from. You can either use these individually, or combine them to provide failover and alternative network connectivity arrangements.

1. Wifi Client. OVMS can connect to a WiFi Access Point, using standard WiFi (802.11 b/g/n) protocols, to connect to a SSID (Access Point name) with associated password. In simple client mode, you can connect only to a single pre-specified SSID. Alternatively, you can use the scanning client mode to connect to any known

WiFi Access Point when within range (note, however, that this is not possible when you run both client and access point on the same OVMS device).

2. Wifi Access Point. OVMS can operate as a WiFi Access Point itself, using standard WiFi (802.11 b/g/n) protocols. This allows users to connect to the OVMS module itself. Note that OVMS v3 is not intended to be a hotspot and users cannot access the Internet via the OVMS module. Wifi Access Point mode can be combined with simple Wifi Client mode, to provide an access point for maintenance of the module, as well as a client to access the Internet via another Access Point within range.
3. Cellular Data. OVMS supports optional modems to provide cellular connectivity. These are configured via Config / Modem.

### 2.3.3 GSM SIM Activation (Hologram)

OVMS has partnered with Hologram and to provide a Hologram GSM SIM pre-installed in every OVMS kit purchased from our partners in Europe and USA. For modules purchased from China, we recommend you purchase a Hologram SIM directly from the [hologram.io](https://hologram.io) store (also available on Amazon). In addition, Hologram have provided OVMS a coupon code valid for US\$5 off data usage:

Hologram Coupon Code: **OVMS**

To activate your Hologram SIM, register at <https://dashboard.hologram.io/>, then invoke “Activate SIM” in the dashboard.

---

**Note:** You don’t need to purchase a phone number for your SIM right now, as there is no SMS support in V3 yet. For the current status of SMS support, see. . .

- [Issue #62 SMS Notifications](#)
  - [Issue #63 SMS Command Gateway](#)
- 

When activating your Hologram SIM, you’ll need to enter the ICCID written on the SIM itself. You can also get that electronically (without having to open up the enclosure) from the OVMS web or terminal shell (Tools > Shell) with the following command:

```
OVMS# metric list m.net.mdm.iccid
```

The ICCID is also displayed during the setup process and on the modem configuration page when using the web user interface.



Modem configuration

Internet

☒ Enable IP networking

APN:

For Hologram, use APN **hologram** with empty username & password

...username:

...password:

DNS:

Set this to i.e. **8.8.8.8 8.8.4.4** (Google public DNS) if you encounter problems with your network provider DNS

Features

☒ Enable SMS

☐ Enable GPS

☐ Use GPS time

Note: GPS & GPS time support can be left disabled, vehicles will activate them as needed

### 2.3.4 Firmware Update



The factory firmware that is provided with the module may be quite out of date. You should perform a firmware update to ensure that you have the latest firmware. You can do this either over Wifi client connections, or via an SD CARD.

We recommend using the auto update system. This will be preconfigured if you have used the setup wizard. The automatic updates are done within a selectable hour of day, and only if Wifi connectivity is available at the time.

### Flash from Web

You can typically just press the **Flash now** button and wait for completion.

### Flash from File

Using an SD CARD formatted as FAT, download the firmware update and place it in a file called **ovms3.bin** in the root directory of the SD CARD. Once the SD CARD is inserted the firmware update will start immediately.

## 2.3.5 12V Monitoring

**Note:** Since release 3.2.006 the 12V calibration and alert setup can be done from the web UI's vehicle configuration page.

As 12V batteries tend to die without warning and need to handle an additional unplanned constant load from the OVMS, the module includes a 12V monitoring and alert system.

### Calibration

The 12V voltage is measured using the incoming voltage that powers the OVMS. As the sensor used by the module has some manufacturing tolerances you should do an initial calibration. Use a voltage meter to measure the actual voltage somewhere suitable (e.g. at a 12V auxiliary equipment plug), calibrate the OVMS to show the same. The calibration factor is set by...

```
config set system.adc factor12v <factor>
```

Calculate the <factor> using:  $\text{oldFactor} * (\text{displayedVoltage} / \text{actualVoltage})$

- oldFactor is the old value set. If you have not changed it yet it is 195.7.
- displayedVoltage is the Voltage as displayed by the OVMS.
- actualVoltage is the Voltage as measured by hand using a voltmeter.

The voltage is read once per second and smoothed over 5 samples, so after changing the factor, wait 5-10 seconds for the new reading to settle.

## Configuration

The default 12V reference voltage (= fully charged & calmed down voltage level) can be set by...:

```
config set vehicle 12v.ref <voltage>
```

This config value initializes metric `v.b.12v.voltage.ref` on boot. The metric will then be updated automatically if your vehicle supports the `v.e.charging12v` flag. The measured reference voltage reflects the health of the 12V battery and serves as the reference for the 12V alert, if it's higher than the configured default.

The 12V alert threshold can be set by...:

```
config set vehicle 12v.alert <voltage>
```

The 12V alert threshold is defined by a relative value to the 12v reference voltage. If the actual 12V reading drops below `12v.ref - 12v.alert`, the 12V alert is raised.

The default reference voltage is 12.6V, the default alert threshold 1.6V, so the alert will be triggered if the voltage drops below 11.0V. This is suitable for standard lead-acid type batteries. If you've got another chemistry, change the values accordingly.

## Related Metrics

Metric	Example Value	Meaning
<code>v.b.12v.current</code>	0.6A	Momentary current level at the 12V battery
<code>v.b.12v.voltage</code>	13.28V	Momentary voltage level at the 12V battery
<code>v.b.12v.voltage.ref</code>	12.51V	Reference voltage of the fully charged & calmed down 12V battery
<code>v.b.12v.voltage.alert</code>	no	If the 12V critical alert is active (yes/no).
<code>v.e.charging12v</code>	yes	If the 12V battery is charging or not (yes/no)

## Related Events

Event	Data	Purpose
<code>vehicle.alert.12v.on</code>		12V system voltage is below alert threshold
<code>vehicle.alert.12v.off</code>		12V system voltage has recovered
<code>vehicle.charge.12v.start</code>		Vehicle 12V battery is charging
<code>vehicle.charge.12v.stop</code>		Vehicle 12V battery has stopped charging

## 2.4 The OVMS Console

### 2.4.1 Console Connections

OVMS v3 includes a powerful command line console that can be accessed in various ways:

1. Using a micro USB cable to a host computer.

If the OVMS is not recognised via USB download the driver from [SILABS website](#). You will also need a suitable terminal emulator. The baud rate is 115200, and you should not enable hardware flow control.

2. TELNET (over wifi).

Note that for security reasons, the telnet server component is not enabled in the default production firmware (but may be available in custom builds). If Telnet is available telnet to the IP address of the module (or <vehicleid>.local MDNS address).

### 3. SSH (over wifi).

SSH to the IP address of the module (or <vehicleid>.local MDNS address). Note that when first booted with a network connection, the module takes a minute or so to generate server side keys (which are stored in the config store).

### 4. Web Console SHELL tab.

Use a web browser to connect to the IP address of the module (or <vehicleid>.local MDNS address). A SHELL tab is available for direct command line console access.

### 5. Remote Apps.

The OVMS Android App currently includes a shell screen that can be used to issue command line console commands via the OVMS server v2. This functionality is not currently available in the iPhone/iPad App.

### 6. SMS.

Console commands can be issued via SMS.

## USB Console

Our recommendations for the USB console are as follows:

1. You can use a Windows, Linux, Mac OSX workstation or an Android device with a USB OTG adapter cable.
2. If your operating system does not have the SILABS USB driver, you can download the driver from [SILABS website](#). If you use Linux and your distribution includes the braille display driver “brltty”, you may need to uninstall that, as it claims any CP2102 device to be a braille device. This applies e.g. to openSuSE 15.0.
3. Plug in the module to your PC/laptop, using a micro USB cable. Check to ensure a serial port appears (using the SILABS driver). For OSX and Linux this will normally appear as /dev/tty.SLAB\_USBtoUART or /dev/ttyUSB0 (or 1/2/... if other serial devices are connected). List your devices using “ls /dev/\*USB\*”.
4. Once the serial port is available you will need a terminal emulator.
  - For OSX, the simplest is the built-in SCREEN utility. You run this as `screen -L /dev/tty.SLAB_USBtoUART 115200` But note that the device path may be different for you - check with ‘ls /dev/\*USB\*’. You can use ‘control-a control-y’ or ‘control-a k y’ (three key sequences) to exit the screen. The “-L” option tells screen to capture a log of your session into the file “screenlog.<n>”.
  - For Linux, the SCREEN utility is also simple to get. If it is not included with your distribution, you can simply *yum install screen*, or *apt-get install screen* (depending on your distribution). From there, the command is the same as for OSX. Alternatively, you can use minicom (which is included with many linux distributions).
  - For Windows, a simple approach is to download the free PUTTY terminal emulator. This supports both direct ASYNC (over USB) connections, as well as SSH (network). You can download putty [using this link](#).
  - For Android, there are [multiple USB serial Apps in the Play store](#). A good recommendation is [Serial USB Terminal by Kai Morich](#).
5. Once you have established the connection, press ENTER to see the “OVMS>” prompt.

## SSH Console

A workstation (Mac, Linux, Window), on the same wifi network as the OVMS module, can use the ssh protocol to connect. In Windows you can use the free PUTTY ssh client. In Linux and OSX ssh is built-in.

The syntax is simply:

```
ssh user@ip
```

Where ‘user’ is the username (normally ‘ovms’) and ‘ip’ is the IP address of the OVMS v3 module. In environments supporting mDNS networking, you should also be able to connect using the mDNS name *<vehicleid>.local*. The password you enter is the module password.

If you use ssh public/private key pairs, you can store your public key on the OVMS v3 module, to take advantage of passwordless login.

```
OVMS# config set ssh.keys <user> <public-key>
```

In this case, ‘user’ is the username you use to ssh, and the public key is your RSA public key (the long base64 blob of text you find in id-rsa.pub between ‘ssh-rsa’ and your username/comment).

You can also use SCP to copy files to and from the OVMS v3 VFS.

---

**Note:** With OpenSSH version 6.6 (or later), cipher `aes128-cbc` has been disabled by default and needs to be enabled manually, either on the command line:

```
ssh -c aes128-cbc user@ip
```

...or by adding a host entry to your `~/.ssh/config` file:

```
Host ovmsname.local
Ciphers +aes128-cbc
```

---

## 2.4.2 Console Basics

Let’s use SSH to demonstrate this:

```
$ ssh ovms@ovms.local

Welcome to the Open Vehicle Monitoring System (OVMS) - SSH Console
Firmware: 3.1.003-2-g7ea18b4-dirty/factory/main
Hardware: OVMS WIFI BLE BT cores=2 rev=ESP32/1

OVMS#
```

When first connecting using USB, the console will be in non-secure mode (as indicated by the “OVMS>” prompt). Here, only a limited number of commands are available (such as viewing network status, modem status, or system time). To get to secure mode, enter the command ‘enable’, and provide the module password. The prompt will then change to “OVMS#”:

```
OVMS> enable
Password:
Secure mode
OVMS#
```

You can enter the ‘disable’ command to get out of secure mode, and ‘exit’ to exit the console completely.

When connecting via a pre-authenticated protocol such as SSH, you will be in secure mode automatically.

At any time, you can use “?” to show the available commands. For example:

```
OVMS# ?
.                Run a script
boot             BOOT framework
can              CAN framework
charge           Charging framework
co               CANopen framework
config           CONFIG framework
disable          Leave secure mode (disable access to most commands)
egpio            EGPIO framework
enable           Enter secure mode (enable access to all commands)
event            EVENT framework
exit             End console session
help             Ask for help
homelink         Activate specified homelink button
location         LOCATION framework
lock             Lock vehicle
log              LOG framework
metrics          METRICS framework
module           MODULE framework
network          NETWORK framework
notify           NOTIFICATION framework
obdii            OBDII framework
ota              OTA framework
power            Power control
re               RE framework
script           Run a script
sd               SD CARD framework
server           OVMS Server Connection framework
simcom           SIMCOM framework
stat             Show vehicle status
store            STORE framework
test             Test framework
time             TIME framework
unlock           Unlock vehicle
unvalet          Deactivate valet mode
valet            Activate valet mode
vehicle          Vehicle framework
vfs              Virtual File System framework
wakeup           Wake up vehicle
wifi             WIFI framework
```

You can also use “?” as part of a command to expand on the available options within that command root:

```
OVMS# wifi ?
mode             WIFI mode framework
scan             Perform a wifi scan
status           Show wifi status

OVMS# wifi mode ?
ap               Acts as a WIFI Access Point
apclient         Acts as a WIFI Access Point and Client
client           Connect to a WIFI network as a client
off              Turn off wifi networking

OVMS# wifi mode client ?
Usage: wifi mode client <ssid> <bssid>
```

Command tokens can be abbreviated so long as enough characters are entered to uniquely identify the command. Op-

tionally pressing TAB at that point will auto-complete the token. If the abbreviated form is not sufficient to be unique (in particular if no characters have been entered yet) then TAB will show a concise list of the possible subcommands and retype the portion of the command line already entered so it can be completed:

```
OVMS# wifi <TAB>
mode scan status
OVMS# wifi
```

Pressing TAB is legal at any point in the command; if there is nothing more that can be completed automatically then there will just be no response to the TAB.

## 2.5 Logging

### 2.5.1 Logging to the Console

Components of the OVMS system output diagnostic logs (information, warnings, etc). You can choose to display these logs on your connected console with the ‘log monitor yes/no’ command:

```
OVMS# log monitor ?
Usage: log monitor [no|yes]
no           Don't monitor log
yes          Monitor log
```

By default, the **USB console** will have log monitoring ‘yes’, **SSH** (and telnet if enabled) ‘no’.

The **web shell** does not use the `log monitor` command but has a checkbox in the upper right corner of the shell panel instead. The keyboard shortcut for the checkbox is `L` (Alt-L or Alt-Shift-L depending on your browser). The web frontend gets the continuous stream of log messages independent of the shell panel or the monitoring being active, and shows the last 100 messages received when opening the shell panel.

Logs are output at various levels of verbosity, and you can control what is shown both globally and on a per-component basis:

```
OVMS# log level ?
none           No logging (0)
error          Log at the ERROR level (1)
warn           Log at the WARN level (2)
info           Log at the INFO level (3)
debug          Log at the DEBUG level (4)
verbose        Log at the VERBOSE level (5)
```

(Note: sorted here for level clarity)

The syntax of this command is `log level <level> [<component>]`. If the component is not specified, the level applies to all components that don’t get a level set explicitly afterwards. The levels increase in verbosity, and setting a particular level will also include all log output at a lower level of verbosity (so, for example, setting level *info* will also include *warn* and *error* output).

A log line typically looks like this:

```
I (32244049) ovms-server-v2: One or more peers have connected
├── Log message
├── Component name
├── Timestamp (milliseconds since boot)
└── Log level (I=INFO)
```

Log levels are applied on log message generation, so a later change to a higher level will not reveal messages generated previously.

### 2.5.2 Logging to SD CARD

You can also choose to store logs on SD CARD. This is very useful to capture debugging information for the developers, as the log will show what happened before a crash or misbehaviour.

We recommend creating a directory to store logs, i.e.:

```
OVMS# vfs mkdir /sd/logs
```

To enable logging to a file, issue for example:

```
OVMS# log file /sd/logs/20180420.log
```

The destination file can be changed any time. To disable logging to the file, issue `log close`, to restart logging after a close issue `log open`. You may choose an arbitrary file name, good practice is using some date and/or bug identification tag. Note: logging will append to the file if it already exists. To remove a file, use `vfs rm ...`.

File logging does not persist over a reboot or crash (unless configured as shown below), you can use a script bound to the `sd.mounted` event to re-enable file logging automatically or configure automatic logging (see below).

You can use the webserver to view and download the files. The webserver default configuration enables directory listings and access to files located under the document root directory, which is `/sd` by default. Any path not bound to an internal webserver function is served from the document root. So you can get an inventory of your log files now at the URL:

```
http://192.168.4.1/logs/
```

... and access your log files from there or directly by their respective URLs. Another option to retrieve the files without unmounting the SD card is by `scp` if you have configured SSH access.

### 2.5.3 Logging Configuration

Use the web UI or `config` command to configure your log levels and file setup to be applied automatically on boot:

```
OVMS# config list log
log (readable writeable)
  file.enable: yes
  file.keepdays: 7
  file.maxsize: 1024
  file.path: /sd/logs/log
  file.syncperiod: 3
  level: info
  level.simcom: info
  level.v-twizy: verbose
  level.webserver: debug
```



Logging configuration

☒ Enable file logging

**Log file path:** 

Logging to SD card will start automatically on mount. Do not remove the SD card while logging is active.

**Download:**

**Max file size:**  

When exceeding the size, the log will be archived suffixed with date & time and a new file will be started. 0 = disable

**Default level:**

**Component levels:**

	Component	Level
<input checked="" type="checkbox"/>	simcom	Info (default)
<input checked="" type="checkbox"/>	v-twizy	Verbose
<input checked="" type="checkbox"/>	webserver	Debug
<input type="checkbox"/>		

The `log` command can be used for temporary changes, if you change the configuration, it will be applied as a whole, replacing your temporary setup.

If a maximum file size `>0` is configured, the file will be closed and archived when the size is reached. The archive name consists of the log file name with added suffix of the timestamp, i.e. `/sd/logs/log.20180421-140356`. Using a logs directory will keep all your archived logs accessible at one place. If `file.keepdays` is defined, older archived logs will automatically be deleted on a daily base.

Take care not to remove an SD card while logging to it is active (or any running file access). The log file should still be consistent, as it is synchronized after every write, but the SD file system currently cannot cope with SD removal with open files. You will need to reboot the module. To avoid this, always use the “Close” button or the `log close` command before removing the SD card.

You don’t need to re-enable logging to an SD path after insertion, the module will watch for the mount event and automatically start logging to it.

## 2.5.4 Performance Impact

SD card I/O has an impact on the module performance. So file logging should generally be switched off or run on a low level (i.e. “info” or “warn”) unless you’re hunting some bug or checking some details of operation. We also recommend using a fast SD card for logging (check the speed with `sd status`, check if you can raise `config sdcard maxfreq.khz` to 20000 kHz).

File logging is done by a separate task, but flushing the file buffers to the SD card still may block the logging CPU core or even both CPU cores for a short period. To reduce the impact of this, the log task by default only flushes the buffer after 1.5 seconds of log inactivity. This means you may lose the last log messages before a crash.

To change the flush behaviour, set config `file.syncperiod` to...

- 0 = never flush (i.e. only at `log close` / log cycle)
- < 0 = flush every `n` log messages (i.e. -1 = flush after every message)
- > 0 = flush after `n/2` seconds idle

The log task counts the time spent for flushes and outputs it with the `log status` command:

```
OVMS# log status
Log listeners      : 3
File logging status: active
  Log file path    : /sd/logs/log
  Current size     : 817.0 kB
  Cycle size       : 1024 kB
  Cycle count      : 8
  Dropped messages : 0
  Messages logged  : 70721
  Total fsync time : 651.1 s
```

This is an example for the default configuration of `file.syncperiod`: 3, the logging here has on average taken  $651.1 / 70721 = 9$  ms per message.

## 2.6 Configuration

OVMS stores all it's configuration in a standardised protected configuration area accessed by the 'config' command. The configurations are organised as follows:

```
<parameter> <instance> = <value>
```

For example:

Parameter	Instance	Value
vehicle	id	OVMSBOX
wifi.ssid	MYSSID	MyPassword
auto	init	yes

Each parameter can be defined (by the component that owns it) as having readable and/or writeable attributes, and these act as access control for the parameters.

- A 'writeable' parameter allows values to be created, deleted and modified.
- A 'readable' parameter allows values to be seen. Instance names can be seen on non-readable parameters (it is just the values themselves that are protected).

For example, the 'vehicle' parameter is readable and writeable:

```
OVMS# config list vehicle
vehicle (readable writeable)
  id: MYCAR
  timezone: HKT-8
```

But the 'wifi.ssid' parameter is only writeable:

```
OVMS# config list wifi.ssid
wifi.ssid (protected writeable)
  MYSSID
  MyOtherSSID
  MyNeighbour
```

The ‘config’ command is used to manipulate these configurations, as is fairly self-explanatory:

```
OVMS# config ?
list          Show configuration parameters/instances
rm            Remove parameter:instance
set           Set parameter:instance=value
```

You can add new instances to parameters simply by setting them.

Beginning with firmware release 3.2.009, a dedicated configuration section `usr` is provided for plugins. Take care to prefix all instances introduced by a unique plugin name, so your plugin can nicely coexist with others.

## 2.7 WiFi Networking

The OVMS WiFi system is based on the ESP32 integrated WiFi transceiver. It uses and provides WiFi protocols 802.11 b/g/n and WPA2 authentication. The current hardware can only use 2.4 GHz frequency bands.

The WiFi antenna is built into the ESP32 module (PCB antenna). It’s possible to replace that by an external antenna, but you’ll need electronics knowledge and SMD soldering skills & equipment.

### 2.7.1 Client & Access Point Modes

The OVMS WiFi network can be configured as a client (connecting to existing WiFi networks) and/or access point (providing it’s own private WiFi network). Both modes can be run simultaneously, which is the recommended default. That way, you can always connect to your module via WiFi.

```
OVMS# wifi status
Power: on
Mode: Access-Point + Client mode

STA SSID: WLAN-214677 (-78.7 dBm) [scanning]
  MAC: 30:ae:a4:5f:e7:ec
  IP: 192.168.2.102/255.255.255.0
  GW: 192.168.2.1
  AP: 7c:ff:4d:15:2f:86

AP SSID: DEX106E
  MAC: 30:ae:a4:5f:e7:ed
  IP: 192.168.4.1
  AP Stations: 0
```

In **client mode**, the module can connect to a fixed network, or automatically scan all channels for known networks to connect to (“scanning mode”). Scanning mode is configured by enabling client mode without a specific client SSID.

The module will normally receive an IP address, gateway and DNS from the WiFi access point by DHCP. See below on how to connect with a manual static IP setup.

In **access point mode**, the module provides access for other WiFi devices on a private network with the IP subnet 192.168.4.0/24. The module's IP address on this network is 192.168.4.1. The module does not provide a DNS or routing to a public WiFi or GSM network on this subnet.

You can define multiple AP networks, but only one can be active at a time.

Use the `wifi mode` command to manually set the mode, use the auto start configuration to configure your default mode and networks.

---

**Note:** The module cannot act as a mobile hotspot, i.e. provide access to the internet via its modem connection. That's in part due to security considerations, and in part due to hardware limitations. If you need this, consider installing a dedicated mobile hotspot and using that one via WiFi instead of the modem for the module's internet access (you won't need the modem in this setup).

---

### 2.7.2 Scanning for Networks

To manually scan your environment for available WiFi networks, do a `wifi scan`:

```
OVMS# wifi scan
Scanning for WIFI Access Points...

AP SSID                               MAC ADDRESS           CHAN  RSSI  AUTHENTICATION
=====
WLAN-214677                           7c:ff:4d:15:2f:86     1     -83  WPA2_PSK
Telekom_FON                           78:dd:12:09:dc:be     11    -84  OPEN
WLAN-184248                           78:dd:12:09:dc:bc     11    -85  WPA2_PSK
=====
Scan complete: 3 access point(s) found
```

This can also be done in the web UI's WiFi client network configuration page by clicking on the *Select* button.

After the module lost connection to a network, a scan is performed automatically every 10 seconds until a new connection can be established. If you'd like to see the background scans and results in the log, enable log level `verbose` for component `esp32wifi`.

### Scan Troubleshooting

If the module doesn't find your access point, or can only see it occasionally in the scan results, you may need to raise your scan times. This has been observed on some older Android hotspots.

By default, the module will scan each channel for 120 milliseconds. To raise that, set your scan time in milliseconds using the `config` command like this, e.g. to 200 milliseconds:

```
OVMS# config set network wifi.scan.tmin 200
```

200 milliseconds have been reported to solve the Android hotspot issue. In addition to the `tmin` configuration you may need to allow more time looking for further access points after finding the first one. Do so by setting `tmax`, e.g.:

```
OVMS# config set network wifi.scan.tmax 300
```

Note that every increase will increase the time a full scan takes, so don't set too high values. Too high values may result in connection drops on the AP network of the module.

### 2.7.3 WiFi Signal Quality

The module monitors the WiFi client signal quality and drops a WiFi connection (switches to modem if available) if it becomes too bad. The WiFi connection will be kept active and monitored, and as the signal recovers, the module will automatically reconnect to the AP.

The default threshold is to stop using the connection if it drops below -89 dBm. A connection is assumed to be usable if the signal is above -87 dBm.

Depending on your WiFi environment, the WiFi connection may still be usable at lower signal levels.

To tweak the thresholds, use the web UI WiFi configuration or change the following configuration variables:

```
OVMS# config set network wifi.sq.good -87.0
OVMS# config set network wifi.sq.bad -89.0
```

### 2.7.4 WiFi Mesh Configuration

In normal operation, the module will try to stick to an established connection as long as possible. If signal quality drops, it will switch to the modem connection, but monitor the WiFi signal and reassociate to the current AP if possible.

If using a mesh network, you may want to force scanning for a better mesh AP as soon as the signal drops below the “bad” threshold. To do so, set the network configuration `wifi.bad.reconnect` to true, either using the web UI or by doing:

```
OVMS# config set network wifi.bad.reconnect yes
```

With this, the module will perform a full WiFi reconnect cycle as soon as the signal becomes bad.

### 2.7.5 Static IP / SSID Configuration

To connect with a **static client address** instead of using DHCP, use the `wifi ip static` command:

```
OVMS# wifi ip static [<ip> <subnet> <gateway>]
```

The gateway will also be used as the DNS.

To configure persistent static details for a known SSID, set these using the following configuration syntax:

```
OVMS# config set wifi.ssid "<ssid>.ovms.staticip" "<ip>,<subnet>,<gateway>"
```

You can also **force connection to a specific AP** by its MAC address, “BSSID” in WiFi terms. To do so, you need to supply the MAC address as a second argument to the `wifi mode client` or as the third argument to the `wifi mode apclient` command:

```
OVMS# wifi mode client <ssid> <bssid>
OVMS# wifi mode apclient <apssid> <ssid> <bssid>
```

This currently needs manual activation by command. Hint: use a script, for example bound to a location.

### 2.7.6 AP Bandwidth Configuration

In normal operation, the default AP bandwidth is set to 20Mhz to reduce interference and improve the signal quality. If a wider 40Mhz bandwidth is preferred, eg for high throughput logging, set the network configuration `wifi.ap.bw`

with the command:

```
OVMS# config set network wifi.ap.bw 40
```

Once the parameter is set the module will require rebooting to take effect. The default bandwidth can be restored by either removing the parameter or setting it to 20.

## 2.8 Virtual File System (VFS)

OVMS includes a Virtual File System (VFS) used to unify all storage in the system. The primary configuration and scripting storage is mounted as '/store', and the SD card as '/sd'. A 'vfs' set of commands is provided for basic manipulation of these stores:

```
OVMS# vfs ?
append          VFS Append a line to a file
cat             VFS Display a file
cp             VFS Copy a file
edit           VFS Edit a file
ls             VFS Directory Listing
mkdir          VFS Create a directory
mv            VFS Rename a file
rm            VFS Delete a file
rmdir         VFS Delete a directory
stat          VFS Status of a file
tail          VFS Output tail of a file
```

Please take care. This is a very small microcontroller based system with limited storage. The /store area should only be used for storage of configurations and small scripts. The /sd SD CARD area is more flexible and can be used for storing of configuration logs, firmware images, etc.

### 2.8.1 Network Access

Network access to the VFS is available via SCP or through the web server.

SCP can access the whole file system and is read/write:

```
#copy a new firmware to the sd card on an OVMS named "leaf" in my ssh config
scp build/ovms3.bin leaf:/sd/dev_ovms3.bin

#copy a config backup from OVMS named "leaf" to my local directory
scp leaf:/sd/backup/cfg-2019-12-05.zip .
```

In the upload example the firmware can then be loaded with the `OTA flash vfs` command

The web server offers read access rooted at the sd card. An example retrieving that same config file from the SCP example would look like:

```
http://leaf-abr.local/backup/cfg-2019-12-05.zip
```

## 2.9 Metrics

Metrics are at the heart of the OVMS v3 system. They are strongly typed named parameters, with values in specific units (and able to be automatically converted to other units). For example, a metric to record the motor temperature

may be an integer in Celsius units, and may be convertible to Fahrenheit.

The full list of metrics available can be shown:

```
OVMS# metrics list
m.freeram          4232852
m.hardware          OVMS WIFI BLE BT cores=2 rev=ESP32/1
m.monotonic         3568Sec
...
v.p.latitude        22.2809
v.p.longitude        114.161
v.p.odometer         100000Km
v.p.satcount         12
v.p.speed            0Kph
v.p.trip             0Km
v.t.alert            0,0,0,1
v.t.health           95,93,96,74%
v.t.pressure          206.843,216.483,275.79,175.79kPa
v.t.temp             33,33,34,38°C
v.type              DEMO
```

You can filter the `metrics list` output for names matching a given substring, for example `metrics list volt` will show all voltage related metrics.

A base OVMS v3 system has more than 100 metrics available (see below), and vehicle modules can add more for their own uses (see vehicle sections).

In general, vehicle modules (and some other system components) are responsible for updating the metrics, and server connections read those metrics, reformat them, and send them on to servers and Apps (for eventual display to the user). Status commands (such as `STAT`) also read these metrics and display them in user-friendly forms:

```
OVMS# stat
Not charging
SOC: 50.0%
Ideal range: 200Km
Est. range: 160Km
ODO: 100000.0Km
CAC: 160.0Ah
SOH: 100%
```

For developer use, there are also some other metric commands used to manually modify a metric's value (for testing and simulation purposes), and trace changes:

```
OVMS# metrics ?
list          Show all metrics
persist       Show persistent metrics info
set           Set the value of a metric
trace         METRIC trace framework
```

Some metrics are persistent across warm reboots. This prevents values such as SOC from being lost when firmware is updated (or in the event of a crash). You can display these with `metrics list -p` and view general information about persistent metrics with `metrics persist`.

## 2.9.1 Standard Metrics

Metric name	Example value	Description
m.freeram	3275588	Total amount of free RAM in bytes
m.hardware	OVMS WIFI BLE BT...	Base module hardware info
m.monotonic	49607Sec	Uptime in seconds
m.net.mdm.iccid	89490240001766080167	SIM ICCID
m.net.mdm.model	35316B09SIM5360E	Modem module hardware info
m.net.mdm.network	congstar	Current GSM network provider
m.net.mdm.sq	-101dBm	... and signal quality
m.net.provider	WLAN-214677	Current primary network provider
m.net.sq	-79dBm	... signal quality
m.net.type	wifi	... and type (none/modem/wifi)
m.net.wifi.network	WLAN-214677	Current Wifi network SSID
m.net.wifi.sq	-79.1dBm	... and signal quality
m.serial		Reserved for module serial no.
m.tasks	20	Task count (use <code>module tasks</code> to list)
m.time.utc	1572590910Sec	UTC time in seconds
m.version	3.2.005-155-g3133466f/...	Firmware version
m.egpio.input	0,1,2,3,4,5,6,7,9	EGPIO input port state (ports 0...9, present=high)
m.egpio.monitor	8,9	EGPIO input monitoring ports
m.egpio.output	4,5,6,7,9	EGPIO output port state
s.v2.connected	yes	yes = V2 (MP) server connected
s.v2.peers	1	V2 clients connected
s.v3.connected		yes = V3 (MQTT) server connected
s.v3.peers		V3 clients connected
v.b.12v.current	0A	Auxiliary 12V battery momentary current
v.b.12v.voltage	12.29V	Auxiliary 12V battery momentary voltage
v.b.12v.voltage.alert		yes = auxiliary battery under voltage alert
v.b.12v.voltage.ref	12.3V	Auxiliary 12V battery reference voltage
v.b.c.temp	13,13,...,13°C	Cell temperatures
v.b.c.temp.alert	0,0,...,0	Cell temperature deviation alert level [0=normal, 1=warning, 2=alert]
v.b.c.temp.dev.max	1.43,0.86,...,-1.29°C	Cell maximum temperature deviation observed
v.b.c.temp.max	19,18,...,17°C	Cell maximum temperatures
v.b.c.temp.min	13,12,...,12°C	Cell minimum temperatures
v.b.c.voltage	4.105,4.095,...,4.105V	Cell voltages
v.b.c.voltage.alert	0,0,...,0	Cell voltage deviation alert level [0=normal, 1=warning, 2=alert]
v.b.c.voltage.dev.max	0.0096,-0.0104,...,0.0125V	Cell maximum voltage deviation observed
v.b.c.voltage.max	4.135,4.125,...,4.14V	Cell maximum voltages
v.b.c.voltage.min	3.875,3.865,...,3.88V	Cell minimum voltages
v.b.cac	90.7796Ah	Calculated battery pack capacity
v.b.consumption	0Wh/km	Main battery momentary consumption
v.b.coulomb.recd	47.5386Ah	Main battery coulomb recovered on trip/charge
v.b.coulomb.recd.total	947.5386Ah	Main battery coulomb recovered total (life time)
v.b.coulomb.used	0.406013Ah	Main battery coulomb used on trip
v.b.coulomb.used.total	835.406013Ah	Main battery coulomb used total (life time)
v.b.current	0A	Main battery momentary current (output=positive)
v.b.energy.recd	2.69691kWh	Main battery energy recovered on trip/charge
v.b.energy.recd.total	3212.69691kWh	Main battery energy recovered total (life time)
v.b.energy.used	0.0209496kWh	Main battery energy used on trip
v.b.energy.used.total	3177.0209496kWh	Main battery energy used total (life time)
v.b.health		General textual description of battery health
v.b.p.level.avg	95.897%	Cell level - pack average

Continued on n



Table 1 – continued from previous page

Metric name	Example value	Description
v.b.p.level.max	96.41%	Cell level - strongest cell in pack
v.b.p.level.min	94.871%	Cell level - weakest cell in pack
v.b.p.level.stddev	0.548%	Cell level - pack standard deviation
v.b.p.temp.avg	13°C	Cell temperature - pack average
v.b.p.temp.max	13°C	Cell temperature - warmest cell in pack
v.b.p.temp.min	13°C	Cell temperature - coldest cell in pack
v.b.p.temp.stddev	0°C	Cell temperature - current standard deviation
v.b.p.temp.stddev.max	0.73°C	Cell temperature - maximum standard deviation observed
v.b.p.voltage.avg	4.1V	Cell voltage - pack average
v.b.p.voltage.grad	0.0032V	Cell voltage - gradient of current series
v.b.p.voltage.max	4.105V	Cell voltage - strongest cell in pack
v.b.p.voltage.min	4.09V	Cell voltage - weakest cell in pack
v.b.p.voltage.stddev	0.00535V	Cell voltage - current standard deviation
v.b.p.voltage.stddev.max	0.00783V	Cell voltage - maximum standard deviation observed
v.b.power	0kW	Main battery momentary power (output=positive)
v.b.range.est	99km	Estimated range
v.b.range.full	50.8km	Ideal range at 100% SOC & current conditions
v.b.range.ideal	48km	Ideal range
v.b.soc	96.3%	State of charge
v.b.soh	85%	State of health
v.b.temp	13°C	Main battery momentary temperature
v.b.voltage	57.4V	Main battery momentary voltage
v.c.12v.current	7.8A	Output current of DC/DC-converter
v.c.12v.power	123W	Output power of DC/DC-converter
v.c.12v.temp	34.5°C	Temperature of DC/DC-converter
v.c.12v.voltage	12.3V	Output voltage of DC/DC-converter
v.c.charging	no	yes = currently charging
v.c.climit	0A	Maximum charger output current
v.c.current	1.25A	Momentary charger output current
v.c.duration.full	25Min	Estimated time remaining for full charge
v.c.duration.range	-1Min	... for sufficient range
v.c.duration.soc	0Min	... for sufficient SOC
v.c.efficiency	87.6%	Momentary charger efficiency
v.c.kwh	2.6969kWh	Energy sum for running charge
v.c.kwh.grid	3.6969kWh	Energy drawn from grid during running session
v.c.kwh.grid.total	256.69kWh	Energy drawn from grid total (life time)
v.c.limit.range	0km	Sufficient range limit for current charge
v.c.limit.soc	80%	Sufficient SOC limit for current charge
v.c.mode	standard	standard, range, performance, storage
v.c.pilot	no	Pilot signal present
v.c.power	125kW	Momentary charger input power
v.c.state	done	charging, toff, done, prepare, timerwait, heating, stopped
v.c.substate		scheduledstop, scheduledstart, onrequest, timerwait, powerwait, stopped, in
v.c.temp	16°C	Charger temperature
v.c.time	0Sec	Duration of running charge
v.c.timermode		yes = timer enabled
v.c.timerstart		Time timer is due to start, seconds since midnight UTC
v.c.type		undefined, type1, type2, chademo, roadster, teslaus, supercharger, ccs
v.c.voltage	0V	Momentary charger supply voltage

Continued on n

Table 1 – continued from previous page

Metric name	Example value	Description
v.d.cp	yes	yes = Charge port open
v.d.fl		yes = Front left door open
v.d.fr		yes = Front right door open
v.d.hood		yes = Hood/frunk open
v.d.rl		yes = Rear left door open
v.d.rr		yes = Rear right door open
v.d.trunk		yes = Trunk open
v.e.alarm		yes = Alarm currently sounding
v.e.aux12v		yes = 12V auxiliary system is on (base system awake)
v.e.awake	no	yes = Vehicle is fully awake (switched on by the user)
v.e.c.config		yes = ECU/controller in configuration state
v.e.c.login		yes = Module logged in at ECU/controller
v.e.cabintemp	20°C	Cabin temperature
v.e.cabinfan	100%	Cabin fan
v.e.cabinsetpoint	24°C	Cabin set point
v.e.cabinintake	fresh	Cabin intake type (fresh, recirc, etc)
v.e.cabinvent	feet,face	Cabin vent type (comma-separated list of feet, face, screen, etc)
v.e.charging12v	no	yes = 12V battery is charging
v.e.cooling		yes = Cooling
v.e.drivemode	33882626	Active drive profile code (vehicle specific)
v.e.drivetime	0Sec	Seconds driving (turned on)
v.e.footbrake	0%	Brake pedal state [%]
v.e.gear		Gear/direction; negative=reverse, 0=neutral
v.e.handbrake		yes = Handbrake engaged
v.e.headlights		yes = Headlights on
v.e.heating		yes = Heating
v.e.hvac		yes = HVAC active
v.e.locked		yes = Vehicle locked
v.e.on	no	yes = Vehicle is in “ignition” state (drivable)
v.e.parktime	49608Sec	Seconds parking (turned off)
v.e.regenbrake		yes = Regenerative braking active
v.e.serv.range	12345km	Distance to next scheduled maintenance/service [km]
v.e.serv.time	1572590910Sec	Time of next scheduled maintenance/service [Seconds]
v.e.temp		Ambient temperature
v.e.throttle	0%	Drive pedal state [%]
v.e.valet		yes = Valet mode engaged
v.g.generating	no	True = currently delivering power
v.g.climit	0A	Maximum generator input current (from battery)
v.g.current	1.25A	Momentary generator input current (from battery)
v.g.duration.empty	25Min	Estimated time remaining for full discharge
v.g.duration.range	-1Min	... for range limit
v.g.duration.soc	0Min	... for SOC limit
v.g.efficiency	87.6%	Momentary generator efficiency
v.g.kwh	2.6969kWh	Energy sum generated in the running session
v.g.kwh.grid	3.6969kWh	Energy sent to grid during running session
v.g.kwh.grid.total	256.69kWh	Energy sent to grid total
v.g.limit.range	0km	Minimum range limit for generator mode
v.g.limit.soc	80%	Minimum SOC limit for generator mode
v.g.mode	standard	Generator mode (TBD)

Continued on n

Table 1 – continued from previous page

Metric name	Example value	Description
v.g.pilot	no	Pilot signal present
v.g.power	125kW	Momentary generator output power
v.g.state	done	Generator state (TBD)
v.g.substate		Generator substate (TBD)
v.g.temp	16°C	Generator temperature
v.g.time	0Sec	Duration of generator running
v.g.timermode	false	True if generator timer enabled
v.g.timerstart		Time generator is due to start
v.g.type		Connection type (chademo, ccs, ...)
v.g.voltage	0V	Momentary generator output voltage
v.i.temp		Inverter temperature
v.i.power	42.7kW	Momentary inverter motor power (output=positive)
v.i.efficiency	98.2%	Momentary inverter efficiency
v.m.rpm		Motor speed (RPM)
v.m.temp	0°C	Motor temperature
v.p.acceleration	0m/s <sup>2</sup>	Vehicle acceleration
v.p.altitude	327.8m	GPS altitude
v.p.direction	31.2°	GPS direction
v.p.gpsdop	1.3	GPS horizontal dilution of precision (smaller=better)
v.p.gpslock	no	yes = has GPS satellite lock
v.p.gpsmode	AA	<GPS><GLONASS>; N/A/D/E (None/Autonomous/Differential/Estimated)
v.p.gpsspeed	0km/h	GPS speed over ground
v.p.latitude	51.3023	GPS latitude
v.p.location	Home	Name of current location if defined
v.p.longitude	7.39006	GPS longitude
v.p.odometer	57913.1km	Vehicle odometer
v.p.satcount	8	GPS satellite count in view
v.p.speed	0km/h	Vehicle speed
v.p.trip	0km	Trip odometer
v.t.alert	0,0,0,1	TPMS tyre alert levels [0=normal, 1=warning, 2=alert]
v.t.health	95,93,96,74%	TPMS tyre health states
v.t.pressure	206.8,216.4,... kPa	TPMS tyre pressures
v.t.temp	33,33,34,38°C	TPMS tyre temperatures
v.type	RT	Vehicle type code
v.vin	VF1ACVYB012345678	Vehicle identification number

## 2.10 Over The Air (OTA) Updates

OVMS v3 includes 16MB flash storage. This is partition as:

```
4MB for factory application image (factory)
4MB for the first OTA application image (ota_0)
4MB for a second OTA application image (ota_1)
1MB for /store configuration and scripting storage
The remainder for bootloader, generic non-volatile storage, and other data
```

In general, the factory application firmware is stored in flash at the factory, during module production. That firmware is never changed on production modules, and is always kept as a failover backup.

That leaves two firmwares for Over The Air (OTA) updates. If the currently running firmware is the factory one, an OTA updated firmware can be written to either of the OTA partitions. If the current running firmware is ota\_0, then

any new OTA updates will be written to ota\_1 (and similarly if ota\_1 is currently running, then new OTA updates will be written to ota\_0). In this way, the currently running firmware is never modified and is always available as a failover backup.

You can check the status of OTA with the 'ota status' command:

```
OVMS# ota status
Firmware:      3.1.003-2-g7ea18b4-dirty/factory/main (build idf v3.1-dev-453-
→g0f978bcb Apr  7 2018 16:26:57)
Server Available: 3.1.003
Running partition: factory
Boot partition:  factory
```

That is showing the currently running firmware as a custom image *v3.1.003-2-g7ea18b4-dirty* running in *factory* partition. The running currently running partition is *factory* and the next time the system is booted, it will run from *factory* as well.

As a convenience, if there is currently active wifi connectivity, a network lookup will be performed and the currently available firmware version on the server will be shown. In this case, that is the standard *3.1.003* release (as shown in the 'Server Available:' line).

If we wanted to boot from ota\_1, we can do this with 'ota boot ota\_1':

```
OVMS# ota boot ota_1
Boot from ota_1 at 0x00810000 (size 0x00400000)

OVMS# ota status
Firmware:      3.1.003-2-g7ea18b4-dirty/factory/main (build idf v3.1-dev-453-
→g0f978bcb Apr  7 2018 16:26:57)
Server Available: 3.1.003
Running partition: factory
Boot partition:  ota_1
```

If the bootloader fails to boot from the specified OTA firmware, it will failover and boot from factory.

We can flash firmware to OTA either from a file on VFS (normally /sd), or over the Internet (via http). Let's try a simple OTA update over HTTP:

```
OVMS# ota flash http
Current running partition is: factory
Target partition is: ota_0
Download firmware from api.openvehicles.com/firmware/ota/v3.1/main/ovms3.bin to ota_0
Expected file size is 2100352
Preparing flash partition...
Downloading... (100361 bytes so far)
Downloading... (200369 bytes so far)
Downloading... (300577 bytes so far)
...
Downloading... (1903977 bytes so far)
Downloading... (2004185 bytes so far)
Download complete (at 2100352 bytes)
Setting boot partition...
OTA flash was successful
  Flashed 2100352 bytes from api.openvehicles.com/firmware/ota/v3.1/main/ovms3.bin
  Next boot will be from 'ota_0'

OVMS# ota status
Firmware:      3.1.003-2-g7ea18b4-dirty/factory/main (build idf v3.1-dev-453-
→g0f978bcb Apr  7 2018 16:26:57)
```

(continues on next page)

(continued from previous page)

```

Server Available: 3.1.003
Running partition: factory
Boot partition:   ota_0

```

Rebooting now (with ‘module reset’) would boot from the new ota\_0 partition firmware:

```

OVMS# ota status
Firmware:      3.1.003/ota_0/main (build idf v3.1-dev-453-g0f978bcb Apr  7 2018,
↳13:11:19)
Server Available: 3.1.003
Running partition: ota_0
Boot partition:   ota_0

```

## 2.11 Boot Status

OVMS maintains a record of the reason for each boot, in RAM that survives a reboot. It can show you how long the module has been running for, and the reason for the last reboot:

```

OVMS# boot status
Last boot was 2244 second(s) ago
This is reset #9 since last power cycle
Detected boot reason: SoftReset
Crash counters: 0 total, 0 early
CPU#0 boot reason was 12
CPU#1 boot reason was 12

```

If an unexpected (not ‘module reset’) reboot occurs within the first 10 seconds of startup (usually during the boot-time auto-loading of modules, scripts, etc), the crash counters are incremented. If those crash counters reach 5 (without a clean reset in between), then the auto-loading of modules is disabled for the 6th boot.

In case of a crash, the output will also contain additional debug information, i.e.:

```

Last crash: abort() was called on core 1
Backtrace:
0x40092ccc 0x40092ec7 0x400dbf1b 0x40176ca9 0x40176c6d 0x400eebd9 0x4013aed9
0x4013b538 0x40139c15 0x40139df9 0x4013a701 0x4013a731

```

If the module can access the V2 server after the crash reboot, it will also store this information along with the crash counters in the server table “\*-OVM-DebugCrash”, which will be kept on the server for 30 days.

Please include this info when sending a bug report, along with the output of “ota status” and – if available – any log files capturing the crash event (see Logging to SD CARD). If you can repeat the crash, please try to capture a log at “log level verbose”.

## 2.12 Events

Internally, OVMS raises events whenever significant events occur. An event is a lightweight message of a name plus optionally associated internal binary data. Event names are top-down structured (so can be filtered by prefix) and sufficient to identify the source and type of the event. Individual vehicle types may also issue their own events, and custom user events are also possible.

To **bind a script to an event**, save the script in directory `/store/events/<eventname>` (hint: directories can be created using the web UI editor). Whenever events are triggered, all the scripts in the corresponding `/store/events/<eventname>` directory are executed. Event scripts are executed in alphanumerical order of their names. Good practice is to prefix script names with 2-3 digit numbers in steps of 10 or 100 (i.e. first script named `50- . . .`), so new scripts can easily be integrated at a specific place. If the event script is written in Javascript, be sure to add the suffix `.js` to the name. Other names will be executed using the standard command interpreter.

If you want to **introduce a custom event** (e.g. for a plugin), prefix the event name by `usr.<pluginname>`, followed by the event purpose. *Example: Foglight*

Be aware **events are processed in series** from a queue, so depending on the system load and the list of registered event listeners, there may be some delay from event generation to e.g. a script execution.

### 2.12.1 Commands

- `event list [<key>]` – Show registered listeners for all or events matching a key (part of the name)
- `event trace <on|off>` – Enable/disable logging of events at the “info” level. Without tracing, events are also logged, but at the “debug” level. Ticker events are never logged.
- `event raise [-d<delay_ms>] <event>` – Manually raise an event, optionally with a delay. You can raise any event you like, but you shouldn’t raise system events without good knowledge of their effects.

### 2.12.2 Standard Events

Event	Data	Purpose
<code>app.connected</code>		One or more remote Apps have connected
<code>app.disconnected</code>		No remote Apps are currently connected
<code>canopen.node.emcy</code>	<code>&lt;event&gt;</code>	CANOpen node emergency received
<code>canopen.node.state</code>	<code>&lt;event&gt;</code>	CANOpen node state change received
<code>canopen.worker.start</code>	<code>&lt;worker&gt;</code>	CANOpen bus worker task started
<code>canopen.worker.stop</code>	<code>&lt;worker&gt;</code>	CANOpen bus worker task stopping
<code>clock.HHMM</code>		Per-minute local time, hour HH, minute MM
<code>clock.dayN</code>		Per-day local time, day N (0=Sun, 6=Sat)
<code>config.changed</code>		Configuration has changed
<code>config.mounted</code>		Configuration is mounted and available
<code>config.unmounted</code>		Configuration is unmounted and unavailable
<code>egpio.input.&lt;port&gt;.&lt;state&gt;</code>		EGPIO input port change (port=0..9, state=high/low)
<code>egpio.output.&lt;port&gt;.&lt;state&gt;</code>		EGPIO output port change (port=0..9, state=high/low)
<code>gps.lock.acquired</code>		GPS lock has been acquired
<code>gps.lock.lost</code>		GPS lock has been lost
<code>housekeeping.init</code>		Housekeeping has initialised
<code>location.alert.flatbed.moved</code>		GPS movement of parked vehicle detected
<code>location.enter.&lt;name&gt;</code>	<code>&lt;name&gt;</code>	The specified geolocation has been entered
<code>location.leave.&lt;name&gt;</code>	<code>&lt;name&gt;</code>	The specified geolcation has been left
<code>network.down</code>		All networks are down
<code>network.interface.change</code>		Network interface change detected
<code>network.interface.up</code>		Network connection is established
<code>network.mgr.init</code>		Network manager has initialised
<code>network.mgr.stop</code>		Network managed has been stopped
<code>network.modem.down</code>		Modem network is down
<code>network.modem.up</code>		Modem network is up

Continued on next page

Table 2 – continued from previous page

Event	Data	Purpose
network.reconfigured		Networking has been reconfigured
network.up		One or more networks are up
network.wifi.down		WIFI network is down
network.wifi.sta.bad		WIFI client has bad signal level
network.wifi.sta.good		WIFI client has good signal level
network.wifi.up		WIFI network is up
retools.cleared.all		RE frame log has been cleared
retools.cleared.changed		RE frame change flags cleared
retools.cleared.discovered		RE frame discovery flags cleared
retools.mode.analyse		RE switched to analysis mode
retools.mode.discover		RE switched to discovery mode
retools.started		RE (reverse engineering) toolkit started
retools.stopped		RE toolkit stopped
retools.pidscan.start		RE OBD2 PID scan started
retools.pidscan.stop		RE OBD2 PID scan stopped
retools.pidscan.done		RE OBD2 PID scan completed
sd.insert		The SD card has just been inserted
sd.mounted		The SD card is mounted and ready to use
sd.remove		The SD card has just been removed
sd.unmounted		The SD card has completed unmounting
sd.unmounting		The SD card is currently unmounting
server.v2.authenticating		V2 server connection is authenticating
server.v2.connected		V2 server connection established online
server.v2.connecting		V2 server connection in progress
server.v2.connectwait		V2 server is pausing before connection
server.v2.disconnected		V2 server connection has been lost
server.v2.stopped		V2 server has been stopped
server.v2.waitnetwork		V2 server connection is waiting for network
server.v2.waitreconnect		V2 server is pausing before re-connection
server.v3.authenticating		V3 server connection is authenticating
server.v3.connected		V3 server connection established online
server.v3.connecting		V3 server connection in progress
server.v3.connectwait		V3 server is pausing before connection
server.v3.disconnected		V3 server connection has been lost
server.v3.stopped		V3 server has been stopped
server.v3.waitnetwork		V3 server connection is waiting for network
server.v3.waitreconnect		V3 server is pausing before re-connection
server.web.socket.closed	<cnt>	Web server lost a websocket client
server.web.socket.opened	<cnt>	Web server has a new websocket client
system.modem.down		Modem has been disconnected
system.modem.gotgps		Modem GPS has obtained lock
system.modem.gotip		Modem received IP address from DATA
system.modem.lostgps		Modem GPS has lost lock
system.modem.muxstart		Modem MUX has started
system.modem.netdeepsleep		Modem is deep sleeping DATA network
system.modem.nethold		Modem is pausing DATA network
system.modem.netloss		Modem has lost DATA network
system.modem.netsleep		Modem is sleeping DATA network
system.modem.netstart		Modem is starting DATA network

Continued on next page

Table 2 – continued from previous page

Event	Data	Purpose
system.modem.netwait		Modem is pausing before starting DATA
system.modem.poweredon		Modem is powered on
system.modem.poweringon		Modem is powering on
system.modem.received.ussd	<ussd>	A USSD message has been received
system.modem.stop		Modem has been shut down
system.shutdown		System has been shut down
system.shuttingdown		System is shutting down
system.start		System is starting
system.vfs.file.changed	<path>	VFS file updated (note: only sent on some file changes)
system.wifi.ap.sta.connected		WiFi access point got a new client connection
system.wifi.ap.sta.disconnected		WiFi access point lost a client connection
system.wifi.ap.sta.ipassigned		WiFi access point assigned an IP address to a client
system.wifi.ap.start		WiFi access point mode starting
system.wifi.ap.stop		WiFi access point mode stopping
system.wifi.down		WiFi is shutting down
system.wifi.scan.done		WiFi scan has been finished
system.wifi.sta.connected		WiFi client is connected to a station
system.wifi.sta.disconnected		WiFi client has disconnected from a station
system.wifi.sta.gotip		WiFi client got an IP address
system.wifi.sta.lostip		WiFi client lost it's IP address
system.wifi.sta.start		WiFi client mode starting
ticker.1		One second has passed since last ticker
ticker.10		Ten seconds have passed
ticker.300		Five minutes have passed
ticker.3600		One hour has passed
ticker.60		One minute has passed
ticker.600		Ten minutes have passed
vehicle.alarm.off		Vehicle alarm has been disarmed
vehicle.alarm.on		Vehicle alarm has been armed
vehicle.alert.12v.off		12V system voltage has recovered
vehicle.alert.12v.on		12V system voltage is below alert threshold
vehicle.alert.bms		BMS cell/pack volts/temps exceeded thresholds
vehicle.asleep		Vehicle systems are asleep
vehicle.awake		Vehicle systems are awake
vehicle.aux.12v.on		Vehicle 12V auxiliary system is on (base system awake)
vehicle.aux.12v.off		Vehicle 12V auxiliary system is off
vehicle.charge.12v.start		Vehicle 12V battery is charging
vehicle.charge.12v.stop		Vehicle 12V battery has stopped charging
vehicle.charge.finished		Vehicle charge has completed normally
vehicle.charge.mode	<mode>	Vehicle charge mode has been set
vehicle.charge.pilot.off		Vehicle charge pilot signal is off
vehicle.charge.pilot.on		Vehicle charge pilot signal is on
vehicle.charge.prepare		Vehicle is preparing to charge
vehicle.charge.start		Vehicle has started to charge
vehicle.charge.state	<state>	Vehicle charge state has changed
vehicle.charge.stop		Vehicle has stopped charging
vehicle.charge.timermode.off		Vehicle charge timer mode has been switched off
vehicle.charge.timermode.on		Vehicle charge timer mode has been switched on
vehicle.headlights.off		Vehicle headlights are off

Continued on next page



Table 2 – continued from previous page

Event	Data	Purpose
vehicle.headlights.on		Vehicle headlights are on
vehicle.locked		Vehicle has been locked
vehicle.off		Vehicle has been switched off
vehicle.on		Vehicle has been switched on
vehicle.require.gps		A vehicle has indicated it requires GPS
vehicle.require.gpstime		A vehicle has indicated it requires GPS time
vehicle.type.cleared		Vehicle module has been unloaded
vehicle.type.set	<type>	Vehicle module has been loaded
vehicle.unlocked		Vehicle has been unlocked
vehicle.valet.off		Vehicle valet mode deactivated
vehicle.valet.on		Vehicle valet mode activated

## 2.13 Geofenced Locations

This section of the manual is still under development.

## 2.14 Notifications

Notifications can be simple text messages or transport structured data. To distinguish by their purpose and origin, notifications have a **type** and a **subtype**.

Notifications are sent by the module via the available communication **channels** (client/server connections). If a channel is temporarily down (e.g. due to a connection loss), the notifications for that channel will be kept in memory until the channel is available again. (Note: this message queue does not survive a crash or reboot of the module.)

Channels process notifications differently depending on the way they work. For example, a v2 server will forward text notifications as push messages to connected smart phones and email readers, a v3 server will publish them under an MQTT topic, and the webserver will display the message as a modal dialog. See the respective manual sections for details.

### Notification types currently defined:

- `info` – informational text messages
- `alert` – alert text messages
- `error` – error code messages
- `data` – historical data records, usually CSV formatted
- `stream` – live data streaming (high bandwidth), usually JSON formatted

The standard subtypes used are listed below, these can be used to filter messages. Vehicles may introduce custom notifications and replace standard notifications, see the respective user guide section for details.

Subtypes by convention are given in lower case, with dots “.” as structural separators.

### Notification channels currently defined:

- `ovmsv2` – server v2 connection
- `ovmsv3` – server v3 connection
- `ovmsweb` – websocket connections
- `pushover` – pushover text messaging service

Channels may have multiple active instances (“readers”), for example you can open multiple websocket connections. Channels may exclude notification types. Currently `stream` records are only supported on a websocket connection, all other types are supported on all channels.

Use `notify status` to see the currently registered channels (“readers”). Example:

```
OVMS# notify status
Notification system has 3 readers registered
  pushover(1): verbosity=1024
  ovmsv2(2): verbosity=1024
  ovmsweb(3): verbosity=65535
Notify types:
  alert: 0 entries
  data: 0 entries
  error: 0 entries
  info: 0 entries
  stream: 0 entries
```

The channel’s “verbosity” defines the supported maximum length of a textual notification message on that channel. Notification senders *should* honor this, but not all may do so. If messages exceed this limit, they may be truncated.

### 2.14.1 Sending notifications

You can send custom notifications from the shell or command scripts by using the `notify raise` command. The command can send the output of another command, an error code (implies type `error`), or any text you enter. For example, to send a custom text message, do:

```
OVMS# notify raise text info usr.anton.welcome "Hello, wonderful person!"
```

In this case, the type would be `info` and the subtype `usr.anton.welcome`. The type must match one of the defined types, the subtype can be chosen arbitrarily. Please use a unique `usr.` prefix for custom notifications to avoid collisions.

To send a battery status command result, do:

```
OVMS# notify raise command info battery.status "stat"
```

To send notifications from Duktape scripts, use the API call `OvmsNotify.Raise()`.

### 2.14.2 Suppress notifications

You can filter the channels to be used for notifications by their subtypes. By default, no subtypes are filtered on any channel, so all notifications are sent to all clients.

To disable (suppress) notifications, create a config entry based on the respective subtype, that lists the channels to include or exclude:

```
OVMS# config set notify <subtype> <channels>
```

**<channels> options are:**

- a) explicit inclusion: e.g. `ovmsv2,pushover` (only enable these)
- b) explicit exclusion: e.g. `*, -ovmsv3, -ovmsweb` (only disable these)
- c) `-` (dash) to disable all
- d) empty/`*` to enable all

**Example:** to disable the OTA update notifications on all channels, do:

```
OVMS# config set notify ota.update -
```

### 2.14.3 Standard notifications

Type	Subtype	Purpose / Content
alert	alarm.sounding	Vehicle alarm is sounding
alert	alarm.stopped	Vehicle alarm has stopped
alert	batt.12v.alert	12V Battery critical
alert	batt.12v.recovered	12V Battery restored
alert	batt.bms.alert	Battery pack/cell alert (critical voltage/temperature deviation)
alert	batt.soc.alert	Battery SOC critical
info	charge.done	stat on charge finished
info	charge.started	stat on start of charge
info	charge.stopped	stat on planned charge stop
alert	charge.stopped	stat on unplanned charge stop
data	debug.crash	Transmit crash backtraces (→ *-OVM-DebugCrash)
data	debug.tasks	Transmit task statistics (→ *-OVM-DebugTasks)
alert	flatbed.moved	Vehicle is being transported while parked - possible theft/flatbed
info	heating.started	stat on start of heating (battery)
data	log.grid	Grid (charge/generator) history log (see below) (→ *-LOG-Grid)
data	log.trip	Trip history log (see below) (→ *-LOG-Trip)
alert	modem.no_pincode	No PIN code for SIM card configured
alert	modem.wrongpincode	Wrong pin code
info	ota.update	New firmware available/downloaded/installed
info	pushover	Connection failure / message delivery response
stream	retools.list.update	RE toolkit CAN frame list update
stream	retools.status	RE toolkit general status update
info	valet.disabled	Valet mode disabled
info	valet.enabled	Valet mode enabled
alert	valet.hood	Vehicle hood opened while in valet mode
alert	valet.trunk	Vehicle trunk opened while in valet mode
alert	vehicle.idle	Vehicle is idling / stopped turned on

### 2.14.4 Grid history log

The grid history log can be used as a source for long term statistics on your charges and typical energy usages and to calculate your vehicle energy costs.

Log entries are created on each change of the charge or generator state (`v.c.state` / `v.g.state`).

You need to enable this log explicitly by configuring a storage time via config param `notify log.grid.storetime` (in days) or via the web configuration page. Set to 0/empty to disable the log. Already stored log entries will be kept on the server until expiry or manual deletion.

Note: the stability of the total energy counters included in this log depends on their source and persistence on the vehicle and/or module. If they are kept on the module, they may lose their values on a power outage.

- Notification subtype: `log.grid`
- History record type: `*-LOG-Grid`

- Format: CSV
- Archive time: `config notify log.grid.storetime (days)`
- Fields/columns:
  - `pos_gpslock`
  - `pos_latitude`
  - `pos_longitude`
  - `pos_altitude`
  - `pos_location`
  - `charge_type`
  - `charge_state`
  - `charge_substate`
  - `charge_mode`
  - `charge_climit`
  - `charge_limit_range`
  - `charge_limit_soc`
  - `gen_type`
  - `gen_state`
  - `gen_substate`
  - `gen_mode`
  - `gen_climit`
  - `gen_limit_range`
  - `gen_limit_soc`
  - `charge_time`
  - `charge_kwh`
  - `charge_kwh_grid`
  - `charge_kwh_grid_total`
  - `gen_time`
  - `gen_kwh`
  - `gen_kwh_grid`
  - `gen_kwh_grid_total`
  - `bat_soc`
  - `bat_range_est`
  - `bat_range_ideal`
  - `bat_range_full`
  - `bat_voltage`
  - `bat_temp`

- charge\_temp
- charge\_12v\_temp
- env\_temp
- env\_cabintemp
- bat\_soh
- bat\_health
- bat\_cac
- bat\_energy\_used\_total
- bat\_energy\_recd\_total
- bat\_coulomb\_used\_total
- bat\_coulomb\_recd\_total

### 2.14.5 Trip history log

The trip history log can be used as a source for long term statistics on your trips and typical trip power usages, as well as your battery performance in different environmental conditions and degradation over time.

Entries are created at the end of a trip (specifically *v.e.* on transition to off). Configure a minimum trip length for logging by the config variable `notify log.trip.minlength` or via the web UI.

You need to enable this log explicitly by configuring a storage time via config param `notify log.trip.storetime` (in days) or via the web configuration page. Set to 0/empty to disable the log. Already stored log entries will be kept on the server until expiry or manual deletion.

- Notification subtype: `log.trip`
- History record type: `*-LOG-Trip`
- Format: CSV
- Archive time: config `notify log.trip.storetime` (days)
- Fields/columns:
  - pos\_gpslock
  - pos\_latitude
  - pos\_longitude
  - pos\_altitude
  - pos\_location
  - pos\_odometer
  - pos\_trip
  - env\_drivetime
  - env\_drivemode
  - bat\_soc
  - bat\_range\_est
  - bat\_range\_ideal

- bat\_range\_full
- bat\_energy\_used
- bat\_energy\_recd
- bat\_coulomb\_used
- bat\_coulomb\_recd
- bat\_soh
- bat\_health
- bat\_cac
- bat\_energy\_used\_total
- bat\_energy\_recd\_total
- bat\_coulomb\_used\_total
- bat\_coulomb\_recd\_total
- env\_temp
- env\_cabintemp
- bat\_temp
- inv\_temp
- mot\_temp
- charge\_12v\_temp
- tpms\_temp\_min
- tpms\_temp\_max
- tpms\_pressure\_min
- tpms\_pressure\_max
- tpms\_health\_min
- tpms\_health\_max

## 2.15 Time

This section of the manual is still under development.

## 2.16 SSL/TLS

### 2.16.1 SSL/TLS Trusted Certificate Authorities

A default minimal list of trusted certificate authorities (CA) is provided with the firmware. you can see the current loaded list with the `tls trust list` command:

```
OVMS# tls trust list
AddTrust External CA Root length 1521
1521 byte certificate: AddTrust External CA Root
cert. version      : 3
serial number      : 01
issuer name        : C=SE, O=AddTrust AB, OU=AddTrust External TTP Network,
↳CN=AddTrust External CA Root
subject name       : C=SE, O=AddTrust AB, OU=AddTrust External TTP Network,
↳CN=AddTrust External CA Root
issued on         : 2000-05-30 10:48:38
expires on        : 2020-05-30 10:48:38
signed using       : RSA with SHA1
RSA key size       : 2048 bits
basic constraints   : CA=true
key usage          : Key Cert Sign, CRL Sign
```

If you want to add to this list, you can place the PEM formatted root CA certificate in the `/store/trustedca` directory on your config partition using the text editor. Then, reload the list with:

```
OVMS# tls trust reload
Reloading SSL/TLS trusted CA list
SSL/TLS has 4 trusted CAs, using 5511 bytes of memory
```

On boot, the trusted Certificate Authorities provided in firmware, and put in your `/store/trustedca` directory, will be automatically loaded and made available.

These trusted certificate authorities are used by the various module in the OVMS system, when establishing SSL/TLS connections (in order to verify the certificate of the server being connected to).

## 2.16.2 How to get the CA PEM for a Server

### Download from CA

If you know the CA, check their download section.

### Using a Browser

This only works for https servers:

1. Open the website you want to access
2. Open the encryption info (e.g. Chrome: lock icon at URL → show certificate)
3. Display the certificate chain (e.g. Chrome: tab “Details”, first element)
4. Select the last entry before the actual server certificate
5. Export the certificate in PEM format (usually the default)
6. Install the file contents as shown above

### Using OpenSSL or GNU TLS CLI

This works for all servers and ports:

```
openssl s_client -connect HOSTNAME:PORT -servername HOSTNAME -showcerts </dev/null \  
| sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
```

...or...:

```
gnutls-cli --print-cert --port PORT HOSTNAME </dev/null \  
| sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
```

Substitute HOSTNAME and PORT accordingly, e.g. https = port 443. The sed just strips the other info from the output and can be omitted to check for errors or details.

There should be two certificates in the output, look for BEGIN and END markers. The first one is the server certificate, the second one the CA certificate. Copy that second certificate into the editor, take care to include the BEGIN and END lines.

## 2.17 Scripting

### 2.17.1 Command Scripts

Lists of commands can be entered into a script file and stored in the VFS for execution (in the `/store/scripts` directory). These are called ‘command scripts’ and are simple sequential lists of OVMS commands. A command script can be executed with:

```
OVMS# . <script>  
OVMS# script run <script>
```

Command scripts can also be stored in the `/store/events/<eventname>` directory structure. Whenever events are triggered, all the scripts in the corresponding `/store/events/<eventname>` directory are executed. Event scripts are executed in alphanumerical order of their names. Good practice is to prefix script names with 2-3 digit numbers in steps of 10 or 100 (i.e. first script named 50- . . .), so new scripts can easily be integrated at a specific place.

Output of background scripts without console association (e.g. event scripts) will be sent to the log with tag `script` at “info” level.

Note that the developer building firmware can optionally set the `OVMS_DEV_SDCARDSSCRIPTS` build flag. If that is set, then the system will also check `/sd/scripts` and `/sd/events` for scripts. This should not be used for production builds, as you could hack the system just by plugging in an SD card.

In addition to command scripts, more sophisticated scripting capabilities may be enabled if the JavaScript environment is enabled in the build. This is discussed in the next section of this guide.

### 2.17.2 JavaScripting

OVMS v3 includes a powerful JavaScript engine. In addition to the standard, relatively fixed, firmware flashed to the module, JavaScripting can be used to dynamically load script code to run alongside the standard firmware. This javascript code can respond to system events, and perform background monitoring and other such tasks.

The simplest way of running javascript is to place a piece of javascript code in the `/store/scripts` directory, with the file extension `.js`. Then, the standard mechanism of running scripts can be employed:

```
OVMS# . <script.js>  
OVMS# script run <script.js>
```



Short javascript snippets can also be directly evaluated with:

```
OVMS# script eval <code>
```

Such javascript code can also be placed in the `/store/events/<eventname>` directories, to be automatically executed when the specified event is triggered. The script file name suffix must be `.js` to run the Javascript interpreter.

**Note:** The scripting engine used is [Duktape](#). Duktape supports [ECMAScript E5/E5.1](#) with some additions from later ECMAScript standards. Duktape does not emulate a browser environment, so you don't have window or document objects etc., just core Javascript plus the OVMS API and plugins.

Duktape builtin objects and functions: <https://duktape.org/guide.html#duktapebuiltins>

### 2.17.3 Persistent JavaScript

When a javascript script is executed, it is evaluated in the global javascript context. Care should be taken that local variables may pollute that context, so it is in general recommended that all JavaScript scripts are wrapped:

```
(function(){
  ... user code ...
})();
```

It is also possible to deliberately load functions, and other code, into the global context persistently, and have that code permanently available and running. When the JavaScript engine initialises, it automatically runs a special startup script:

```
/store/scripts/ovmsmain.js
```

That script can in turn include other code. If you make a change to such persistent code, and want to reload it, you can with:

```
OVMS# script reload
```

### 2.17.4 JavaScript Modules

The OVMS JavaScript engine supports the concept of modules (using the node.js style of exports). Such modules can be written like this:

```
exports.print = function(obj, ind) {
  var type = typeof obj;
  if (type == "object" && Array.isArray(obj)) type = "array";
  if (!ind) ind = '';

  switch (type) {
    case "string":
      print('"' + obj.replace(/\\/g, '\\\\') + '"');
      break;
    case "array":
      print('[\n');
      for (var i = 0; i < obj.length; i++) {
        print(ind + '  ');
        exports.print(obj[i], ind + '  ');
        if (i != obj.length-1) print(', ');
      }
  }
```

(continues on next page)

(continued from previous page)

```

        print('\n');
    }
    print(ind + ']');
    break;
case "object":
    print('{\n');
    var keys = Object.keys(obj);
    for (var i = 0; i < keys.length; i++) {
        print(ind + '  ' + keys[i] + ': ');
        exports.print(obj[keys[i]], ind + ' ');
        if (i != keys.length-1) print(', ');
        print('\n');
    }
    print(ind + '}');
    break;
default:
    print(obj);
}

if (ind == '') print('\n');
}

```

By convention, modules such as this are placed in the `/store/scripts/lib` directory as `<modulename>.js`. These modules can be loaded with:

```
JSON = require("lib/JSON");
```

And used as:

```
JSON.print(this);
```

To automatically load a custom module on startup, add the `MyPlugin = require("lib/MyPlugin");` line to `ovmsmain.js`.

There are a number of **internal modules** already provided with the firmware, and by convention these are provided under the `int/<modulename>` namespace. The above JSON module is, for example, provided as `int/JSON` and automatically loaded into the global context. These internal modules can be directly used (so `JSON.print(this)` works directly).

## 2.17.5 Testing JavaScript / Modules

Use the **editor** (see Tools menu) to test or evaluate arbitrary Javascript code. This can be done on the fly, i.e. without saving the code to a file first. Think of it as a server side Javascript shell.

**Testing modules** normally involves reloading the engine, as the `require()` call caches all loaded modules until restart. To avoid this during module development, use the following template code. This mimics the `require()` call without caching and allows to do tests within the same evaluation run:

```

// Load module:
mymodule = (function() {
    exports = {};

    // ... insert module code here ...

    return exports;

```

(continues on next page)

(continued from previous page)

```

}) ();

// Module API tests:
mymodule.myfunction1();
JSON.print(mymodule.myfunction2());

```

As the module is actually loaded into the global context this way just like using `require()`, anything else using the module API (e.g. a web plugin) will also work after evaluation.

## 2.17.6 Internal Objects and Functions/Methods

A number of OVMS objects have been exposed to the JavaScript engine, and are available for use by custom scripts via the global context.

The global context is the analog to the `window` object in a browser context, it can be referenced explicitly as `this` on the JavaScript toplevel or as `globalThis` from any context.

You can see the global context objects, methods, functions and modules with the `JSON.print(this)` method:

```

OVMS# script eval 'JSON.print(this)'
{
  "performance": {
    "now": function now() { [native code] }
  },
  "assert": function () { [native code] },
  "print": function () { [native code] },
  "OvmsCommand": {
    "Exec": function Exec() { [native code] }
  },
  "OvmsConfig": {
    "Delete": function Delete() { [native code] },
    "Get": function Get() { [native code] },
    "Instances": function Instances() { [native code] },
    "Params": function Params() { [native code] },
    "Set": function Set() { [native code] }
  },
  "OvmsEvents": {
    "Raise": function Raise() { [native code] }
  },
  "OvmsLocation": {
    "Status": function Status() { [native code] }
  },
  "OvmsMetrics": {
    "AsFloat": function AsFloat() { [native code] },
    "AsJSON": function AsJSON() { [native code] },
    "Value": function Value() { [native code] }
  },
  "OvmsNotify": {
    "Raise": function Raise() { [native code] }
  },
  "OvmsVehicle": {
    "ClimateControl": function ClimateControl() { [native code] },
    "Homelink": function Homelink() { [native code] },
    "Lock": function Lock() { [native code] },
    "SetChargeCurrent": function SetChargeCurrent() { [native code] },
    "SetChargeMode": function SetChargeMode() { [native code] },

```

(continues on next page)

(continued from previous page)

```

    "SetChargeTimer": function SetChargeTimer() { [native code] },
    "StartCharge": function StartCharge() { [native code] },
    "StartCooldown": function StartCooldown() { [native code] },
    "StopCharge": function StopCharge() { [native code] },
    "StopCooldown": function StopCooldown() { [native code] },
    "Type": function Type() { [native code] },
    "Unlock": function Unlock() { [native code] },
    "Unvalet": function Unvalet() { [native code] },
    "Valet": function Valet() { [native code] },
    "Wakeup": function Wakeup() { [native code] }
  },
  "JSON": {
    "format": function () { [ecmascript code] },
    "print": function () { [ecmascript code] }
  },
  "PubSub": {
    "publish": function () { [ecmascript code] },
    "subscribe": function () { [ecmascript code] },
    "clearAllSubscriptions": function () { [ecmascript code] },
    "clearSubscriptions": function () { [ecmascript code] },
    "unsubscribe": function () { [ecmascript code] }
  }
}

```

## Global Context

- **assert(condition, message)** Assert that the given condition is true. If not, raise a JavaScript exception error with the given message.
- **print(string)** Print the given string on the current terminal. If no terminal (for example a background script) then print to the system console as an informational message.
- **performance.now()** Returns monotonic time since boot in milliseconds, with microsecond resolution.

## JSON

The JSON module extends the native builtin `JSON.stringify` and `JSON.parse` methods by a `format` and a `print` method, to format and/or print out a given javascript object in JSON format. Both by default insert spacing and indentation for readability and accept an optional `false` as a second parameter to produce a compact version for transmission.

- **JSON.print(data)** Output data (any Javascript data) as JSON, readable
- **JSON.print(data, false)** ...compact (without spacing/indentation)
- **str = JSON.format(data)** Format data as JSON string, readable
- **str = JSON.format(data, false)** ...compact (without spacing/indentation)
- **JSON.stringify(value[, replacer[, space]])** see [MDN JSON/stringify](#)
- **JSON.parse(text[, reviver])** see [MDN JSON/parse](#)

**Note:** The JSON module is provided for compatibility with standard Javascript object dumps and for readability. If performance is an issue, consider using the Duktape native builtins `JSON.stringify()` / `Duktape.enc()` and `JSON.parse()` / `Duktape.dec()` (see Duktape builtins and [Duktape JSON](#) for explanations of these).

For example, `Duktape.enc('JC', data)` is equivalent to `JSON.format(data, false)` except for the representation of functions. Using the JX encoding will omit unnecessary quotations.

## HTTP

The HTTP API provides asynchronous GET & POST requests for HTTP and HTTPS. Requests can return text and binary data and follow 301/302 redirects automatically. Basic authentication is supported (add username & password to the URL), digest authentication is not yet implemented.

The handler automatically excludes the request objects from garbage collection until finished (success/failure), so you don't need to store a global reference to the request.

- **`req = HTTP.Request(cfg)`** Perform asynchronous HTTP/HTTPS GET or POST request.

Pass the request parameters using the `cfg` object:

- `url`: standard URL/URI syntax, optionally including user auth and query string
- `post`: optional POST data, set to an empty string to force a POST request. Note: you need to provide this in encoded form. If no `Content-Type` header is given, it will default to `x-www-form-urlencoded`.
- `headers`: optional array of objects containing key-value pairs of request headers. Note: `User-Agent` will be set to the standard OVMS user agent if not present here.
- `timeout`: optional timeout in milliseconds, default: 120 seconds.
- `binary`: optional flag: `true` = perform a binary request (see response object).
- `done`: optional success callback function, called with the response object as argument, with `this` pointing to the request object.
- `fail`: optional error callback function, called with the error string as argument, with `this` pointing to the request object.
- `always`: optional final callback function, no arguments, `this` = request object.

The `cfg` object is extended and returned by the API (`req`). It will remain stable at least until the request has finished and callbacks have been executed. On completion, the `req` object may contain an updated `url` and a `redirectCount` if redirects have been followed. Member `error` (also passed to the `fail` callback) will be set to the error description if an error occurred. The `always` callback if present is called in any case, after a `done` or `fail` callback has been executed. Check `this.error` in the `always` callback to know if an error occurred.

On success, member object `response` will be present and contain:

- `statusCode`: the numerical HTTP Status response code
- `statusText`: the HTTP Status response text
- `headers`: array of response headers, each represented by an object { `<name>`: `<value>` }
- `body`: only for text requests: response body as a standard string
- `data`: only for binary requests: response body as a `Uint8Array`

Notes: any HTTP response from the server is considered success, check `response.statusCode` for server specific errors. Callbacks are executed without an output channel, so all `print` outputs will be written to the system log. Hint: use `JSON.print(this, false)` in the callback to get a debug log dump of the request.

**Examples:**

```
// simple POST, ignore all results:
HTTP.Request({ url: "http://smartplug.local/switch", post: "state=on&when=now
↪ " });

// fetch and inspect a JSON object:
HTTP.Request({
  url: "http://solarcontroller.local/status?fmt=json",
  done: function(resp) {
    if (resp.statusCode == 200) {
      var status = JSON.parse(resp.body);
      if (status["power"] > 5000)
        OvmsVehicle.StartCharge();
      else if (status["power"] < 3000)
        OvmsVehicle.StopCharge();
    }
  }
});

// override user agent, log completed request object:
HTTP.Request({
  url: "https://dexters-web.de/f/test.json",
  headers: [{ "User-Agent": "Mr. What Zit Tooya" }],
  always: function() { JSON.print(this, false); }
});
```

- **HTTP.request()** Legacy alias for `HTTP.Request()`, please do not use.

---

**Note:** **SSL requests (https)** can take up to 12 seconds on an idle module. SSL errors also may not reflect the actual error, for example an empty server response with code 400 may be reported as a general “SSL error”. If you get “SSL error” on a valid request, you may need to install a custom root CA certificate; see [SSL/TLS](#).

---

## VFS

The VFS API provides asynchronous loading and saving of files on `/store` and `/sd`. Text and binary data is supported. Currently only complete files can be loaded, the saver supports an append mode. In any case, the data to save/load needs to fit into RAM twice, as the buffer needs to be converted to/from Javascript.

The handler automatically excludes the request objects from garbage collection until finished (success/failure), so you don’t need to store a global reference to the request.

Loading or saving protected paths (`/store/ovms_config/...`) is not allowed. Saving to a path automatically creates missing directories.

See [AuxBatMon: 12V History Chart](#) for a complete application usage example.

- **req = VFS.Load(cfg)** Perform asynchronous file load.

Pass the request parameters using the `cfg` object:

- `path`: full file path, e.g. `/sd/mydata/telemetry.json`
- `binary`: optional flag: `true` = perform a binary request, returned data will be an `Uint8Array`
- `done`: optional success callback function, called with the data content read as the single argument, `this` pointing to the request object
- `fail`: optional error callback function, called with the error string as argument, with `this` pointing to the request object

- `always`: optional final callback function, no arguments, `this` = request object

The `cfg` object is extended and returned by the API (`req`). It will remain stable at least until the request has finished and callbacks have been executed. On success, the `req` object contains a `data` property (also passed to the `done` callback), which is either a string (text mode) or a `Uint8Array` (binary mode).

Member `error` (also passed to the `fail` callback) will be set to the error description if an error occurred. The `always` callback if present is called in any case, after a `done` or `fail` callback has been executed. Check `this.error` in the `always` callback to know if an error occurred.

#### Example:

```
// Load a custom telemetry object from a JSON file on SD card:
var telemetry;
VFS.Load({
  path: "/sd/mydata/telemetry.json",
  done: function(data) {
    telemetry = Duktape.dec('jx', data);
    // ...process telemetry...
  },
  fail: function(error) {
    print("Error loading telemetry: " + error);
  }
});
```

- **`req = VFS.Save(cfg)`** Perform asynchronous file save.

Pass the request parameters using the `cfg` object:

- `data`: the string or `Uint8Array` to save
- `path`: full file path (missing directories will automatically be created)
- `append`: optional flag: `true` = append to the end of the file (also creating the file as necessary)
- `done`: optional success callback function, called with no arguments, `this` pointing to the request object
- `fail`: optional error callback function, called with the error string as argument, with `this` pointing to the request object
- `always`: optional final callback function, no arguments, `this` = request object

The `cfg` object is extended and returned by the API (`req`). It will remain stable at least until the request has finished and callbacks have been executed.

Member `error` (also passed to the `fail` callback) will be set to the error description if an error occurred. The `always` callback if present is called in any case, after a `done` or `fail` callback has been executed. Check `this.error` in the `always` callback to know if an error occurred.

#### Example:

```
// Save the above telemetry object in JSON format on SD card:
VFS.Save({
  path: "/sd/mydata/telemetry.json",
  data: Duktape.enc('jx', telemetry),
  fail: function(error) {
    print("Error saving telemetry: " + error);
  }
});
```

### **Warning: File I/O, especially saving, can cause short freezes of the module!**

Minimize save frequency and, if possible, avoid saving while the vehicle is in operation (driving / charging), by using a check like:

```
// Saving to VFS may cause short blockings, so only allow when vehicle is off:
function allowSave() {
    return !OvmsMetrics.Value("v.e.on") && !OvmsMetrics.Value("v.c.charging");
}
```

## PubSub

The PubSub module provides access to a Publish-Subscribe framework. In particular, this framework is used to deliver events to the persistent JavaScript framework in a high performance flexible manner. An example script to print out the ticker.10 event is:

```
var myTicker=function(msg,data){ print("Event: "+msg+"\n"); };
PubSub.subscribe("ticker.10",myTicker);
```

The above example created a function `myTicker` in global context, to print out the provided event name. Then, the `PubSub.subscribe` module method is used to subscribe to the `ticker.10` event and have it call `myTicker` every ten seconds. The result is “Event: ticker.10” printed once every ten seconds.

- **id = PubSub.subscribe(topic, handler)** Subscribe the function `handler` to messages of the given topic. Note that types are not limited to OVMS events. The method returns an `id` to be used to unsubscribe the handler.
- **PubSub.publish(topic, [data])** Publish a message of the given topic. All subscribed handlers will be called with the topic and data as arguments. `data` can be any Javascript data.
- **PubSub.unsubscribe(id | handler | topic)** Cancel a specific subscription, all subscriptions of a specific handler or all subscriptions to a topic.

## OvmsCommand

- **str = OvmsCommand.Exec(command)** The `OvmsCommand` object exposes one method “Exec”. This method is passed a single parameter as the command to be executed, runs that command, and then returns the textual output of the command as a string. For example:

```
print(OvmsCommand.Exec("boot status"));
Last boot was 14 second(s) ago
This is reset #0 since last power cycle
Detected boot reason: PowerOn (1/14)
Crash counters: 0 total, 0 early
```

## OvmsConfig

- **array = OvmsConfig.Params()** Returns the list of available configuration parameters.
- **array = OvmsConfig.Instances(param)** Returns the list of instances for a specific parameter.
- **string = OvmsConfig.Get(param, instance, default)** Returns the specified parameter/instance value.



- **object = OvmsConfig.GetValues(param, [prefix])** Gets all param instances matching the optional prefix with their associated values. If a prefix is given, the returned property names will have the prefix removed. Note: all values are returned as strings, you need to convert them as needed.
- **OvmsConfig.Set(param, instance, value)** Sets the specified parameter/instance value.
- **OvmsConfig.SetValues(param, prefix, object)** Sets all properties of the given object as param instances after adding the prefix. Note: non-string property values will be converted to their string representation.
- **OvmsConfig.Delete(param, instance)** Deletes the specified parameter instance.

Beginning with firmware release 3.2.009, a dedicated configuration parameter `usr` is provided for plugins. You can add new config instances simply by setting them, for example by `OvmsConfig.Set("usr", "myplugin.level", 123)` or by the `config set` command.

Read plugin configuration example:

```
// Set default configuration:
var cfg = { level: 100, enabled: "no" };

// Read user configuration:
Object.assign(cfg, OvmsConfig.GetValues("usr", "myplugin.));

if (cfg["enabled"] == "yes") {
  print("I'm enabled at level " + Number(cfg["level"]));
}
```

Keep in mind to prefix all newly introduced instances by a unique plugin name, so your plugin can nicely coexist with others.

## OvmsEvents

This provides access to the OVMS event system. While you may raise system events, the primary use is to raise custom events. Sending custom events is a lightweight method to inform the web UI (or other plugins) about simple state changes. Use the prefix `usr.` on custom event names to prevent conflicts with later framework additions.

Another use is the emulation of the `setTimeout()` and `setInterval()` browser methods by subscribing to a delayed event. Pattern:

```
function myTimeoutHandler() {
  // raise the timeout event again here to emulate setInterval()
}
PubSub.subscribe('usr.myplugin.timeout', myTimeoutHandler);

// start timeout:
OvmsEvents.Raise('usr.myplugin.timeout', 1500);
```

- **OvmsEvents.Raise(event, [delay\_ms])** Signal the event, optionally with a delay (milliseconds, must be given as a number). Delays are handled by the event system, the method call returns immediately.

## OvmsLocation

- **isatlocation = OvmsLocation.Status(location)** Check if the vehicle is currently in a location's geofence (pass the location name as defined). Returns `true` or `false`, or `undefined` if the location name passed is not valid.

Note: to get the actual GPS coordinates, simply read metrics `v.p.latitude`, `v.p.longitude` and `v.p.altitude`.

### OvmsMetrics

- **str = OvmsMetrics.Value(metricname [,decode])** Returns the typed value (default) or string representation (with `decode = false`) of the metric value.
- **num = OvmsMetrics.AsFloat(metricname)** Returns the float representation of the metric value.
- **str = OvmsMetrics.AsJSON(metricname)** Returns the JSON representation of the metric value.
- **obj = OvmsMetrics.GetValues([filter] [,decode])** Returns an object of all metrics matching the optional name filter/template (see below), by default decoded into Javascript types (i.e. numerical values will be JS numbers, arrays will be JS arrays etc.). The object returned is a snapshot, the values won't be updated.

The `filter` argument may be a string (for substring matching as with `metrics list`), an array of full metric names, or an object of which the property names are used as the metric names to get. The object won't be changed by the call, see `Object.assign()` for a simple way to merge objects. Passing an object is especially convenient if you already have an object to collect metrics data.

The `decode` argument defaults to `true`, pass `false` to retrieve the metrics string representations instead of typed values.

With the introduction of the `OvmsMetrics.GetValues()` call, you can get multiple metrics at once and let the system decode them for you. Using this you can for example do:

```
// Get all metrics matching substring "v.b.c." (vehicle battery cell):
var metrics = OvmsMetrics.GetValues("v.b.c.");
print("Temperature of cell 3: " + metrics["v.b.c.temp"][2] + " °C\n");
print("Voltage of cell 7: " + metrics["v.b.c.voltage"][6] + " V\n");

// Get some specific metrics:
var ovmsinfo = OvmsMetrics.GetValues(["m.version", "m.hardware"]);
JSON.print(ovmsinfo);
```

This obsoletes the old pattern of parsing a metric's JSON representation using `eval()`, `JSON.parse()` or `Duktape.dec()` you may still find in some plugins. Example:

```
var celltemps = eval(OvmsMetrics.AsJSON("v.b.c.temp"));
print("Temperature of cell 3: " + celltemps[2] + " °C\n");
```

**Warning:** Never use `eval()` on unsafe data, e.g. user input! `eval()` executes arbitrary Javascript, so can be exploited for code injection attacks.

### OvmsNotify

- **id = OvmsNotify.Raise(type, subtype, message)** Send a notification of the given type and subtype with message as contents. Returns the message id allocated or 0 in case of failure. Examples:

```
// send an info notification to the user:
OvmsNotify.Raise("info", "usr.myplugin.status", "Alive and kicking!");
```

(continues on next page)

(continued from previous page)

```
// send a JSON stream to a web plugin:
OvmsNotify.Raise("stream", "usr.myplugin.update", JSON.format(streamdata,
↪false));

// send a CSV data record to a server:
OvmsNotify.Raise("data", "usr.myplugin.record", "*-MyStatus,0,86400,Alive");
```

## OvmsVehicle

The OvmsVehicle object is the most comprehensive, and exposes several methods to access the current vehicle. These include:

- **str = OvmsVehicle.Type()** Return the type of the currently loaded vehicle module
- **success = OvmsVehicle.Wakeup()** Wakeup the vehicle (return TRUE if successful)
- **success = OvmsVehicle.Homelink(button, durationms)** Fire the given homelink button
- **success = OvmsVehicle.ClimateControl(onoff)** Turn on/off climate control
- **success = OvmsVehicle.Lock(pin)** Lock the vehicle
- **success = OvmsVehicle.Unlock(pin)** Unlock the vehicle
- **success = OvmsVehicle.Valet(pin)** Activate valet mode
- **success = OvmsVehicle.Unvalet(pin)** Deactivate valet mode
- **success = OvmsVehicle.SetChargeMode(mode)** Set the charge mode (“standard” / “storage” / “range” / “performance”)
- **success = OvmsVehicle.SetChargeCurrent(limit)** Set the charge current limit (in amps)
- **success = OvmsVehicle.SetChargeTimer(onoff, start)** Set the charge timer
- **success = OvmsVehicle.StartCharge()** Start the charge
- **success = OvmsVehicle.StopCharge()** Stop the charge
- **success = OvmsVehicle.StartCooldown()** Start a cooldown charge
- **success = OvmsVehicle.StopCooldown()** Stop the cooldown charge

## 2.17.7 Test Utilities

You can use the web UI editor and shell to edit, upload and test script files. If you need many test cycles, a convenient alternative is to use shell scripts to automate the process.

If you’ve configured ssh public key authentication, you can simply use `scp` to upload scripts and `ssh` to execute commands:

```
#!/bin/bash
# Upload & execute a script file:

FILE="test.js"
PATH="/store/scripts/"

OVMS_HOST="yourovms.local"
```

(continues on next page)

(continued from previous page)

```

SCP="/usr/bin/scp -q"
SSH="/usr/bin/ssh"

# Upload:
$SCP "${FILE}" "${OVMS_HOST}:${PATH}${FILE}"

# Execute:
$SSH "${OVMS_HOST}" "script run ${FILE}"

```

Customize to your needs. If you want to test a plugin, simply replace the `script run` command by `script reload` followed by some `script eval` calls to your plugin API.

Note: this may be slow, as the `ssh` session needs to be negotiated for every command.

A faster option is using the OVMS HTTP REST API. The following script uses `curl` to upload and execute a script:

```

#!/bin/bash
# Upload & execute a script file:

FILE="test.js"
PATH="/store/scripts/"

OVMS_HOST="http://yourovms.local"
OVMS_USER="admin"
OVMS_PASS="yourpassword"

CURL="/usr/bin/curl -c .auth -b .auth"
SED="/usr/bin/sed"
DATE="/usr/bin/date"

# Login?
if [[ -e ".auth" ]] ; then
    AUTHAGE=$((($DATE +%s) - $($DATE +%s -r ".auth"))
else
    AUTHAGE=3600
fi
if [[ "$AUTHAGE" -ge 3600 ]] ; then
    RES=$((CURL "${OVMS_HOST}/login" --data-urlencode "username=${OVMS_USER}" --data-
    ↪urlencode "password=${OVMS_PASS}" 2>/dev/null)
    if [[ "$RES" =~ "Error" ]] ; then
        echo -n "LOGIN ERROR: "
        echo $RES | $SED -e 's:.*<li>\([^<]*\).*:\1:g'
        rm .auth
        exit 1
    fi
fi

# Upload:
RES=$((CURL "${OVMS_HOST}/edit" --data-urlencode "path=${PATH}${FILE}" --data-
    ↪urlencode "content@${FILE}" 2>/dev/null)
if [[ "$RES" =~ "Error" ]] ; then
    echo -n "UPLOAD ERROR: "
    echo $RES | $SED -e 's:.*<li>\([^<]*\).*:\1:g'
    rm .auth
    exit 1
fi

```

(continues on next page)

(continued from previous page)

```
# Execute:
$CURL "${OVMS_HOST}/api/execute" --data-urlencode "command=script run ${FILE}"
```

## 2.18 Factory Reset

### 2.18.1 Module Configuration

A standard factory reset erases the configuration store. After a factory reset, you will be able to access the USB console with an empty module password and the “OVMS” wifi access point with the initial password “OVMSinit”. We recommend using the setup wizard to configure the module or restoring a configuration backup as soon as possible, as the module is accessible by anyone knowing the initial password.

The standard factory reset does not revert OTA firmware installs. See below for methods to switch back to and optionally replace the factory firmware.

#### Method 1: Command

If you have console command access, a factory reset can be accomplished with this command:

```
OVMS# module factory reset
Reset configuration store to factory defaults, and lose all configuration data (y/n):_
↪y
Store partition is at 00c10000 size 00100000
Unmounting configuration store...
Erasing 1048576 bytes of flash...
Factory reset of configuration store complete and reboot now...
```

That command will erase all configuration store, and reboot to an empty configuration.

Note: to issue this command from the web shell or a remote shell (App, Server, ...), you need to skip the confirmation step by adding the option `-noconfirm`, i.e.:

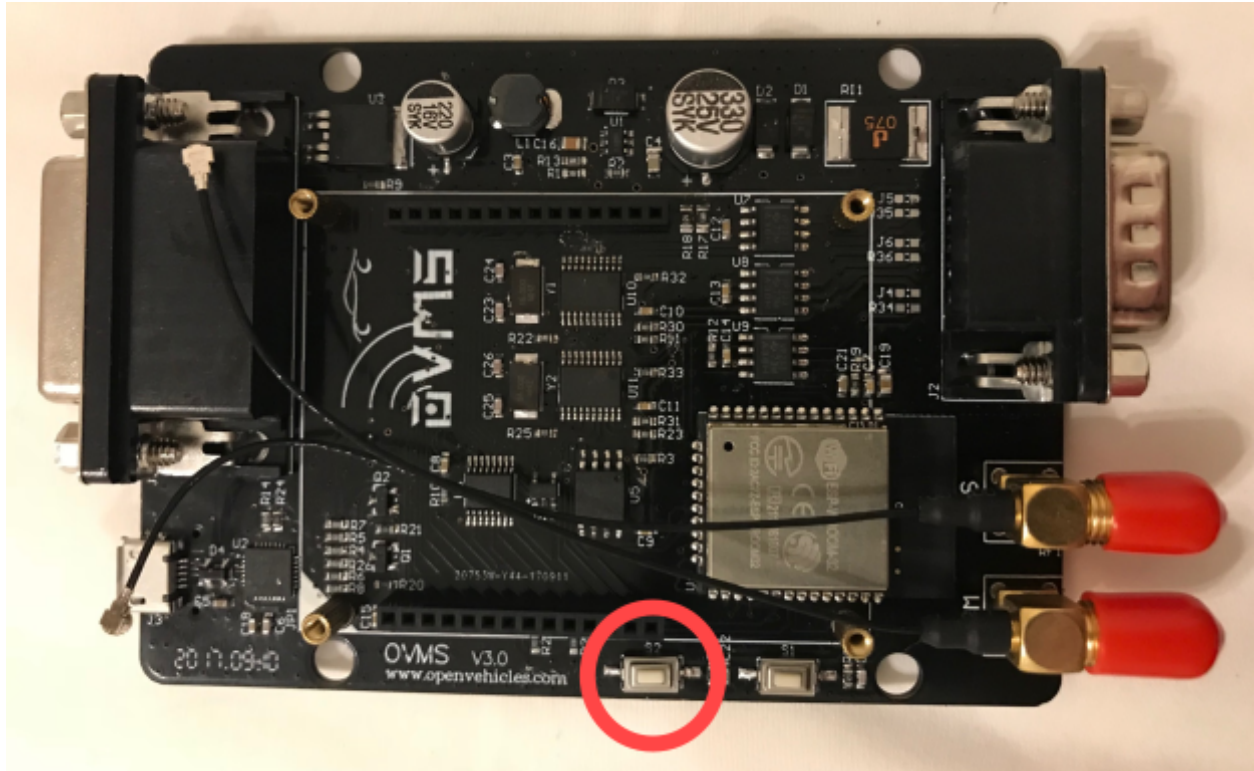
```
OVMS# module factory reset -noconfirm
```

#### Method 2: SD card

If you don't have console access, you can perform a factory reset by placing an empty file named `factoryreset.txt` in the root directory of an SD card and insert that SD into the (running) module. The file will be deleted and the module will reboot within about 30 seconds.

#### Method 3: Switch S2

You can also open the module case, remove any SD card (important!), power on the module, wait 10 seconds, then push and hold switch “S2” for 10 seconds. “S2” is located here:



#### Method 4: USB

If you don't have console access and don't have an SD card, you can perform a factory reset from a PC via USB using the `esptool.py` tool from the Espressif ESP-IDF toolkit (see below for installation):

```
esptool.py \
  --chip esp32 --port /dev/tty.SLAB_USBtoUART --baud 921600 \
  erase_region 0xC10000 0x100000
```

Note: the port needs to be changed to the one assigned by your system, i.e. `/dev/ttyUSB0` on a Linux system or `COMx` on Windows. After using `esptool.py` to manually erase the config region, you should go into the console and do the module factory reset step to properly factory reset.

---

**Note:** Methods 1, 2 and 3 need a running system, i.e. will not work if your module cannot boot normally. In this case first try method 4. If that doesn't help also switch back to the factory firmware as shown below.

---

### 2.18.2 Module Factory Firmware

You can switch back to factory firmware with this command:

```
OVMS# ota boot factory
Boot from factory at 0x00010000 (size 0x00400000)
```

Or, without console access (lost module password), using the `esptool.py` from the Espressif ESP-IDF toolkit:

```
esptool.py \
  --chip esp32 --port /dev/tty.SLAB_USBtoUART --baud 921600 \
  erase_region 0xd000 0x2000
```

Note: the device needs to be changed to the one assigned by your system, i.e. `/dev/ttyUSB0` on a Linux system or `COMx` on Windows.

### 2.18.3 Flash Factory Firmware via USB

`esptool.py` can also be used to flash a new factory firmware. Download the firmware file `ovms3.bin` you want to flash, then issue:

```
esptool.py \
  --chip esp32 --port /dev/tty.SLAB_USBtoUART --baud 921600 \
  --before "default_reset" --after "hard_reset" \
  write_flash --compress --flash_mode "dio" --flash_freq "40m" --flash_size detect \
  0x10000 ovms3.bin
```

Note: if you were running an OTA partition before, you also need to switch back to the factory partition as shown above.

### 2.18.4 Full Reflash via USB

If you accidentally did an `erase_flash` or erased the wrong region, you will need to do a full reflash of your module (including the boot loader and partitioning scheme).

The need for a full reflash will typically show by the USB output of the module boot being just something like:

```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x3f (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57
```

To do a full reflash, download the three `.bin` files from the release you want to flash, e.g. from

<https://ovms.dexters-web.de/firmware/ota/v3.2/edge/>

Then issue:

```
esptool.py \
  --chip esp32 --port /dev/tty.SLAB_USBtoUART --baud 921600 \
  --before "default_reset" --after "hard_reset" \
  write_flash --compress --flash_mode "dio" --flash_freq "40m" --flash_size detect \
  0x1000 bootloader.bin 0x10000 ovms3.bin 0x8000 partitions.bin
```

...replacing the port and file paths accordingly for your system.

If this fails, open a support ticket on <https://www.openvehicles.com> and attach a log of the boot process, or install the developer environment and do a `make flash`.

### 2.18.5 Installing esptool.py

The `esptool.py` package and installation instructions can be found here:

<https://github.com/espressif/esptool>

The package normally can be installed without manual download using the python package manager “pip”, i.e. on Unix/Linux:

```
sudo pip install esptool
```

**Warning:** You can brick your module using the esptool. Only use the commands shown above.

## 2.19 OBDII ECU

### 2.19.1 Purpose

The OBDII interface is a connector, electrical specification, and protocol used for viewing the operation and status of a car. It is mandated in all light duty vehicles sold in North America beginning in 1996, and while the focus of the information it provides is for monitoring the emissions of Internal Combustion Engines, it has proven to be a handy port for connecting a variety of aftermarket displays and monitors to a car. Electric Vehicles have no need for emissions monitoring, so often omit the port from the car, thus making these aftermarket devices incompatible. The OBDII ECU capability of the OVMS v3 is used to create a simulated OBDII port, which can be used to attach many of these aftermarket devices to the car.

### 2.19.2 Cabling

The cable and OBDII connector are not provided with the OVMSv3 module, and can be built using the diagram below. Alternatively, you can purchase the [OVMS V3 HUD / OBD II-F Cable from FastTech](#).

DA26	OBDII Female	Signal name
6	14	CAN3-L
16	6	CAN3-H
8	4 & 5	Chassis & Signal Ground
18	16	+12v switched output to HUD/dongle

N.B. Also place a 120 ohm resistor between OBDII pins 14 & 6 for bus termination

### 2.19.3 Setup

From the Web Interface, check the “Start OBD2ECU” box, and select CAN3 from the dropdown menu (if using the wiring diagram above). This will enable the OBDII ECU Task to run the next time the OVMSv3 module is powered on or reset. Also check the “Power on external 12V” box in order to feed 12v power through to the device. Click on Save at the bottom of the page.

From the command line, the following commands are available:

```
obdii ecu start can3  Starts the OBDII ECU task.
obdii ecu stop        Stops the OBDII ECU task
obdii ecu list         Displays the parameters being served, and their current value
obdii ecu reload       Reloads the map of parameters, after a config change

power ext12v on        Turns on power feed to the device
power ext12v off       Turns off power feed to the device
```



## 2.19.4 Operation

During operation, an OBDII device, for example, a Head-Up Display (HUD) or OBDII Diagnostic module, will make periodic requests, usually a few times per second, for a set of parameters. The OVMSv3 module will reply to those parameters with the metric if configured to do so, on an individual basis. These parameters can be common items such as vehicle speed, engine RPM, and engine coolant temperature, but because of the differences between ICE and EV vehicles, many of the parameters do not have equivalent values in an EV. Speed and engine (motor) RPM can be directly mapped, but there is typically no “engine coolant”. That parameter (in fact, most parameters) can be mapped to some other value of interest. For example, the Engine Coolant display on the HUD can be configured to display motor or battery temperature instead. Engine Load (PID 4, a percentage value), is mapped by default to battery State of Charge (also a percentage). However, note that not all vehicle metrics may be supported by all vehicles.

Parameters requested by the OBDII device are referred to by “PID value”. Note that each PID has a specific range of allowed values, and that it is not possible to directly represent values outside that range. For example, PID 5 (Engine coolant temperature) has a range of -40 to +215; it would not be possible to map the full range of motor RPMs to this parameter. Values outside the allowed range are limited to the range boundary value. The complete table of possible parameters are described here: [https://en.wikipedia.org/wiki/OBD-II\\_PIDs#Mode\\_01](https://en.wikipedia.org/wiki/OBD-II_PIDs#Mode_01)

Vehicle metrics are referred to by name. See the table in Appendix 1 for a list of available metrics, which vehicles report them, and which of those are of a format that can be mapped by the OBDII ECU task.

## 2.19.5 Defaults and customization

By default, the following mapping of PID value to OVMSv3 metric is used:

PID (Dec)	PID (Hex)	Requested Parameter	Mapped Metric Value	Metric Name
4	0x04	Engine Load	Battery SOC	v.b.soc
5	0x05	Coolant Temp	Motor Temp	v.m.temp
12	0x0c	Motor RPM	Motor RPM	v.m.rpm
13	0x0d	Vehicle Speed	Vehicle Speed	v.p.speed
16	0x10	MAF Air Flow	12v Battery	v.b.12v.voltage

The OBDII ECU task supports the remapping and reporting of PIDs 4-18, 31, 33, 47, and 51 (all decimal). PIDs #0 and 32 are used by the OBDII protocol for management, and not available for assignment. Other PIDs return zero.

PIDs requested for which no mapping has been established are ignored by the OBDII ECU task. As an aid for discovering which PIDs are being requested, the configuration parameter ‘autocreate’ may be used to populate entries into the obdii ecu list display. (‘config set obd2ecu autocreate yes’). Such autocreated entries are marked as “Unimplemented”, and return zero to the device. They are not added to the configured (saved) obd2ecu.map, and get cleared at each boot or reset. They may be mapped to a supported metric, if desired, using:

```
config set obd2ecu.map <PID> <metric name>
```

Example:

```
OVMS# config set obd2ecu.map 5 v.b.temp
(map battery temp to engine coolant temp)

OVMS# vfs edit /store/obd2ecu/10
(script for fuel pressure PID; see below)

OVMS# obdii ecu reload
OBDII ECU pid map reloaded
```

(continues on next page)

(continued from previous page)

OVMS# *obdii ecu list*

PID	Type	Value	Metric
0 (0x00)	internal	0.000000	
4 (0x04)	internal	95.000000	v.b.soc
5 (0x05)	metric	16.000000	v.b.temp
10 (0x0a)	script	0.000000	
11 (0x0b)	unimplemented	0.000000	
12 (0x0c)	internal	0.000000	v.m.rpm
13 (0x0d)	internal	0.000000	v.p.speed
16 (0x10)	internal	13.708791	v.b.12v.voltage
32 (0x20)	internal	0.000000	

Types: \* “internal” means default internal handling of the PID. \* “metric” means a user-set mapping of PID to the named metric \* “unimplemented” are PIDs requested by the device, but for which no map has been set \* “script” means the user has configured a script to handle the PID

## 2.19.6 Special handling

Several PIDs are handled specially by the OBDII ECU task.

- PIDs 0 and 32 are bit masks that indicate what other PIDs are being reported by the OBDII ECU task. These are maintained internally based on the default, mapped, and scripted PID table. Note that some OBDII devices use PID 0 as a test for ECU presence and operating mode (standard or extended), and ignore the returned values. The OBDII ECU task supports both modes.
- PID 12, Engine RPM, is often monitored by the OBDII device to detect when the car is turned off. Since an EV’s motor is not rotating when the car is stopped, a HUD may decide to power down when it sees the RPM drop below a particular value, or if there is no variation (jitter) in its value. To prevent this, the OBDII ECU task will source a fake value of 500 rpm, plus a small periodic variation, if the car is not moving (vehicle speed is less than 1). To actually let the device turn off, see “External Power Control”, below.
- PID 16, MAF Air Flow, is commonly used by OBDII devices to display fuel flow, by measuring the amount of air entering the engine in support of combustion. Since this is irrelevant to an EV, the OBDII ECU task maps this metric to a simple integer. Most HUDs displays limit this to a range of 0-19.9 liter/hr, which is acceptable to display the +12v battery voltage. Since the conversion factors are complicated, this value is at best approximate, in spite of its implied precision.
- Mode 9, PID 2, VIN, is used to report the car’s DMV VIN to the attached OBDII device. Since the rest of the parameters reported by the OBDII ECU task are simulated, and some OBDII devices may use the VIN for tracking purposes, the reporting of the VIN may be turned off by setting the privacy flag to “yes”. The command is ‘config set obd2ecu privacy yes’. Setting it to ‘no’, which is the default, allows the reporting of the VIN.
- Mode 9, PID 10, ECU Name, is statically mapped to report the OVMSv3’s Vehicle ID field (vehicle name, not VIN). This string may be customized to any printable string of up to 20 characters, if not used with the OVMS v2 or v3 mobile phone applications. (‘config set vehicle id car\_name’)

## 2.19.7 Metric Scripts

Should one desire to return a value not directly available by a single named metric, it is possible to map a PID to a short script, where combinations of metrics, constants, etc. may be used to create a custom value. Note that the restrictions on PID value ranges still applies. Also note that the special handling for PID 12 (engine RPM) is not applied in the OBDII ECU task, so it must be included in the script if driving a HUD.

Scripts should be placed in the directory `/store/obd2ecu/PID`, where PID is the decimal value of the PID to be processed. Example for creating a “kw per km” sort of metric:

```
ret1=OvmsMetrics.AsFloat("v.p.speed");
ret2=OvmsMetrics.AsFloat("v.b.power");
out=0.0;
if (ret1 > 0) { out=ret2/ret1; }
out;
```

Put this text in a file `/store/obd2ecu/4` to map it to the “Engine Load” PID. See “Simple Editor” chapter for file editing, or use ‘vfs append’ commands (tedious). Note however, that Vehicle Power (v.b.power) is not supported on all cars (which is why this is not the default mapping for this PID).

Warning: The error handling of the scripting engine is very rough at this writing, and will typically cause a full module reboot if anything goes wrong in a script.

## 2.19.8 External Power Control

Since the OVMSv3 module remains powered at all times, and the normal means for deducing that a car has been turned off don’t work on an EV (see PID #12, above), an external OBDII device needs to be explicitly turned on and off. This is currently done with short event scripts. The following commands configure the OVMSv3 to make the external 12v feed follow the vehicle on/off state, or use the vfs edit command to create or modify the files:

```
vfs mkdir /store/events
vfs mkdir /store/events/vehicle.on
vfs mkdir /store/events/vehicle.off

vfs append 'power ext12v on' /store/events/vehicle.on/ext12v
vfs append 'power ext12v off' /store/events/vehicle.off/ext12v
```

## 2.20 EGPIO

### 2.20.1 Hardware

The OVMS has 10 general purpose I/O ports, provided by a [MAX7317 I/O expander](#).

The MAX7317 ports can be individually configured as either an open-drain output, or an overvoltage-protected Schmitt input. Being open-drain, you need to add pull-up resistors for input and output switch uses to get a defined high level (e.g. 10K to 3.3V). See the datasheet for details on maximum current & voltage ratings.

Depending on your hardware configuration, up to 4 ports may be used by the module. 6 are generally free to use.

Port	Signal(s)	Exp.Pin	Default usage
0	MDM_EN	15	Modem enable
1	SW_CTL	–	Ext12V control (SW_12V)
2	CAN1_EN + EIO_1	17	CAN1 transceiver enable
3	MDM_DTR + EIO_2	19	Modem sleep control
4	EIO_3	21	-free-
5	EIO_4	20	-free-
6	EIO_5	18	-free-
7	EIO_6	16	-free-
8	EIO_7	14	-free-
9	EIO_8	12	-free-

SW\_CTL (port 1) controls a [BTS452R power switch](#), which *could* deliver a nominal output of 25W or 1.8A, and up to 40W or 2.9A short term at the SW\_12V pin, *but* the main fuse of the module (located at the corner near the DB9 plug) limits the 12V current total sum of the board plus any external hardware to continuous 0.75A on a revision 3.1 board and 1.0A on a revision 3.2 board. The unswitched EXT\_12V pin is also behind the fuse.

The module including modem needs a 12V current share of ~80mA in full operation. Calculate with 100mA to be on the safe side. That leaves continuous **0.65A** on a revision 3.1 board and **0.9A** on a revision 3.2 board for external devices and addons powered by the module. The fuse has a little headroom, but don't rely on that.

SW\_12V is meant to power auxiliary devices from the OVMS, for example head-up displays. Of course you can as well power a standard 12V automotive relay or fan directly from this without additional hardware.

The EGPIO (EIO) ports are not connected directly to the DA26 connector but are available at the internal expansion port. To route an EGPIO port to the DA26 connector, connect it to one of the GEP\_1...7 lines at the expansion port, either directly or via some additional driver.

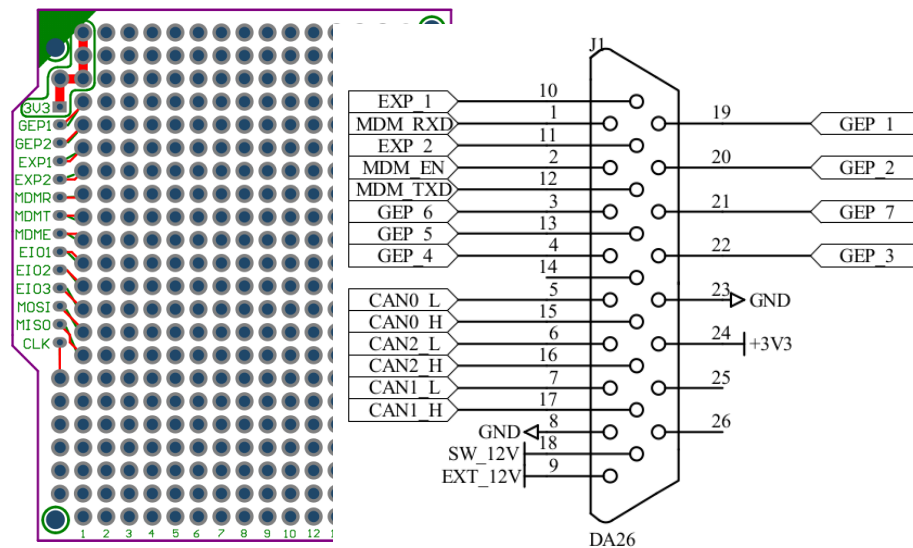


Fig. 1: Expansion Port (Prototype PCB)

Fig. 2: DA26 Connector

Example: to route EIO\_8 (port 9) to GEP\_7 (pin 21 on the DA26), set a jumper on pins 10+12 on the expansion port.

## 2.20.2 Commands

EGPIO control is provided by the `egpio` command set:

- `egpio output <port> <level> [<port> <level> ...]` – set output level(s)
- `egpio input <port> [<port> ...]` – query input level(s)
- `egpio status` – show output, input & monitor status
- `egpio monitor <on|off> [ports]` – enable/disable monitoring
- `egpio monitor status` – show current monitoring status

To configure a port for input, it needs to be switched to output level high (1). That is done automatically by the `input` and `monitor` commands.

If you set multiple outputs, the ports will be set one at a time, so output levels will change with a slight delay. You can use this behaviour to set data lines before a clock line, e.g. when sending bits serially into a shift register.

---

**Note:** The MAX7317 needs active polling to detect input state changes. Monitoring is disabled by default, it can be enabled manually or configured to start automatically on module init. Without monitoring, only manual input queries will update the input state, trigger input events and input metric updates.

---

### 2.20.3 Configuration

Parameter	Instance	Description
auto	egpio	yes = Start monitoring on boot (default: no)
egpio	monitor.ports	List of ports to monitor by default (space separated)
egpio	monitor.interval	Polling interval in milliseconds (min 10, default 50)

The default interval of 50 ms (= 20 Hz) means an input signal needs to be at least 50 ms long to be detected. This polling frequency produces a CPU load of ~0.5% on core 1 and is normally sufficient to detect even very short button pushes.

### 2.20.4 Metrics

Metric name	Example value	Description
m.egpio.input	0,1,2,3,4,5,6,7,9	EGPIO input port state (ports 0...9, present=high)
m.egpio.monitor	8,9	EGPIO input monitoring ports
m.egpio.output	4,5,6,7,9	EGPIO output port state

Hint: to process these metrics from Javascript, read them into an array using `eval()` and test for the presence of a port number using e.g. the `includes()` method in a browser plugin. Duktape does not support `includes()`, you can test `indexOf(port)` instead.

Example:

```
var input = eval(OvmsMetrics.AsJSON("m.egpio.input"));
if (input.indexOf(9) < 0)
    print("Input port 9 (EI08) is currently low\n");
```

### 2.20.5 Events

Event	Data	Purpose
egpio.input.<port>.<state>	–	EGPIO input port change (port=0...9, state=high/low)
egpio.output.<port>.<state>	–	EGPIO output port change (port=0...9, state=high/low)

Hint: to listen to events from Javascript, bind to `msg:event` on a `.receiver` object from browser context or use `PubSub` from module context.

Example:

```
PubSub.subscribe("egpio.input.9.low", function() {  
  print("Input port 9 (EI08) is now low\n");  
});
```

## 2.21 TPMS

OVMS v3 includes a capability to read and write sets of TPMS wheel sensors IDs from the vehicle TPMS ECU, and store in your configuration. Support for this is vehicle-specific, and may require optional hardware boards, so please follow your individual vehicle guides for further information.

The command to read the current TPMS wheel sensor IDs from the car is:

```
OVMS# tpms read SET
```

(replace SET with your own identifier for this set of wheels, such as 'summer', 'winter', etc)

You can check the stored wheel sensor ID sets with:

```
OVMS# tpms list
```

If you change wheels, you can write the new wheel IDs to the TPMS ECU in the car with:

```
OVMS# tpms write SET
```

(replace SET with your own identifier for this set of wheels, such as 'summer', 'winter', etc)

Note that in most cases the car must be switched on for the above commands to work. Please refer to your vehicle specific user guide for more information on this.

Note also that OVMS doesn't have any radio capable of receiving TPMS signals and cannot read the IDs from the wheel sensors themselves. OVMS can only read the IDs from the TPMS ECU in the car itself. You can, however, drop by pretty much any garage (or use any of a large number of TPMS tools) to trigger and read these IDs from the wheels. Once your garage gives you the sensor IDs (each expressed as an 8 character hexadecimal ID), you can enter them into OVMS as:

```
OVMS# tpms set SET ID1 ID2 ID3 ID4 ... (replace SET with your own identifier for this set of wheels,  
and ID1.. with the hexadecimal sensor ID)
```

## 2.22 Optional K-Line Expansion Board

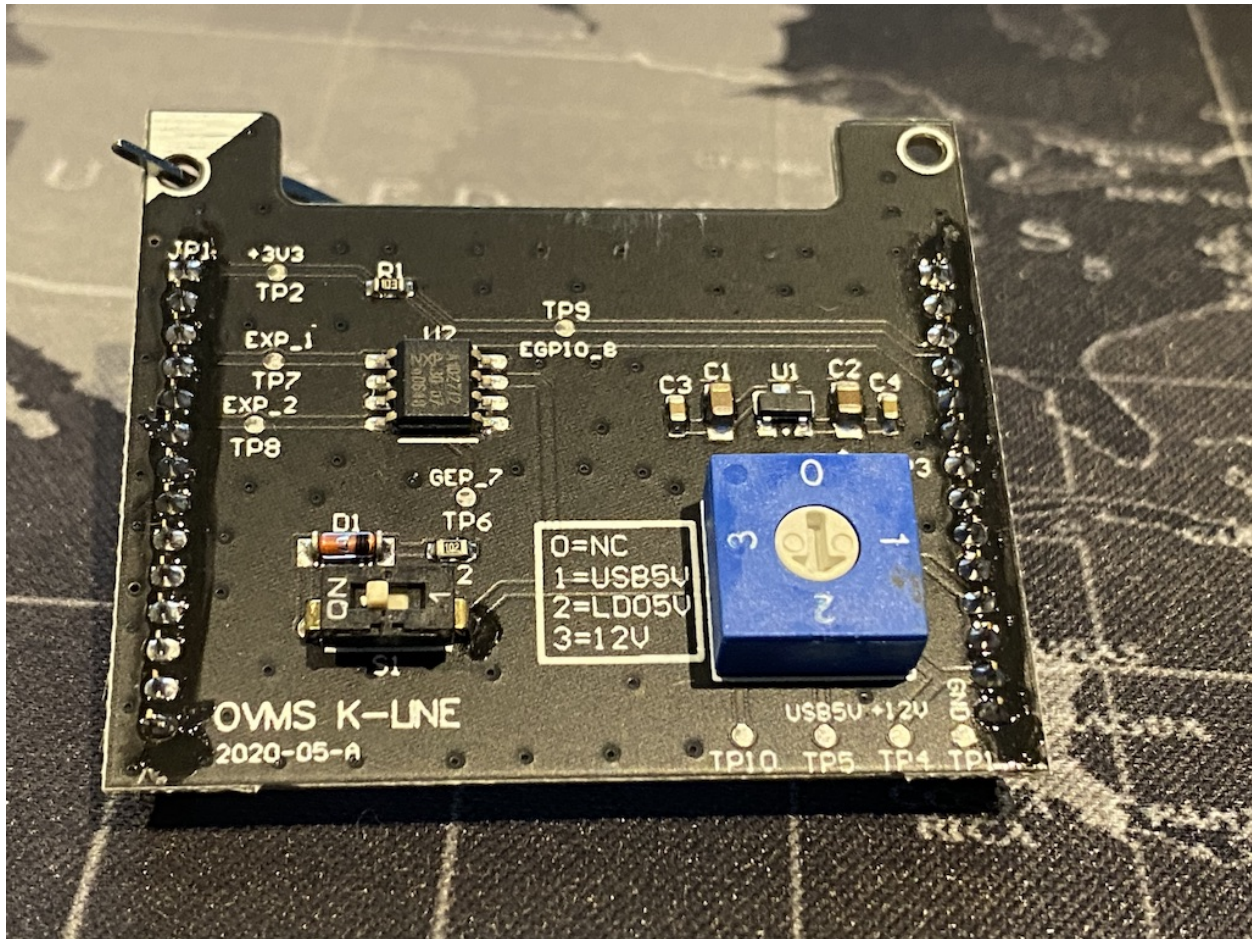
The optional K-Line Expansion Board is available to extend the capabilities of the OVMS system. It can be used to talk to vehicle ECUs using K-Line communication standards.





There are two installation options:

1. If you are using a modem module, you install the K-Line Expansion Board on top of the modem. Unscrew the four black screws holding the modem in place. In your K-Line expansion board kit you will find four brass stand offs with holes at one end and small bolts at the other. Use these to screw down the modem. Now, orient the K-line Expansion Board so the top left corner (marked with a white triangle) matches the white triangle on the modem board. Carefully push the K-Line Expansion Board into place (making sure that the pins align and are not bent). Once done, the four black screws can be used to secure it in place.
2. If you are not using a modem module, you install the K-Line Expansion Board directly onto the main OVMS board. Screw the provided standoffs into the main board, and Carefully attach the provided pin headers into the main OVMS board. Now, orient the K-line Expansion Board so the top left corner (marked with a white triangle) matches the white triangle on the main OVMS board. Carefully push the K-Line Expansion Board into place (making sure that the pins align and are not bent). Once done, the four black screws can be used to secure it in place.



Next, the K-Line expansion board will need to be configured to match your vehicle's K-Line bus and ECU hardware requirements. Refer to your vehicle specific guide for information on what your vehicle requires:

- S1 is used to turn on/off the extra 'master' mode components. For most vehicles, this should be OFF.
- SW1 is used to select the power source for the K-Line bus. It can be #0 (no power connected), #1 (5v power from USB bus), #2 (5v power converted from vehicle's 12v power), or #3 (12v power direct from the vehicle). For most vehicles, either #2 or #3 is used.

The K-Line bus from the vehicle is connected via a vehicle specific cable. This connection is usually to pin #1 on the DB9 connector. On OVMS modules v3.2 and later (as well as v3.1 boards labelled on board as July 2018 or later), this is internally connected to GEP7 (General Expansion Pin #7). This is necessary to bring the K-Line bus up to the internal expansion pins (for the K-Line Expansion Board to be able to reach it).

If you have an earlier board, there is a relatively simply wiring modification that can be made to upgrade to this:

1. You will need a soldering iron, solder, and a small length of insulated wire.
2. Pin #1 on the DB9 connector is the K-Line pin from the vehicle.
3. Pin #21 on the DA26 connector is GEP 7 (which is also connected to a pin on internal expansion connector).
4. On the underside of the board you can solder a jumper wire between DB9 pin #1 and DA26 pin #21 to make this connection.

Please refer to the vehicle specific guides for further information on this.



The OVMS can easily be extended by plugins. Plugins may consist of any combination of module scripts (Javascript) and web UI extensions (HTML).

Installation currently needs to be done manually, but is simple. See below and the plugin documentation on specific steps and on how to use the plugin functions. A plugin store with simplified download and optional automatic updating will be provided in the future.

This page is intended as an overview of all plugins currently available in the OVMS repository. If you're a plugin developer and want to add your plugin here, please submit a pull request containing your plugin directory. Organize your files and include a `README.rst` as shown here.

### 3.1 ABRP: [abetterrouteplanner.com](https://abetterrouteplanner.com)

#### Send live data to [abetterrouteplanner.com](https://abetterrouteplanner.com)

##### Overview

[abetterrouteplanner](https://abetterrouteplanner.com) is probably the best website to plan a route with an EV and optimize the number of stops needed and where to charge your vehicle.

Moreover, it has a wonderful functionality, to update your plan, taking into account your live data, to adjust with the real consumption of the vehicle, by connecting an obd device and soft like *EVNotify* or *Torque Pro*.

This plugin sends your live data from the OVMS box.

**Author** David S Arnold

**New in version 1.2:** Your own parameters (car model, abrp token but also api-url) are now stored as config parameters, you can consult them using the shell with `config list usr` or modify with `config set` command.

### Contents

- [ABRP: abetterrouteplanner.com](#)
  - [Installation](#)
  - [Script code](#)
  - [Usage](#)

### 3.1.1 Installation

You can use the embedded website *tools/shell* and *tools/editor* of the OVMS box to create the directory and file. This editor will allow you to paste the plugin code from the documentation.

#### Files to be created in /store/scripts/:

- **ovmsmain.js**, if not already exists
- **sendlivedata2abrp.js**, by copying the file below

Next, configure your car model, API URL & token:

---

**Note:** ABRP only supports live data feeds for some car models. You can request a notification when support has been added for your car. **Do not feed live data using other car models!** Also, ABRP has not yet added this OVMS plugin as a general data source, so you need to disguise as “Torque”.

---

#### If using the ABRP 4 UI:

- Login to [abetterrouteplanner](#)
- Open “Settings”, enable detailed setup
- Add your car (if ABRP has no live support for your car model, it will display “Live data not available”)
- Click “Link Torque”, click “Next” 3 times
- Set the “Webserver URL” by `config set usr abrp.url "..."` \* Note: according to Iternio, the URL `http://api.iternio.com/1/tlm/send` should generally be used here instead of the Torque specific one displayed
- Set the “User Email Address” (API token) by `config set usr abrp.user_token "..."`

#### If using ABRP classic UI:

- Login to [abetterrouteplanner in classic view](#)
- In Settings/more Settings, there’s an item ‘Live Car Connection’ with 2 buttons: ‘Setup’ and ‘View live data’
- Click on Setup (if there is no Setup button, live data support isn’t available for your car model)
- Click on Torque
- Set the “Webserver URL” by `config set usr abrp.url "..."`
- Set the “User Email Address” (API token) by `config set usr abrp.user_token "..."`

#### Determine your car model code:

- Open [API car models list](#)

- To improve readability, optionally paste the page into [JSONLint.com](https://jsonlint.com)
- Search for your car brand and model, the code is the field following the car specification, for example  
renault:zoe:20:52:r110
- Set the code by config set usr abrp.car\_model "..."

Finally reboot your OVMS module. This was a one-time configuration.

You're now ready. Test it with the shell page in the embedded web server using the command `script eval abrp.info()` and then with the command `script eval abrp.onetime()`. You can also do it with the mobile app.

### 3.1.2 Script code

This is a javascript code, to extract in the file `sendlivedata2abrp.js`.

**Warning:** note the minimum firmware version: 3.2.008-147

```

1  /**
2   *      /store/scripts/sendlivedata2abrp.js
3   *
4   * Module plugin:
5   *   Send live data to a better route planner
6   *   This version uses the embedded GSM of OVMS, so there's an impact on data_
7   *   ↪consumption
8   *   /\ requires OVMS firmware version 3.2.008-147 minimum (for HTTP call)
9   *
10  * Version 1.3   2020   inf0mike (forum https://www.openvehicles.com)
11  *
12  * Enable:
13  *   - install at above path
14  *   - add to /store/scripts/ovmsmain.js:
15  *       abrp = require("sendlivedata2abrp");
16  *   - script reload
17  *
18  * Usage:
19  *   - script eval abrp.info()           => to display vehicle data to be sent to abrp
20  *   - script eval abrp.onetime()        => to launch one time the request to abrp server
21  *   - script eval abrp.send(1)          => toggle send data to abrp
22  *   - script eval abrp.send(0)          => stop sending data
23  *   - script eval abrp.resetConfig()    => reset configuration to defaults
24  *
25  * Version 1.3 updates:
26  *   - Fix for rounding of fractional SOC causing abrp to report SOC off by 1
27  *   - Fix for altitude never being sent
28  *   - New convenience method to reset config to defaults
29  *
30  * Version 1.2 updates:
31  *   - based now on OVMS configuration to store user token, car model and url
32  *   - review messages sent during charge
33  *   - send a message when vehicle is on before moving to update abrp
34  *
35  * Version 1.1 fix and update:
36  *   - fixed the utc refreshing issue
37  *   - send notifications
38  *   - send live data only if necessary

```

(continues on next page)

(continued from previous page)

```

38  **/
39
40
41
42  /*
43   * Declarations:
44   *   CAR_MODEL: find your car model here: https://api.iternio.com/1/tlm/get_carmodels_
45   *   list?api_key=32b2162f-9599-4647-8139-66e9f9528370
46   *   OVMS_API_KEY : API_KEY to access to ABRP API, given by the developer
47   *   MY_TOKEN : Your token (corresponding to your abrp profile)
48   *   TIMER_INTERVAL : to subscribe to a ticker event
49   *   URL : url to send telemetry to abrp following: https://iternio.com/index.php/
50   *   iternio-telemetry-api/
51   *   CR : Carriage Return for console prints
52   *
53   *   objTLM : JSON object containing data read
54   *   objTimer : timer object
55   */
56
57  const CAR_MODEL = "@@:@@:@@:@@";
58  const OVMS_API_KEY = "32b2162f-9599-4647-8139-66e9f9528370";
59  const MY_TOKEN = "@@@@@@@@-@@@@-@@@@-@@@@-@@@@@@@@@@@@";
60  const TIMER_INTERVAL = "ticker.60"; // every minute
61  const EVENT_MOTORS_ON = "vehicle.on";
62  const URL = "http://api.iternio.com/1/tlm/send";
63
64  const DEFAULT_CFG = {
65    "url": URL,
66    "user_token": MY_TOKEN,
67    "car_model": CAR_MODEL
68  };
69
70  const CR = '\n';
71
72  var objTLM;
73  var objTimer, objEvent;
74  var sHasChanged = "";
75  var bMotorsOn = false;
76
77  // initialise from default
78  var abrp_cfg = JSON.parse(JSON.stringify(DEFAULT_CFG));
79
80  // check if json object is empty
81  function isJsonEmpty(obj) {
82    for(var key in obj) {
83      if(obj.hasOwnProperty(key))
84        return false;
85    }
86    return true;
87  }
88
89  // Read & process config:
90  function readConfig() {
91    // check if config exist
92    var read_cfg = OvmsConfig.GetValues("usr", "abrp.");
93    print(JSON.stringify(read_cfg) + CR);
94    if (isJsonEmpty(read_cfg) == true) {

```

(continues on next page)

(continued from previous page)

```

93     // no config yet, set the default values
94     OvmsConfig.SetValues("usr","abrp.",abrp_cfg);
95 } else {
96     // config existing
97     abrp_cfg.url = read_cfg.url;
98     abrp_cfg.user_token = read_cfg.user_token;
99     abrp_cfg.car_model = read_cfg.car_model;
100 }
101 }
102
103 // Make json telemetry object
104 function InitTelemetryObj() {
105     return {
106         "utc": 0,
107         "soc": 0,
108         "soh": 0,
109         "speed": 0,
110         "car_model": abrp_cfg.car_model,
111         "lat": 0,
112         "lon": 0,
113         "alt": 0,
114         "ext_temp": 0,
115         "is_charging": 0,
116         "batt_temp": 0,
117         "voltage": 0,
118         "current": 0,
119         "power": 0
120     };
121 }
122
123 // Fill json telemetry object
124 function UpdateTelemetryObj(myJSON) {
125     if(!myJSON){
126         // if the data object is undefined or null then return early
127         return false;
128     }
129     var read_num = 0;
130     var read_str = "";
131     var read_bool = false;
132
133     sHasChanged = "";
134
135     if (bMotorsOn) {
136         sHasChanged = "_MOTORS-ON";
137         bMotorsOn = false;
138     }
139
140     // using Math.floor avoids rounding up of .5 values to next whole number
141     // as some vehicles report fractional values. Abrp seems to only support integer
142     ↪ values
143     read_num = Math.floor(Number(OvmsMetrics.Value("v.b.soc")));
144     if (myJSON.soc != read_num) {
145         myJSON.soc = read_num;
146         sHasChanged += "_SOC:" + myJSON.soc + "%";
147     }
148
149     read_num = Number(OvmsMetrics.Value("v.b.soh"));

```

(continues on next page)

(continued from previous page)

```

149     if (myJSON.soh != read_num) {
150         myJSON.soh = read_num;
151         sHasChanged += "_SOH:" + myJSON.soh + "%";
152     }
153
154     if ( (myJSON.soh + myJSON.soc) == 0 ) {
155         // Sometimes the canbus is not readable, and abrp doesn't like 0 values
156         print("canbus not readable: reset module and then put motors on" + CR);
157         return false;
158     }
159
160     //myJSON.lat = OvmsMetrics.AsFloat("v.p.latitude").toFixed(3);
161     //above code line works, except when value is undefined, after reboot
162
163     read_num = OvmsMetrics.AsFloat("v.p.latitude");
164     read_num = read_num.toFixed(3);
165     if (myJSON.lat != read_num) {
166         myJSON.lat = read_num;
167         sHasChanged += "_LAT:" + myJSON.lat + "°";
168     }
169
170     read_num = Number(OvmsMetrics.AsFloat("v.p.longitude"));
171     read_num = read_num.toFixed(3);
172     if (myJSON.lon != read_num) {
173         myJSON.lon = read_num;
174         sHasChanged += "_LON:" + myJSON.lon + "°";
175     }
176
177     read_num = Number(OvmsMetrics.AsFloat("v.p.altitude"));
178     read_num = read_num.toFixed(1);
179     if (read_num <= (myJSON.alt - 2) || read_num >= (myJSON.alt + 2)) {
180         myJSON.alt = read_num;
181         sHasChanged += "_ALT:" + myJSON.alt + "m";
182     }
183
184     read_num = Number(OvmsMetrics.Value("v.b.power"));
185     myJSON.power = read_num.toFixed(1);
186
187     myJSON.speed=Number(OvmsMetrics.Value("v.p.speed"));
188     myJSON.batt_temp=Number(OvmsMetrics.Value("v.b.temp"));
189     myJSON.ext_temp=Number(OvmsMetrics.Value("v.e.temp"));
190     myJSON.voltage=Number(OvmsMetrics.Value("v.b.voltage"));
191     myJSON.current=Number(OvmsMetrics.Value("v.b.current"));
192
193     myJSON.utc = Math.trunc(Date.now()/1000);
194     //myJSON.utc = OvmsMetrics.Value("m.time.utc");
195
196     // read_bool = Boolean(OvmsMetrics.Value("v.c.charging"));
197     // v.c.charging is also on when regen => not wanted here
198     read_str = OvmsMetrics.Value("v.c.state");
199     if ( (read_str == "charging") || (read_str == "topoff") ) {
200         myJSON.is_charging = 1;
201         read_str = OvmsMetrics.Value("v.c.mode");
202         if (sHasChanged != "") {
203             sHasChanged += "_CHRG:" + read_str + "(" + OvmsMetrics.Value("v.c.charging")
↩+ " )";
204             print("Charging in mode " + read_str + CR);

```

(continues on next page)

(continued from previous page)

```

205     }
206   } else {
207     myJSON.is_charging = 0;
208   }
209
210   myJSON.car_model = abrp_cfg.car_model;
211
212   return (sHasChanged !== "");
213 }
214
215 // Show available vehicle data
216 function DisplayLiveData(myJSON) {
217   var newcontent = "";
218   newcontent += "altitude = " + myJSON.alt      + "m" + CR;    //GPS altitude
219   newcontent += "latitude = " + myJSON.lat      + "°" + CR;    //GPS latitude
220   newcontent += "longitude= " + myJSON.lon      + "°" + CR;    //GPS longitude
221   newcontent += "ext temp = " + myJSON.ext_temp + "°C" + CR;    //Ambient_
↪temperature
222   newcontent += "charge   = " + myJSON.soc      + "%" + CR;    //State of charge
223   newcontent += "health   = " + myJSON.soh      + "%" + CR;    //State of health
224   newcontent += "bat temp = " + myJSON.batt_temp + "°C" + CR;    //Main battery_
↪momentary temperature
225   newcontent += "voltage  = " + myJSON.voltage  + "V" + CR;    //Main battery_
↪momentary voltage
226   newcontent += "current  = " + myJSON.current  + "A" + CR;    //Main battery_
↪momentary current
227   newcontent += "power    = " + myJSON.power    + "kW" + CR;    //Main battery_
↪momentary power
228   newcontent += "charging = " + myJSON.is_charging + CR;    //yes = currently_
↪charging
229   print(newcontent);
230 }
231
232 function InitTelemetry() {
233   objTLM = InitTelemetryObj();
234   sHasChanged = "";
235 }
236
237 function UpdateTelemetry() {
238   var bChanged = UpdateTelemetryObj(objTLM);
239   if (bChanged) { DisplayLiveData(objTLM); }
240   return bChanged;
241 }
242
243 function CloseTelemetry() {
244   objTLM = null;
245   sHasChanged = "";
246 }
247
248 // http request callback if successful
249 function OnRequestDone(resp) {
250   print("response="+JSON.stringify(resp)+CR);
251   //OvmsNotify.Raise("info", "usr.abrp.status", "ABRP:." + sHasChanged);
252 }
253
254 // http request callback if failed
255 function OnRequestFail(error) {

```

(continues on next page)

(continued from previous page)

```

256     print("error="+JSON.stringify(error)+CR);
257     OvmsNotify.Raise("info", "usr.abrp.status", "ABRP:." + JSON.stringify(error));
258 }
259
260 // Return full url with JSON telemetry object
261 function GetUrlABRP() {
262     var urljson = abrp_cfg.url;
263     urljson += "?";
264     urljson += "api_key=" + OVMS_API_KEY;
265     urljson += "&";
266     urljson += "token=" + abrp_cfg.user_token;
267     urljson += "&";
268     urljson += "tlm=" + encodeURIComponent(JSON.stringify(objTLM));
269     print(urljson + CR);
270     return urljson;
271 }
272
273 // Return config object for HTTP request
274 function GetURLcfg() {
275     var cfg = {
276         url: GetUrlABRP(),
277         done: function(resp) {OnRequestDone(resp)},
278         fail: function(err) {OnRequestFail(err)}
279     };
280     return cfg;
281 }
282
283 function SendLiveData() {
284     if (UpdateTelemetry()) {
285         HTTP.Request( GetURLcfg() );
286     }
287 }
288
289 function Reactivate_MotorsOn() {
290     bMotorsOn = true;
291     SendLiveData();
292 }
293
294 function InitTimer() {
295     objTimer = PubSub.subscribe(TIMER_INTERVAL, SendLiveData);
296     objEvent = PubSub.subscribe(EVENT_MOTORS_ON, SendLiveData);
297 }
298
299 function CloseTimer() {
300     PubSub.unsubscribe(objEvent);
301     PubSub.unsubscribe(objTimer);
302     objEvent = null;
303     objTimer = null;
304 }
305
306 // API method abrp.onetime():
307 // Read and send data, but only once, no timer launched
308 exports.onetime = function() {
309     readConfig();
310     InitTelemetry();
311     SendLiveData();
312     CloseTelemetry();

```

(continues on next page)



(continued from previous page)

```

313     }
314
315     // API method abrp.info():
316     //   Do not send any data, just read vehicle data and writes in the console
317     exports.info = function() {
318         readConfig();
319         InitTelemetry();
320         UpdateTelemetry();
321         CloseTelemetry();
322     }
323
324     // API method abrp.resetConfig()
325     //   Resets stored config to default
326     exports.resetConfig = function() {
327         OvmsConfig.SetValues("usr","abrp.", DEFAULT_CFG);
328         print(JSON.stringify(abrp_cfg));
329         OvmsNotify.Raise("info", "usr.abrp.status", "ABRP::config changed");
330     }
331
332     // API method abrp.send():
333     //   Checks every minut if important data has changed, and send it
334     exports.send = function(onoff) {
335         if (onoff) {
336             readConfig();
337             if (objTimer != null) {
338                 print("Already running !" + CR);
339                 return;
340             }
341             print("Start sending data..." + CR);
342             InitTelemetry();
343             SendLiveData();
344             InitTimer();
345             OvmsNotify.Raise("info", "usr.abrp.status", "ABRP::started");
346         } else {
347             if (objTimer == null) {
348                 print("Already stopped !" + CR);
349                 return;
350             }
351             print("Stop sending data" + CR);
352             CloseTimer();
353             CloseTelemetry();
354             OvmsNotify.Raise("info", "usr.abrp.status", "ABRP::stopped");
355         }
356     }

```

### 3.1.3 Usage

#### How to make it run:

- install as described in ‘Installation’
- **add to /store/scripts/ovmsmain.js:** `abrp = require("sendlivedata2abrp");`
- script reload

**With command lines in the OVMS android or iOS app, in the messages part:**

- `script eval abrp.info()` => to display vehicle data to be sent to abrp
- `script eval abrp.onetime()` => to launch one time the request to abrp server
- `script eval abrp.send(1)` => toggle send data to abrp
- `script eval abrp.send(0)` => stop sending data
- `script eval abrp.resetConfig()` => reset configuration to defaults

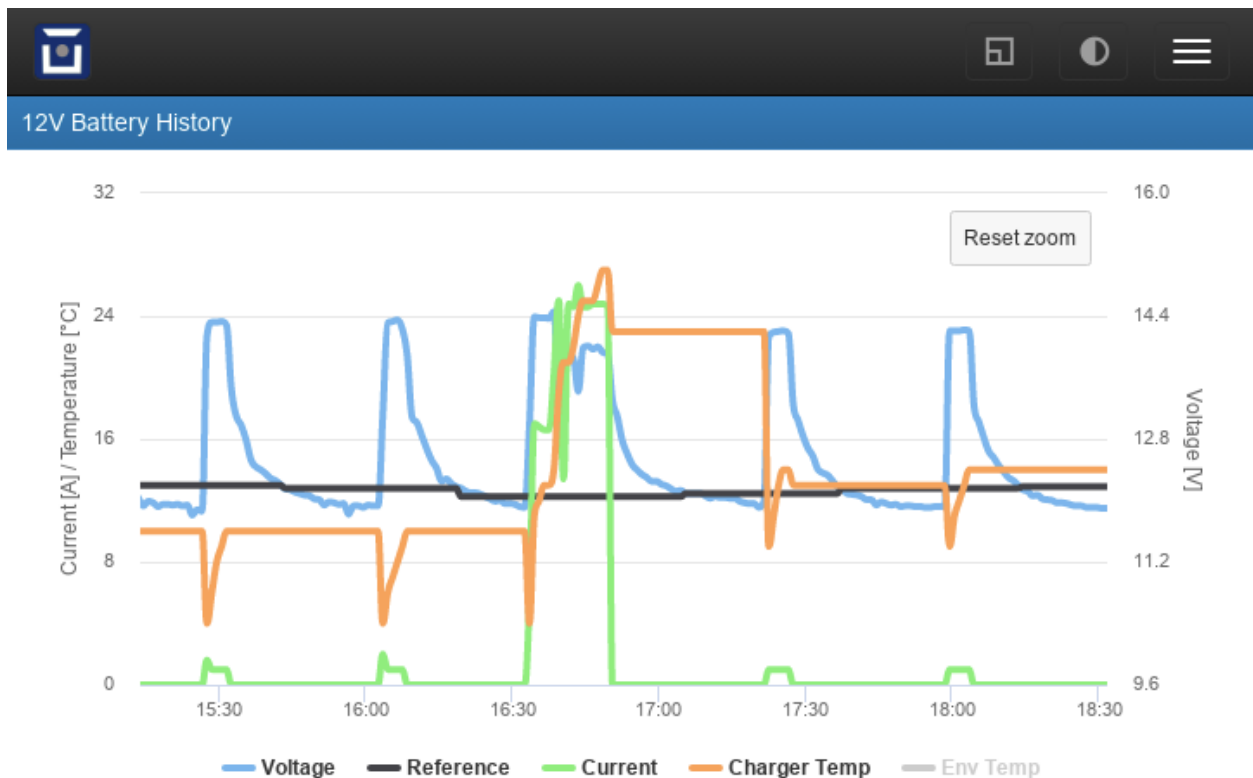
Also in the messages part, configuration can be updated with:

- `config set usr abrp.car_model <value>`
- `config set usr abrp.url <value>`
- `config set usr abrp.user_token <value>`

## 3.2 AuxBatMon: 12V History Chart

Web chart showing last 24 hours of 12V battery metrics (similar to Android App)

Version 2.0 by Michael Balzer <dexter@dexters-web.de>



The module plugin records the relevant 12V battery metrics (voltage, reference voltage, current, charger and environment temperatures) once per minute for up to 24 hours by default.

The web plugin renders these recordings into a time series chart and continues feeding live updates into the chart. The chart is zoomable and can be panned along the X axis.

Since version 2.0, metrics history data is stored in a file and restored automatically on reboot/reload. This needs OVMS firmware  $\geq$  3.2.008-235 to work (will fallback to no saving on earlier versions).

### 3.2.1 Installation

1. Save `auxbatmon.js` as `/store/scripts/lib/auxbatmon.js`
2. Add line to `/store/scripts/ovmsmain.js`:
  - `auxbatmon = require("lib/auxbatmon");`
3. Issue `script reload` or evaluate the `require` line
4. Install `auxbatmon.htm` web plugin, recommended setup:
  - Type: Page
  - Page: `/usr/auxbatmon`
  - Label: 12V History
  - Menu: Tools
  - Auth: Cookie

### 3.2.2 Configuration

No live config currently. You can customize the sample interval (default: 60 seconds) and the history size (default: 24 hours) by changing the constants in the code. Take care to match a customization in both the module and the web plugin.

The persistent history storage file is `/store/usr/auxbatmon.jx`. It needs ~25 kB with the default sample configuration. If space is tight on your `/store` partition you can change the file location to `/sd/...` in the source code.

### 3.2.3 Usage

```
[script eval] auxbatmon.dump()           -- dump recorded history data in JSON format
```

## 3.3 ChgInd: Charge State Indicator

### Control three LEDs to indicate charging & SOC level

Version 1.0 by Michael Balzer <dexter@dexters-web.de>

Following an idea by “green\_fox” on the german Twizy forum (not limited to the Twizy) to implement an externally visible charge indicator.

Use a red, a green and a blue LED to signal charge state:

- OFF = not charging
- RED = charging, SOC below 20%
- YELLOW (RED+GREEN) = charging, SOC 20% ... 80%
- GREEN = charging, SOC above 80%
- BLUE = fully charged (held for 15 minutes)

This plugin uses three EGPIO lines to control the three LEDs (KISS). By using a shift register this could be reduced to two lines, by using a serial protocol to one line. The latter would need protocol support in the EGPIO (todo).

### 3.3.1 Installation

1. Save `chgind.js` as `/store/scripts/lib/chgind.js`
2. Add line to `/store/scripts/ovmsmain.js`:
  - `chgind = require("lib/chgind");`
3. Issue script reload or evaluate the require line

### 3.3.2 Configuration

```
// EGPIO ports to use:
const LED_red    = 4;  // EIO_3
const LED_green  = 5;  // EIO_4
const LED_blue   = 6;  // EIO_5

// Time to hold BLUE state after charge stop:
const holdTimeMinutes = 15;
```

### 3.3.3 Plugin API

```
chgind.set(state)  -- set LED state ('off'/'red'/'yellow'/'green'/'blue')
```

## 3.4 ChgThrottle: Protect Charger

**Throttle charge current / stop charge if charger gets too hot**

Version 1.2 by Michael Balzer <dexter@dexters-web.de>

The plugin monitors the charger temperature. It can reduce the charge current or stop a running charge process if the charger temperature exceeds on of three defined thresholds.

### 3.4.1 Installation

1. Save `chgthrottle.js` as `/store/scripts/lib/chgthrottle.js`
2. Add line to `/store/scripts/ovmsmain.js`:
  - `chgthrottle = require("lib/chgthrottle");`
3. Issue script reload

### 3.4.2 Configuration

Param	Instance	Description	Default
usr	<code>chgthrottle.enabled</code>	yes = enable throttling	yes
usr	<code>chgthrottle.t1.temp</code>	temperature threshold 1	50
usr	<code>chgthrottle.t1.amps</code>	charge current at threshold 1	25
usr	<code>chgthrottle.t2.temp</code>	threshold 2	55
usr	<code>chgthrottle.t2.amps</code>	current 2	15

(continues on next page)

(continued from previous page)

usr	chgthrottle.t3.temp	threshold 3	60
usr	chgthrottle.t3.amps	current 3	-1

Set t1...3 to ascending temperature thresholds. Below t1.temp, the current is set to unlimited.

Set temp to empty/0 to disable the level.

Set amps to -1 to stop the charge at that level.

### 3.4.3 Usage

Simply configure as desired, the plugin will monitor the charger temperature automatically and react as configured.

Use `config set usr chgthrottle.enabled no` to disable the plugin, ... `yes` to enable again.

State changes are logged and sent as notifications. Use `info()` to show the current state:

```
script eval chgthrottle.info()
```

Notifications are sent initially, then only on level-up during charge, and only on level-down while not charging.

## 3.5 Edimax: Smart Plug Control

### Smart Plug control for Edimax models SP-1101W, SP-2101W et al

Version 2.0 by Michael Balzer <dexter@dexters-web.de>

Note: may need `HTTP.request()` digest auth support to work with newer Edimax firmware (untested)

The smart plug can be bound to a defined location. Automatic periodic recharging or charge stop can be configured via main battery SOC level and/or 12V battery voltage level. The plugin can also switch off power at the charge stop event of the main and/or 12V battery.

### 3.5.1 Installation

1. Save `edimax.js` as `/store/scripts/lib/edimax.js`
2. Add line to `/store/scripts/ovmsmain.js`:
  - `edimax = require("lib/edimax");`
3. Issue `script reload`
4. Install `edimax.htm` web plugin, recommended setup:
  - Type: Page
  - Page: `/usr/edimax`
  - Label: Edimax Smart Plug
  - Menu: Config
  - Auth: Cookie
5. Install `edimax-status.htm` web plugin, recommended setup:
  - Type: Hook

- Page: /status
- Hook: body.post

The `edimax-status.htm` plugin adds power control to the Status page's vehicle panel.

### 3.5.2 Configuration

Use the web frontend for simple configuration.

Param	Instance	Description
usr	edimax.ip	Edimax IP address
usr	edimax.user	... username, "admin" by default
usr	edimax.pass	... password, "1234" by default
usr	edimax.location	optional: restrict auto switch to this location
usr	edimax.soc_on	optional: switch on if SOC at/below
usr	edimax.soc_off	optional: switch off if SOC at/above
usr	edimax.chg_stop_off	optional: yes = switch off on vehicle.charge.stop
usr	edimax.aux_volt_on	optional: switch on if 12V level at/below
usr	edimax.aux_volt_off	optional: switch off if 12V level at/above
usr	edimax.aux_stop_off	optional: yes = switch off on vehicle.charge.12v.stop

### 3.5.3 Usage

```
script eval edimax.get()
script eval edimax.set("on" | "off")
script eval edimax.info()
```

Note: `get()` & `set()` do an async update (if the location matches), the result is logged. Use `info()` to show the current state.

### 3.5.4 Events

```
usr.edimax.on
usr.edimax.off
usr.edimax.error
```

## 3.6 Foglight: Speed Adaptive Foglight

Automatic speed adaptive fog light using OVMS GPIO port

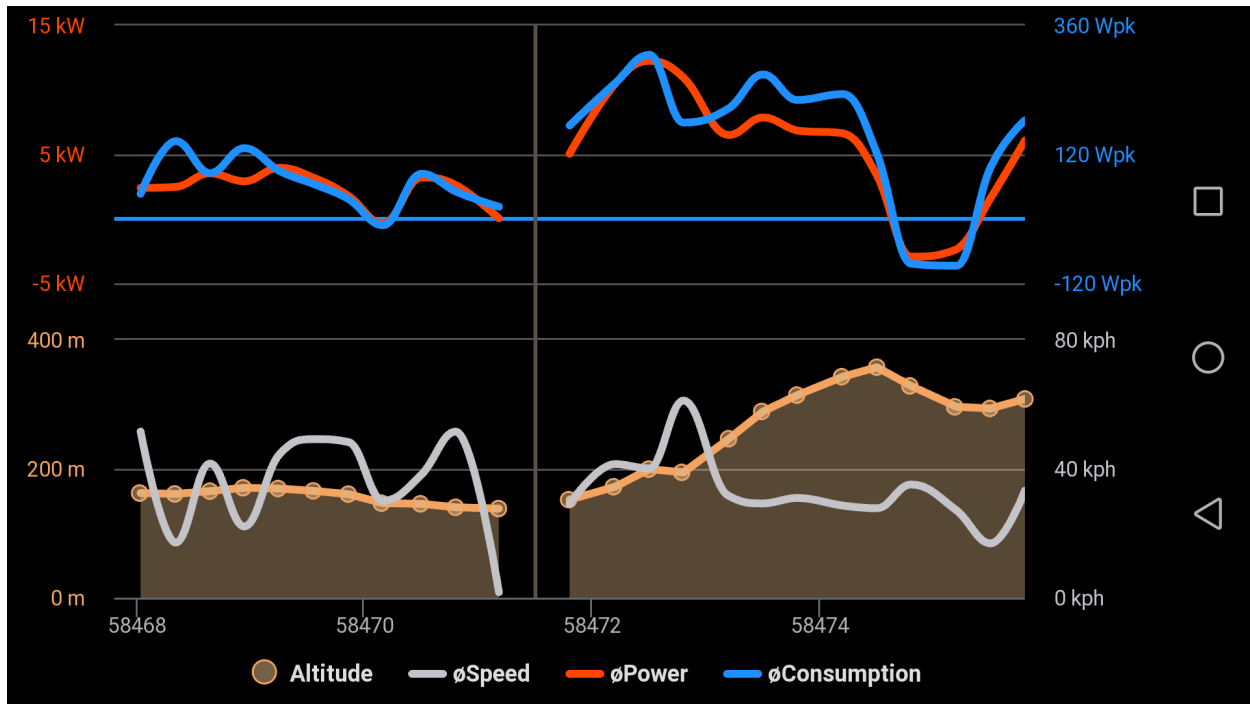
- `foglight.js` – module script
- `foglight.htm` – web dashboard extension

Details: [https://docs.openvehicles.com/en/latest/components/ovms\\_script/docs/foglight.html](https://docs.openvehicles.com/en/latest/components/ovms_script/docs/foglight.html)

## 3.7 PwrMon: Trip Power Chart

Web chart showing average speed, power & energy use on the road

Version 1.0 by Michael Balzer <dexter@dexters-web.de>



The plugin calculates average speed (kph), power (kW) & energy use (Wpk = Wh/km) from the odometer, altitude and energy counters (metrics `v.b.energy.used` & `v.b.energy.recd`), so the vehicle needs to provide these with reasonable precision & resolution.

The optional module plugin continuously records and stores the metrics history, so the chart can load the last 20 km (by default) when opened. The chart will then continue to add live metrics data. Without the module plugin, the chart will only display live data.

The chart is zoomable and can be panned along the X axis. The four data series can be selected and shown/hidden as usual by clicking on the series names in the legend, click into the chart to highlight a specific data point.

The live data may be more accurate than the data stored by the module plugin, as it will react directly to the odometer change, while the module plugin can only check the odometer once per second.

### 3.7.1 Installation

1. Install `pwrmon.htm` web plugin, recommended setup:

- Type: Page
- Page: `/usr/pwrmon`
- Label: Trip Power Chart
- Menu: Tools
- Auth: Cookie

2. Optionally:

- Save `pwrmon.js` as `/store/scripts/lib/pwrmon.js`
- Add line to `/store/scripts/ovmsmain.js`:  

```
- pwrmon = require("lib/pwrmon");
```

- Issue `script reload` or evaluate the `require` line

### 3.7.2 Configuration

No live config currently. You can customize the sample interval (default: 0.3 km) and the history size (default: 20 km) by changing the constants in the code. The storage file and interval (1 km) can also be changed.

On the web plugin, you can customize the live sample interval (default 0.3 km, doesn't need to match the module plugin) and the initial zoom (default: 8 km).

The persistent history storage file is `/store/usr/pwrmon.jx`. It needs ~9 kB with the default sample configuration. If space is tight on your `/store` partition you can change the file location to `/sd/...` in the source code or disable the file by setting an empty string as the filename.

### 3.7.3 Plugin API

<pre>pwrmon.dump()</pre>	<code>-- dump (print) recorded history data in JSON format</code>
<pre>pwrmon.data()</pre>	<code>-- get a copy of the history data object</code>

## 3.8 RegenMon: Regen Brake Monitor

**Live monitoring for regenerative brakelight & acceleration level**

- `regenmon.htm`

**Details:** [https://docs.openvehicles.com/en/latest/components/ovms\\_webserver/docs/regenmon.html](https://docs.openvehicles.com/en/latest/components/ovms_webserver/docs/regenmon.html)

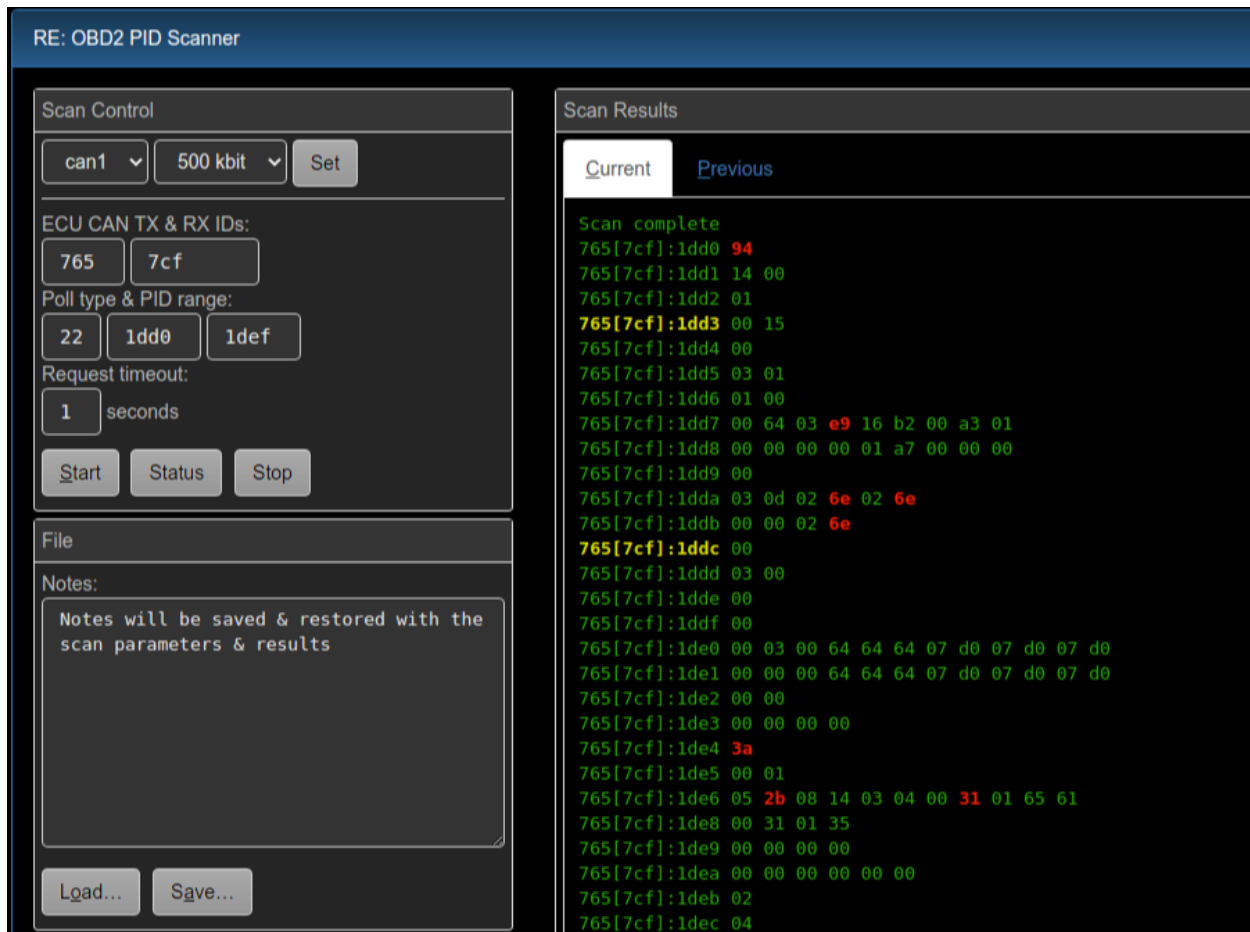
## 3.9 REPIDScan: OBD2 PID Scanner UI

**Web UI for the OVMS3 RE toolkit's OBD2 PID scanner**

Version 2.1 by Michael Balzer <[dexter@dexters-web.de](mailto:dexter@dexters-web.de)>

- 1.0: Initial release
- 2.0: Highlight differences between scan results
- 2.1: PID step support





This is a web frontend for the `re obdii scan` commands.

Some basic usage notes are included, but you really should have some knowledge of how accessing ECU and other onboard devices via OBD-II/UDS works. Some pointers to get started:

- [https://en.wikipedia.org/wiki/OBD-II\\_PIDs](https://en.wikipedia.org/wiki/OBD-II_PIDs)
- [https://en.wikipedia.org/wiki/Unified\\_Diagnostic\\_Services](https://en.wikipedia.org/wiki/Unified_Diagnostic_Services)

You will need OVMS firmware release 3.2.015-324 or higher to be able to use all features.

The UI allows to easily define and keep multiple scans along with notes. Scan results of two successive runs can be compared directly. The UI checks if vehicle `NONE` is loaded and shows a button to switch to that otherwise. The CAN buses can be started from the UI as well.

Scan results are shown in plain text form as delivered by the underlying commands.

Feel free to improve and extend (and submit your results).

### 3.9.1 Installation

1. Install `repidscan.htm` web plugin, recommended setup:

- Type: Page
- Page: `/usr/repidscan`
- Label: RE PID Scanner

- Menu: Tools
- Auth: Cookie

### 3.9.2 Usage

**Warning:** The scanner allows to use any poll type! Requests will be sent with PIDs only though, but some devices may fail to validate the request length, so be careful not to use any write or control poll types.

The OBD2 PID scanner performs a series of OBD/UDS requests for a range of PIDs and displays the results. Only PIDs with positive results (responses) will be shown in the results.

Enter all values hexadecimal (case irrelevant). Default RXID is TXID+8, try RXID range 500-7FF if you don't know the responding ID. To send broadcast requests, set TXID to 7DF and RXID to 7E8-7EF. The PID range must match the poll type PID constraints (8/16 bit).

Click "Start" to run a scan. Only one scan can be active at a time, but you don't need to stop a scan to start another run.

Scan results will be shown automatically when the scan has been completed. Scanning a large range of PIDs may take some time, to get intermediate results, click "Status". You can also start the scan, do something else and return to the tool later.

Differences between two scans are highlighted in both result tabs, with added PIDs being marked green (yellow in night mode), and byte value changes being marked red. No highlighting is applied if the scans don't share any PIDs.

On the next scanner start, the previous results (if any) will be copied to the "Previous" tab. You can switch tabs by the mouse or keyboard to compare them and look for differences, e.g. after having changed some control on the car.

All inputs and outputs can be saved to and loaded from files on the module or SD card. Use this to define your areas of interest, so you can easily re-run a previously defined scan the next day. File format is plain readable JSON, so you can continue analysis or prepare scans offline.

Notes: to avoid clashing of scan polls with vehicle polls, use the scanner only with the vehicle module NONE. The CAN bus to use needs to have been started in active mode. The scanner does not send session or tester presence messages, if you need these, use the `re obdii tester` tool or the `obdii canX request` command.

## 3.10 REtools: Reverse Engineering UI

### Preliminary web UI for the OVMS3 RE toolkit

Version 0.1 by Michael Balzer <dexter@dexters-web.de>

Reverse Engineering

Control

Start Stop

Set mode:

Discover Analyse

Serve
Clear:

all discovered

changed
Status

Mode: Discovering  
Key Map: 23 entries  
0 keys are ignored  
0 keys are new and changed  
8 bytes are changed  
0 keys are new and discovered  
8 bytes are discovered

Show 10 entries
Search:

Key	Records	Interval	Last data	ASCII
can1/155	37755	9	04 98 22 54 97 a0 00 73	.. "T...s
can1/423	3705	99	03 32 ff ff 00 e0 00 e8	.2.....
can1/424	3706	99	11 40 05 26 3f 51 00 40	.0.&?Q.0
can1/425	3316	99	0a 2f 44 ff fe 72 01 37	./D..r.7
can1/426	2394	99	00 38 01 00 eb 16 00	.8.....
can1/436	3708	99	00 1d 96 85 00 00	.....
can1/554	378	997	40 40 40 40 40 3f 00	00000??.
can1/556	3776	99	33 a3 37 33 83 37 33 8a	3.73.73.
can1/557	371	997	33 73 37 33 a3 3a 33 b0	3s73.:3.
can1/55e	378	997	33 a3 3a 33 a3 3a 17 82	3.:3:...

Showing 1 to 10 of 23 entries

Previous 1 2 3 Next

This is a first implementation of a web frontend for the `re` commands.

You'll need some understanding of the `re` system, the UI lacks user guidance. You can read about the `re` system on the developer list.

Feel free to improve and extend (and submit your results ;-)).

### 3.10.1 Installation

1. Install `retools.htm` web plugin, recommended setup:

- Type: Page
- Page: `/usr/retools`
- Label: RE Toolkit
- Menu: Tools
- Auth: Cookie

## 3.11 Renault Twizy

### 3.11.1 Dashboard-Tuneslider

Add recuperation control to the dashboard

- `dashboard-tuneslider.htm`

**Details:** [https://docs.openvehicles.com/en/latest/components/ovms\\_webserver/docs/plugin-twizy/dashboard-tuneslider.html](https://docs.openvehicles.com/en/latest/components/ovms_webserver/docs/plugin-twizy/dashboard-tuneslider.html)

### 3.11.2 eDriver BMS Monitor

Status monitor for the Twizy custom eDriver BMS by Pascal Ripp

Version 2.0 by Michael Balzer <dexter@dexters-web.de>

Name	Value
Status	CHARGING
Error	INTERLOCK ERROR <span style="color: red;">⚠</span>
Temperature	77 °C <span style="color: red;">⚠</span>
AUX Relay	ON
SOC	70.0 %
Voltage	57.4 V
Current	58.3 A
Coulomb used	0.01 Ah
Coulomb recd	0.00 Ah
Energy used	0.00 kWh
Energy recd	0.00 kWh

#	A	Temp	Dev	Alert
1		29 °C	-2.0 °C	<span style="color: orange;">?</span>
2		30 °C	-1.0 °C	
3		34 °C	3.0 °C	<span style="color: red;">⚠</span>

Clear Alerts

This is a simple web page plugin to display the standard battery cell info along with the additional custom data provided by the eDriver BMS on one page.

Additional custom data:

- Main state
- Error state
- BMS internal temperature
- AUX relay state
- Cell balancing states

Note: these are read from their respective custom metrics `xrt.bms...`, see [Custom Metrics](#).

Overall SOC, voltage, current and coulomb/energy counters are also shown to simplify battery capacity and current sensor calibration.

Check out the eDriver BMS Manual for details or contact Pascal in case of questions.

### Installation

1. Install `edrvmon.htm` web plugin, recommended setup:

- Type: Page
- Page: `/usr/edrvmon`
- Label: eDriver BMS Monitor
- Menu: Vehicle
- Auth: Cookie

### History

- V2.0 – Added cell balancing time totals + SOC & voltage use/charge range
- V1.0 – Initial release

### 3.11.3 Page-Command

only command options as UI

Version 0.2 Matthias Greiling <matthias@greiling.de>



#### Details:

This view only presents commands to OVMS. While driving you can choose between different drive modes. The coloured area indicates the active drive mode. Activate an other mode by touching or clicking the grey bordered area. To change again simply click or touch again the same or another ares. Furthermore you can limit the speed online to predefined values. The speed limit signs are common in germany and display km/h. Just activate the sign you want to be limited to - another touch will deactivates the limit of the chosen one.

#### Installation

##### 1. Install `page-command.html` web plugin, recommended setup:

- Type: Page
- Page: `/usr/command`
- Label:
- Menu: Main
- Auth: Cookie

#### History

- V0.2 – Simplfy backgrounds/ icons of command row
- V0.1 – Initial release

### 3.11.4 WifiConsole

#### Backend for WifiConsole hardware extension

- `WifiConsole.js`

**Details:** <https://github.com/dexterbg/WifiConsole>

## 3.12 Installing Module Plugins

A module plugin normally consists of a single Javascript file that needs to be placed in the `/store/scripts/lib/` directory. The plugin is then loaded by a `require()` call, which can be done manually as needed or automatically on boot by adding it to the `/store/scripts/ovmsmain.js` file.

1. Menu **Tools** → **Editor**
2. Cancel the open dialog
3. Paste the plugin source into the editor
4. *Save as...* → `/store/scripts/lib/...` using the name as shown in the plugin documentation
5. *Open* → (one level up) → `ovmsmain.js`
6. Add the `require()` call as shown in the plugin documentation
7. *Save*
8. *Reload JS Engine*

The plugin is now installed and activated.

**Hint:** you can now try out the commands provided by the plugin directly from the editor. Clear the editor, fill in the command or Javascript snippet, click *Evaluate JS*. The output will be shown below the input field.

To deactivate a plugin, comment out the `require()` call by prefixing the line with `//` and do another JS engine reload. To remove a plugin, remove the `require()` call and delete the file using `vfs rm /store/scripts/lib/....`

## 3.13 Installing Web Plugins

A web plugin normally consists of a single HTML file. Web plugins can be meant to render new pages or to hook into an existing page. They need an initial registration to work:

1. Menu **Config** → **Web Plugins**
2. (add)
3. Set type and name as suggested in the plugin documentation
4. *Save*
5. *Edit*
6. Set attributes as suggested in the plugin documentation
7. Paste the plugin source into the content area
8. *Save*

The plugin is now installed and activated.

**Hint:** you can use the text editor (tools menu) to change or update an already installed web plugin. Simply edit the plugin file in folder `/store/plugin/` directly, the system will reload the plugin as soon as you save it.

To deactivate a web plugin, set the state to “off”. To remove a plugin, click → *Save*.





Vehicle Type: **BMW I3**

This vehicle type supports the BMW i3 and i3s models. All model years should be supported.

The OVMS support was developed Jan 2021.

It was developed against the author's 2018 120Ah i3s BEV. I would welcome engagement from the owner of a REX type to further develop metrics related to the REX engine. Testing by drivers of LHD models, as well as those with the smaller batteries will also be helpful.

As of this release this vehicle support is read-only and cannot send commands to the car.

### 4.1 Support Overview

“tba” item are still on the to-do list and are not currently supported.

Function	Support Status
Hardware	Any OVMS v3 (or later) module.
Vehicle Cable	OBD-II cable: left-handed cable worked best in my RHD car
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
Cabin Pre-heat/cool Control	tba
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	tba
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	tba
Lock/Unlock Vehicle	tba
Valet Mode Control	No
Others	12v battery voltage/current, battery true SOC, etc

## 4.2 WARNINGS

### 4.2.1 Alarm behaviour

As standard, the i3 will sound the alarm if anything is left connected to the OBD-II port when the car is locked.

A tool like Bimmercode will allow you to disable this. Alternatively you will need to disconnect the OVMS unit before locking the car.

A future version may add a command to allow you to disable this alarm directly from your OVMS shell.

### 4.2.2 12V Battery drain

The i3 has a small 20Ah AGM 12v battery. Whilst care has been taken to minimize OVMS' power usage, OVMS could eventually drain this battery if the car is left unplugged and locked. OVMS will also send an alert if 12V drops under 12V alert threshold. (See 12V Calibration section).

HOWEVER: If you are going to leave the car for a few days, it is recommended to unplug OVMS.

## 4.3 Car status

The car is accessible over the OBD-II port when it is running (ignition on) and for a short time (40 seconds or so) after it is turned off or the car is "tweaked" (lock button pushed, connected-drive command received, etc).

Unfortunately this means that when your car is standing or charging OVMS only has intermittent access to data from the car.

By observation, whilst the car is charging it wakes up now and then (seems to be every 30 minutes). So at those times we can update our SOC etc.

Metrics “v.e.awake” tells you if the car is awake or not. Metric “xi3.s.age” will tell you how many minutes have passed since we last received data from the car.

You may also refer to metric xi3.s.pollermode as follows:

Mode	Meaning
0	Car is asleep - no OBD-II data traffic
1	Car OBD-II is awake - we are seeing data traffic
2	Car is ready to drive or driving
3	Car is charging

## 4.4 Custom metrics

Metric name	Example value	Description
xi3.s.age	5Min	How long since we last got data from the car
xi3.s.pollermode	0	OBD-II polling mode as explained above
xi3.v.b.p.ocv.avg	4.0646V	Main battery pack - average open-circuit voltage
xi3.v.b.p.ocv.max	4.067V	Main battery pack - highest open-circuit voltage
xi3.v.b.p.ocv.min	4.063V	Main battery pack - lowest open-circuit voltage
xi3.v.b.range.bc	245km	Available range per trip computer (based on current driving mode and style)
xi3.v.b.range.comfort	217km	Available range if you use Comfort mode
xi3.v.b.range.ecopro	245km	Available range if you use EcoPro mode
xi3.v.b.range.ecoproplus	247km	Available range if you use EcoPro+ mode
xi3.v.b.soc.actual	85%	Actual physical state-of-charge of the main battery pack
xi3.v.b.soc.actual.highlimit	93.7%	Highest physical charge level permitted (shown as 100% SOC)
xi3.v.b.soc.actual.lowlimit	10.5%	Minimum physical charge level permitted (shown as 0% SOC)
xi3.v.c.chargecablecapacity	0A	Maximum power capacity of connected charge cable per the charging interface
xi3.v.c.chargeledstate	0	Colour of the “ring light” on the charging interface.
xi3.v.c.chargeplugstatus	Not connected	Charging cable connected?
xi3.v.c.current.dc	0A	Power flowing on the DC side of the AC charger
xi3.v.c.current.dc.limit	0.100003A	Limit
xi3.v.c.current.dc.maxlimit	16A	Maximum limit
xi3.v.c.current.phase1	0A	Power being drawn on AC phase 1
xi3.v.c.current.phase2	0A	Power being drawn on AC phase 2
xi3.v.c.current.phase3	0A	Power being drawn on AC phase 3
xi3.v.c.dc.chargevoltage	0V	Voltage seen on the DC charger input
xi3.v.c.dc.contactorstatus	open	DC contactor state (closed implies we are DC charging)
xi3.v.c.dc.controlsymbols	0	DC charger control signals (always see 0?)
xi3.v.c.dc.inprogress	no	DC charging in progress?
xi3.v.c.dc.plugconnected	no	Is DC charger plug connected (doesn't seem to work)
xi3.v.c.deratingreasons	0	Reasons why charging rate is derated
xi3.v.c.error	0	Charging error codes
xi3.v.c.failsafetrigger	0	Failsafe trigger reasons
xi3.v.c.interruptionreasons	0	Charging interruption reasons
xi3.v.c.pilotsignal	0A	Charge rate pilot signal being received from EVSE
xi3.v.c.readytocharge	no	Are we ready to charge
xi3.v.c.temp.gatedriver	40°C	Charger gatedrive mosfet temperature
xi3.v.c.voltage.dc	8.4V	Charger output DC voltage being seen (for AC charging, not DC)
xi3.v.c.voltage.dc.limit	420V	Maximum permitted DC voltage

Continued on next page

Table 1 – continued from previous page

Metric name	Example value	Description
xi3.v.c.voltage.phase1	0V	Voltage seen on AC charger input phase 1
xi3.v.c.voltage.phase2	0V	Voltage seen on AC charger input phase 2
xi3.v.c.voltage.phase3	0V	Voltage seen on AC charger input phase 3
xi3.v.d.chargeport.dc	no	Is the charger port DC cover open (doesn't seem to work)
xi3.v.e.autorecirc	no	Ventilation is in "auto-recirculate" mode
xi3.v.e.obdtraffic	no	Are we seeing OBD-II frames from the car?
xi3.v.p.tripconsumption	127Wh/km	Average consumption for the current or most recent trip
xi3.v.p.wheel1_speed	0km/h	Wheel 1 speed
xi3.v.p.wheel2_speed	0km/h	Wheel 2 speed
xi3.v.p.wheel3_speed	0km/h	Wheel 3 speed
xi3.v.p.wheel4_speed	0km/h	Wheel 4 speed
xi3.v.p.wheel_speed	0km/h	Average wheel speed

## 4.5 To be researched

Can we start/stop charging?

Can we pre-heat?

Can we lock/unlock the car?

Can we disable the OBD-II alarm

Still looking for the trip regen kWh

Can we get the voltage state of each individual cells rather than just the battery min / max / average?

## DBC Based Vehicles

Vehicle Type: **DBC**

The DBC based vehicle reads a specified DBC file describing CAN bus messages and produces vehicle metrics from that. It is under development, experimental, and not generally available.

### 5.1 Support Overview

Function	Support Status
Hardware	This is vehicle specific. The DBC vehicle module can be configured to use any or all of the 3 CAN buses.
Vehicle Cable	Vehicle specific
GSM Antenna	Standard GSM antenna
GPS Antenna	Vehicle specific. Standard OVMS GPS supported.
SOC Display	Yes, if DBC maps it
Range Display	Yes, if DBC maps it
GPS Location	Yes, if DBC maps it, otherwise OVMS GPS
Speed Display	Yes, if DBC maps it
Temperature Display	Yes, if DBC maps it
BMS v+t Display	Not currently supported
TPMS Display	Yes, if DBC maps it
Charge Status Display	Yes, if DBC maps it
Charge Interruption Alerts	Yes, if DBC maps it
Charge Control	Not supported by DBC format, maybe by extension
Cabin Pre-heat/cool Control	Not supported by DBC format, maybe by extension
Lock/Unlock Vehicle	Not supported by DBC format, maybe by extension
Valet Mode Control	Not supported by DBC format, maybe by extension
Others	None

## 5.2 Contents

### 5.2.1 DBC Introduction

DBC is a CAN data description file format introduced by [Vector Informatik GmbH](#). DBC files are text files so can be created and edited using a simple text editor like the one built into the OVMS web UI.

DBC files can be used to support vehicles that don't have a dedicated native adaption yet. This is done using the generic DBC vehicle type, which can use DBC files to translate CAN data into metrics.

This section tries to show you how to create and use a DBC specification on the OVMS. Of course you'll need to know how to decode your vehicle's CAN frames first. You can use the OVMS RE (reverse engineering) toolkit to identify which messages are on the bus and which changes correlate to actions and status on the car.

There's also a very good general introduction to DBC from CSS Electronics including an explanatory video: <https://www.csselectronics.com/screen/page/can-dbc-file-database-intro/language/en>

DBC files only specify the passive (reading) part, they don't provide a means to define transmissions. If you need to send frames to request certain information, you can still use the OVMS DBC engine to decode the results (instead of doing the decoding in C++ code). So a DBC file can be used as a base for a real vehicle module. To request OBD2 data during development, you can use the `re obdii` commands.

### Basic Example

**Warning:** The DBC support is usable, but considered to be an alpha release. The way it works may change in the future.

This example is taken from the **Twizy**, which sends some primary BMS status data at CAN ID 0x155 (341). Note: the Twizy module actually does not use DBC, this is just an example how you would decode this message if using DBC.

**Message example:** 05 96 E7 54 6D 58 00 6F

- Byte 0: charge current limit [A], scaled by 5, no offset: 05 → **25A**
- Bytes 1+2: momentary battery current [A], big endian, lower 12 bits, scaled by -0.25, offset +500: \_6 E7 → **58.25A**
- Bytes 4+5: SOC [%], big endian, scaled by 0.0025, no offset: 6D 58 → **69.98%**

These can be translated to metrics directly:

- Charge current limit: `v.c.climit`
- Momentary battery current: `v.b.current`
- Battery SOC: `v.b.soc`

### Create DBC File

**Copy & paste** the following into a new editor window:

```
VERSION "DBC Example 1.0"

BS_: 500000 : 0,0

BO_ 341 BMS_1: 8 Vector__XXX
```

(continues on next page)

(continued from previous page)

SG_ v_c_climit	:	7 8@0+	(5,0)	[0 35]	"A"	Vector__XXX
SG_ v_b_current	:	11 12@0+	(-0.25,500)	[-500 1000]	"A"	Vector__XXX
SG_ v_b_soc	:	39 16@0+	(0.0025,0)	[0 100]	"%"	Vector__XXX

**What does this mean?**

- BS\_ defines the bus speed (500 kbit in this example) and bit timings (unused)
- BO\_ defines the data object, a CAN frame of length 8 with ID 341 (0x155) ("BMS\_1" is just an arbitrary name)
- SG\_ lines define the signals (values) embedded in the object (see below)
- Vector\_\_XXX is just a placeholder for any sender/receiver (currently unused by the OVMS)

**Signals** are defined by their...

- Name (= metric name with . replaced by \_)
- Start position (bit position) and bit length (7|8)
- Endianness: 0 = big endian (most significant byte first), 1 = little endian (least significant byte first) (Note: the DBC format documentation is wrong on this)
- Signedness: + = unsigned, - = signed
- Scaling and offset:  $(-0.25, 500) \Rightarrow \text{real value} = \text{raw value} * -0.25 + 500$
- Minimum/maximum:  $[-500|1000]$  = valid real values are in the range -500 to 1000
- Unit: e.g. "A" (ignored/irrelevant, defined by the metric)

The metric to set can be given as the name of the signal. You may use the metric name directly on the OVMS, but to conform to the DBC standard, replace the dots by \_.

**Bit positions** are counted from byte 0 upwards by their significance, regardless of the endianness. The first message byte has bits 0...7 with bit 7 being the most significant bit of the byte. The second byte has bits 8...15 with bit 15 being the MSB, and so on:

```
[ 7 6 5 4 3 2 1 0 ] [ 15 14 13 12 11 10 9 8 ] [ 23 22 21 20 ...
`----- Byte 0 -----` `----- Byte 1 -----` `----- Byte 2 ...`
```

For big endian values, signal start bit positions are given for the most significant bit. For little endian values, the start position is that of the least significant bit.

**Note: On endianness:** "big endian" means a byte sequence of 0x12 0x34 decodes into 0x1234 (most significant byte first), "little endian" means the same byte sequence decodes into 0x3412 (least significant byte first). "Big endian" is the natural way of writing numbers, i.e. with their most significant digit coming first, "little endian" is vice versa. Endianness only applies to values of multiple bytes, single byte values are always written naturally / "big endian".

**Use DBC File**

**Save** the DBC example as: /store/dbc/twizy1.dbc (the directory will be created by the editor)

**Open the shell.** To see debug logs, issue `log level debug dbc-parser` and `log level debug v-dbc`.

Note: the DBC parser currently isn't graceful on errors, a **wrong DBC file may crash the module**. So you should only enable automatic loading of DBC files on boot when you're done developing and testing it.

So let's first try if the **DBC engine can parse** our file. The `dbc autoload` command loads all DBC files from the `/store/dbc` directory:

```
OVMS# dbc autoload
Auto-loading DBC files...
D (238062) dbc-parser: VERSION parsed as: DBC Example 1.0
D (238062) dbc-parser: BU_ parsed 1 nodes
D (238062) dbc-parser: BO_ parsed message 341
D (238072) dbc-parser: SG_ parsed signal v_c_climit
D (238072) dbc-parser: SG_ parsed signal v_b_current
D (238082) dbc-parser: SG_ parsed signal v_b_soc
```

Looks good. `dbc list` can tell us some statistics:

```
OVMS# dbc list
twizyl: DBC Example 1.0: 1 message(s), 3 signal(s), 56% coverage, 1 lock(s)
```

The coverage tells us how much of our CAN data bits are covered by signal definitions.

Now let's **load the file into the DBC vehicle**:

```
OVMS# config set vehicle dbc.can1 twizyl
Parameter has been set.

OVMS# vehicle module NONE
I (1459922) v-none: Generic NONE vehicle module

OVMS# vehicle module DBC
I (249022) v-dbc: Pure DBC vehicle module
I (249022) v-dbc: Registering can bus #1 as DBC twizyl
```

Nice. Let's **simulate receiving our test frame** and check the decoded metrics:

```
OVMS# can can1 rx standard 155 05 96 E7 54 6D 58 00 6F
OVMS# me li v.b.soc
v.b.soc                                69.98%
OVMS# me li v.b.current
v.b.current                            58.25A
OVMS# me li v.c.climit
v.c.climit                             25A
```

So the decoding apparently works.

**To configure DBC mode for autostart** we now just need to set the DBC vehicle mode to be loaded on vehicle startup, and to enable autoloading of the DBC files from `/store/dbc`. You can do so either by using the user interface page [Config](#) → [Autostart](#) (check “Autoload DBC files” and set the vehicle type to “DBC”), or from the shell by...

```
OVMS# config set auto dbc yes
OVMS# config set auto vehicle.type DBC
```

Try a reboot to see if everything works.

You can now add more objects and signals to your DBC file.

---

**Note:** During development of a DBC file, you'll need to reload the file frequently. The DBC engine locks the currently used vehicle, so you'll need to unload the DBC vehicle (`vehicle module NONE`), then reload the DBC file (`dbc autoload`), then reactivate the DBC vehicle (`vehicle module DBC`).

---



## DEMO Vehicle

Vehicle Type: **DEMO**

The demonstration vehicle can be used to verify module functionality, but is mostly used for development purposes. The vehicle itself updates system metrics with simulated data. It can respond to charge, and other similar, commands.

### 6.1 Support Overview

Function	Support Status
Hardware	Demonstration data only, no vehicle connection
Vehicle Cable	None, not required (other than power)
GSM Antenna	Standard GSM antenna
GPS Antenna	None, not required
SOC Display	Yes
Range Display	Yes
GPS Location	Yes
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Not currently supported
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Yes
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Yes
Valet Mode Control	Yes
Others	None



Vehicle Type: **FT5E**

The Fiat 500e vehicle support will be documented here.

## 7.1 Support Overview

Function	Support Status
Hardware	tba
Vehicle Cable	tba
GSM Antenna	tba
GPS Antenna	tba
SOC Display	tba
Range Display	tba
GPS Location	tba
Speed Display	tba
Temperature Display	tba
BMS v+t Display	tba
TPMS Display	tba
Charge Status Display	tba
Charge Interruption Alerts	tba
Charge Control	tba
Cabin Pre-heat/cool Control	tba
Lock/Unlock Vehicle	tba
Valet Mode Control	tba
Others	tba



---

### Hyundai Ioniq vFL

---

#### Hyundai Ioniq Electric (28 kWh)

- Vehicle Type: **HIONVFL**
- Log tag: `v-hyundaivfl`
- Namespace: `xhi`
- Maintainers: [Michael Balzer](#)
- Sponsors: [Henri Bachmann](#)
- Credits: [EVNotify](#)

## 8.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	No
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Partial
BMS v+t Display	Yes
SOH Display	Yes
TPMS Display	No
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	

## 8.2 Custom Metrics

Metric name	Example value	Description
xhi.b.soc.bms	78.5%	Internal BMS SOC
xhi.c.state	128	Charge state flags

## 8.3 Debug Logs

To see all PID poll results in your log, set log level `verbose` for component `v-hyundaiivfl`.

Vehicle Type: **KN**

The Kia e-Niro vehicle support will be documented here.

## 9.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module
Vehicle Cable	9658635 Kia OBD-II to DB9 Data Cable for OVMS
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes (based on consumption of past 20 trips longer than 1km)
GPS Location	Yes
Speed Display	Not currently supported
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Not currently supported
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Yes
Valet Mode Control	Not currently supported
Others	ODO Not currently supported

### 9.2 OBD-II cable

The Kia e-Niro have one CAN-bus available on the OBD-II port: D-can. You can use the standard OBD-II to DB9 cable from Fasttech.

In case you want to build your own cable, here's the pinout:

J1962-M	DB9-F	Signal
4	3	Chassis / Power GND
6	7	CAN-0H (C-can High)
14	2	CAN-0L (C-can Low)
16	9	+12V Vehicle Power

A simple approach is to buy an OBDII (J1962-M) pigtail, and solder the DB9-F connector end appropriately.

### 9.3 Configuration

TODO. Configuration is quite similar to the Kia Soul, so please check that out. Please use the Web based configuration!

### 9.4 Estimated Range

Currently, there is no known way to get the estimated range directly from the car, so the estimated range is calculated by looking at the consumption from the last 20 trips that are longer than 1 km.

### 9.5 12V battery drain

OVMS will eventually drain the 12V battery, but steps have been taken to minimize the drain. However, if you are going to leave the car for a few days, it is recommended to unplug OVMS. OVMS will send an alert if 12V drops under 12V alert threshold. See 12V Calibration section.



## CHAPTER 10

---

### Kia Soul EV

---

Vehicle Type: **KS**

The Kia Soul EV vehicle support will be documented here.

### 10.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module
Vehicle Cable	9658635 Kia OBD-II to DB9 Data Cable for OVMS
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes
Speed Display	?
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Not currently supported
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Yes
Valet Mode Control	Not currently supported
Others	

## 10.2 OBD-II cable

The Kia Soul EV have two different CAN-busses available on the OBD-II port: C-can and M-can.

C-can is the main can-bus and M-can is the multimedia bus. The latter one is not necessary for OVMS, but some metrics are fetched from the M-bus and these metrics will be empty if you don't have the proper cable. The standard OBD-II to DB9 cable from Fasttech supports only C-can, so make sure you buy the Kia Soul specific one.

In case you want to build your own cable, here's the pinout:

J1962-M	DB9-F	Signal
1	5	CAN-1H (M-can High)
4	3	Chassis / Power GND
6	7	CAN-0H (C-can High)
9	4	CAN-1L (M-can Low)
14	2	CAN-0L (C-can Low)
16	9	+12V Vehicle Power

A simple approach is to buy an OBDII (J1962-M) pigtail, and solder the DB9-F connector end appropriately.

## 10.3 Configuration

There are a few Kia Soul EV specific settings that you have to consider in order to get the most out of OVMS. These are battery size, real life ideal range, remote command pincode and charge port remote control.

## 10.4 Battery size

Up until the 2018 version of Kia Soul, all models had a 27kWh battery. The 2018-version have a 30kWh battery. OVMS is by default set up with 27000Wh, but you can change this configuration to fit your car using this command in the OVMS-shell:

```
config set xks cap_act_kwh 27000
```

NB! Even though it says cap\_act\_kwh, the number must be in Wh.

## 10.5 Real life ideal range

Even though the Kia Soul EV is equipped with a pretty good and conservative GOM, most experienced drivers know how far the car can go on a charge. This is what we call the ideal range. You can set the ideal range in kilometers, experienced at 20 degrees celsius, by using this command:

```
config set xks maxrange 160
```

This setting is set to 160 km by default, and matches the author driving a 2015 Kia Soul Classic in 20 degrees in southern Norway. Your mileage may vary, so please set it accordingly.

The ideal range, as shown in the OVMS APP, are then derived from this number, multiplied by the state of charge and adjusted using a combination of the outside temperature and the battery temperature.

## 10.6 Open charge port using key fob

By default, OVMS listens for the the third button on the key fob and opens the charge port if Pressed. If you don't want this behaviour, you can disable it by using this command:

```
config set xks remote_charge_port 0
```

## 10.7 Security pin code for remote commands

Remote commands like lock and unlock doors, among others, require a pin code. This pin code can be set using the web configuration or using this command:

```
config set password pin 1234
```

Please set this pin as soon as possible.

## 10.8 Soul specific metrics

NB! Not all metrics are correct or tested properly. This is a work in progress.

There are a lot of extra metrics from the Kia Soul. Here's the current ones:

Metric	Description
xks.b.cell.volt.max	The highest cell voltage
xks.b.cell.volt.max.no	The cell number with the highest voltage
xks.b.cell.volt.min	The lowest cell voltage
xks.b.cell.volt.min.no	The cell number with the lowest voltage
xks.b.cell.det.max	The highest registered cell deterioration in percent.
xks.b.cell.det.max.no	The cell with the highest registered deterioration.
xks.b.cell.det.min	The lowest registered cell deterioration in percent.
xks.b.cell.det.min.no	The cell with the lowest registered deterioration.
xks.b.min.temp	The lowest temperature in the battery
xks.b.max.temp	The highest temperature in the battery
xks.b.inlet.temp	The air temperature at the air inlet to the battery
xks.b.heat1.temp	The temperature of the battery heater 1
xks.b.heat2.temp	The temperature of the battery heater 2
xks.b.bms.soc	The internal state of charge from BMS
xks.c.power	Charge power in kW.
xks.c.speed	The charge speed in kilometer per hour.
xks.ldc.out.volt	The voltage out of the low voltage DC converter.
xks.ldc.in.volt	The voltage into the low voltage DC converter.
xks.ldc.out.amps	The power drawn from the low voltage DC converter.
xks.ldc.temp	The temperature of the LDC.
xks.obc.pilot.duty	The duty cycle of the pilot signal
xks.e.lowbeam	Low beam on/off
xks.e.highbeam	High beam on/off
xks.e.inside.temp	Actual cabin temperature
xks.e.climate.temp	Climate temperature setting
xks.e.climate.driver.only	Climate is set to driver only
xks.e.climate.resirc	Climate is set to recirculate

Continued on next page

Table 1 – continued from previous page

Metric	Description
xks.e.climate.auto	Climate is set to auto
xks.e.climate.ac	Air condition on/off
xks.e.climate.fan.speed	Climate fan speed
xks.e.climate.mode	Climate mode
xks.e.preheat.timer1.enabled	Preheat timer 1 enabled/disabled
xks.e.preheat.timer2.enabled	Preheat timer 2 enabled/disabled
xks.e.preheating	Preheating on/off
xks.e.pos.dist.to.dest	Distance to destination (nav unit)
xks.e.pos.arrival.hour	Arrival time, hour part (nav unit)
xks.e.pos.arrival.minute	Arrival time, minute part(nav unit)
xks.e.pos.street	Current street? Or Next street?
xks.v.seat.belt.driver	Seat belt sensor
xks.v.seat.belt.passenger	Seat belt sensor
xks.v.seat.belt.back.right	Seat belt sensor
xks.v.seat.belt.back.left	Seat belt sensor
xks.v.traction.control	Traction control on/off
xks.v.cruise.control.enabled	Cruise control enabled/disabled
xks.v.emergency.lights	Emergency lights enabled/disabled
xks.v.steering.mode	Steering mode: Sport, comfort, normal.
xks.v.power.usage	Power usage of the car
xks.v.trip.consumption.kWh/100km	Battery consumption for current trip
xks.v.trip.consumption.km/kWh	Battery consumption for current trip

Note that some metrics are polled at different rates than others and some metrics are not available when car is off. This means that after a restart of the OVMS, some metrics will be missing until the car is turned on and maybe driven for few minutes.

Climate and navigation-metrics are fetched from navigation unit and needs the Kia Soul compatible OBDII-cable.

## 10.9 Soul specific shell commands

There are a few shell commands made for the Kia Soul. Some are read only, others can enable functions and some are used to write directly to a ECU and must therefore be used with caution.

### 10.9.1 Read only commands

```
xks trip
```

Returns info about the last trip, from you put the car in drive (D or B) and til you parked the car.

```
xks tpms
```

Shows the tire pressures.

```
xks aux
```

Prints out the voltage level of the auxiliary battery.

```
xks vin
```

Prints out some more information taken from the cars VIN-number. Not complete.

## 10.9.2 Active Commands

`xks trunk <pin code>`

Opens up the trunk

`xks chargeport <pin code>`

Opens up the charge port

`xks ParkBreakService <on/off>`

Not yet working.

`xks IGN1 <on/off><pin>`

Turn on or off IGN1-relay. Can be used to wake up part of the car.

`xks IGN2 <on/off><pin>`

Turn on or off IGN2-relay. Can be used to wake up part of the car.

`xks ACC <on/off><pin>`

Turn on or off ACC-relay. Can be used to wake up part of the car.

`xks START <on/off><pin>`

Turn on or off START-relay. Can be used to wake up part of the car.

`xks headlightdelay <on/off>`

Turn on/off the “follow me home” head light delay function.

`xks onetouchturnsignal <0=Off, 1=3 blinks, 2=5 blinks, 3=7 blinks>`

Configure one touch turn signal settings.

`xks autodoorunlock <1=Off, 2=On vehicle off, 3=On shift to P, 4=On driver door unlock>`

Configure auto door unlock settings.

`xks autodoorlock <0=Off, 1=On speed, 2=On shift>`

Configure auto door unlock settings.

### 10.9.3 ECU-write commands

These commands are for the extra playful people. Use with caution.

```
xks s jb <b1><b2><b3>
```

Send command to smart junction box.

```
xks bcm <b1><b2><b3>
```

Send command to body control module.

## 10.10 12V battery drain

OVMS will eventually drain the 12V battery, but steps have been taken to minimize the drain. However, if you are going to leave the car for a few days, it is recommended to unplug OVMS.

# CHAPTER 11

## Maxus eDeliver3

Vehicle Type: **MED3**

The Maxus eDeliver3 will be documented here.

### 11.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	No
Temperature Display	Yes (External Temp and Battery)
BMS v+t Display	No
TPMS Display	No
Charge Status Display	No
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	





## CHAPTER 12

---

### Mercedes-Benz B250E W242

---

Vehicle Type: **SE**

The Mercedes-Benz B-Klasse Electric Drive will be documented here.

#### 12.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	No
Range Display	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	No
BMS v+t Display	No
TPMS Display	No
Charge Status Display	No
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	

## 12.2 Contents

### 12.2.1 Standard Metrics

Metric name	Notes
ms_v_mot_rpm	
ms_v_env_throttle	Resolution of 0.4 %
ms_v_pos_speed	Resolution of 0.1 km/h
ms_v_pos_odometer	Resolution of 0.1 km
ms_v_bat_12v_voltage	Resolution of 0.1 V
ms_v_bat_power	Car reports percents, so the kW is just calculated by multiplying by 132 kW. Negative side is saturated to -50 kW. This is just guess work..
ms_v_bat_range_est	
ms_v_bat_consumption	Instantaneous consumption, Wh/km
ms_v_tpms_*_p	
ms_v_env_cabinsetpoint	There are two figures, left and right. Using just one for now.

### 12.2.2 Custom Metrics

Metric name	Example value	Description
xmb.v.display.trip.reset	2733.2km	Dashbord Trip value since Reset
xmb.v.display.trip.start	17.5km	Dashbord Trip value since Start (today)
xmb.v.display.consumption.start	25.9 kWh	Dashbord Trip consumption since Start (today)
xmb.v.display.accel	89%	Eco score on acceleration over last 6 hours
xmb.v.display.const	18%	Eco score on constant driving over last 6 hours
xmb.v.display.coast	100%	Eco score on coasting over last 6 hours
xmb.v.display.ecoscore	69%	Eco score shown on dashboard over last 6 hours
xmb.v.fl_speed	37.83 km/h	Front Left wheel speed
xmb.v.fr_speed	37.85 km/h	Front Right wheel speed
xmb.v.rl_speed	37.80 km/h	Rear Left wheel speed
xmb.v.rr_speed	37.82 km/h	Rear Right wheel speed

Vehicle Type: **MGEV**

This vehicle type supports the MG ZS EV (2019-).

MG5 is not yet supported in this build.

## 13.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: 2019-
Vehicle Cable	Right hand OBDII cable (RHD), Left hand OBDII cable (LHD)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (or any compatible antenna)
SOC Display	Yes
Range Display	Yes (BMS calculated and WLTP range from SoC)
Cabin Pre-heat/cool Control	tba
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	tba
Lock/Unlock Vehicle	tba
Valet Mode Control	No
Others	tba

### 13.2 Development notes

To compile the code you will need to check out the repository, check out the components mongoose, libzip and zlib and copy the file

```
sdkconfig.default.hw31
```

from the OVMS.V3/support folder to the OVMS.V3 folder and rename it to

```
sdkconfig
```

### 13.3 Community documentation

This module is developed from the work provided by the My MG ZS EV community at <https://discourse.mymgzsev.com/> Please join the Slack channel for support and the latest builds.

### 13.4 Car status

The car is accessible over the OBD port when it is running (ignition on) and for around 40 seconds after it is turned off or the car is “tweaked” (lock button pushed, etc).

The OBD port may be kept awake by using the “tester present” message to the gateway ECU. This keeps a lot of systems awake and draws roughly 5A on the 12V bus, so it’s not a good idea to do.

The MGEV module now monitors (and automatically calibrates) the 12V status and will automatically switch on when the 12V exceeds 12.9V. When it does this it will try to poll the vehicle for data.

#### **There are 4 Poll States**

- 0 ListenOnly: the OVMS module is quiet and stops sending polls, it will enter this state after 50s of being < 12.9V
- 1 Charging: the OVMS module sends charging specific queries
- 2 Driving: the OVMS module sends driving specific queries
- 3 Backup: the OVMS module cannot get data from the car when it is charging so just retries SoC queries

The Gateway (GW, GWM) is the keeper of all the data of the car and will enter a locked state when it is woken by the car starting charging and the car is locked. This we have called “Zombie Mode”, and we have developed an override for this.

#### **This override, however causes a few strange things to happen:**

- If Zombie mode override is active, the car will not unlock the charge cable. To fix this disrupt the charge and wait 50s for OVMS to go back to sleep and the cable should release (or unplug OVMS)
- Zombie mode override resets the “Accumulated Total Trip” on the Cluster
- Zombie mode override sets the gearshift LEDs switch on

# CHAPTER 14

## Mitsubishi Trio

Vehicle Type: **MI**

This should be used to support the Mitsubishi i-Miev (Citroen C-Zero, Peugeot iOn) vehicles.

### 14.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes
Speed Display	Yes
Temperature Display	No
BMS v+t Display	Yes
TPMS Display	Not currently supported
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	No
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Not currently supported
Valet Mode Control	Not currently supported
Others	

## 14.2 Trio specific metrics

NB! Not all metrics are correct or tested properly. This is a work in progress.

Metric	Description
xmi.b.power.min	The battery minimum power usage
xmi.b.power.max	The battery maximum power usage
xmi.e.lowbeam	Low beam light status
xmi.e.highbeam	High beam light status
xmi.e.frontfog	Front fog light status
xmi.e.rearfog	Rear fog light status
xmi.e.rightblinker	Right blinker status
xmi.e.leftblinker	Left blinker status
xmi.e.warninglight	Warning light status
xmi.c.kwh.dc	Charge energy on DC side in kWh
xmi.c.kwh.ac	Charge energy on AC side in kWh
xmi.c.efficiency	Charge efficiency (DC/AC)
xmi.c.power.ac	Charge power on AC side in kW
xmi.c.power.dc	Charge power on DC side in kW
xmi.c.time	Charge time (h:mm:ss)
xmi.c.soc.start	Charge start soc in %
xmi.c.soc.stop	Charge stop soc in %
xmi.e.heating.amp	Heating energy usage in A
xmi.e.heating.watt	Heating energy usage in W
xmi.e.heating.temp.return	Heater return water temperature
xmi.e.heating.temp.flow	Heater flow water temperature
xmi.e.ac.amp	AC energy usage in A
xmi.e.ac.watt	AC energy usage in W
xmi.e.trip.park	Trip start odometer in km
xmi.e.trip.park.energy.used	Energy used in kWh
xmi.e.trip.park.energy.recuperated	Recuperated energy in kWh
xmi.e.trip.park.heating.kwh	Heater energy usage on trip in kWh
xmi.e.trip.park.ac.kwh	AC energy usage on trip in kWh
xmi.e.trip.park.soc.start	Trip start soc
xmi.e.trip.park.soc.stop	Trip stop soc
xmi.e.trip.charge	Trip start odometer in km since charge
xmi.e.trip.charge.energy.used	Energy used in kWh since charge
xmi.e.trip.charge.energy.recuperated	Recupe.. energy in kWh since charge
xmi.e.trip.charge.heating.kwh	Heating usage in kWh since charge
xmi.e.trip.charge.ac.kwh	AC energy usage in kWh since charge
xmi.e.trip.charge.soc.start	Trip start soc since charge
xmi.e.trip.charge.soc.stop	Trip stop soc since charge

Note that some metrics are polled at different rates than others and some metrics are not available when car is off. This means that after a restart of the OVMS, some metrics will be missing until the car is turned on and maybe driven for few minutes.

## 14.3 Custom Commands

### 14.3.1 Read only commands

`xmi trip`

Returns info about the last trip, from you start the car with key.

`xmi tripc`

Returns info about the trip since last charge.

`xmi aux`

Prints out the voltage level of the auxiliary battery.

`xmi vin`

Prints out some more information taken from the cars VIN-number.

## 14.4 Trio Regen Brake Light Hack

### 14.4.1 Parts required

- 1x DA26 / Sub-D 26HD Plug & housing \* Note: housings for Sub-D 15 fit for 26HD \* e.g. Assmann-WSW-A-HDS-26-LL-Z \* Encitech-DPPK15-BK-K-D-SUB-Gehaeuse
- 1x 12V Universal Car Relay + Socket \* e.g. Song-Chuan-896H-1CH-C1-12V-DC-Kfz-Relais \* GoodSky-Relaissocket-1-St.-GRL-CS3770
- 1x 1 Channel 12V Relay Module With Optocoupler Isolation \* 12V 1 channel relay module with Optocoupler Isolation

Car wire-tap connectors, car crimp connectors, 0.5 mm<sup>2</sup> wires, zipties, shrink-on tube, tools

Note: if you already use the switched 12V output of the OVMS for something different, you can use one of the free EGPIO outputs. That requires additionally routing an EGPIO line to the DA26 connector at the expansion slot (e.g. using a jumper) and using a relay module (2/b: relay shield) with separate power input instead of the standard car relay.

I use 2/b (relay shield) variant: Be aware the MAX71317 outputs are open drain, so you need a pull up resistor to e.g. +3.3. According to the data sheet, the current should stay below 6 mA.

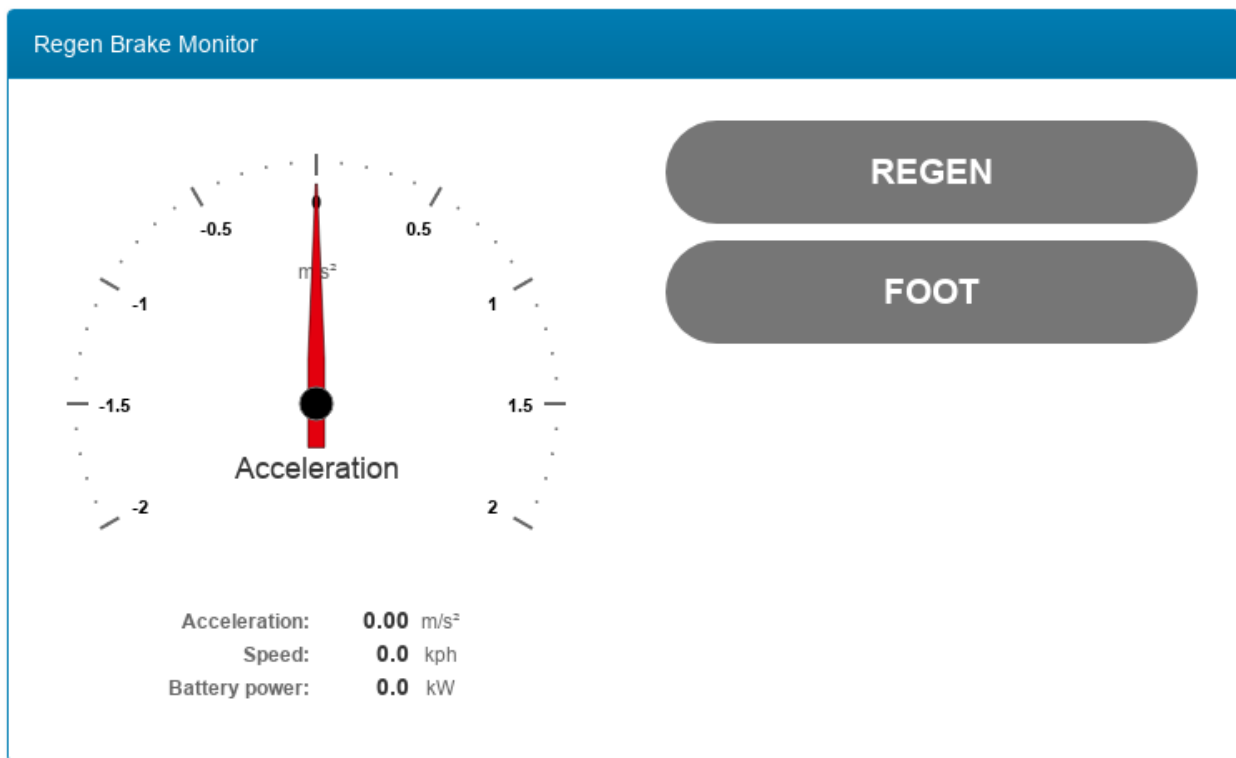
Inside OVMS Box: Connect JP1 Pin10 (GEP7) to Pin12 (EGPIO\_8) with jumper

In DA26 connector:

```
pin 24(+3.3) ----- [ 680 Ohms ] ---+---- [ Relay board IN ]
                                   |
                                   pin 21 (EGPIO_8)
pin 9 ----- [Relay board DC+]
pin 8 ----- [Relay board DC-]
[Relay board COM] ----- Brake pedal switch one side
[Relay board NO]  ----- Brake pedal switch other side
```

### 14.4.2 Configuration

See OVMS web user interface, menu Trio → Brake Light:



Set the port as necessary and the checkbox to enable the brakelight.

For monitoring and fine tuning, use the „regenmon“ web plugin: [https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3/blob/master/vehicle/OVMS.V3/components/ovms\\_webserver/dev/regenmon.htm](https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3/blob/master/vehicle/OVMS.V3/components/ovms_webserver/dev/regenmon.htm)



## Regen Brake Light

**Acceleration  
smoothing:**
    

Speed/acceleration smoothing eliminates road bump and gear box backlash noise.

Lower value = higher sensitivity. Set to zero if your vehicle speed is already smoothed.

☒ Enable regenerative braking signal

**... control port:**

**... activation level:**
 m/s<sup>2</sup>    

Deceleration threshold to activate regen brake light.

Under UN regulation 13H, brake light illumination is required for decelerations >1.3 m/s<sup>2</sup>.

**... deactivation level:**
 m/s<sup>2</sup>    

Deceleration threshold to deactivate regen brake light.

Under UN regulation 13H, brake lights must not be illuminated for decelerations ≤0.7 m/s<sup>2</sup>.

The OVMS generated regenerative braking signal needs a hardware modification to drive your vehicle brake lights. See your OVMS vehicle manual section for details on how to do this.

The regen brake signal is activated when the deceleration level exceeds the configured threshold, the speed is above 1 m/s (3.6 kph / 2.2 mph) and the battery power is negative (charging).

The signal is deactivated when deceleration drops below the deactivation level, speed drops below 1 m/s or battery power indicates discharging. The signal will be held activated for at least 500 ms.



# CHAPTER 15

## Nissan Leaf/e-NV200

Vehicle Type: **NL**

This vehicle type supports the Nissan Leaf (24kWh & 30kWh) and Nissan e-NV200 (24kWh & 40kWh).

### 15.1 Support Overview

#### 15.1.1 Hardware

Item	Support Status
Module	Any OVMS v3 (or later) module. Vehicle support: 2011-2017 (24kWh & 30kWh LEAF, 24kWh & 40KWh e-Nv200 & custom battery e.g Muxsan)
Vehicle Cable	1779000 Nissan Leaf OBD-II to DB9 Data Cable for OVMS
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)

#### 15.1.2 Controls

Function	Support Status
Charge Control	Start charge only (Stop charge in beta firmware stage)
Cabin Pre-heat/cool Control	Yes <sup>1</sup> (see info below)
Lock/Unlock Vehicle	Yes <sup>3</sup> (see info below)
Valet Mode Control	Not currently supported

<sup>1</sup> OVMS currently supports 2011-2017 Nissan LEAF and Nissan e-NV200

<sup>3</sup> Lock/Unlock will work if CAR can bus is awake, this can be activated by turning on A/C

### 15.1.3 Metrics

Item	Support Status
SOC	Yes (by default based on GIDS)[4]
Range	Yes (by default based on GIDS)
GPS Location	Yes (from modem module GPS)
Speed	Yes (from vehicle speed PID)
Cabin Temperature	Yes (from vehicle temperature PIDs)
Ambient Temperature	Yes (from vehicle temperature PIDs)
SetPoint Temperature	Yes (from vehicle hvac PIDs) <sup>2</sup>
HVAC Fan Speed	Yes (from vehicle hvac PIDs) <sup>2</sup>
HVAC Heating/Cooling Status	Yes (from vehicle hvac PIDs) <sup>2</sup>
HVAC On Status	Yes (from vehicle hvac PIDs) <sup>2</sup>
HVAC Temperature Setpoint	Yes (from vehicle hvac PIDs) <sup>2</sup>
HVAC Ventilation Mode	Yes (from vehicle hvac PIDs) <sup>2</sup>
BMS v+t	Yes
TPMS	Yes (If hardware available)
Charge Status	Yes
Charge Interruption Alerts	Yes

## 15.2 Remote Climate Control

### 15.2.1 2011-2013 models (ZE0)

Gen1 Cars (ZE0, 2011-2013) require a hardware modification to enable OVMS to control remote climate. Wire RC3 to TCU pin 11, [more info](#)

### 15.2.2 2014-2016 models (AZE0)

To use OVMS to activate remote climate the Nissan TCU (Telematics Control Unit) module must be unplugged if fitted (only on Acenta and Tekna models). The TCU is located behind the glovebox on LHD cars or on the right hand side of the drivers foot well on RHD cars. The large white plug on the rear of the TCU should be unplugged, push down tab in the middle and pull to unplug, [see video for RHD cars](#) and [this page for LHD cars](#).

Note: Unplugging the TCU will disable Nissan EV connect / CARWINGS features e.g Nissan mobile app. All other car functions will not be effected e.g GPS, maps, radio, Bluetooth, microphone all work just the same as before. OVMS can be used to more than substitute the loss of Nissan Connect features. The TCU can be plugged back in at any point in the future if required.

OVMS remote climate support will 'just work' on LEAF Visia models and Visia/Acenta e-NV200 since these models do not have a TCU fitted.

Note: If you prefer not to unplug the Nissan TCU, all OVMS functions appart from remote climate will function just fine alongside the Nissan TCU.

---

<sup>2</sup> Some HVAC Status Items have been only verified with 2013-2016 MY cars and will only work if the year is set in configuraiton. Also HVAC needs to be in ON position before powering down the vehicle for the metrics to work during pre-heat.

### 15.2.3 2016-2017 models (AZE0)

**Remote climate control will only work when plugged in and actively charging on 2016-2017 models.** This is because in 2016 Nissan moved the TCU from the EV CAN bus to the CAR CAN bus.

Set the model year as follows and if necessary configure 30 kWh model:

```
config set xnl modelyear 2016
```

or

```
config set xnl modelyear 2017
```

*Note: in latest OVMS firmware version model year and battery size can be set via the web config interface.*

### 15.2.4 2018+ models (ZE1)

2018+ 40/62kWh LEAF is not yet supported. Please get in touch if your interested in helping to add support. Relevant 2018 CANbus messages have already been decoded and documented, see [MyNissanLEAF thread](#).

### 15.2.5 Specific battery configs

For models with a 30 kWhr battery pack, set the capacity manually with:

```
config set xnl maxGids 356 config set xnl newCarAh 79
```

For models with a 40 kWhr battery pack, set the capacity manually with:

```
config set xnl maxGids 502 config set xnl newCarAh 115
```

For models with a 62 kWhr battery pack, set the capacity manually with:

```
config set xnl maxGids 775 config set xnl newCarAh 176
```

*Note: In latest OVMS firmware version, model year and battery size can be set via the web config interface. This is easier and also the preferred method.*

*Note 2: OVMS fully supports battery upgraded LEAFs, just set the capacity according to what battery is currently installed.*

## 15.3 Range Calculation

The OVMS uses two configuration options to calculate remaining range, whPerGid (default 80Wh/gid) and km-PerKWh (default 7.1km/kWh). The range calculation is based on the remaining gids reported by the LBC and at the moment does not hold 5% in reserve like LeafSpy. Feedback on this calculation is welcomed.

## 15.4 Resources

- Nissan LEAF support added by Tom Parker, see [his wiki](#) for lots of documentation and resources. Some info is outdated e.g climate control now turns off automatically.
- Nissan LEAF features are being added by Jaunius Kapkan, see [his github profile](#) to track the progress.
- [MyNissanLEAF thread](#) for Nissan CANbus decoding discussion
- Database files (.DBC) for ZE0 and AZE0 Leaf can be found here: [Github LEAF Canbus database files](#)

Assistance is appreciated as I haven't had time to try to override the TCU using the OVMS or find an alternative solution to prevent the TCU overriding the messages while still allowing the hands free microphone to work.

## CHAPTER 16

---

### OBDII Vehicles

---

Vehicle Type: **O2**

Support for generic OBDII is provided by this module.

## 16.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: Not widely tested, but should be all that support OBDII standard PIDs over CAN bus.
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes (based on fuel level PID)
Range Display	No
GPS Location	Yes (from modem module GPS)
Speed Display	Yes (from vehicle speed PID)
Temperature Display	Yes (from vehicle temperature PIDs)
BMS v+t Display	No
TPMS Display	No
Charge Status Display	No
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	VIN and RPMs should be available



## CHAPTER 17

---

Renault Twizy

---

Vehicle Type: **RT**

## 17.1 Support Overview

Function	Support Status
Hardware	No specific requirements (except for regen brake light control, see below)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	No
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Current limit, stop, SOC/range limit
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	Mapped to dynamic speed lock
Valet Mode Control	Mapped to odometer based speed lock
Others	<ul style="list-style-type: none"> <li>• Battery &amp; motor power &amp; energy monitoring</li> <li>• Battery health &amp; capacity</li> <li>• SEVCON monitoring &amp; tuning (Note: tuning only SC firmware &lt;= 0712.0002 / ~07/2016)</li> <li>• Kickdown</li> <li>• Auto power adjust</li> <li>• Regen brake light</li> <li>• Extended trip &amp; GPS logging</li> </ul>

Note: regarding the Twizy, V3 works very similar to V2. All telemetry data tables and tuning profiles are fully compatible, and the commands are very similar. Therefore, the basic descriptions and background info from the V2 manual are valid for V3 as well.

- [V2 FAQ \(english\)](#)
- [V2 FAQ \(german\)](#)
- [V2 documents directory](#)
- [V2 user manual \(PDF\)](#)

## 17.2 Hints on V3 commands vs. V2

- V3 commands need to be in lower case.
- When accessing via USB terminal, first issue enable (login).
- V3 commands are similar to V2 commands, just structured slightly different.

- Try `help`, `?` and `xrt ?`. Twizy commands are subcommands of `xrt`.
- TAB only works via USB / SSH. Shortcuts can be used generally, e.g. `mo t` instead of `module tasks`.
- A usage info is shown if a command syntax is wrong or incomplete.

## 17.3 Contents

### 17.3.1 Configuration

The Twizy configuration is stored in the OVMS config parameter (section) `xrt`. To list the current values, issue:

```
config list xrt
```

For reference, the following table includes the equivalent OVMS V2 feature or parameter number & bit value where applicable. You may use the parameter & feature editors on the App (values are mapped), the commands have been merged into the `config` command in V3.

Config instance name	Example value	V2 F/P	Description
autopower	yes	F15/8	Bool: SEVCON automatic power level adjustment (Default: yes)
autoreset	yes	F15/2	Bool: SEVCON reset on error (Default: yes)
aux_charger_port	0	–	EGPIO port to control auxiliary charger (Default: 0 = disabled)
aux_fan_port	0	–	EGPIO port to control auxiliary charger fan (Default: 0 = disabled)
canwrite	yes	F15/1	Bool: CAN write enabled (Default: no)
cap_act_prc	82.2288	F13	Battery actual capacity level [%] (Default: 100.0)
cap_nom_ah	108	–	Battery nominal capacity [Ah] (Default: 108.0)
chargelevel	0	F7	Charge power level [1-7] (Default: 0=unlimited)
chargemode	0	F6	Charge mode: 0=notify, 1=stop at sufficient SOC/range (Default: 0)
console	no	F15/16	-unused- (reserved for possible V3 SimpleConsole presence)
dtc_autoreset	no	–	Reset DTC statistics on every drive/charge start (Default: no)
gpslogint	5	(F8)	Seconds between RT-GPS-Log entries while driving (Default: 0 = disabled)
kd_compzero	120	F2	Kickdown pedal compensation (Default: 120)
kd_threshold	35	F1	Kickdown threshold (Default: 35)
kickdown	yes	F15/4	Bool: SEVCON automatic kickdown (Default: yes)
lock_on	6	–	Speed limit [kph] to engage lock mode at (Default: undefined = off)
maxrange	55	F12	Maximum ideal range at 20 °C [km] (Default: 80)
motor_rpm Rated	2050	–	Powermap V3 control: rated speed [RPM] (Default: 0 = V2)
motor_trq_breakdown	210.375	–	Powermap V3 control: breakdown torque [Nm] (Default: 0 = V2)
profileNN	XaZwt5ehZQ..	(P16-21)	Tuning profile #NN [NN=01...99] base64 code
profileNN.label	PWR	–	... button label
profileNN.title	Power	–	... title
profile_buttons	0,1,2,3	–	Tuning drivemode button configuration
profile_cfgmode	2	–	Tuning profile last loaded in config (pre-op) mode
profile_user	2	P15	Tuning profile last loaded in user (op) mode
suffrange	0	F11	Sufficient range [km] (Default: 0=disabled)
suffsoc	80	F10	Sufficient SOC [%] (Default: 0=disabled)
type	–	–	Twizy custom SEVCON/Gearbox type: [SC80GB45,SC45GB80] (Default: - = auto detect)
valet_on	12345.6	–	Odometer limit [km] to engage valet mode at (Default: undefined = off)

### 17.3.2 Custom Metrics

Metric name	Example value	Description
co.can1.nd1.emcy.code		SEVCON emergency code
co.can1.nd1.emcy.type		SEVCON emergendy type
co.can1.nd1.state	Operational	SEVCON state (preop/op)
xrt.b.u.soc.max	70.52%	Current trip/charge pack SOC max
xrt.b.u.soc.min	50.8%	Current trip/charge pack SOC min
xrt.b.u.temp.max	20°C	Current trip/charge pack temp max
xrt.b.u.temp.min	18.1429°C	Current trip/charge pack temp min
xrt.b.u.volt.max	55V	Current trip/charge pack volt max
xrt.b.u.volt.min	50.2V	Current trip/charge pack volt min

Continued on next page

Table 1 – continued from previous page

Metric name	Example value	Description
xrt.bms.balancing	2,4,8,10,13,14	Custom BMS: cell balancing status
xrt.bms.error	0	Custom BMS: error status
xrt.bms.state1	4	Custom BMS: main state
xrt.bms.state2	1	Custom BMS: auxiliary state
xrt.bms.temp	52°C	Custom BMS: internal temperature
xrt.bms.type	1	Custom BMS: type (0=VirtualBMS, 1=eDriver BMS, 7=Standard)
xrt.cfg.applied	yes	Tuning working set has been applied to SEVCON
xrt.cfg.base	2	Tuning base profile (preop mode params)
xrt.cfg.profile	144,110,111,182,...	Tuning profile params
xrt.cfg.type	Twizy80	Tuning vehicle type
xrt.cfg.unsigned	yes	Tuning working set has unsigned changes
xrt.cfg.user	2	Tuning user/live profile (op mode params)
xrt.cfg.ws	0	Tuning profile last loaded into working set
xrt.i.cur.act	0A	SC monitor: motor current level
xrt.i.frq.output	0	SC monitor: motor output frequency
xrt.i.frq.slip	0	SC monitor: motor slip frequency
xrt.i.pwr.act	0kW	SC monitor: motor power level
xrt.i.trq.act	0	SC monitor: motor torque level
xrt.i.trq.demand	0	SC monitor: motor torque demand
xrt.i.trq.limit	0	SC monitor: motor torque limit
xrt.i.vlt.act	0V	SC monitor: output voltage level (motor)
xrt.i.vlt.bat	0	SC monitor: input voltage level (battery)
xrt.i.vlt.cap	0A	SC monitor: capacitor voltage level
xrt.i.vlt.mod	0%	SC monitor: motor modulation factor
xrt.m.version	1.2.4 Sep 17 2019	OVMS Twizy component version
xrt.p.stats.acc.dist	0.6267km	Power stats: accelerating: distance
xrt.p.stats.acc.recd	1.30133e-05kWh	Power stats: accelerating: energy recovered
xrt.p.stats.acc.spdavg	2.7km/h/s	Power stats: accelerating: speed average
xrt.p.stats.acc.used	0.17692kWh	Power stats: accelerating: energy used
xrt.p.stats.cst.dist	0.5331km	Power stats: coasting: distance
xrt.p.stats.cst.recd	4.23289e-05kWh	Power stats: coasting: energy recovered
xrt.p.stats.cst.spdavg	26.18km/h	Power stats: coasting: speed average
xrt.p.stats.cst.used	0.09703kWh	Power stats: coasting: energy used
xrt.p.stats.dec.dist	0.5436km	Power stats: decelerating: distance
xrt.p.stats.dec.recd	0.0140886kWh	Power stats: decelerating: energy recovered
xrt.p.stats.dec.spdavg	2.6km/h/s	Power stats: decelerating: speed average
xrt.p.stats.dec.used	0.0379027kWh	Power stats: decelerating: energy used
xrt.p.stats.ldn.dist	0.2km	Power stats: downwards: distance
xrt.p.stats.ldn.hsum	3m	Power stats: downwards: height sum
xrt.p.stats.ldn.recd	0.00300565kWh	Power stats: downwards: energy recovered
xrt.p.stats.ldn.used	0.0356104kWh	Power stats: downwards: energy used
xrt.p.stats.lup.dist	1.31km	Power stats: upwards: distance
xrt.p.stats.lup.hsum	66m	Power stats: upwards: height sum
xrt.p.stats.lup.recd	0.010866kWh	Power stats: upwards: energy recovered
xrt.p.stats.lup.used	0.263831kWh	Power stats: upwards: energy used
xrt.s.b.pwr.drv		SC monitor: virtual dyno drive power levels
xrt.s.b.pwr.rec		SC monitor: virtual dyno recup power levels
xrt.s.m.trq.drv		SC monitor: virtual dyno drive torque levels
xrt.s.m.trq.rec		SC monitor: virtual dyno recup torque levels

Continued on next page

Table 1 – continued from previous page

Metric name	Example value	Description
xrt.v.b.alert.12v	no	Display service indicator: 12V alert
xrt.v.b.alert.batt	no	Display service indicator: Battery alert
xrt.v.b.alert.temp	no	Display service indicator: Temperature alert
xrt.v.b.status	0	Internal BMS status (CAN frame 628)
xrt.v.c.status	0	Internal Charger status (CAN frame 627)
xrt.v.i.status	0	Internal SEVCON status (CAN frame 629)
xrt.v.e.locked.speed	0	Speed limit [kph] set for Twizy lock mode
xrt.v.e.valet.odo	0	Odometer limit [km] set for Twizy valet mode

### 17.3.3 Events

The Renault Twizy module emits these specific events additionally to the general OVMS events:

Event	Data	Purpose
vehicle.charge.substate.scheduledstop		Charge stopped by user request (command)
vehicle.ctrl.cfgmode		SEVCON in configuration mode (preop)
vehicle.ctrl.loggedin		SEVCON session established
vehicle.ctrl.runmode		SEVCON in normal mode (op)
vehicle.drivemode.changed		SEVCON tuning profile changed
vehicle.dtc.present	<dtc_descr>	OBD2 DTC (diagnostic trouble code) present
vehicle.dtc.stored	<dtc_descr>	OBD2 DTC stored
vehicle.fault.code	<code>	SEVCON fault code received
vehicle.kickdown.engaged		Kickdown detected, drive power changed
vehicle.kickdown.released		Kickdown mode end, normal power restored
vehicle.kickdown.releasing		Kickdown mode about to end

### 17.3.4 Custom Commands

**Note:** This is currently just a brief overview to feed the search engine. See each command's usage info and where applicable corresponding V2 manual entries on details.

- V3 commands need to be in lower case.
- When accessing via USB terminal, first issue enable (login).
- V3 commands are similar to V2 commands, just structured slightly different.
- Try `help`, `?` and `xrt ?`. Twizy commands are subcommands of `xrt`.
- TAB only works via USB / SSH. Shortcuts can be used generally, e.g. `mo t` instead of `module tasks`.
- A usage info is shown if a command syntax is wrong or incomplete.

```
OVMS# xrt ?
batt          Battery monitor
ca            Charge attributes
cfg           SEVCON tuning
dtc           Show DTC report / clear DTC
mon           SEVCON monitoring
obd           OBD2 tools
power         Power/energy info
```

## Battery Monitor

```
OVMS# xrt batt ?
data-cell      Output cell record
data-pack      Output pack record
reset          Reset alerts & watches
status         Status report
tdev           Show temperature deviations
temp           Show temperatures
vdev           Show voltage deviations
volt           Show voltages
```

**Note:** Also see standard BMS commands (bms ?).

## Charge Attributes

```
OVMS# xrt ca ?
Usage: xrt ca [R] | [<range>] [<soc>%] [L<0-7>] [S|N|H]
```

**Note:** Also see standard charge control commands (charge ?).

## SEVCON Tuning

```
OVMS# xrt cfg ?
brakelight     Tune brakelight trigger levels
clearlogs      Clear SEVCON diag logs
drive          Tune drive power level
get            Get tuning profile as base64 string
info           Show tuning profile
load           Load stored tuning profile
op             Leave configuration mode (go operational)
power          Tune torque, power & current levels
pre           Enter configuration mode (pre-operational)
querylogs      Send SEVCON diag logs to server
ramplimits     Tune max pedal reaction
ramps          Tune pedal reaction
read           Read register
recup          Tune recuperation power levels
reset          Reset tuning profile
save           Save current tuning profile
set            Set tuning profile from base64 string
showlogs       Display SEVCON diag logs
smooth         Tune pedal smoothing
speed          Tune max & warn speed
tmap           Tune torque/speed maps
write          Read & write register
writeonly      Write register
```

## Open Vehicles

---

### Show DTC Report / Clear DTC

OVMS# xrt dtc ?	
clear	Clear stored DTC in car
reset	Reset OVMS DTC statistics
show	Show DTC report

### SEVCON Monitoring

OVMS# xrt mon ?	
reset	Reset monitoring
start	Start monitoring
stop	Stop monitoring

### OBD2 Tools

OVMS# xrt obd ?	
request	Send OBD2 request, output response
OVMS# xrt obd request ?	
bms	Send OBD2 request to BMS
broadcast	Send OBD2 request as broadcast
charger	Send OBD2 request to charger (BCB)
cluster	Send OBD2 request to cluster (display)
device	Send OBD2 request to a device

### Power/Energy Info

OVMS# xrt power ?	
report	Trip efficiency report
stats	Generate RT-PWR-Stats entry
totals	Power totals

## 17.3.5 Notifications

The Renault Twizy module sends the custom or customized notifications described here additionally to the system notifications.

See [Notifications](#) for general info on notifications.



Type	Subtype	Purpose / Content
alert	battery.status	Battery pack/cell alert status report (alerts & watches)
alert	bms.status	Battery management system alert (for custom BMS only)
info	charge.status.sufficient	Sufficient charge reached (SOC/range as configured)
alert	valetmode.odolimit	Odometer limit reached, speed reduction engaged
alert	vehicle.dtc	OBD2 DTC (diagnostic trouble code) alert
data	xrt.battery.log	Battery pack/cell monitoring log
data	xrt.gps.log	Extended GPS log
data	xrt.obd.cluster.dtc	OBD2 DTC log for cluster (display/UCH)
data	xrt.power.dyno	SEVCON live monitoring (virtual dyno) data
data	xrt.power.log	Power statistics
info	xrt.power.totals	Current usage cycle power totals
data	xrt.trip.log	Trip history log
info	xrt.trip.report	Trip energy usage report
alert	xrt.sevcon.fault	SEVCON fault condition detected
data	xrt.sevcon.log	SEVCON faults & events logs
info	xrt.sevcon.profile.reset	A tuning RESET has been performed
info	xrt.sevcon.profile.switch	Tuning profile switch result

## Trip history log

The trip history log can be used as a source for long term statistics on your trips and typical trip power usages, as well as your battery performance in different environmental conditions and degradation over time.

Entries are created at the end of a trip and on each change in the charge state, so you can also see where charges stopped or how long they took and how high the temperatures got.

- Notification subtype: `xrt.trip.log`
- History record type: RT-PWR-Log
- Format: CSV
- Maximum archive time: 365 days
- Fields/columns:
  - `odometer_km`
  - `latitude`
  - `longitude`
  - `altitude`
  - `chargestate`
  - `parktime_min`
  - `soc`
  - `soc_min`
  - `soc_max`
  - `power_used_wh`
  - `power_recd_wh`
  - `power_distance`

- range\_estim\_km
- range\_ideal\_km
- batt\_volt
- batt\_volt\_min
- batt\_volt\_max
- batt\_temp
- batt\_temp\_min
- batt\_temp\_max
- motor\_temp
- pem\_temp
- trip\_length\_km
- trip\_soc\_usage
- trip\_avg\_speed\_kph
- trip\_avg\_accel\_kps
- trip\_avg\_decel\_kps
- charge\_used\_ah
- charge\_recd\_ah
- batt\_capacity\_prc
- chg\_temp

### Extended GPS log

The extended GPS log contains additional details about power and current levels, the BMS power limits and the automatic power level adjustments done by the OVMS. You can use this to create detailed trip power charts and to verify your auto power adjust settings.

The log frequency is once per minute while parking/charging, and controlled by config `xrt gpslogint` (web UI: Twizy → Features) while driving. Logging only occurs if logged metrics have changed.

- Notification subtype: `xrt.gps.log`
- History record type: RT-GPS-Log
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - odometer\_mi\_10th
  - latitude
  - longitude
  - altitude
  - direction
  - speed

- gps\_fix
- gps\_stale\_cnt
- gsm\_signal
- current\_power\_w
- power\_used\_wh
- power\_recd\_wh
- power\_distance
- min\_power\_w
- max\_power\_w
- car\_status
- max\_drive\_pwr\_w
- max\_recup\_pwr\_w
- autodriven\_level
- autorecup\_level
- min\_current\_a
- max\_current\_a

### Battery pack/cell monitoring log

The extended GPS log contains additional details about power and current levels, the BMS power limits and the automatic power level adjustments done by the OVMS. You can use this to create detailed trip power charts and to verify your auto power adjust settings.

The standard log frequency is once per minute, logging only occurs if logged metrics have changed. Additional records are created on battery alert events. Note: an entry consists of a pack level record (RT-BAT-P) and up to 16 (for LiFePO4 batteries) cell entries (RT-BAT-C).

- Notification subtype: `xrt.battery.log`
- History record type: RT-BAT-P (pack status)
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - packnr
  - volt\_alertstatus
  - temp\_alertstatus
  - soc
  - soc\_min
  - soc\_max
  - volt\_act
  - volt\_min

- volt\_max
  - temp\_act
  - temp\_min
  - temp\_max
  - cell\_volt\_stddev\_max
  - cmod\_temp\_stddev\_max
  - max\_drive\_pwr
  - max\_recup\_pwr
  - bms\_state1
  - bms\_state2
  - bms\_error
  - bms\_temp
- Notification subtype: `xrt.battery.log`
- History record type: RT-BAT-C (cell status)
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - cellnr
  - volt\_alertstatus
  - temp\_alertstatus
  - volt\_act
  - volt\_min
  - volt\_max
  - volt\_maxdev
  - temp\_act
  - temp\_min
  - temp\_max
  - temp\_maxdev
  - balancing
  - been\_balancing
  - balancetime

### Power statistics

The power statistics are the base for the trip reports and can be used to analyze trip sections regarding speed and altitude changes and their respective effects on power usage. The log is also written when charging, that data can be used to log changes in the charge current, for example triggered externally by some solar charge controller.

Log frequency is once per minute, logging only occurs if metrics have changed.

- Notification subtype: `xrt.power.log`
- History record type: `RT-PWR-Stats`
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - `speed_const_dist`
  - `speed_const_use`
  - `speed_const_rec`
  - `speed_const_cnt`
  - `speed_const_sum`
  - `speed_accel_dist`
  - `speed_accel_use`
  - `speed_accel_rec`
  - `speed_accel_cnt`
  - `speed_accel_sum`
  - `speed_decel_dist`
  - `speed_decel_use`
  - `speed_decel_rec`
  - `speed_decel_cnt`
  - `speed_decel_sum`
  - `level_up_dist`
  - `level_up_hsum`
  - `level_up_use`
  - `level_up_rec`
  - `level_down_dist`
  - `level_down_hsum`
  - `level_down_use`
  - `level_down_rec`
  - `charge_used`
  - `charge_recd`

### OBD2 cluster DTC log

This server table stores DTC occurrences for one week. This is mostly raw data, and the DTCs are internal Renault values that have not yet been decoded.

See: <https://www.twizy-forum.de/ovms/86362-liste-df-codes-dtc>

- Notification subtype: `xrt.obd.cluster.dtc`

- History record type: RT-OBD-ClusterDTC
- Format: CSV
- Maximum archive time: 7 days
- Fields/columns:
  - EntryNr
  - EcuName
  - NumDTC
  - Revision
  - FailPresentCnt
  - G1
  - G2
  - ServKey
  - Customer
  - Memorize
  - Bt
  - Ef
  - Dc
  - DNS
  - Odometer
  - Speed
  - SOC
  - BattV
  - TimeCounter
  - IgnitionCycle

### SEVCON faults & events

These logs are created on request only, e.g. by the SEVCON logs tool in the Android App, or by using the `xrt cfg querylogs` command. The command queries the SEVCON (inverter) alerts, faults, events and statistics (SEVCON needs to be online). The results are then transmitted to the server using the following records.

- Notification subtype: `xrt.sevcon.log`
- History record type: RT-ENG-LogKeyTime
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - KeyHour
  - KeyMinSec
- Notification subtype: `xrt.sevcon.log`

- History record type: `RT-ENG-LogAlerts`
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - Code
  - Description
  - TimeHour
  - TimeMinSec
  - Data1
  - Data2
  - Data3
- Notification subtype: `xrt.sevcon.log`
- History record type: `RT-ENG-LogSystem`
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - Code
  - Description
  - TimeHour
  - TimeMinSec
  - Data1
  - Data2
  - Data3
- Notification subtype: `xrt.sevcon.log`
- History record type: `RT-ENG-LogCounts`
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - Code
  - Description
  - LastTimeHour
  - LastTimeMinSec
  - FirstTimeHour
  - FirstTimeMinSec
  - Count
- Notification subtype: `xrt.sevcon.log`

- History record type: RT-ENG-LogMinMax
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns:
  - BatteryVoltageMin
  - BatteryVoltageMax
  - CapacitorVoltageMin
  - CapacitorVoltageMax
  - MotorCurrentMin
  - MotorCurrentMax
  - MotorSpeedMin
  - MotorSpeedMax
  - DeviceTempMin
  - DeviceTempMax

### SEVCON live monitoring

These records store the measurement results of the virtual dyno included in the SEVCON live monitor (Twizy → SEVCON Monitor). The virtual dyno records a torque/power profile from the actual car performance during driving. The profile has four data sets:

- Maximum battery drive power over speed (metric `xrt.s.b.pwr.drv`, unit kW)
- Maximum battery recuperation power over speed (metric `xrt.s.b.pwr.rec`, unit kW)
- Maximum motor drive torque over speed (metric `xrt.s.m.trq.drv`, unit Nm)
- Maximum motor recuperation torque over speed (metric `xrt.s.m.trq.rec`, unit Nm)

Power is measured at the battery, so you can derive the efficiency. Speed is truncated to integer, the value arrays take up to 120 entries (0 ... 119 kph).

These datasets are visualized by the web UI using a chart, and transmitted to the server on any monitoring “stop” or “reset” command in the following records:

- Notification subtype: `xrt.power.dyno`
- History record types: RT-ENG-BatPwrDrv, RT-ENG-BatPwrRec, RT-ENG-MotTrqDrv, RT-ENG-MotTrqRec
- Format: CSV
- Maximum archive time: 24 hours
- Fields/columns: max 120 values for speed levels beginning at 0 kph

### 17.3.6 Twizy Regen Brake Light Hack



## Why?

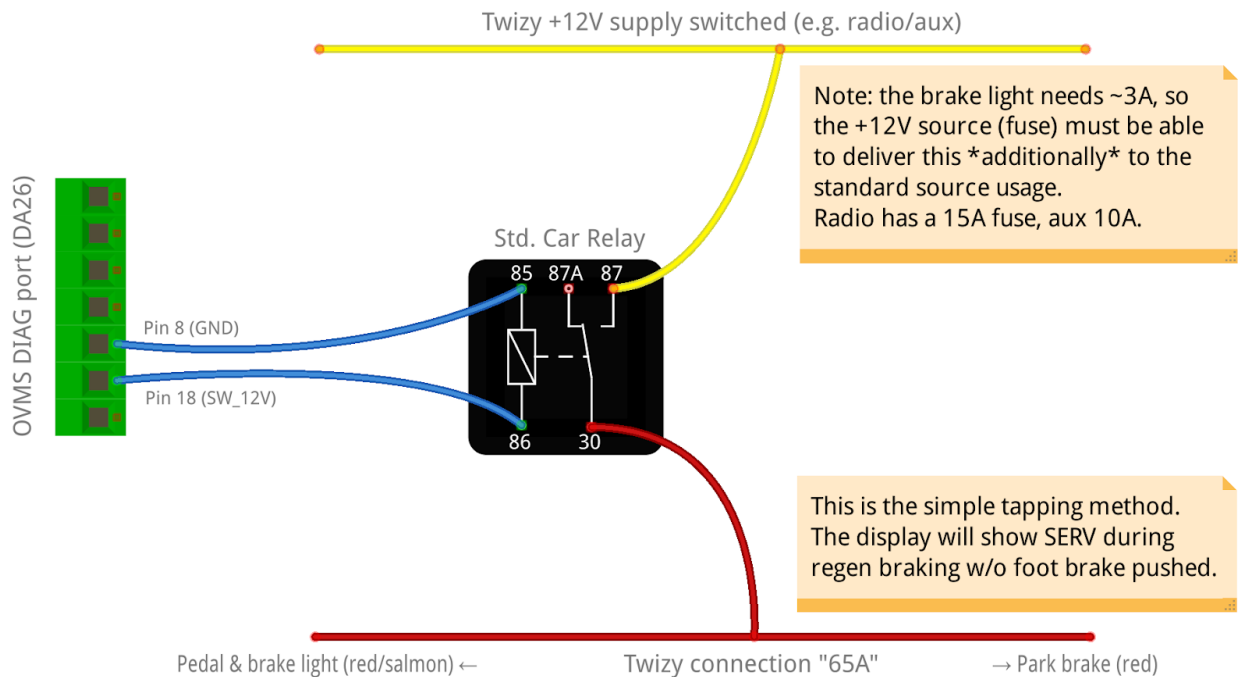
The Twizy recuperation levels are very low by default, regenerative braking is so weak it doesn't need to trigger the brake light – so a regen braking indication has been omitted by Renault. The Twizy can be tuned to one pedal driving though with high neutral braking power.

The SEVCON normally can generate a regen braking signal itself. That signal can be configured using the `xrt cfg brakelight` tuning command. Pin 11 on the SEVCON main connector will go to GND on activation. The problem is, that pin is not connected on the Twizy, using it needs modification of the connector. To my knowledge this hasn't been done successfully up to now, so we don't even know if the Twizy SEVCON firmware includes that functionality.

## How does it work?

The OVMS generates a secondary regen braking signal itself. That signal is used to control a relay to provide power to the brake light independent of the brake pedal.

Integration is simple and does not require to change the existing connections:



fritzing

The Twizy UCH/display recognizes the regen brake light activation despite the foot brake being pushed and turns on the SERV indicator. Depending on your preferences you may see this as a problem or as an indicator for the regen braking.

Switching the 65A line does not avoid the SERV indicator (tested). Most probably the brake light is monitored by the current flowing at the fuse box, so the SERV indicator cannot be avoided without changing that system.

## Parts required

- 1x DA26 / Sub-D 26HD Plug & housing
  - Note: housings for Sub-D 15 fit for 26HD
  - e.g. [Assmann-WSW-A-HDS-26-LL-Z](#)

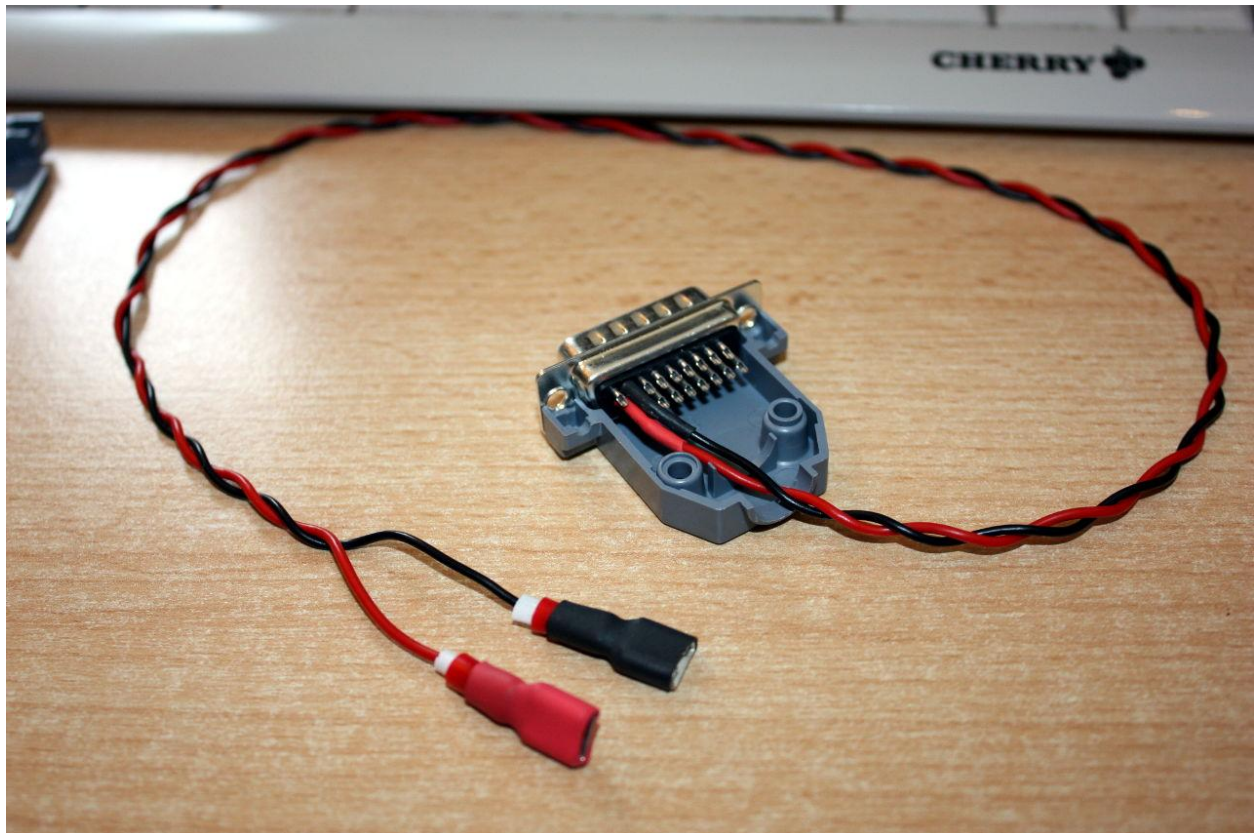
- [Encitech-DPPK15-BK-K-D-SUB-Gehaeuse](#)
2. 1x 12V Universal Car Relay + Socket
    - e.g. [Song-Chuan-896H-1CH-C1-12V-DC-Kfz-Relais](#)
    - [GoodSky-Relaissockel-1-St.-GRL-CS3770](#)

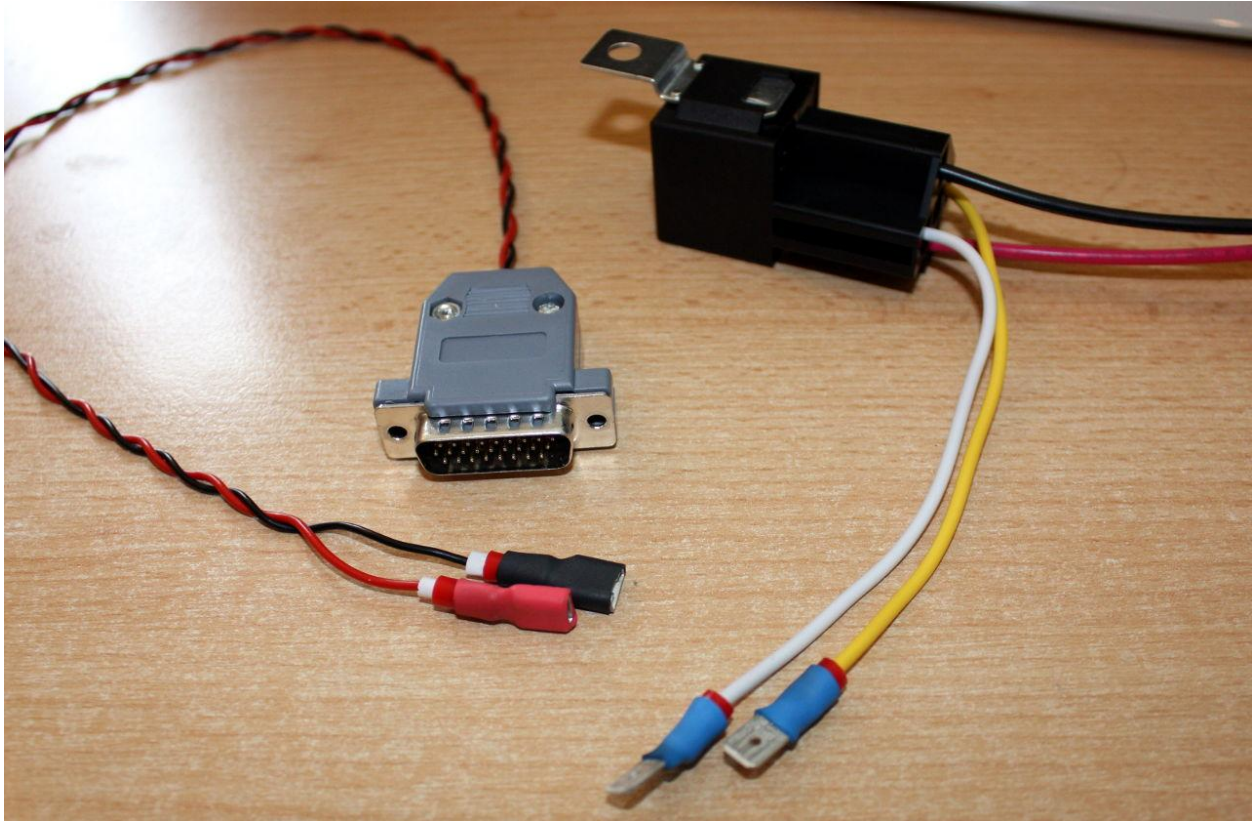
3. Car wire-tap connectors, car crimp connectors, 0.5 mm<sup>2</sup> wires, zipties, shrink-on tube, tools

Note: if you already use the switched 12V output of the OVMS for something different, you can use one of the free EGPIO outputs. That requires additionally routing an EGPIO line to the DA26 connector at the expansion slot (e.g. using a jumper) and using a relay module (relay shield) with separate power input instead of the standard car relay.

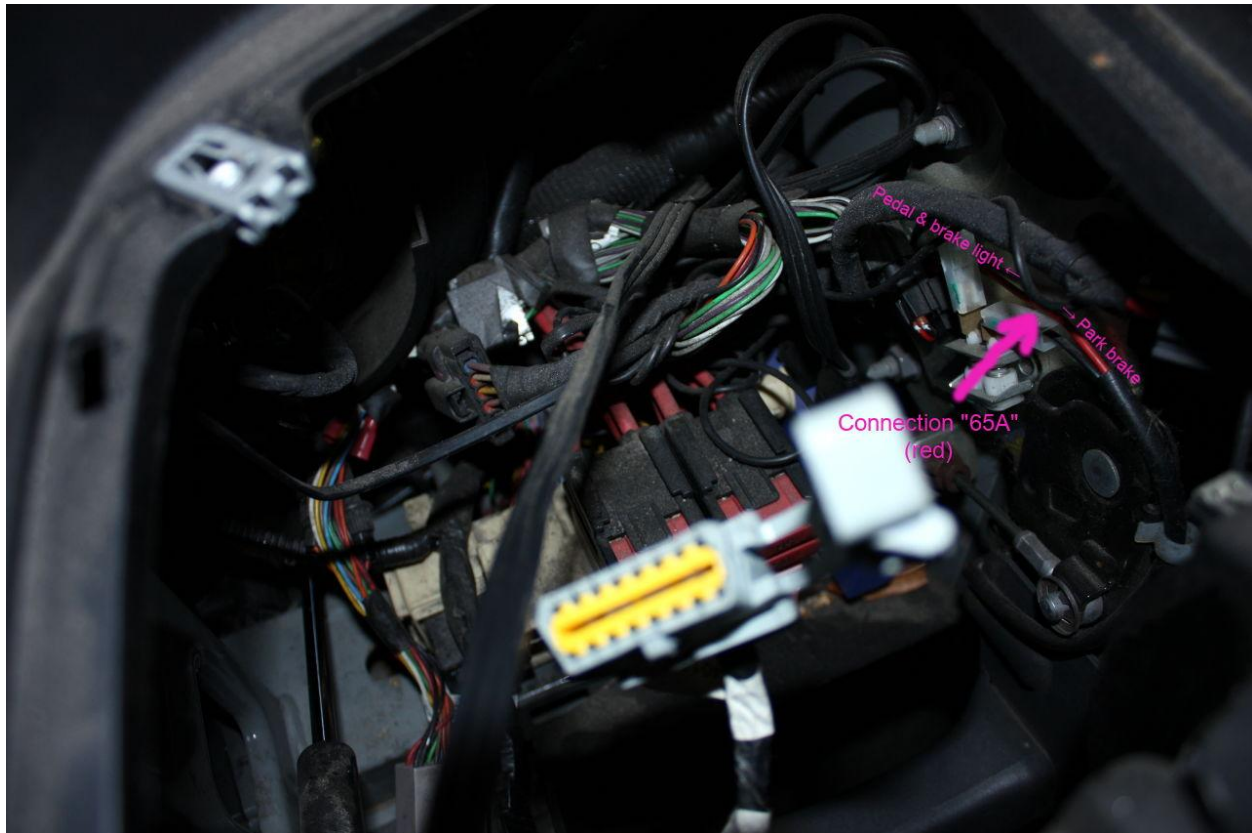
### Build

1. Solder ~ 40 cm two-core wire to the DA26 plug pins 8 (GND) and 18 (+12V switched), use shrink-on tubes to protect the terminals, mount the housing
2. Crimp 6.3mm sockets to the other ends of the wire, isolate using shrink-on tube, crimp 6.3mm plugs to the relay coil (pins 85 + 86), isolate using shrink-on tube





3. Extend the relay switch terminals by ~ 30 cm wires, crimp on plugs for the tap connectors
4. Unmount the Twizy glovebox; locate the red wire on the right above the parking brake:



5. Crimp a tap connector onto the red wire and connect it to the relay (pin 30):





6. Connect switched +12V likewise to pin 87 of the relay
  - +12V may be taken from the radio or 12V aux socket to simplify things
  - the dedicated brake light supply is fuse no. 23, which is very hard to reach without dismounting the whole fuse holder
7. Place the relay in the free area on the upper left of the fuse holder (not below the door dampener, that may crush the relay), secure the relay using a zip tie
8. Pull the OVMS DIAG cable through the glovebox bottom hole and connect it to the relay inputs (pins 85/86, polarity doesn't matter)
9. Do a test: plug in the OVMS (Note: the relay switches on during the first OVMS boot), switch on the Twizy, issue commands:
  - `egpio output 1 1` should activate the brakelight
  - `egpio output 1 0` should deactivate the brakelight
  - Note: if you're using another EGPIO port, use the according port number (3-9) instead
10. Mount the glovebox and you're done.

## Configuration

See OVMS web user interface, menu Twizy → Brake Light:

Regen Brake Light

Acceleration smoothing:

2

-

+

Speed/acceleration smoothing eliminates road bump and gear box backlash noise.  
Lower value = higher sensitivity. Set to zero if your vehicle speed is already smoothed.

☒ Enable regenerative braking signal

... control port:

SW\_12V (DA26 pin 18)

... activation level:

1,3

m/s<sup>2</sup>

-

+

Deceleration threshold to activate regen brake light.  
Under UN regulation 13H, brake light illumination is required for decelerations >1.3 m/s<sup>2</sup>.

... deactivation level:

0,7

m/s<sup>2</sup>

-

+

Deceleration threshold to deactivate regen brake light.  
Under UN regulation 13H, brake lights must not be illuminated for decelerations ≤0.7 m/s<sup>2</sup>.

Save

The OVMS generated regenerative braking signal needs a hardware modification to drive your vehicle brake lights. See your OVMS vehicle manual section for details on how to do this.

The regen brake signal is activated when the deceleration level exceeds the configured threshold, the speed is above 1 m/s (3.6 kph / 2.2 mph) and the battery power is negative (charging).

The signal is deactivated when deceleration drops below the deactivation level, speed drops below 1 m/s or battery power indicates discharging. The signal will be held activated for at least 500 ms.

Set the port as necessary and the checkbox to enable the brakelight.

For monitoring and fine tuning, use the *regenmon* web plugin:

[https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3/blob/master/vehicle/OVMS.V3/components/ovms\\_webserver/dev/regenmon.htm](https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3/blob/master/vehicle/OVMS.V3/components/ovms_webserver/dev/regenmon.htm)

### 17.3.7 Auxiliary Charge Fan

The builtin **Elips 2000W** charger's officially specified temperature operation range is -20..+50 °C. This is exceeded by the Twizy easily even on moderate summer days. The charger temperature can easily rise above 60 °C, and this is

possibly the main reason why Twizy chargers eventually die. It also leads to the Twizy reducing the maximum charge power available, resulting in charges taking much longer than usual.

A **counter measure** can be to add an additional fan to cool the charger, and switch that fan on as necessary (see below). The fan can be controlled using one of the OVMS EGPIO output ports (see [EGPIO](#)). The easiest way is to use the switched 12V supply available at the DA26 expansion plug without additional hardware. The 12V port can deliver a nominal output of 25W or 1.8A, and up to 40W or 2.9A short term, so can be used to power many auxiliary fans directly.

**To enable the auxiliary fan**, simply configure the EGPIO to be used in config parameter `xrt aux_fan_port`. For example, to use the switched 12V port = EGPIO port 1, do:

```
OVMS# config set xrt aux_fan_port 1
```

Set the port to 0 to disable the feature.

#### Mode of operation:

- The OVMS **continuously monitors** the charger temperature. It turns on the fan when the charger temperature rises **above 45 °C** ( $\geq 46$  °C), and off when the temperature drops **below 45 °C** ( $\leq 44$  °C).
- **After end of charge** (Twizy system switched off), the OVMS does not get temperature updates, so it keeps the fan running for another **5 minutes**.

The fan control works during **charging and driving**, as during driving the charger also has to supply +12V by the builtin DC/DC converter (heating up the charger), and a trip may start with an already hot charger.

### 17.3.8 Auxiliary Charger

The **Elips 2000W** (as the name says) does only charge with ~2 kW, thus needing 3-3.5 hours to charge the original Twizy battery. That's quite slow and gets worse when [replacing the battery by a larger one](#).

The Elips is running a special Renault firmware and also taking a crucial role in the Twizy system communication, for example to control the BMS. So it cannot easily be replaced by a third party charger.

But a **third party charger can run in parallel** to the Elips to boost the charge power. This can be done best from the "rear end", i.e. using the battery main connector as the power input (like the SEVCON does during regenerative braking).

#### Some documented examples of additional chargers:

- <https://www.twizy-forum.de/werkstatt-twizy/84935-twizy-schnelllader>
- <https://www.twizy-forum.de/projekte-twizy/83603-schnell-lader-extern>
- <https://www.twizy-forum.de/tipps-und-tricks-twizy/80814-twizy-schnellladen-die-wave-kann-kommen>

**The caveat:** to avoid damaging the battery, additional charge power should not be supplied when the battery is nearly full. So the additional charger needs to be switched off before the main charger finishes.

The OVMS can **automate** switching the additional charger both on and off. Any of the available EGPIO output ports (see [EGPIO](#)) can be used for this task. If the charger has no digital power control input, a convenient way is to use the switched 12V supply available at the DA26 expansion plug without additional hardware. The 12V port can deliver a nominal output of 25W or 1.8A, and up to 40W or 2.9A short term, so can be used to power a standard automotive relay directly.

#### Configuration:

Simply set the EGPIO to be used in config parameter `xrt aux_charger_port`. For example, to use the switched 12V port = EGPIO port 1, do:

```
OVMS# config set xrt aux_charger_port 1
```

Set the port to 0 to disable the feature.

### Mode of operation:

- The port is **switched ON** when the Twizy charges, is below 94% SOC, is still within the CC phase (i.e. not topping off / balancing), and the charge current is not under user control.
- The port is **switched OFF** when the Twizy stops charging, reaches 94% SOC or enters the topping off phase (whichever comes first), or when you set a custom charge current.

### Charge current control:

If you set the **charge current** to level 7 (35 A), only the original charger will run, but at full power. You can set levels 6...1 (30...5 A) to reduce the charge power, only using the Elips charger. Setting the current to level 0 (no charge throttling) will allow maximum charge current from the Elips charger and additionally enable the second charger.



# CHAPTER 18

## Renault Zoe

Vehicle Type: **RZ**

The Renault Zoe will be documented here.

### 18.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes (External Temp and Battery)
BMS v+t Display	No
TPMS Display	Yes (pressure only)
Charge Status Display	Yes
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	



# CHAPTER 19

## Smart ED Gen.3

Vehicle Type: **SE**

The Smart ED will be documented here.

### 19.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	No
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Current limit
Cabin Pre-heat/cool Control	When charging and CAN-Bus Active
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	

## 19.2 Contents

### 19.2.1 Custom Metrics

Metric name	Example value	Description
xse.mybms.HW.rev	12,38,0	BMS hardware-revision year, week, patchlevel
xse.mybms.SW.rev	12,35,1	BMS soft-revision year, week, patchlevel
xse.mybms.adc.cvolts.max	4171	maximum cell voltages in mV, add offset +1500
xse.mybms.adc.cvolts.mean	4165	average cell voltage in mV, no offset
xse.mybms.adc.cvolts.min	4145	minimum cell voltages in mV, add offset +1500
xse.mybms.adc.volts.offset	103	calculated offset between RAW cell voltages and ADCref, about 90mV
xse.mybms.amps	65483A	battery current in ampere (x/32) reported by by BMS
xse.mybms.amps2	2046.34A	battery current in ampere read by live data on CAN or from BMS
xse.mybms.batt.vin	WME4513901K661441	VIN stored in BMS
xse.mybms.dc.fault	0	Flag to show DC-isolation fault
xse.mybms.hv		total voltage of HV system in V
xse.mybms.hv.contact.cycles.left	281991	counter related to ON/OFF cycles of the car
xse.mybms.hv.contact.cycles.max	300000	static, seems to be maximum of contactor cycles
xse.mybms.isolation	10239	Isolation in DC path, resistance in kOhm
xse.mybms.power	794.186kW	power as product of voltage and amps in kW
xse.v.b.c.capacity	18491,18536,18536,...	Cell capacity [As]
xse.v.b.c.capacity.dev.max	45.1,90.1,90.1,...	Cell maximum capacity deviation observed [As]
xse.v.b.c.capacity.max	18491,18536,18536,...	Cell maximum capacity [As]
xse.v.b.c.capacity.min	18491,18536,18536,...	Cell minimum capacity [As]
xse.v.b.energy.used.reset	16.34kWh	Energy used Reset (Dashbord)
xse.v.b.energy.used.start	16.9kWh	Energy used Start (Dashbord)
xse.v.b.hv.active	no	HV Batterie Status
xse.v.b.p.capacity.as.average	18625	cell capacity statistics from BMS measurement cycle
xse.v.b.p.capacity.as.maximum	19193	cell capacity statistics from BMS measurement cycle
xse.v.b.p.capacity.as.minimum	18843	cell capacity statistics from BMS measurement cycle
xse.v.b.p.capacity.avg	18445.9	Cell capacity - pack average [As]
xse.v.b.p.capacity.combined.quality	0.515328	some sort of estimation factor??? constantly updated
xse.v.b.p.capacity.max	18670	Cell capacity - strongest cell in pack [As]
xse.v.b.p.capacity.max.cell	9	Cell capacity - number of strongest cell in pack
xse.v.b.p.capacity.min	18229	Cell capacity - weakest cell in pack [As]
xse.v.b.p.capacity.min.cell	59	Cell capacity - number of weakest cell in pack
xse.v.b.p.capacity.quality	0.742092	some sort of estimation factor??? after measurement cycle
xse.v.b.p.capacity.stddev	90.8	Cell capacity - current standard deviation [As]
xse.v.b.p.capacity.stddev.max	90.8	Cell capacity - maximum standard deviation observed [As]
xse.v.b.p.hv.lowcurrent	0Sec	counter time of no current, reset e.g. with PLC heater or driving
xse.v.b.p.hv.off.time	0Sec	HighVoltage contactor off time in seconds
xse.v.b.p.last.meas.days	13	days elapsed since last successful measurement
xse.v.b.p.ocv.timer	2676Sec	counter time in seconds to reach OCV state
xse.v.b.p.voltage.max.cell	1	Cell volatage - number of strongest cell in pack
xse.v.b.p.voltage.min.cell	5	Cell volatage - number of weakest cell in pack
xse.v.b.real.soc	97.8%	real state of charge
xse.v.bus.awake	no	CAN-Bus Status
xse.v.c.active	no	Charging Status
xse.v.display.time	738Min	Dashbord Time
xse.v.display.trip.reset	2733.2km	Dashbord Trip value at Reset

Continued on next page

Table 1 – continued from previous page

Metric name	Example value	Description
xse.v.display.trip.start	17.5km	Dashbord Trip value at Start
xse.v.nlg6.amps.cablecode	0A	Onboard Charger Ampere Cable
xse.v.nlg6.amps.chargingpoint	0A	Onboard Charger Ampere ...
xse.v.nlg6.amps.setpoint	41A	Onboard Charger Ampere setpoint
xse.v.nlg6.dc.current	0A	Onboard Charger LV current Ampere
xse.v.nlg6.dc.hv	0V	Onboard Charger HV Voltage
xse.v.nlg6.dc.lv	13.5V	Onboard Charger LV Voltage
xse.v.nlg6.main.amps	0,0,0A	Onboard Charger main Phasen Ampere
xse.v.nlg6.main.volts	0,0,0V	Onboard Charger main Phasen Voltage
xse.v.nlg6.pn.hw	4519820621	Onboard Charger Hardware Serial No.
xse.v.nlg6.present	no	Onboard Charger OBL or NLG6 present
xse.v.nlg6.temp.coolingplate	0°C	Onboard Charger coolingplate Temperature
xse.v.nlg6.temp.reported	0°C	Onboard Charger reported Temperature
xse.v.nlg6.temp.socket	0°C	Onboard Charger socket Temperature
xse.v.nlg6.temps		Onboard Charger Temperatures
xse.v.pos.odometer.start	41940km	Odo at last Ignition on. For Trip calc Value
xse.v.display.accel	89%	Eco score on acceleration over last 6 hours
xse.v.display.const	18%	Eco score on constant driving over last 6 hours
xse.v.display.coast	100%	Eco score on coasting over last 6 hours
xse.v.display.ecoscore	69%	Eco score shown on dashboard over last 6 hours

## 19.2.2 Custom Commands

xse trip

show last Trip value

xse recu up

xse recu down

set Rekuperation up or down. Works only if Rekuperation is enabled in car

xse charge

set charge and leave Time.

Example:

xse charge 7 15 off

set Time at 7:15 clock with pre climate off

xse charge 9 55 on

set Time at 9:55 clock with pre climate on

## 19.2.3 BMS Diagnose Tool

xse rptdata get a Battery Status Report

xse bmsdiag get Battery Diagnose Report

Original Tool [ED BMSdiag](#)

[description DE](#)

[description EN](#)



## Smart ED/EQ Gen.4 (453)

Vehicle Type: **SQ**

The Smart ED/EQ Gen.4 will be documented here.

### 20.1 Support Overview

Function	Support Status
Hardware	OVMS v3 (or later)
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes (External Temp and Battery)
BMS v+t Display	only Cell Volts atm.
TPMS Display	No
Charge Status Display	Yes
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	





# CHAPTER 21

## Tesla Model 3

### Vehicle Type: T3

At present, support for the Tesla Model 3 in OVMS is experimental and under development. This vehicle type should not be used by anyone other than those actively involved in development of support for this vehicle.

## 21.1 Support Overview

Function	Support Status
Hardware	tba
Vehicle Cable	tba
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	tba
SOC Display	tba
Range Display	tba
GPS Location	tba
Speed Display	tba
Temperature Display	tba
BMS v+t Display	tba
TPMS Display	tba
Charge Status Display	tba
Charge Interruption Alerts	tba
Charge Control	tba
Cabin Pre-heat/cool Control	tba
Lock/Unlock Vehicle	tba
Valet Mode Control	tba
Others	tba



## CHAPTER 22

---

### Tesla Model S

---

Vehicle Type: **TS**

At present, support for the Tesla Model S in OVMS is experimental and under development. This vehicle type should not be used by anyone other than those actively involved in development of support for this vehicle.

### 22.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: Not widely tested, but should be all.
Vehicle Cable	9665972 OVMS Data Cable for Early Teslas
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	None, not required
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from car's built in GPS)
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes
TPMS Display	Not currently supported
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Not currently supported
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Not currently supported
Valet Mode Control	Not currently supported
Others	Adhesive velcro strips useful for vehicle attachment

## 22.2 TPMS Option

Reading and writing TPMS wheel sensor IDs from/to the Baolong TPMS ECU is supported by OVMS in Tesla Model S cars using the Baolong system (vehicles produced up to around August 2014). You can easily tell if you have this Tesla Model S Baolong TPMS system as Tesla doesn't support displaying tyre pressures in the instrument cluster. By contrast, the later Continental TPMS system does show the pressures in the instrument cluster.

OVMS directly supports the Baolong TPMS in Tesla Model S cars, without any extra hardware required. You simply need the usual OVT1 cable.

To read the current wheel sensor IDs from the Baolong TPMS ECU, ensure that the car is ON (simplest is to wake the car up and put in drive), and issue the 'tpms read' command in OVMS.

Similarly, to write wheel sensor IDs to the Baolong TPMS ECU, ensure that the car is ON, and issue the 'tpms write' command in OVMS.

## CHAPTER 23

---

### Tesla Roadster

---

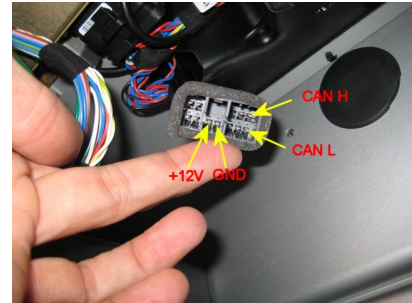
Vehicle Type: **TR**

The Tesla Roadster support in OVMS is perhaps the most mature in the project. All versions (1.x, 2.x, and 3.x) are supported, for both North American and other variants.

### 23.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: 1.x, 2.x, and 3.0 roadsters (all markets).
Vehicle Cable	9665972 OVMS Data Cable for Early Teslas
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	None, not required
SOC Display	Yes
Range Display	Yes
GPS Location	Yes (from car's built in GPS)
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Not currently supported
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	Yes
Cabin Pre-heat/cool Control	Not currently supported
Lock/Unlock Vehicle	Yes
Valet Mode Control	Yes
Others	Adhesive velcro strips useful for vehicle attachment

## 23.2 Module Installation in the vehicle



The OVMS module is connected to your Roadster via the CAN bus diagnostic port connector, which is located in the passenger footwell. It is made of plastic that is wrapped in grey foam, as shown in the photo. Typically, the connector is wedged into the front wall near the center console so it won't rattle. Pull the connector out and note the orientation of the pins, especially the void above the +12V and Ground pins.

The OVMS Data Cable for Tesla Roadster can then be plugged in, connecting the car to the OVMS module. Make sure to tighten the screws holding the module-side DB9 connector.

At this point, check the car. Tap on the VDS in the centre console and make sure it turns on. Insert the key, turn on the car, and make sure everything works as expected. If you see any problems at all with the car, disconnect the OVMS module and contact Open Vehicles support for assistance.



### Warning!

If, when you plug the OVMS into the car, you see any interference to, or strange behaviour of, car systems immediately unplug the module and contact Open Vehicles for assistance. Never leave a module connected in such circumstances.

The OVMS Module is best secured to the front wall of the passenger footwell using adhesive velcro tape. With the module connected to the car diagnostics port connector, experiment with various placements until you find a suitable spot. For the velcro attachment to work, you'll want to choose a spot on the front wall that's flat for the entire size of the module (hint: avoid the big round black plastic plug).

Mounting is straightforward:

1. Ensure that both velcro strips are fixed together.
2. Remove the adhesive backing from one side; fasten it to the back of the OVMS module.
3. Using a clean dry cloth, clean the area of the car passenger footwell wall to which you are going to attach the module.
4. Remove the adhesive backing from the side of the velcro strip facing the car, and then firmly push the OVMS module into place - holding it still for a few seconds to allow the adhesive to work.
5. You can then remove and reinstall the OVMS module as desired via the velcro.

## 23.3 Antenna Installation

You will find the performance of this antenna fantastic - and much better than even your cellphone, but proper placement is essential. Since it has a very long cable, you can place the antenna just about wherever you want, but please ensure it is high-up on the car and away from any metal objects that might interfere with the signal.

Possible areas include the bottom of the windscreen/windshield on the passenger side, the top of the windscreen on the passenger side (hidden by the sun visor), behind the passenger on the side pillar, in the rear window, or under the dashboard (for the brave and experienced at dismantling Tesla Roadster dashboards).

The antenna is connected to the port marked GSM on the OVMS v3 module. The GPS port is not used on the Tesla Roadster (as your car already has GPS installed and we can read that positioning information off the CAN bus directly).

### 23.3.1 Antenna beside rear passenger headrest

The antenna cable is long enough to reach back to the area around the left side seat head. This approach is generally easiest. The module is placed in the left seat footwell, near the diagnostic port connector. From there, the antenna cable is routed through the base of the waterfall, under the door sill, and up the side of the door frame.

The door sill is held in place by velcro and is easily removed. You may have to loosen the *waterfall* (held in place by four screws around the fuse box area). The antenna cable can then be placed on top of the metal of the chassis sill, between the velcro strips, and routed up through the existing plastic trunking. At this point, the door sill can then be put back in place.

### 23.3.2 Antenna on windshield/windscreen

To route the antenna cable up to the front windshield/windscreen, you will need to remove the fuse-box cover (one screw that needs to be turned 90 degrees), then two screws from the box below the fuse box (these screws should be completely removed in order to be able to remove the box and access the compartment beneath). You do not need to adjust anything in the fuse box - you only need the cover removed to make it easier to route the cable.

Start with the cable at the windscreen/windshield and route it down the side of the left side door front pillar. The plastic corner can be pulled back slightly, and you can push the cable through into the open bottom compartment. Pull the cable through there so that the antenna is where you want it and there is no loose cable outside the box. The antenna itself can be mounted to the windscreen/windshield by first cleaning the area with a clean dry cloth, removing the adhesive backing tape, then firmly pushing the antenna against the glass and waiting a few seconds for the adhesive to stick.

Now for the tricky bit. You need to get the cable through to the passenger footwell, but it is tight. It is much easier to get a guide wire up into the fuse box compartment than to get the antenna cable down into the passenger footwell. So, we recommend you use a small (12 inches / 30 cm or so) piece of stiff wire to use as a guide and push it up from the area marked by the green arrow on the bottom right of the picture below. Once the guide wire is in the fuse box, push it down into the lower compartment you opened and wrap it around the antenna cable. You can then pull the guide wire back down into the passenger footwell, bringing the antenna cable with it.

The antenna cable can then be screwed in to the OVMS module. You can then tidy up any loose cable, and screw-back the lower compartment box (two screws) and fuse box cover (one screw 90 degrees to lock).

## 23.4 Configuration Options

The Tesla Roadster specific configuration options are in configuration parameter **xtr**:

Instance	Default	Description
digital.speedo	no	Set to <i>yes</i> to enable digital speedometer (2.x cars only)
digital.speedo.reps	3	Number of CAN bus repeat transmissions
cooldown.timelimit	60	Number of minutes after which cooldown is stopped
cooldown.templimit	31	Temperature (in Celcius) after which cooldown is stopped
protect.lock	yes	Refuse to lock vehicle when switched on

## 23.5 Tesla Roadster Notes

1. In general, the OVMS module in a Tesla Roadster acts exactly like the little VDS screen. We should be able to do anything that screen can do, but no more. Here are some notes:
2. The lock/unlock and valet functions rely on a PIN code. This is the same PIN code you enter into the vehicle using the VDS screen when activating valet mode. If you don't know the PIN code, either try the default 1234 or contact Tesla for assistance.
3. While OVMS can lock/unlock the doors of all Tesla Roadster models, cars outside North America are fitted with an immobiliser and neither OVMS nor the VDS will disarm/arm that. The OVMS lock/unlock functionality only applies to the doors, not the alarm in vehicles sold outside North America.
4. OVMS v3 can calculate an overall battery health metric. This metric is calculated using our own algorithm and is in no way approved by Tesla. Battery health is dependent on many factors, and hard to bring down to just one simple number.
5. The Tesla Roadster requires the ignition key to be on, and manual switches turned, to cool/heat the cabin. It is not technically possible to do this remotely via OVMS.

The digital speedometer function replaces the AMPS display in the dashboard instrument cluster with the vehicle speed. It is only supported on v2.x cars (not v1.5). This is an experimental feature, and works 99% of the time, but sometimes the car *wins* and displays AMPS for a split second. A better solution is to use the HUD functionality of OVMS v3 and install an external Heads Up Display in the car.

## 23.6 TPMS Option

Reading and writing TPMS wheel sensor IDs from/to the Baolong TPMS ECU is supported in v2.x Tesla Roadsters using the optional OVMS K-Line Expansion Board. You will need a v3.2 module (or v3.1 module labelled on board as July 2018 or later, with K-line pin connected), and an OVT1 vehicle cable (clearly labelled with "OVT1" on the cable).

The optional OVMS K-line Expansion Board should be configured with SW1 set to position #2 (LDO 5v), and S1 in to OFF position (away from the ON label).

To read the current wheel sensor IDs from the Baolong TPMS ECU, ensure that the ignition switch is ON (so instrument panel lights are on), and issue the 'tpms read' command in OVMS.

Similarly, to write wheel sensor IDs to the Baolong TPMS ECU, ensure that the ignition switch is ON (so instrument panel lights are on), and issue the 'tpms write' command in OVMS.

Note that this functionality will not work with v1.5 Tesla Roadsters (that use a different TPMS ECU to the v2.x cars).



## 23.7 Thanks

There are so many people to thank for Tesla Roadster support in OVMS. W.Petefish for sourcing the car connector, Fuzzylogic for the original hardware and software design and demonstration of it working, Scott451 for figuring out many of the Roadster CAN bus messages, Tom Saxton for v1.5 Roadster testing, Michael Thwaite for pictures of antenna installation, Bennett Leeds for wordsmithing the manual, Mark Webb-Johnson for CAN bus hacking and writing the vehicle module support, Sonny Chen for beta testing and tuning, and many others for showing that this kind of thing can work in the real world.



## Tracking Vehicles

Vehicle Type: **XX**

The track vehicle will be documented here.

## 24.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: Not widely tested, but should be all.
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left), or any cable proving power
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	No
Range Display	No
GPS Location	Yes (from modem module GPS)
Speed Display	Yes (from modem module GPS)
Temperature Display	No
BMS v+t Display	No
TPMS Display	No
Charge Status Display	No
Charge Interruption Alerts	No
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	None



## CHAPTER 25

---

### VW e-Up

---

#### Vehicle Type: **VWUP**

This vehicle type supports the VW e-UP (2013-, 2020-), Skoda Citigo E IV and the Seat MII electric (2020-). Connection can be made via the OBD2 port to the top left of the driving pedals and/or the Comfort CAN bus, e.g. below the passenger seat (T26 connector, instead of the VW OCU there).

The main difference currently is that the OBD connection enables access to way more metrics (e.g. down to cell voltages), while the Comfort CAN connection is necessary if write access is needed, e.g. for remote climate control. The Comfort CAN also provides data in more cases without turning on the car or charging, as the bus wakes on many events (e.g. opening of doors) and can also be woken via OVMS.

For the full experience, making both connections is recommended.

Connection to OBD2 is done with the standard OVMS OBD2-cable just below the fuses left of the driving pedals:





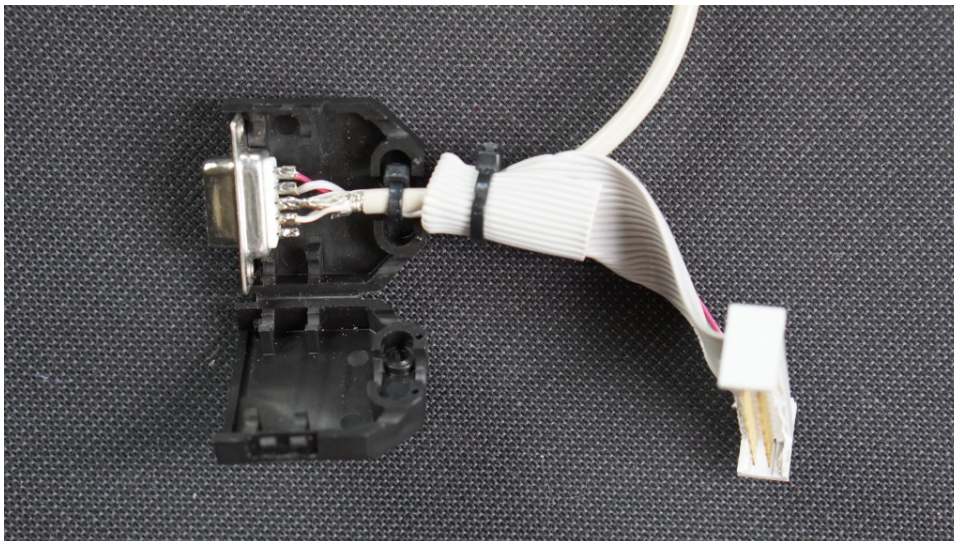
Connection to Comfort CAN can be done by removing the OCU below the passenger seat using a custom 26-pin adapter cable to the T26A plug (advantage: the connections for the GSM & GPS-antennas of the car can be used using a Fakra-SMA adapter):





(The passenger seat doesn't have to be removed, it can also be done by fiddling around a bit :))

If both connections are to be made simultaneously, an adapter cable has to be made with the following connections:



The cable used between the OBD plug and the DB9-F plug needs to be twisted to avoid transmission problems. A good cable to use here is a CAT-5 or CAT-6 double shielded network cable. Be shure to not only connect CAN hi and CAN lo, but also connect ground.

T26	OBD	DB9-F	Signal
26	4	3	Chassis / Power GND
.	14	2	can1 L (Can Low)
.	6	7	can1 H (Can High)
.	.	4	can2 L (Can Low, not used)
.	.	5	can2 H (Can High, not used)
2	.	6	can3 L (Comfort-can Low)
14	.	8	can3 H (Comfort-can High)
1	.	9	+12V Vehicle Power

After selecting the VW e-Up vehicle module, the corresponding settings have to be made in the web interface via the “VW e-Up” menu under “Features”:

The screenshot shows the 'VW e-Up feature configuration' web interface. The top navigation bar includes 'Dashboard', 'Status', 'Tools', 'VW e-Up', and 'Config'. A dropdown menu is open under 'VW e-Up', showing 'Custom Metrics', 'Features' (selected), 'Climate control', and 'Charging Metrics'. The main content area is titled 'VW e-Up feature configuration' and contains three sections: 'Vehicle Settings', 'Connection Types', and 'Remote Control'. In 'Vehicle Settings', the 'Model year' is set to '2019'. Below this, there are two lines of explanatory text. In 'Connection Types', there are two checked options: 'OBD2' (with the note 'CAN1 connected to OBD2 port?') and 'T26A' (with the note 'CAN3 connected to T26A plug underneath passenger seat?'). In 'Remote Control', there is one checked option: 'Enable CAN writes' (with the note 'Controls overall CAN write access, climate control depends on this.'). At the bottom of the form is a 'Save' button.

By default, both connections are activated.

For more details on the two connection types, please see the corresponding projects:

*VW e-Up via Comfort CAN (T26A)*

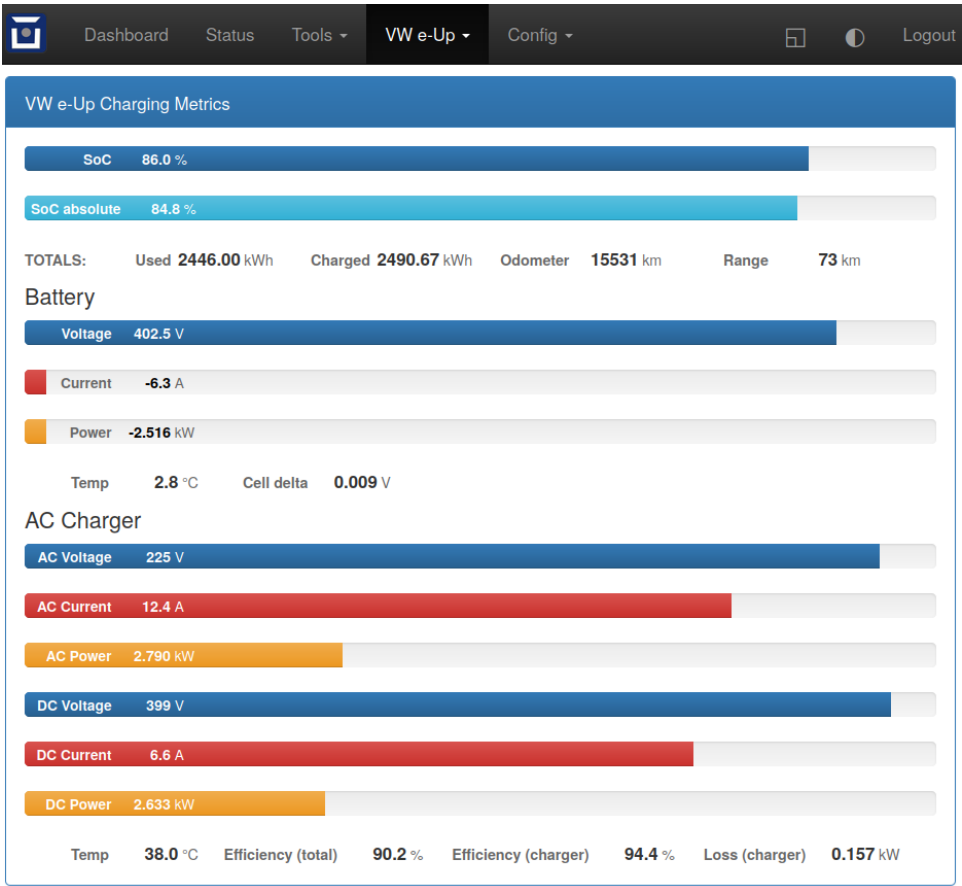
*VW e-Up via OBD2*

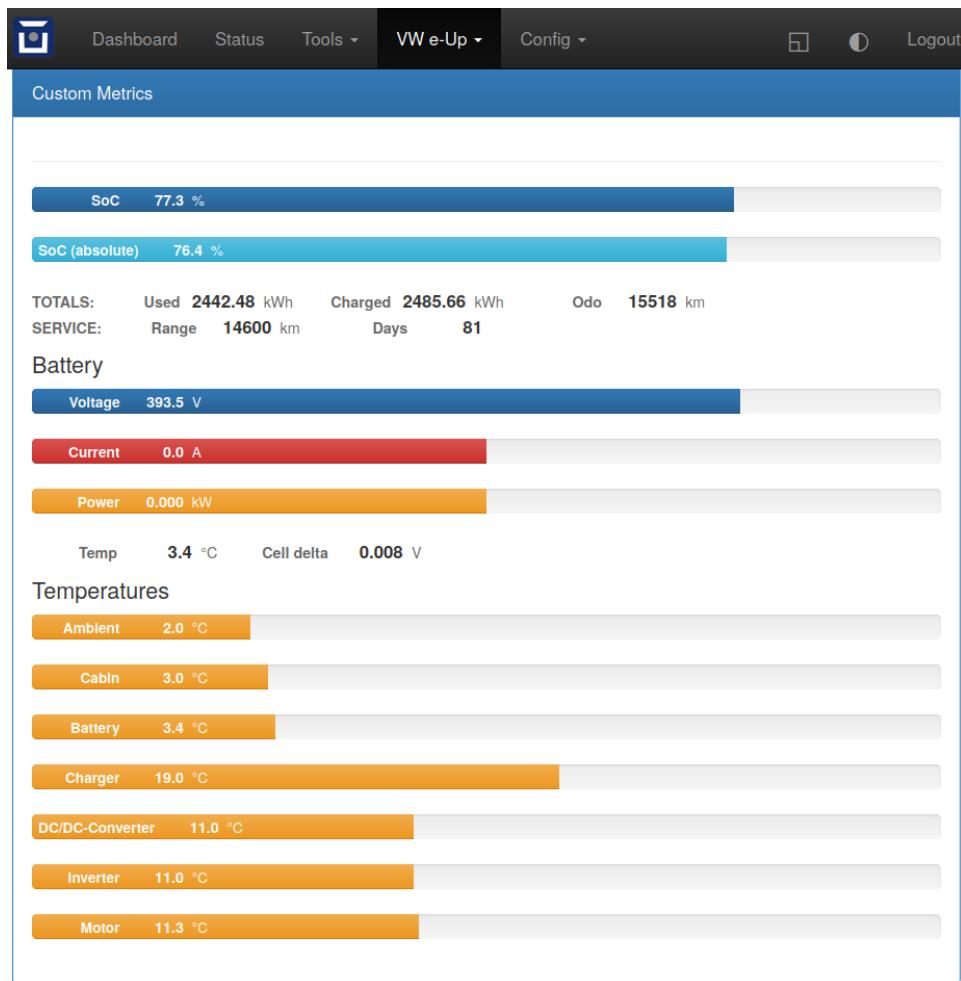
The initial code is shamelessly copied from the original projects for the Comfort CAN by Chris van der Meijden and for the OBD2 port by SokoFromNZ.

List of (possible) metrics via OBD2: <https://www.goingelectric.de/wiki/Liste-der-OB2-Codes/>



If OBD is selected, a sample page with some charging metrics is shown in the web interface:





Beware: obviously, these values have great uncertainties (in my car, the DC output voltage of the charger is always lower than the voltage of the battery...) But e.g. the internal energy counters are very informative :)

Additional custom web pages (code for the example above is below) can be defined as described here: <https://docs.openvehicles.com/en/latest/plugin/README.html?highlight=web%20plugin#installing-web-plugins>

## 25.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: 2013-
Vehicle Cable	Comfort CAN T26A (OCU connector cable, located under front passenger seat) to DB9 Data Cable for OVMS using pin 6 and 8 for can3 _AND_ OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left) for can1
GSM Antenna	T4AC - R205 with fakra_sma adapter cable or 1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	T4AC - R50 with fakra_sma adapter cable or 1020200 Universal GPS Antenna (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
Cabin Pre-heat/cool Control	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes (see list of metrics below)
BMS v+t Display	Yes
TPMS Display	tba
Charge Status Display	Yes
Charge Interruption Alerts	Yes (per notification on the charging state)
Charge Control	tba
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	<b>See list of metrics below</b>

## 25.2 Supported Standard Metrics

The second column specifies the bus from which the metrics are obtained. Metrics via OBD are only updated when the vehicle is on (ignition started) or some in charging mode. Metrics via T26 (Comfort CAN) can be updated on demand by waking the Comfort CAN from the OVMS module. During charging, the Comfort CAN automatically wakes every 5% of SoC.

Metric name	bus	Example value	Description
v.b.12v.voltage	direct	12.9 V	Current voltage of the 12V battery
v.b.consumption	OBD	0Wh/km	Main battery momentary consumption
v.b.current	OBD	23.2 A	Current current into (negative) or out of (positive) the main battery
v.b.energy.recd.total	OBD	578.323 kWh	Energy recovered total (life time) of the main battery (charging and re
v.b.energy.used.total	OBD	540.342 kWh	Energy used total (life time) of the main battery
v.b.power	OBD	23.234 kW	Current power into (negative) or out of (positive) the main battery.
v.b.range.est	T26	99km	Estimated range
v.b.range.ideal	T26	48km	Ideal range
v.b.soc	OBD, T26	88.2 %	Current usable State of Charge (SoC) of the main battery
v.b.temp	OBD	22.5 °C	Current temperature of the main battery
v.b.voltage	OBD	320.2 V	Current voltage of the main battery
v.c.12v.current	OBD	7.8A	Output current of DC/DC-converter
v.c.12v.power	OBD	0.123kW	Output power of DC/DC-converter
v.c.12v.temp	OBD	34.5°C	Temperature of DC/DC-converter
v.c.12v.voltage	OBD	12.3V	Output voltage of DC/DC-converter
v.c.charging	T26	true	Is vehicle charging (true = “Vehicle CHARGING” state. v.e.on=false
v.c.climit	fixed	16/32A	Maximum charger output current
v.c.current	OBD	1.25A	Momentary charger output current
v.c.efficiency	OBD	91.3 %	Charging efficiency calculated by v.b.power and v.c.power
v.c.kwh	OBD	2.6969kWh	Energy sum for running charge
v.c.mode	T26	standard	standard, range, performance, storage
v.c.pilot	T26	no	Pilot signal present
v.c.power	OBD	7.345 kW	Input power of charger
v.c.state	T26	done	charging, topoff, done, prepare, timerwait, heating, stopped
v.c.substate	T26		scheduledstop, scheduledstart, onrequest, timerwait, powerwait, stopp
v.c.temp	OBD	16°C	Charger temperature
v.c.time	T26	0Sec	Duration of running charge
v.c.voltage	OBD	0V	Momentary charger supply voltage
v.d.cp	T26	yes	yes = Charge port open
v.d.fl	T26		yes = Front left door open
v.d.fr	T26		yes = Front right door open
v.d.hood	T26		yes = Hood/frunk open
v.d.rl	T26		yes = Rear left door open
v.d.rr	T26		yes = Rear right door open
v.d.trunk	T26		yes = Trunk open
v.e.awake	T26	no	yes = Vehicle/bus awake (switched on)
v.e.cabintemp	T26	20°C	Cabin temperature
v.e.drivetime	T26	0Sec	Seconds driving (turned on)
v.e.headlights	T26		yes = Headlights on
v.e.hvac	T26		yes = HVAC active
v.e.locked	T26		yes = Vehicle locked
v.e.on	T26	true	Is ignition on and drivable (true = “Vehicle ON”, false = “Vehicle OF
v.e.parktime	T26	49608Sec	Seconds parking (turned off)
v.e.serv.range	OBD	12345km	Distance to next scheduled maintenance/service [km]
v.e.serv.time	OBD	1572590910Sec	Time of next scheduled maintenance/service [UTC]
v.e.temp	OBD, T26		Ambient temperature
v.i.temp	OBD		Inverter temperature
v.m.temp	OBD	0°C	Motor temperature
v.p.odometer	OBD, T26	2340 km	Total distance traveled
v.p.speed	T26	0km/h	Vehicle speed

Continued

Table 1 – continued from previous page

Metric name	bus	Example value	Description
v.vin	T26	VF1ACVYB012345678	Vehicle identification number

## 25.3 Custom Metrics

In addition to the standard metrics above the following custom metrics are read from the car or internally calculated by OVMS using read values.

Metric name	bus	Example value	Description
xvu.b.cell.delta	OBD	0.012 V	Delta voltage between lowest and highest cell voltage
xvu.b.soc.abs	OBD	85.3 %	Current absolute State of Charge (SoC) of the main battery
xvu.c.soc.norm	OBD	80.5 %	Current normalized State of Charge (SoC) of the main battery as reported by charge management ECU
xvu.c.ac.i1	OBD	5.9 A	AC current of AC charger phase 1
xvu.c.ac.i2	OBD	7.0 A	AC current of AC charger phase 2 (only for model 2020+)
xvu.c.ac.p	OBD	7.223 kW	Current charging power on AC side (calculated by ECU's AC voltages and AC currents)
xvu.c.ac.u1	OBD	223 V	AC voltage of AC charger phase 1
xvu.c.ac.u2	OBD	233 V	AC voltage of AC charger phase 2 (only for model 2020+)
xvu.c.dc.i1	OBD	1.2 A	DC current of AC charger 1
xvu.c.dc.i2	OBD	1.2 A	AC current of AC charger 2 (only for model 2020+)
xvu.c.dc.p	OBD	6.500 kW	Current charging power on DC side (calculated by ECU's DC voltages and DC currents)
xvu.c.dc.u1	OBD	380 V	DC voltage of AC charger 1
xvu.c.dc.u2	OBD	375 V	DC voltage of AC charger 2 (only for model 2020+)
xvu.c.eff.calc	OBD	90.0 %	Charger efficiency calculated by AC and DC power
xvu.c.eff.ecu	OBD	92.3 %	Charger efficiency reported by the Charger ECU
xvu.c.loss.calc	OBD	0.733 kW	Charger power loss calculated by AC and DC power
xvu.c.loss.ecu	OBD	0.620 kW	Charger power loss reported by the Charger ECU
xvu.e.serv.days	OBD	78 days	Time to next scheduled maintenance/service [days]
xvu.m.soc.abs	OBD	85.3 %	Current absolute State of Charge (SoC) of the main battery as reported by motor electronics ECU
xvu.m.soc.norm	OBD	80.5 %	Current normalized State of Charge (SoC) of the main battery as reported by motor electronics ECU

## 25.4 Example Code for Web Plugin with some custom metrics:

```
<div class="panel panel-primary">
  <div class="panel-heading">Custom Metrics</div>
  <div class="panel-body">

    <hr/>

    <div class="receiver">
      <div class="clearfix">
        <div class="metric progress" data-metric="v.b.soc" data-prec="1">
          <div class="progress-bar value-low text-left" role="progressbar">
```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

    <span class="label">Voltage</span>
    <span class="value">?</span>
    <span class="unit">V</span>
  </div>
</div>
</div>
<div class="metric progress" data-metric="v.b.current" data-prec="1">
  <div class="progress-bar progress-bar-danger value-low text-left" role=
↪ "progressbar"
    aria-valuenow="0" aria-valuemin="-200" aria-valuemax="200" style="width:0%">
    <div>
      <span class="label">Current</span>
      <span class="value">?</span>
      <span class="unit">A</span>
    </div>
  </div>
</div>
<div class="metric progress" data-metric="v.b.power" data-prec="3">
  <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
    aria-valuenow="0" aria-valuemin="-70" aria-valuemax="70" style="width:0%">
    <div>
      <span class="label">Power</span>
      <span class="value">?</span>
      <span class="unit">kW</span>
    </div>
  </div>
</div>
</div>
<div class="clearfix">
  <div class="metric number" data-metric="v.b.temp" data-prec="1">
    <span class="label">Temp</span>
    <span class="value">?</span>
    <span class="unit">°C</span>
  </div>
  <div class="metric number" data-metric="xvu.b.cell.delta" data-prec="3">
    <span class="label">Cell delta</span>
    <span class="value">?</span>
    <span class="unit">V</span>
  </div>
</div>
<h4>Temperatures</h4>

<div class="clearfix">
  <div class="metric progress" data-metric="v.e.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">Ambient</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.e.cabintemp" data-prec="1">

```

(continues on next page)

(continued from previous page)

```

    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">Cabin</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.b.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">Battery</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.c.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">Charger</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.c.12v.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">DC/DC-Converter</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.i.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"
      aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">
      <div>
        <span class="label">Inverter</span>
        <span class="value">?</span>
        <span class="unit">°C</span>
      </div>
    </div>
  </div>
  <div class="metric progress" data-metric="v.m.temp" data-prec="1">
    <div class="progress-bar progress-bar-warning value-low text-left" role=
↪ "progressbar"

```

(continues on next page)



(continued from previous page)

```
aria-valuenow="0" aria-valuemin="-10" aria-valuemax="40" style="width:0%">>
<div>
  <span class="label">Motor</span>
  <span class="value">?</span>
  <span class="unit">°C</span>
</div>
</div>
</div>
</div>
</div>
</div>
```



## CHAPTER 26

---

### VW e-Up via OBD2

---

Vehicle Type: **VWUP.OBD**

This vehicle type supports the VW e-UP (new model from year 2020 onwards). Untested (so far) but probably working: The older models of the e-Up as well as Skoda Citigo E IV and Seat MII electric.

Connection is via the standard OBD-II port (above the drivers left foot):

All communication with the car is read-only. For changing values (i.e. climate control) see the T26A connection to the Comfort CAN bus.

### 26.1 Support Overview

Function	Support Status
Hardware	No specific requirements
Vehicle Cable	OBD-II to DB9 Data Cable for OVMS (1441200 right, or 1139300 left)
GSM Antenna	1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	1020200 Universal GPS Antenna (SMA Connector) (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
GPS Location	Yes
Speed Display	Yes
Temperature Display	Yes
BMS v+t Display	Yes (including cell details)
TPMS Display	Yes
Charge Status Display	Yes
Charge Interruption Alerts	Yes
Charge Control	No
Cabin Pre-heat/cool Control	No
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	<b>See list of metrics below</b>

## 26.2 Vehicle States

**Warning:** For proper state detection, the **12V calibration is crucial**. Calibrate using the OVMS Web UI: Config → Vehicle → 12V Monitor

Three vehicle states are supported and detected automatically:

**Vehicle ON** The car is on: It is drivable.

**Vehicle CHARGING** The car is charging: The car's Charger ECU is responsive and reports charging activity.

**Vehicle OFF** The car is off: It hasn't drawn (or charged) any current into the main battery for a period of time and the 12V battery voltage is smaller than 12.9V.

## 26.3 Supported Standard Metrics

Metrics updated in state “Vehicle ON” or “Vehicle CHARGING”

Metric name	Example value	Description
v.e.on	yes	Is ignition on and drivable (true = “Vehicle ON”, false = “Vehicle OFF” state)
v.c.charging	yes	Is vehicle charging (true = “Vehicle CHARGING” state. v.e.on=false if this is true)
v.c.limit.soc	100%	Current/next charge timer mode SOC destination
v.c.mode	range	“range” = charging to 100% SOC, else “standard”
v.c.timermode	no	Yes = current/next charge under timer control
v.c.state	done	charging, stopped, done
v.c.substate	scheduledstop	scheduledstop, scheduledstart, onrequest, timerwait, stopped, interrupted
v.b.12v.voltage <sup>1</sup>	12.9 V	Current voltage of the 12V battery
v.b.voltage	320.2 V	Current voltage of the main battery
v.b.current	23.2 A	Current current into (negative) or out of (positive) the main battery
v.b.power	23.234 kW	Current power into (negative) or out of (positive) the main battery.
v.b.energy.used.total	540.342 kWh	Energy used total (life time) of the main battery
v.b.energy.recd.total	578.323 kWh	Energy recovered total (life time) of the main battery (charging and recuperation)
v.b.temp	22.5 °C	Current temperature of the main battery
v.p.odometer	2340 km	Total distance traveled

Metrics updated only in state “Vehicle ON”

Metric name	Example value	Description
v.b.soc <sup>2</sup>	88.2 %	Current usable State of Charge (SoC) of the main battery

Metrics updated only in state “Vehicle CHARGING”

<sup>1</sup> Also updated in state “Vehicle OFF”

<sup>2</sup> Restriction by the ECU. Supplied when the ignition is on during charging. Use xv.u.b.soc as an alternative when charging with ignition off.

Metric name	Example value	Description
v.c.power	7.345 kW	Input power of charger
v.c.efficiency	91.3 %	Charging efficiency calculated by v.b.power and v.c.power

## 26.4 Custom Metrics

In addition to the standard metrics above the following custom metrics are read from the car or internally calculated by OVMS using read values.

### State metrics

Metric name	Example value	Description
xvu.e.hv.chgmode	0	High voltage charge mode; 0=off, 1=Type2, 4=CCS
xvu.e.lv.autochg	1	Auxiliary battery (12V) auto charge mode (0/1)
xvu.e.lv.pwrstate	0	Low voltage (12V) power state (0=off, 4=12V, 8=HVAC, 15=on)

### Timed charge metrics

Metric name	Example value	Description
xvu.c.limit.soc.max	80%	Charge schedule maximum SOC
xvu.c.limit.soc.min	20%	Charge schedule minimum SOC
xvu.c.timermode.def	yes	Charge timer defined & default

`xvu.c.timermode.def` tells if a charge schedule has been configured and enabled. If so, the car uses timed charging by default (the charge mode button will be lit). `v.c.timermode` tells if the charge timer is or will actually be used for the current or next charge, i.e. reflects the mode selected by pushing the button.

With timed charging, the car first charges to the minimum SOC as soon as possible (when connected). If the maximum SOC configured for the schedule hasn't been reached by then, it will then wait for the timer to signal the second phase to charge up to the maximum SOC. `v.c.limit.soc` reflects the current phase, i.e. will be the minimum SOC during phase 1, the maximum (if configured) during phase 2. After reaching the timer defined final SOC, it will switch to 100%.

Note: `xvu.c.limit.soc.min` will show the configured minimum SOC also if no schedule is currently enabled. `xvu.c.limit.soc.max` shows the maximum for the current/next schedule to apply. If no schedule is enabled, it will be zero.

### Metrics updated in state “Vehicle ON” or “Vehicle CHARGING”

Metric name	Example value	Description
xvu.b.cell.delta	0.012 V	Delta voltage between lowest and highest cell voltage
xvu.b.soc	85.3 %	Current absolute State of Charge (SoC) of the main battery

### Metrics updated only in state “Vehicle CHARGING”

Metric name	Example value	Description
xvu.c.eff.ecu <sup>3</sup>	92.3 %	Charger efficiency reported by the Charger ECU
xvu.c.loss.ecu <sup>3</sup>	0.620 kW	Charger power loss reported by the Charger ECU
xvu.c.ac.p	7.223 kW	Current charging power on AC side (calculated by ECU's AC voltages and AC currents)
xvu.c.dc.p	6.500 kW	Current charging power on DC side (calculated by ECU's DC voltages and DC currents)
xvu.c.eff.calc	90.0 %	Charger efficiency calculated by AC and DC power
xvu.c.loss.calc	0.733 kW	Charger power loss calculated by AC and DC power
xvu.c.ccs.u <sup>4</sup>	331.5V	CCS charger supplied voltage [V]
xvu.c.ccs.i <sup>4</sup>	62.2A	CCS Charger supplied current [A]
xvu.c.ccs.p <sup>4</sup>	20.6193kW	CCS Charger supplied power [kW]

## 26.5 Battery Capacity & SOH

e-Up Model	Total capacity	Usable capacity
Gen 1 (2016)	18.7 kWh / 50 Ah	16.4 kWh / 43.9 Ah (87.7%)
Gen 2 (2020)	36.8 kWh / 120 Ah	32.3 kWh / 105.3 Ah (87.7%)

There are currently two ways to get an estimation of the remaining capacity of the e-Up:

1. By deriving a usable energy capacity from the MFD range estimation.
2. By deriving a total coulomb capacity from the coulombs charged.

---

**Note:** Consider the capacity estimations as experimental / preliminary. We need field data to optimize the readings. If you'd like to help us, see below.

---

The **MFD range estimation** seems to include some psychological factors with an SOC below 30%, so we only provide this and the derived capacity in two custom metrics. The capacity derivation is only calculated with SOC  $\geq$  30%, but if so is available immediately after switching the car on. This can serve as a quick first estimation, relate it to the usable capacity of your model.

The **charge coulomb based estimation** provides a better estimation but will need a little more time to settle. Usable measurements need charges of at least 30% SOC, the more the better. Estimations are only calculated if a charge has exceeded 30% SOC, and results are smoothed over multiple charges to provide stable readings.

- To get a rough capacity estimation, charge at least 30% normalized SOC difference.
- To get a good capacity estimation, do at least three charges with each covering 60% or more normalized SOC difference.

Charging by CCS (DC) apparently yields higher results, especially on the energy estimations. We don't know yet the reason or if we need to compensate this.

Note: the **SOH** (state of health) is currently coupled directly and solely to the calculated amp-hour capacity **CAC**.

To **log your capacity data** on a connected V2 server, do:

---

<sup>3</sup> Only supplied by ECU when the car ignition is on during charging.

<sup>4</sup> These are not measurements by the car but provided as is by the charger and typically deviate from the battery metrics. According to IEC 61851, CCS currents may be off by +/- 3% and voltages by +/- 5%. The power figures displayed by some chargers also typically won't match these values, possibly because the charger displays the power drawn from the grid (including losses).

```
OVMS# config set xvu log.chargecap.storetime 30
```

30 is the number of days to keep the data, set to 0 to disable. The counters will be stored in table XVU-LOG-ChargeCap, with one entry every 2.4% absolute SOC difference. Resulting CAC/SOH updates will be logged in table XVU-LOG-ChargeCapSOH. You can also extract the data from your module log file by filtering lines matching ChargeCap.

### 26.5.1 Capacity and SOH metrics

Metric name	Example value	Description
xvu.b.cap.ah.abs	122.71Ah	Total coulomb capacity estimation
xvu.b.cap.ah.norm	113.63Ah	Usable coulomb capacity estimation
xvu.b.cap.kwh.abs	39.1kWh	Total energy capacity estimation
xvu.b.cap.kwh.norm	36.21kWh	Usable energy capacity estimation
xvu.b.cap.kwh.range	32.8947kWh	Usable energy capacity estimation from MFD range
xvu.b.energy.range	18.5kWh	Current energy used by MFD range estimation

### 26.5.2 Provide Data to the Developers

To help us with optimizing the capacity estimations, first of all enable file logging if not already enabled. Then enable extended polling and logging before a charge by...:

```
OVMS# config set xvu dc_interval 30
OVMS# log level verbose v-vweup
```

After the charge, disable the extended polling and logging:

```
OVMS# config set xvu dc_interval 0
OVMS# log level info v-vweup
```

Then download all log files written during the charge (archived and current), zip them and mail the zip to Michael Balzer <dexter@dexters-web.de>. The log data will only be used for technical analysis and deleted afterwards.

Note: if you forgot enabling the local log but still have chargecap logs on the server: these can help as well.

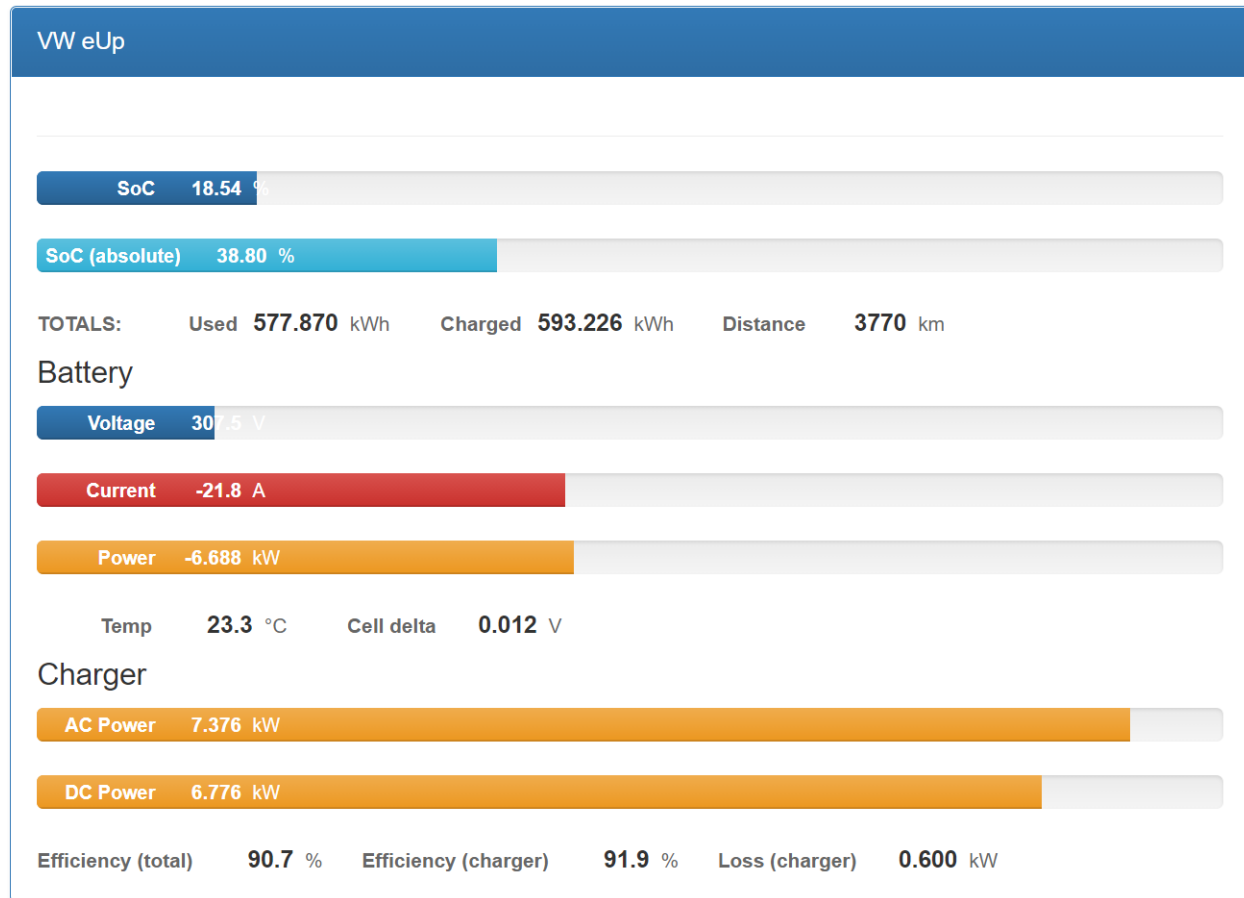
**Thanks!**

## 26.6 Custom Status Page for Web UI

**Note:** This plugin is obsolete, use the standard page **VW e-Up → Charging Metrics** instead. We keep the source here as a base for user customization.

The easiest way to display custom metrics is using the *Web Plugins* feature of OVMS (see *Installing Web Plugins*).

This page plugin content shows the metrics in a compact form which can be displayed on a phone in landscape mode on the dashboard of the car. Best approach is to connect the phone directly to the OVMS AP-WiFi and access the web UI via the static IP (192.168.4.1) of OVMS.



```

<div class="panel panel-primary">
  <div class="panel-heading">VW eUp</div>
  <div class="panel-body">

    <hr/>

    <div class="receiver">
      <div class="clearfix">
        <div class="metric progress" data-metric="v.b.soc" data-prec="2">
          <div class="progress-bar value-low text-left" role="progressbar"
            aria-valuenow="0" aria-valuemin="0" aria-valuemax="100" style="width:0%">
            <div>
              <span class="label">SoC</span>
              <span class="value">?</span>
              <span class="unit">%</span>
            </div>
          </div>
        </div>
        <div class="metric progress" data-metric="xvu.b.soc" data-prec="2">
          <div class="progress-bar progress-bar-info value-low text-left" role=
            ↪ "progressbar"
            aria-valuenow="0" aria-valuemin="0" aria-valuemax="100" style="width:0%">
            <div>
              <span class="label">SoC (absolute)</span>
              <span class="value">?</span>
              <span class="unit">%</span>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

```

(continues on next page)



```

</div>
</div>
</div>
</div>
<div class="clearfix">
<div class="metric number" data-metric="v.b.energy.used.total" data-prec="3">
  <span class="label">TOTALS:<span>
  ↳<span>Used</span>
  <span class="value">?</span>
  <span class="unit">kWh</span>
</div>
<div class="metric number" data-metric="v.b.energy.recd.total" data-prec="3">
  <span class="label">Charged</span>
  <span class="value">?</span>
  <span class="unit">kWh</span>
</div>
<div class="metric number" data-metric="v.p.odometer" data-prec="0">
  <span class="label">Distance</span>
  <span class="value">?</span>
  <span class="unit">km</span>
</div>
</div>
<h4>Battery</h4>
<div class="clearfix">
<div class="metric progress" data-metric="v.b.voltage" data-prec="1">
  <div class="progress-bar value-low text-left" role="progressbar"
  aria-valuenow="0" aria-valuemin="300" aria-valuemax="350" style="width:0%">
  <div>
    <span class="label">Voltage</span>
    <span class="value">?</span>
    <span class="unit">V</span>
  </div>
</div>
<div class="metric progress" data-metric="v.b.current" data-prec="1">
  <div class="progress-bar progress-bar-danger value-low text-left" role=
  ↳"progressbar"
  aria-valuenow="0" aria-valuemin="-200" aria-valuemax="200" style="width:0%">
  <div>
    <span class="label">Current</span>
    <span class="value">?</span>
    <span class="unit">A</span>
  </div>
</div>
<div class="metric progress" data-metric="v.b.power" data-prec="3">
  <div class="progress-bar progress-bar-warning value-low text-left" role=
  ↳"progressbar"
  aria-valuenow="0" aria-valuemin="-70" aria-valuemax="70" style="width:0%">
  <div>
    <span class="label">Power</span>
    <span class="value">?</span>
    <span class="unit">kW</span>
  </div>
</div>

```

(continues on next page)

(continued from previous page)

```

</div>
</div>
<div class="clearfix">
<div class="metric number" data-metric="v.b.temp" data-prec="1">
  <span class="label">Temp</span>
  <span class="value">?</span>
  <span class="unit">°C</span>
</div>
<div class="metric number" data-metric="xvu.b.cell.delta" data-prec="3">
  <span class="label">Cell delta</span>
  <span class="value">?</span>
  <span class="unit">V</span>
</div>
</div>

<h4>Charger</h4>

<div class="clearfix">
<div class="metric progress" data-metric="xvu.c.ac.p" data-prec="3">
  <div class="progress-bar progress-bar-warning value-low text-left" role=
→ "progressbar"
    aria-valuenow="0" aria-valuemin="0" aria-valuemax="8" style="width:0%">
    <div>
      <span class="label">AC Power</span>
      <span class="value">?</span>
      <span class="unit">kW</span>
    </div>
  </div>
</div>
<div class="metric progress" data-metric="xvu.c.dc.p" data-prec="3">
  <div class="progress-bar progress-bar-warning value-low text-left" role=
→ "progressbar"
    aria-valuenow="0" aria-valuemin="0" aria-valuemax="8" style="width:0%">
    <div>
      <span class="label">DC Power</span>
      <span class="value">?</span>
      <span class="unit">kW</span>
    </div>
  </div>
</div>
</div>
<div class="clearfix">
<div class="metric number" data-metric="v.c.efficiency" data-prec="1">
  <span class="label">Efficiency (total)</span>
  <span class="value">?</span>
  <span class="unit">%</span>
</div>
<div class="metric number" data-metric="xvu.c.eff.calc" data-prec="1">
  <span class="label">Efficiency (charger)</span>
  <span class="value">?</span>
  <span class="unit">%</span>
</div>
<div class="metric number" data-metric="xvu.c.loss.calc" data-prec="3">
  <span class="label">Loss (charger)</span>
  <span class="value">?</span>
  <span class="unit">kW</span>
</div>

```

(continues on next page)

(continued from previous page)

```
</div>  
</div>  
</div>  
</div>
```



## CHAPTER 27

---

### VW e-Up via Comfort CAN (T26A)

---

Vehicle Type: integrated in **VWUP**

This part was the initial code for the OVMS VWUP vehicle module. Development started in January 2020 by Chris van der Meijden.

It supports the VW e-UP (2013-, 2020-), Skoda Citigo E IV and the Seat MII electric (2020-) directly connected to the comfort can bus through the 'T26A' socket.

## 27.1 Support Overview

Function	Support Status
Hardware	Any OVMS v3 (or later) module. Vehicle support: 2020- (2013- VW e-Up as well)
Vehicle Cable	Comfort CAN T26A (OCU connector cable, located under front passenger seat) to DB9 Data Cable for OVMS using pin 6 and 8 for can3
GSM Antenna	T4AC - R205 with fakra_sma adapter cable or 1000500 Open Vehicles OVMS GSM Antenna (or any compatible antenna)
GPS Antenna	T4AC - R50 with fakra_sma adapter cable or 1020200 Universal GPS Antenna (or any compatible antenna)
SOC Display	Yes
Range Display	Yes
Cabin Pre-heat/cool Control	Yes
GPS Location	Yes (from modem module GPS)
Speed Display	Yes
Temperature Display	Yes (outdoor, cabin)
BMS v+t Display	No
TPMS Display	No
Charge Status Display	Yes
Charge Interruption Alerts	Yes (per notification on the charging state)
Charge Control	tba
Lock/Unlock Vehicle	No
Valet Mode Control	No
Others	Odometer, trip, VIN, status of lock, plug, lights, doors, trunk and bonnet

## 27.2 Pinout OCU T26A - OVMS DB9 adapter

For the T26A approach we directly tap into the comfort can bus via the original OCU cable.

The OCU connector is located under the passenger seat.



Advantage is the direct write access to the comfort can bus.

Disadvantage is that we won't be able to access all control units of the car.

OCU	DB9-F	Signal
26	3	Chassis / Power GND
.	2	can1 L (Can Low, not used)
.	7	can1 H (Can High, not used)
.	4	can2 L (Can Low, not used)
.	5	can2 H (Can High, not used)
2	6	can3 L (Comfort-can Low)
14	8	can3 H (Comfort-can High)
1	9	+12V Vehicle Power

For confectioning the T26A adapter cable you can use a standard 26 pin ribbon cable (2x13 pins, 2,54mm grid dimension) and a DB9 female D-Sub connector. You will need to grind down the rim of the socket of the ribbon cable.

To make a GSM/GPS adapter cable to connect to the original VW fakra socket you can use a double fakra male connector with two SMA male connectors attached.

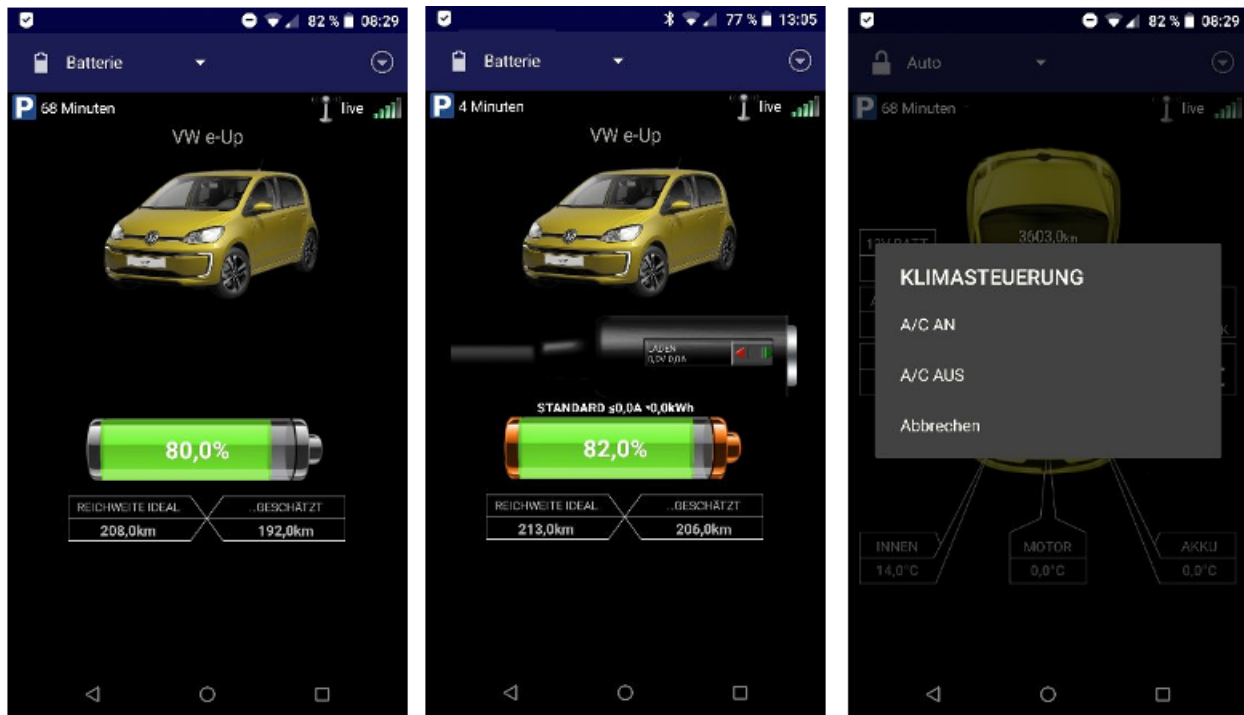


### 27.3 User notes

- Remove the passenger seat (on right hand drive cars the driver's seat).
- Open the carpet lid.
- Disconnect the T26A cable and the fakra cable from the OCU (online communication unit).
- Remove the OCU.
- Connect the confectioned T26A adapter DB9 cable attached to the OVMS to the VW T26A connector of the car.
- Connect your GMS/GPS fakra adapter to the VW fakra cable.
- Fit OVMS in the compartment.
- Close the carpet lid.
- Reinstall the passenger seat.
- Configure OVMS as described in the OVMS user manual.
- Configure 'Model year', 'Can write access' and 'Connection type' under VW e-Up -> Features.
- 'Model year' and 'Can write access' can also be set from within the app (FEATURES 20 and 15).
- Register and connect OVMS to a server (as guided within the OVMS setup).
- Turn the ignition in the car on and off to receive initial values (also needed after updates).
- Install the OVMS app on your smartphone or tablet and configure it to connect to the server.



- Enjoy :-)



## 27.4 Climate control

Climate control works, as long as write access to the comfort can has been enabled in the app or in the OVMS webinterface (VW e-Up -> Features).

To turn on or off the AC from within the Android app just press the “A/C” button. Within the iOS app press “Homelink 1” for AC on and “Homelink 2” for AC off.

Once the AC is turned on by the app there will be a delay of about 17 seconds until the AC actually starts in the car. Further 10 seconds all communication from the app to the car is blocked.

The communication from the app to the car is also blocked for 10 seconds after the “AC off” command from the app to the car. There is no delay between the “AC off” signal of the app and the actually turning off in the car.

In rare cases ‘AC off’ does not respond. There will be a delay of 40 seconds before you can try again.

The cabin target temperature can be set from the OVMS webinterface (VW e-Up (Komfort CAN) -> Climate control) or via the app under FEATURES 21.

Dashboard
Status
Tools
VW e-Up
Config
Logout

VW e-Up climate control configuration

This page offers remote climate configuration.

The target temperature for the cabin can be set here (15 to 30 °C).

### Climate control

**Cabin target temperature:**

°C

Default 22 °C, 15=Lo, 30=Hi.

This parameter can also be set in the app under FEATURES 21.

## 27.5 IDs on Comfort CAN Bus

ID	Conversion	Unit	Function
61A	d7/2	%	State of Charge (relative)
320	(d4<<8+d3-1)/190	km/h	Speed
65F	3 Msg d5-7,d1-7,d1-7	String	VIN number
571	5+(.05*d0)	Volt	12 Volt battery voltage
65D	d3&f<<12ld2<<8ld1	km	Odometer
3E3	(d2-100)/2	°C	Cabin temperature
527	(d5/2)-50	°C	Outdoor temperature
531	d0 00		Headlights off
52D	d0 +255 if d1 41	km	Calculated range
381	d0 02		Status doors locked
470	d1 1,2,4,8,20,10		Integer Doors, trunk, hood opened or closed
3E1	d4		Integer Blower speed?(57,66,7D,98,BB,DE,FA)
575	d0 00 to 0F		Integer Key position
575	d3 00 or 10		windshield heater (off or on)
569	b07		“AC”-LED
69C	d1/10+10	°C	temperature setpoint for remote AC (only in message D2 <d1> 00 1E 1E 0A 00 00)
61C	d2 < 07	bool	Charging detection
43D	d1 01 or 11		TX: Working or sleeping in the ring
5A7	d1 16		TX: OCU AC blocking signal
5A9	all 00		TX: OCU heartbeat
69E	multiple msg d0 C1 d6 xx	°C	TX: AC on / off signals TX: set cabin temperature for 69C

## 27.6 Development notes

Under this vehicle component part we use the original T26A approach, which can write to the comfort can and is able to manage the climate control of the car.

You will normally use the OVMS binaries provided i.e. [here](#).

The VWUP T26A component part with working climate control is publicly available within the OVMS binary version 3.2.15 'edge' and upwards ('main', 'eap' and 'edge'). The Android app version has to be 3.17.1 or higher to have access to the climate control functions for this vehicle component.

If you want to compile the binary yourself you will need to read the OVMS development documentation on how to set up the tool chain, check out the repository and the submodules and copy the file

```
sdkconfig.default.hw31
```

from the OVMS.V3/support folder to the OVMS.V3 folder and rename it to

```
sdkconfig
```

## 27.7 Vehicle log files

To be able to implement the VWUP vehicle component for OVMS the CAN logging of the VW e-UP provided by 'sharkcow' was of tremendous help.

The implementation of this vehicle component could not have been done without these great files.

They can be found here:

<https://github.com/sharkcow/VW-e-UP-OBD-CAN-logs/>



---

### Command Line Interpreter

---

The command line interpreter or command parser presented by the OVMS async serial console is constructed as a tree of command word tokens. Enter a single question mark followed by RETURN to get the root command list, like this:

```
OVMS# ?  
.  
bms  
boot  
can  
...  
Run a script  
BMS framework  
BOOT framework  
CAN framework
```

A root command may be followed by one of a list of additional tokens called subcommands which in turn may be followed by further subcommands down multiple levels, forming a tree. The command and subcommand tokens may be followed by parameters. Use question mark at any level in the command sequence to get the list of subcommands applicable at that point. If the next item to be entered is a parameter rather than a subcommand, then a usage message will be displayed to indicate the required or optional parameters. The usage message will also be shown if the command as entered is not valid. The usage message is described in further detail below.

Command tokens can be abbreviated so long as enough characters are entered to uniquely identify the command. Optionally pressing TAB at that point will auto-complete the token. If the abbreviated form is not sufficient to be unique (in particular if no characters have been entered yet) then TAB will show a concise list of the possible subcommands and retype the portion of the command line already entered so it can be completed. Pressing TAB is legal at any point in the command; if there is nothing more that can be completed automatically then there will just be no response to the TAB.

### 28.1 OvmsCommand API

Each command word token in the tree is represented by an `OvmsCommand` object. The `OvmsCommand::RegisterCommand()` function is used to create and add a command or subcommand token object into the tree, returning an `OvmsCommand*` pointer to the new object. New commands are added to the root of the tree using the global `MyCommandApp.RegisterCommand()`. Subcommands are added as children of a command by calling `RegisterCommand()` using the `OvmsCommand*` pointer to the parent object, thus building the tree. For example:

```
OvmsCommand* cmd_wifi = MyCommandApp.RegisterCommand("wifi","WIFI framework", wifi_
↪status);
cmd_wifi->RegisterCommand("status","Show wifi status",wifi_status);
cmd_wifi->RegisterCommand("reconnect","Reconnect wifi client",wifi_reconnect);
```

The `RegisterCommand()` function takes the following arguments:

- `const char* name` – the command token
- `const char* title` – one-line description for the command list
- `void (*execute) (...)` – does the work of the command
- `const char *usage` – parameter description for “Usage:” message
- `int min` – minimum number of parameters allowed
- `int max` – maximum number of parameters allowed
- `bool secure` – true for commands permitted only after *enable*
- `int (*validate) (...)` – validates parameters as explained later

The `RegisterCommand()` function tolerates duplicate registrations of the same `name` at the same node of the tree by assuming that the other arguments are also the same and returning the existing object. This allows multiple modules that can be configured independently to share the same top-level command. For example, the *obdii* command is shared by the **vehicle** and **obd2ecu** modules.

Modules that can be dynamically loaded and unloaded must remove their commands from the tree using `UnregisterCommand(const char* name)` before unloading.

It’s important to note that many of the arguments to `RegisterCommand()` can and should be defaulted. The default values are as follows:

```
execute = NULL
usage = ""
min = 0
max = 0
secure = true
validate = NULL
```

For example, for secure, non-terminal commands (those with child subcommands), such as the top-level framework commands like *bms* in the list of root commands shown earlier, the model should simply be:

```
RegisterCommand("name", "Title");
```

For secure, terminal subcommands (those with no children) that don’t require any additional parameters, the model should be:

```
RegisterCommand("name", "Title", execute);
```

This model also applies if the command has children but the command itself wants to execute a default operation if no subcommand is specified. It is incorrect to specify `min = 0, max = 1` to indicate an optional subcommand; that is indicated by the presence of the `execute` function along with a non-empty array of child subcommands.

Any command with required or optional parameters should provide a “usage” string hinting about the parameters in addition to specifying the minimum and maximum number of parameters allowed:

```
RegisterCommand("name", "Title", execute, "usage", min, max);
```

The `usage` argument only needs to describe the parameters that follow this (sub)command because the full usage message is dynamically generated. The message begins with the text “Usage: ” followed by the names of the ancestors of this subcommand back to the root of the tree plus the name of this subcommand itself. That is, the message starts with all the tokens entered to this point. The message continues with a description of subcommands and/or parameters that may be entered next, as specified by the `usage` string.

**Note:** The usage message is *not* restricted to a single line; the `usage` string can include additional lines of explanatory text, separated by `\n` (newline) characters, to help convey the meaning of the parameters and the purpose of the command.

The `usage` string syntax conventions for specifying alternative and optional parameters are similar to those of usage messages in Unix-like systems. The string can also include special codes to direct the dynamic generation of the message:

- `$C` expands to the list of children commands as `child1|child2|child3`.
- `[$C]` expands to list optional children as `[child1|child2|child3]`.
- `$G$` expands to the usage string of the first child; this would typically be used after `$C` so the usage message shows the list of children and then the parameters or next-level subcommands that can follow the children. This is useful when the usage string is the same for all or most of the children as in this example:

```
Usage: power adc|can1|can2|can3|egpio|esp32|sdcard|simcom|spi|wifi_
↳deepsleep|devel|off|on|sleep|status
```

- `$Gfoo$` expands to the usage of the child named “foo”; this variant would be used when not all the children have the same usage but it would still be helpful to show the usage of one that’s not first.
- `$L` lists a full usage message for each of the children on separate lines. This provides more help than just showing the list of children but at the expense of longer output.

For subcommands that take parameters, the `usage` string contains explicit text to list the parameters:

- Parameter names or descriptions are enclosed in angle brackets to distinguish them from command tokens, for example `<metric> <value>`. Since the angle brackets demarcate each parameter, spaces may be included in the description.
- Parameters that are optional are further enclosed in square brackets, like `<id> <name> [<value>]`.
- When there are alternative forms or meanings for a parameter, the alternatives are separated by vertical bar as in `<task names or ids>|*|=` which indicates that the parameter can be either of the characters `*` or `=` instead of a list of task names or ids. A variant form encloses the alternatives in curly braces as in `<param> {<instance> | *}`.
- One or more additional lines of explanatory text can be included like this:

```
"<id>\nUse ID from connection list / 0 to close all"
```

For non-terminal commands (those with children subcommands) the `usage` argument can be omitted because the default value of `" "` is interpreted as `$C`. For commands that have children subcommands that are optional (because an `execute` function is included) the default `usage` argument is interpreted as `[$C]`.

### 28.1.1 Execute Function

The `execute` function performs whatever work is required for the command. Its signature is as follows:

```
void (*execute)(int verbosity, OvmsWriter* writer, OvmsCommand* cmd, int argc, const_
↳char* const* argv)
```

- `int verbosity` – tells how much output is appropriate (e.g., shell vs. SMS)
- `OvmsWriter* writer` – object to which output is delivered, e.g. console
- `OvmsCommand* cmd` – the command that held the `execute` function pointer
- `int argc` – how many parameters are being supplied to the function
- `const char* const* argv` – the parameter list

Any output appropriate for the command is accomplished through `puts()` or `printf()` calls on the `writer` object. The `cmd` pointer may allow sharing one `execute` function among multiple related command objects and provides access to members of the command object such as `GetName()`.

The `argc` count will be constrained to the `min` and `max` values specified for the `cmd` object, so if the minimum and maximum are the same then the `execute` function does not need to check. However, if parameters are expected then their values must be validated.

### 28.1.2 Validate Function

Most commands do not need to specify a `validate` function. It supports extensions of the original command parser design for two use cases:

1. For commands that store the possible values of a parameter in a `NameMap<T>` or `CNameMap<T>`, the `validate` function enables TAB auto-completion when entering that parameter.
2. The original design only allowed parameters to be collected by the terminal subcommand. That forced an unnatural word order for some commands. The `validate` function enables non-terminal subcommands to take one or more parameters followed by multiple levels of children subcommands. The parameters may be strings looked up in a `NameMap<T>` or `CNameMap<T>` or they could be something else like a number that can be validated by value. The `validate` function must indicate success for parsing to continue to the children subcommands. The return value is the number of parameters validated if successful or -1 if not.

The signature of the `validate` function is as follows:

```
int (*validate)(OvmsWriter* writer, OvmsCommand* cmd, int argc, const char* const*_
↳argv, bool complete)
```

- `OvmsWriter* writer` – object to which output is delivered, e.g. console
- `OvmsCommand* cmd` – the command that held the `validate` function pointer
- `int argc` – how many parameters are being supplied to the function
- `const char* const* argv` – the parameter list
- `bool complete` – true for TAB completion of the last parameter (case 1), false when validating intermediate parameters before calling `execute` on the terminal descendant command (case 2)

The `writer` and `cmd` arguments are the same as for the `execute` function. The `argc` count is never more than `max` and, if `complete` is false, never less than `min`. However, when `complete` is true to request TAB auto-completion and `max` is greater than 1, `argc` will be at least 1 but may be less than `min` because it indicates how many parameters have been entered so far. The TAB auto-completion is performed on the last parameter entered after validating any preceding parameters. If `min` and `max` are both 1 then it is not necessary to check `argc`.

If the acceptable values of a parameter are stored in a `NameMap<T>` or `CNameMap<T>`, those maps implement a `Validate()` function that will perform the validation needed for the `validate` function covering both the true



and false cases of complete. Those maps also implement a `FindUniquePrefix()` function that may be used to validate preceding parameters for commands that take multiple parameters.

The `config_validate()` function for the *config* command in `main/ovms_config.cpp` is an example implementation of use case 1 for a command taking three parameters with TAB auto-completion on the first two:

```
int config_validate(OvmsWriter* writer, OvmsCommand* cmd, int argc, const char*  
↳const* argv, bool complete)  
{  
    if (!MyConfig.ismounted())  
        return -1;  
    // argv[0] is the <param>  
    if (argc == 1)  
        return MyConfig.m_map.Validate(writer, argc, argv[0], complete);  
    // argv[1] is the <instance>  
    if (argc == 2)  
    {  
        OvmsConfigParam* const* p = MyConfig.m_map.FindUniquePrefix(argv[0]);  
        if (!p) // <param> was not valid, so can't check <instance>  
            return -1;  
        return (*p)->m_map.Validate(writer, argc, argv[1], complete);  
    }  
    // argv[2] is the value, which we can't validate  
    return -1;  
}
```

The *location* command in `components/ovms_location/src/ovms_location.cpp` is an example of use case 2 as it includes an intermediate parameter and also utilizes the `$L` form of the usage string:

```
OVMS# location action enter ?  
Usage: location action enter <location> acc <profile>  
Usage: location action enter <location> homelink 1|2|3  
Usage: location action enter <location> notify <text>
```

The following excerpt shows the implementation of the `location_validate()` function and a subset of the `RegisterCommand()` calls to build the command subtree. This example shows how simple the validation code can be – sometimes just one line to call `Validate()`. In this case the code does need to check `argc` because the function is shared by multiple subcommand objects taking 1 or 2 parameters.

```
int location_validate(OvmsWriter* writer, OvmsCommand* cmd, int argc, const char*  
↳const* argv, bool complete)  
{  
    if (argc == 1)  
        return MyLocations.m_locations.Validate(writer, argc, argv[0], complete);  
    return -1;  
}  
  
OvmsCommand* cmd_location = MyCommandApp.RegisterCommand("location", "LOCATION_  
↳framework");  
OvmsCommand* cmd_action = cmd_location->RegisterCommand("action", "Set an action for_  
↳a location");  
OvmsCommand* cmd_enter = cmd_action->RegisterCommand("enter", "Set an action upon_  
↳entering a location", NULL, "<location> $L", 1, 1, true, location_validate);  
OvmsCommand* enter_homelink = cmd_enter->RegisterCommand("homelink", "Transmit_  
↳Homelink signal");  
enter_homelink->RegisterCommand("1", "Homelink 1 signal", location_homelink, "", 0, 0,   
↳true);  
enter_homelink->RegisterCommand("2", "Homelink 2 signal", location_homelink, "", 0, 0,   
↳true);
```

(continues on next page)

(continued from previous page)

```
enter_homelink->RegisterCommand("3","Homelink 3 signal",location_homelink,"", 0, 0,␣  
↪true);  
cmd_enter->RegisterCommand("acc","ACC profile",location_acc,"<profile>", 1, 1,␣  
↪true);  
cmd_enter->RegisterCommand("notify","Text notification",location_notify,"<text>", 1,  
↪ INT_MAX, true);
```

---

CAN Bus Data Logging

---

OVMS can be used as CAN bus datalogging tool.

## 29.1 Physical Connections

OVMS hardware V3 supports up to three CAN bus connections. The connections to OVMS are as follows:

DB9-F	Signal
-----	-----
2	CAN1-L
7	CAN1-H
4	CAN2-L
5	CAN2-H
6	CAN3-L
8	CAN3-H

Note: the board schematics refer to CAN0,1,2. These correspond to CAN1,2,3 here in the documentation and source code. The first is handled by ESP32 i/o lines directly, and CAN2/3 are handled by MCP2515 ICs via SPI from the ESP.

CAN1 is the fastest bus, use this one if possible. The CAN logging tool is able to log all buses at the same time, to the same file or stream.

Vehicle CAN bus(s) are usually accesable via the vehicle's OBD2 port. Most modern cars have multiple CAN busses. The OBD2 'standard' CAN will be available on OBD2 **pin 6: CAN-H** and **pin 14: CAN-L**. However, modern vehicles (especially EV's) often have other CAN buses available on non-standard OBD2 pins.

A voltmeter (ideally oscilloscope) can be used to determine which OBD2 pins contain CAN data:

- Can high pins should normally be between 2.5 and 3.5 volts (to ground) - maybe 2.7 to 3.3 volts if there is traffic.
- Can low pins should normally be between 1.5 and 2.5 volts (to ground) - maybe 1.7 to 2.3 volts if there is traffic.

Pre-fabricated OBD2 > DB9-F for several specific vehicles can be purchased via OpenVehicles.

## 29.2 Enable OVMS CAN bus

Once physical connections has been made and OVMS is up and running connect to OVMS shell via web browser / SSH or serial.

If a specific vehicle module is loaded the CAN bus will already be enabled in OVMS. To check which CAN buses are enabled use:

```
OVMS# can list
```

If no vehicle module is selected the CAN bus must be started e.g

```
OVMS# can can1 start listen 500000
```

This will enable CAN1 in listen mode (read only) at 500k baud, active can be used instead of listen to enable read-write mode. To stop a CAN1 bus:

```
OVMS# can can1 stop
```

OVMS supports the following CAN bauds rates: 100000, 125000, 250000, 500000, 1000000.

## 29.3 Logging to SD card

It is possible to view CAN data directly in OVMS monitor shell, however since modern cars have very busy CAN buses there is often too much data which swamps the monitor or exceeds the logging queue, resulting in dropped messages. Logging to SD card is the better option.

If using a good quality SD card with a current OVMS V3 module (i.e. PCB revision 3.2 / 2019.05.23 or later), increase the SD card speed for best performance with:

```
config set sdcard maxfreq.khz 20000
```

Start logging all CAN messages using CRTD log file format with:

```
ovms# can log start vfs crtd /sd/can.crtcd
```

or log specific CAN packets by applying a filter e.g 0x55b the Nissan LEAF SoC CAN message

```
ovms# can log start vfs crtd /sd/can.crtcd 55b
```

Other CAN log file formats are supported e.g crtd, gvret-a, gvret-b, lawricel, pcap, raw.

Check CAN logging status with:

```
ovms# can log status
```

To Stop CAN logging:

```
ovms# can log stop
```

**Note: the can logging must be stopped before the file can be viewed**

To View the CAN log:

```
ovms# vfs head /sd/can.crtcd
```

tail and cat commands can also be used. However, be careful the log file can quickly become very large, cat may overwhelm the shell.

The log file can also be viewed in a browser with `http://<ovms-ipaddress>/sd/can.crtcd`

The logfiles can then be imported into a tool like SavvyCan for analysis.

## 29.4 Network Streaming

CAN data can be streamed directly to SavvyCan (or other compatible application) using the OVMS tcpserver CAN logging feature over a local network. Start tcpserver CAN logging with:

```
OVMS# can log start tcpserver discard gvret-b :23
```

This will start a tcpserver on port 23 (as required by SavvyCan) using the GVRET format supported by SavvyCAN.

Once OVMS CAN logging tcpserver is running open up SavvyCan and select:

```
Connection > Add New Device Connection > Network Connection
```

then enter the OVMS WiFi local network IP address (no port number required). CAN packets should now appear streaming into SavvyCan.

*Note: CAN tcpserver network streaming is a beta feture currently in edge firmware and may be buggy*

## 29.5 Optimizing the Performance

On `can log stop`, the system will output some statistics. Check especially the dropped frame count. Frame drops can occur because the system was busy with other tasks like handling network traffic. There are two options to optimize this:

- a) Reduce background activities, i.e. stop all services not needed for the logging. If possible, do the logging without an active vehicle module (e.g. set the “empty” vehicle via `vehicle module NONE`).
- b) Raise the log queue size. The default queue size has a capacity of 100 frames. To e.g. allow 200 frames, do:  
`config set can log.queuesize 200`.



---

## CRTD CAN Log Format

---

### 30.1 Introduction

The CRTD CAN Log format is a textual log file format designed to store information on CAN bus frames and events.

Files, or network data streams, in CRTD format contain only textual data in UTF-8, with each record being on an individual line terminated by a single linefeed (ascii 10) character.

Each line is made up from the following fields separated by single spaces (ASCII 32):

- Timestamp: Julian timestamp, seconds and milliseconds/microseconds, separated by a decimal point
- Record Type: Mnemonic to denote the record type
- Record Data: The remaining data is dependant on the record type

Here are some examples:

```
1542473901.020305 1R11 ...  
1542473901.020305 2T11 ...  
1542473901.020305 R11 ...  
1542473901.021 1R11 ...
```

### 30.2 Timestamps

In general, timestamps should be in UTC. A comment may optionally be placed at the start of the file to describe the timezone.

If the timestamp contains 3 digits after the decimal point, it should be consider millisecond precision, and handled appropriately. Similarly, if the timestamp contains 6 digits after the decimal point, it should be considered microsecond precision.

## 30.3 CAN Bus Designation

Record types may optionally be prefixed by the CAN bus number (1 ... n). If no bus number is provided, #1 should be assumed.

## 30.4 Record Types

### 30.4.1 Comment Record

Comment records start with the letter 'C', and the record type has two characters following to denote the sub-type. The rest of the record is to be considered a comment.

The following comment record types are defined:

- CXX: General textual comment
- CER: An indication of a (usually recoverable) error
- CST: Periodical statistics
- CEV: An indication of an event

Here are some examples:

```
169.971289 CXX Info Type:crted; Path:'/sd/can3.crted'; Filter:3:0-ffffffff;
↪Vehicle:TSHK;
19292.299819 CEV vehicle.alert this is a textual vehicle alert
198923.283738 CST intr=0 rxpkt=0 txpkt=0 errflags=0 rxerr=0 txerr=0 rxovr=0 txovr=0
↪txdelay=0 wdgreset=0
2783.384726 CER intr=0 rxpkt=0 txpkt=0 errflags=0 rxerr=0 txerr=0 rxovr=0 txovr=0
↪txdelay=0 wdgreset=0
```

### 30.4.2 Received Frame Record

Received frame records describe a frame received from the CAN bus, and start with the letter 'R'. Two types are defined:

- R11: A standard 11bit ID CAN frame
- R29: An extended 29bit ID CAN frame

The record type is followed by the frame ID (in hexadecimal), and then up to 8 bytes of CAN frame data.

Here are some examples:

```
1542473901.020305 1R11 213 00 00 00 00 c0 01 00 00
1542473901.020970 2R11 318 92 0b 13 10 11 3a 00 00
1542473901.021259 2R11 308 00 ff f6 a6 06 03 80 00
1542473901.021560 2R11 408 00
1542473901.030341 1R11 358 18 08 20 00 00 00 00 20
1542473901.034872 2R11 418 80
1542473901.035514 1R11 408 10
1542473901.036694 3R11 41C 10
1542473901.040289 R11 428 00 30
1542473901.042516 2R11 168 e0 7f 70 00 ff ff ff
1542473901.042809 2R11 27E c0 c0 c0 c0 00 00 00 00
1542473901.043073 1R11 248 29 29 0f bc 01 10 00
```



### 30.4.3 Transmitted Frame Record

Transmitted frame records describe a frame transmitted onto the CAN bus, and start with the letter 'T'. Two types are defined:

- T11: A standard 11bit ID CAN frame
- T29: An extended 29bit ID CAN frame

The record type is followed by the frame ID (in hexadecimal), and then up to 8 bytes of CAN frame data.

Here are some examples:

```
1542473901.020305 1T11 213 00 00 00 00 c0 01 00 00
1542473901.020970 2T11 318 92 0b 13 10 11 3a 00 00
1542473901.021259 2T11 308 00 ff f6 a6 06 03 80 00
1542473901.021560 2T11 408 00
1542473901.030341 1T11 358 18 08 20 00 00 00 00 20
1542473901.034872 2T11 418 80
1542473901.035514 1T11 408 10
1542473901.036694 3T11 41C 10
1542473901.040289 T11 428 00 30
1542473901.042516 2T11 168 e0 7f 70 00 ff ff ff
1542473901.042809 2T11 27E c0 c0 c0 c0 00 00 00 00
1542473901.043073 1T11 248 29 29 0f bc 01 10 00
```

## 30.5 Conclusions

Being a textual format, CRTD files are designed to be human readable and manipulated/analysed with standard text processing tools. They are not at all sophisticated, or compact.



### 31.1 TL;DR: Examples

The following examples include their documentation in the HTML page and source. Read the source and install them as plugins (see below) to see how they work.

#### 31.1.1 Metric Displays

OVMS V3 is based on metrics. Metrics can be single numerical or textual values or complex values like sets and arrays. The web framework keeps all metrics in a global object, which can be read simply by e.g. `metrics["v.b.soc"]`.

Metrics updates (as well as other updates) are sent to all DOM elements having the `receiver` class. To hook into these updates, simply add an event listener for `msg:metrics`.

Listening to the event is not necessary though if all you need is some metrics display. This is covered by the `metric` widget class family as shown here.

## Single Values & Charts

### Basic usage

All elements of class `metric` in a `receiver` are checked for the `data-metric` attribute. If no specific metric class is given, the metric value is simply set as the element text: `hologram` is your current network provider.

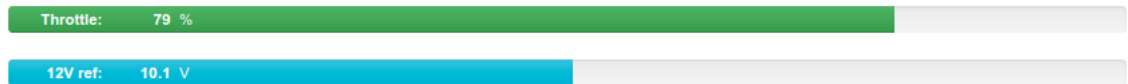
### Text & Number

`number` & `text` displays get the metric value set in their child of class `value`. They may additionally have labels and units. `data-prec` can be used on `number` to set the precision. They have fixed min widths and float by default, so you can simply put multiple displays into the same container:

Throttle: 79 %      10.1 V<sub>ref</sub>      Network: hologram

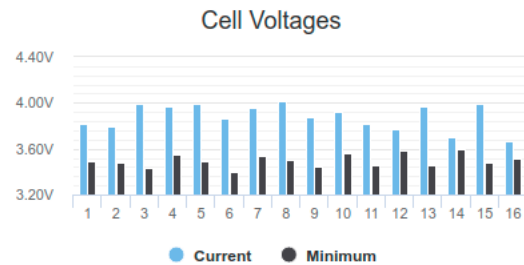
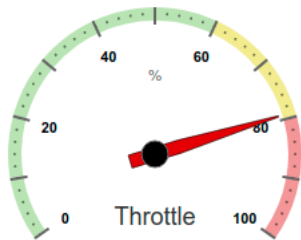
### Progress Bar

Bootstrap `progress` bars can be used as lightweight graphical indicators. Labels and units are available as well. Again, all you need is a bit of markup:



### Gauges & Charts

The OVMS web framework has builtin support for the highly versatile **Highcharts** library with loads of chart types and options. `chart` metric examples:



The following example covers...

- Text (String) displays
- Number displays
- Progress bars (horizontal light weight bar charts)
- Gauges
- Charts

Gauges & charts use the HighCharts library, which is included in the web server. The other widgets are simple standard Bootstrap widgets extended by an automatic metrics value update mechanism.

Highcharts is a highly versatile charting system. For inspiration, have a look at:

- <https://www.highcharts.com/demo>
- <https://www.highcharts.com/docs>

We're using `styled mode` so some options don't apply, but everything can be styled by standard CSS.

Install the example as a web page plugin:

`metrics.htm` (hint: right click, save as)

```

1 <!--
2   Test/Development/Documentation page; install as plugin to test
3 -->
4
5 <div class="panel panel-primary">
6   <div class="panel-heading">Metrics Displays Test/Demo</div>
7   <div class="panel-body">
8
9     <p>OVMS V3 is based on metrics. Metrics can be single numerical or textual values,
10    ↪ or complex values
11    ↪ like sets and arrays. The web framework keeps all metrics in a global object,
12    ↪ which can be read
13    ↪ simply by e.g. <code>metrics["v.b.soc"]</code>.</p>
14
15    <p>Metrics updates (as well as other updates) are sent to all DOM elements having
16    ↪ the
17    ↪ <code>receiver</code> class. To hook into these updates, simply add an event
18    ↪ listener for
19    ↪ <code>msg:metrics</code>. Listening to the event is not necessary if all you
20    ↪ need is some metrics
21    ↪ display. This is covered by the <code>metric</code> class family as shown here.
22    ↪ </p>
23
24    <p>
25    ↪ <button type="button" class="btn btn-default action-gendata">Generate random
26    ↪ data</button>
27    ↪ <button type="button" class="btn btn-default action-showsrc">Show page source</
28    ↪ button>
29    ↪ </p>
30
31    <hr/>
32
33    <div class="receiver">
34
35      <h4>Basic usage</h4>
36
37      <p>All elements of class <code>metric</code> in a <code>receiver</code> are
38      ↪ checked for the
39      ↪ <code>data-metric</code> attribute. If no specific metric class is given, the
40      ↪ metric value
41      ↪ is simply set as the element text: <span class="metric" data-metric="m.net.
42      ↪ provider">?</span>
43      ↪ is your current network provider.</p>
44
45      <h4>Text & Number</h4>
46
47      <p><code>number</code> & <code>text</code> displays get the metric value
48      ↪ set in their child of
49      ↪ class <code>value</code>. They may additionally have labels and units. <code>
50      ↪ data-prec</code> can
51      ↪ be used on <code>number</code> to set the precision, <code>data-scale</code>
52      ↪ to scale the raw
53      ↪ values by a factor. They have fixed min widths and float by default, so you
54      ↪ can simply put
55      ↪ multiple displays into the same container:</p>
56
57      <div class="clearfix">

```

(continues on next page)

(continued from previous page)

```

43     <div class="metric number" data-metric="v.e.throttle" data-prec="0">
44         <span class="label">Throttle:</span>
45         <span class="value">?</span>
46         <span class="unit">%</span>
47     </div>
48     <div class="metric number" data-metric="v.b.12v.voltage.ref" data-prec="1">
49         <span class="value">?</span>
50         <span class="unit">V<sub>ref</sub></span>
51     </div>
52     <div class="metric text" data-metric="m.net.provider">
53         <span class="label">Network:</span>
54         <span class="value">?</span>
55     </div>
56 </div>
57
58 <h4>Progress Bar</h4>
59
60 <p>Bootstrap <code>progress</code> bars can be used as lightweight graphical_
↪ indicators.
61     Labels and units are available, also <code>data-prec</code> and <code>data-
↪ scale</code>.
62     Again, all you need is a bit of markup:</p>
63
64     <div class="clearfix">
65         <div class="metric progress" data-metric="v.e.throttle" data-prec="0">
66             <div class="progress-bar progress-bar-success value-low text-left" role=
↪ "progressbar"
67                 aria-valuenow="0" aria-valuemin="0" aria-valuemax="100" style="width:0%">
68                 <div>
69                     <span class="label">Throttle:</span>
70                     <span class="value">?</span>
71                     <span class="unit">%</span>
72                 </div>
73             </div>
74         </div>
75         <div class="metric progress" data-metric="v.b.12v.voltage.ref" data-prec="1">
76             <div class="progress-bar progress-bar-info value-low text-left" role=
↪ "progressbar"
77                 aria-valuenow="0" aria-valuemin="5" aria-valuemax="15" style="width:0%">
78                 <div>
79                     <span class="label">12V ref:</span>
80                     <span class="value">?</span>
81                     <span class="unit">V</span>
82                 </div>
83             </div>
84         </div>
85     </div>
86
87 <h4>Gauges & Charts</h4>
88
89 <p>The OVMS web framework has builtin support for the highly versatile <b>
↪ Highcharts library</b>
90     with loads of chart types and options. <code>chart</code> metric examples:</p>
91
92     <div class="row">
93         <div class="col-sm-6">
94             <div class="metric chart" data-metric="v.e.throttle" style="height:220px">

```

(continues on next page)

(continued from previous page)

```

95         <div class="chart-box gaugechart" id="throttle-gauge"/>
96     </div>
97 </div>
98 <div class="col-sm-6">
99     <div class="metric chart" data-metric="v.b.c.voltage,v.b.c.voltage.min"
100     ↪style="height:220px">
101         <div class="chart-box barchart" id="cell-voltages"/>
102     </div>
103 </div>
104 </div>
105 <p><button type="button" class="btn btn-default action-gendata">Generate random
106 ↪data</button></p>
107 <p>For charts, a little bit of scripting is necessary.
108     The scripts for these charts contain the chart configuration, part of which
109 ↪is the update
110     function you need to define. The update function translates metrics data into
111 ↪chart data.
112     This is trivial for single values like the throttle, the cell voltage chart
113 ↪is an example
114     on basic array processing.</p>
115 <p>Also, while charts <em>can</em> be defined with few options, you'll <em>love
116 ↪</em> to explore
117     all the features and fine tuning options provided by Highcharts. For
118 ↪inspiration,
119     have a look at the <a target="_blank" href="https://www.highcharts.com/demo">
120 ↪Highcharts demos</a>
121     and the <a target="_blank" href="https://www.highcharts.com/docs/">Highcharts
122 ↪documentation</a>.
123     We're using <a target="_blank" href="https://www.highcharts.com/docs/chart-
124 ↪design-and-style/style-by-css">
125     styled mode</a>, so some options don't apply, but everything can be styled by
126 ↪standard CSS.</p>
127 </div>
128 </div>
129 </div>
130 <script>
131 (function() {
132     /* Get page source before chart rendering: */
133     var pagesrc = $('#main').html();
134     /* Init throttle gauge: */
135     $("#throttle-gauge").chart({
136         chart: {
137             type: 'gauge',
138             spacing: [0, 0, 0, 0],
139             margin: [0, 0, 0, 0],
140             animation: { duration: 500, easing: 'easeOutExpo' },
141         },
142         title: { text: "Throttle", verticalAlign: "middle", y: 75 },
143         credits: { enabled: false },

```

(continues on next page)

(continued from previous page)

```

141     tooltip: { enabled: false },
142     plotOptions: {
143         gauge: { dataLabels: { enabled: false }, overshoot: 1 }
144     },
145     pane: [{
146         startAngle: -125, endAngle: 125, size: '100%', center: ['50%', '60%']
147     }],
148     yAxis: [{
149         title: { text: '%' },
150         className: 'throttle',
151         reversed: false,
152         min: 0, max: 100,
153         plotBands: [
154             { from: 0, to: 60, className: 'green-band' },
155             { from: 60, to: 80, className: 'yellow-band' },
156             { from: 80, to: 100, className: 'red-band' },
157         ],
158         minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
159         tickPixelInterval: 40, tickPosition: 'inside', tickLength: 13,
160         labels: { step: 2, distance: -28, x: 0, y: 5, zIndex: 2 },
161     }],
162     series: [{
163         name: 'Throttle', data: [0],
164         className: 'throttle',
165         animation: { duration: 0 },
166         pivot: { radius: '10' },
167         dial: { radius: '88%', topWidth: 1, baseLength: '20%', baseWidth: 10,
168         ↪ rearLength: '20%' },
169     }],
170     /* Update method: */
171     onUpdate: function(update) {
172         // Create gauge data set from metric:
173         var data = [ metrics["v.e.throttle"] ];
174         // Update chart:
175         this.series[0].setData(data);
176     },
177     /* Init cell voltages chart */
178     $("#cell-voltages").chart({
179         chart: {
180             type: 'column',
181             animation: { duration: 500, easing: 'easeOutExpo' },
182         },
183         title: { text: "Cell Voltages" },
184         credits: { enabled: false },
185         tooltip: {
186             enabled: true,
187             shared: true,
188             headerFormat: 'Cell #{point.key}<br/>',
189             pointFormat: '{series.name}: <b>{point.y}</b><br/>',
190             valueSuffix: " V"
191         },
192         legend: { enabled: true },
193         xAxis: {
194             categories: []
195         },
196     },

```

(continues on next page)



(continued from previous page)

```

197     yAxis: [{
198         title: { text: null },
199         labels: { format: "{value:.2f}V" },
200         tickAmount: 4, startOnTick: false, endOnTick: false,
201         floor: 3.3, ceiling: 4.2,
202         minorTickInterval: 'auto',
203     }],
204     series: [{
205         name: 'Current', data: [],
206         className: 'cell-voltage',
207         animation: { duration: 0 },
208     }, {
209         name: 'Minimum', data: [],
210         className: 'cell-voltage-min',
211         animation: { duration: 0 },
212     }],
213     /* Update method: */
214     onUpdate: function(update) {
215         // Note: the 'update' parameter contains the actual update set.
216         // You can use this to reduce chart updates to the actual changes.
217         // For this demo, we just use the global metrics object:
218         var
219             m_vlt = metrics["v.b.c.voltage"] || [],
220             m_min = metrics["v.b.c.voltage.min"] || [];
221         // Create categories (cell numbers) & rounded values:
222         var cat = [], val0 = [], val1 = [];
223         for (var i = 0; i < m_vlt.length; i++) {
224             cat.push(i+1);
225             val0.push(Number((m_vlt[i]||0).toFixed(3)));
226             val1.push(Number((m_min[i]||0).toFixed(3)));
227         }
228         // Update chart:
229         this.xAxis[0].setCategories(cat);
230         this.series[0].setData(val0);
231         this.series[1].setData(val1);
232     },
233 });
234
235 /* Test metrics generator: */
236 $('action-gendata').on('click', function() {
237     var td = {};
238     td["m.net.provider"] = ["hologram", "Vodafone", "Telekom"][Math.floor(Math.
239 ↪random()*3)];
240     td["v.e.throttle"] = Math.random() * 100;
241     td["v.b.12v.voltage.ref"] = 10 + Math.random() * 4;
242     var m_vlt = [], m_min = [];
243     for (var i = 1; i <= 16; i++) {
244         m_vlt.push(3.6 + Math.random() * 0.5);
245         m_min.push(3.4 + Math.random() * 0.2);
246     }
247     td["v.b.c.voltage"] = m_vlt;
248     td["v.b.c.voltage.min"] = m_min;
249     $('receiver').trigger('msg:metrics', $.extend(metrics, td));
250 });
251
252 /* Display page source: */
253 $('action-showsrc').on('click', function() {

```

(continues on next page)

(continued from previous page)









```

253     $('<div/>').dialog({
254         title: 'Source Code',
255         body: '<pre style="font-size:85%; height:calc(100vh - 230px);">'
256             + encode_html(pagesrc) + '</pre>',
257         size: 'lg',
258     });
259 });
260
261 }) ();
262 </script>

```

## Vector Tables

Show 10 entries

#	▲	Voltage	◆	Minimum	◆	Maximum	◆	Max.Dev.	◆	Alert	◆
1		3.82		3.49		3.81		-0.017			
2		3.80		3.48		3.87		-0.054			
3		3.99		3.43		3.85		0.140			
4		3.97		3.55		3.97		-0.033			
5		3.99		3.50		3.97		-0.137			
6		3.87		3.40		3.97		-0.114			
7		3.96		3.54		3.93		-0.098			
8		4.02		3.50		3.87		-0.192			
9		3.88		3.44		3.95		0.163			
10		3.92		3.56		3.85		0.178			

Previous 1 2 Next

Some metrics, for example the battery cell voltages or the TPMS tyre health data, may contain vectors of arbitrary size. Besides rendering into charts, these can also be displayed by their textual values in form of a table.

The following example shows a live view of the battery cell voltages along with their recorded minimum, maximum, maximum deviation and current warning/alert state. Alert states 0-2 are translated into icons.

The metric table widget uses the DataTables library, which is included in the web server. The DataTables Javascript library offers a wide range of options to create tabular views into datasets.

Install the example as a web page plugin:

metrics-table.htm (hint: right click, save as)

```

1 <!--
2   Web UI page plugin: DataTables metrics widget demonstration
3 -->
4
5 <style>
6 td i {

```

(continues on next page)

(continued from previous page)

```

7   font-style: normal;
8   font-size: 140%;
9   line-height: 90%;
10  font-weight: bold;
11 }
12 td i.warning { color: orange; }
13 td i.danger { color: red; }
14 </style>
15
16
17 <div class="panel panel-primary panel-single receiver" id="my-receiver">
18   <div class="panel-heading">Metrics Table Widget Example</div>
19   <div class="panel-body">
20
21     <p>The following table shows a live view of the battery cell voltages along with
22     ↳ their recorded
23       minimum, maximum, maximum deviation and current warning/alert state.</p>
24     <p>Try resizing the window or using a mobile phone to see how the table adapts to
25     ↳ the screen
26       width. The table will also keep the selected sorting over data updates.</p>
27     <p>Hint: if you don't have live battery cell data, click the generator button to
28     ↳ create
29       some random values. The random data is only generated in your browser, not on
30     ↳ the module.</p>
31
32     <div class="metric table"
33       data-metric="v.b.c.voltage,v.b.c.voltage.min,v.b.c.voltage.max,v.b.c.voltage.
34     ↳ dev.max,v.b.c.voltage.alert">
35       <table class="table table-striped table-bordered table-hover" id="v-table" />
36     </div>
37
38     <p>See <a target="_blank" href="https://datatables.net/manual/">DataTables manual
39     ↳ </a> for all
40       options and API methods available.</p>
41
42   </div>
43   <div class="panel-footer">
44     <p><button type="button" class="btn btn-default action-gendata">Generate random
45     ↳ data</button></p>
46   </div>
47 </div>
48
49 <script>
50 (function() {
51
52   // Utilities:
53   var alertMap = {
54     0: '',
55     1: '<i class="warning"></i>',
56     2: '<i class="danger"></i>',
57   };
58
59   function fmtCode(value, map) {
60     return (map[value] !== undefined) ? map[value] : null;
61   }
62
63   function fmtNumber(value, prec) {

```

(continues on next page)

(continued from previous page)

```

57     return (value !== undefined) ? Number(value).toFixed(prec) : null;
58 }
59
60 // Init table:
61 $('#v-table').table({
62     responsive: true,
63     paging: true,
64     searching: false,
65     info: false,
66     autoWidth: false,
67     columns: [
68         { title: "#",          className: "dt-body-center", width: "6%",
↪ responsivePriority: 1 },
69         { title: "Voltage",    className: "dt-body-right",  width: "22%",
↪ responsivePriority: 3 },
70         { title: "Minimum",    className: "dt-body-right",  width: "22%",
↪ responsivePriority: 4 },
71         { title: "Maximum",    className: "dt-body-right",  width: "22%",
↪ responsivePriority: 5 },
72         { title: "Max.Dev.",    className: "dt-body-right",  width: "22%",
↪ responsivePriority: 2 },
73         { title: "Alert",      className: "dt-body-center", width: "6%",
↪ responsivePriority: 1 },
74     ],
75     rowId: 0,
76     onUpdate: function(update) {
77         // Get vector metrics to display:
78         var v = [
79             metrics["v.b.c.voltage"] || [],
80             metrics["v.b.c.voltage.min"] || [],
81             metrics["v.b.c.voltage.max"] || [],
82             metrics["v.b.c.voltage.dev.max"] || [],
83             metrics["v.b.c.voltage.alert"] || [],
84         ];
85         var lcnt = 0;
86         v.map(el => lcnt = Math.max(lcnt, el.length));
87         // Transpose vectors to columns:
88         var l, d = [];
89         for (l = 0; l < lcnt; l++) {
90             d.push([
91                 l+1,
92                 fmtNumber(v[0][l], 2),
93                 fmtNumber(v[1][l], 2),
94                 fmtNumber(v[2][l], 2),
95                 fmtNumber(v[3][l], 3),
96                 fmtCode(v[4][l], alertMap),
97             ]);
98         }
99         // Display new data:
100         this.clear().rows.add(d).draw();
101     },
102 });
103
104
105 // Test data generator:
106 $('.action-gendata').on('click', function() {
107     var td = {};

```

(continues on next page)

(continued from previous page)

```

108   var m_vlt = [], m_min = [], m_max = [], m_devmax = [], m_alert = [];
109   for (var i = 1; i <= 16; i++) {
110     m_vlt.push(3.6 + Math.random() * 0.5);
111     m_min.push(3.4 + Math.random() * 0.2);
112     m_max.push(3.8 + Math.random() * 0.2);
113     m_devmax.push(-0.2 + Math.random() * 0.4);
114     m_alert.push(Math.floor(Math.random() * 3));
115   }
116   td["v.b.c.voltage"] = m_vlt;
117   td["v.b.c.voltage.min"] = m_min;
118   td["v.b.c.voltage.max"] = m_max;
119   td["v.b.c.voltage.dev.max"] = m_devmax;
120   td["v.b.c.voltage.alert"] = m_alert;
121   $('.receiver').trigger('msg:metrics', $.extend(metrics, td));
122   });
123
124 }) ();
125 </script>

```

### 31.1.2 Command Buttons & Monitors

commands.htm (hint: right click, save as)

```

1  <!--
2   Test/Development/Documentation page; install as plugin to test
3  -->
4
5  <style>
6  h3 {
7    margin-top: 40px;
8    margin-bottom: 20px;
9  }
10 </style>
11
12 <div class="panel panel-primary">
13   <div class="panel-heading">Commands & Monitors</div>
14   <div class="panel-body">
15
16     <p>Command execution is only allowed for an authorized session. The command API will
17       output "Unauthorized" and a Login button as necessary. The login state is also
18       available in the global variable <code>loggedin</code>. To send the user to the
19       login page and return to the current page after login, call <code>login()</code>:
20     </p>
21
22     <p>
23       <button type="button" class="btn btn-default action-login" onclick="login()">Login
24     </button>
25       <button type="button" class="btn btn-default action-logout" onclick="logout()">
26     </button> Logout</p>
27
28     <script>
29       $('.action-login').prop('disabled', loggedin);
30       $('.action-logout').prop('disabled', !loggedin);
31     </script>

```

(continues on next page)

(continued from previous page)

```

30
31
32 <h3>Command Execution on Load</h3>
33
34 <p>To execute a command automatically on page load, simply add the <code>monitor</code> class
35 to an output element, set <code>data-updcmd</code> to the command to be executed,
36 and
37 <code>data-updcnt</code> to 1:</p>
38
39 <pre class="monitor" data-updcmd="boot status" data-updcnt="1">Fetching boot
40 status...</pre>
41
42 <p>Output elements typically are <code>samp</code> or <code>pre</code>, as commands
43 normally output formatted plain text, but any element can be used. <code>samp</code>
44 by default compresses white space and has no visible area, while <code>pre</code>
45 is visible and preserves all spacing.</p>
46
47 <p><code>updcnt</code> will count down to zero and stop, or run indefinitely if
48 started
49 below zero. The execution interval can be given in <code>data-updint</code> (in
50 seconds).
51 You can set the data attributes any time using jQuery, all monitors are checked
52 and
53 updated by the framework once per second.</p>
54
55 <h3>Command Execution on Events</h3>
56
57 <p>To automatically trigger a monitor update on OVMS events, additionally set the
58 <code>data-events</code> attribute to a regular expression matching the event(s)
59 of
60 interest. This example monitor lists the active network channels and automatically
61 updates on all server connection events:</p>
62
63 <p><pre class="monitor"
64 data-events="server.*(connect|open|close|stop)"
65 data-updcmd="network list"
66 data-updcnt="1"></pre></p>
67
68 <p>
69 Try:
70 <a class="btn btn-default" onclick="window.open('/', '_blank', 'width=400,
71 height=500')">Open new window</a>
72 <a class="btn btn-default" href="#" data-cmd="server v2 stop">Stop V2 server</a>
73 <a class="btn btn-default" href="#" data-cmd="server v2 start">Start V2 server</a>
74 <a class="btn btn-default" href="#" data-cmd="event raise server.fake.open">Send
75 fake event</a>
76 </p>
77
78 <h3>Command Buttons</h3>
79
80 <p>A bootstrap button has the base class <code>btn</code>. Add the command to
81 execute
82 as attribute <code>data-cmd</code> and optionally the output element as

```

(continues on next page)

(continued from previous page)

```

76     <code>data-target</code> and you're done.</p>
77
78     <div class="row">
79
80         <div class="col-sm-6">
81             <button type="button" class="btn btn-default" data-target="#out1" data-cmd=
↪ "wifi status">
82                 Wifi Status → <code>&lt;samp&gt;</code>
83             </button>
84             <samp id="out1" />
85         </div>
86
87         <div class="col-sm-6">
88             <button type="button" class="btn btn-default" data-target="#out2" data-cmd=
↪ "wifi status">
89                 Wifi Status → <code>&lt;pre&gt;</code>
90             </button>
91             <pre id="out2" />
92         </div>
93
94     </div>
95
96     <p>Add class <code>samp-inline</code> or wrap in a <code>ul.list-inline</code> to
↪ place
97     the output element on the same line with the button. Prefix the target with a "+"
↪ to
98     append to it:</p>
99
100    <p>
101        External 12V power
102        <button type="button" class="btn btn-info" data-target="+#out3" data-cmd="power
↪ ext12v on">on</button>
103        <button type="button" class="btn btn-info" data-target="+#out3" data-cmd="power
↪ ext12v off">off</button>
104        <samp class="samp-inline" id="out3" />
105    </p>
106
107
108    <h3>Combining Buttons & Monitors</h3>
109
110    <p>By combination of a button and a monitor, you can let the button start a repeated
111    execution of a command: set <code>data-watchcnt</code> on the button to the
↪ number of
112    repetitions (default 0) and <code>data-watchint</code> to the interval in seconds
113    (default 2).</p>
114
115    <p>
116        <button type="button" class="btn btn-default" data-target="#out4"
117            data-cmd="power simcom on" data-watchcnt="-1" data-watchint="3">
118            Power on modem & get status updates every 3 seconds
119        </button>
120        <pre class="monitor" id="out4" data-updcmd="simcom status" />
121    </p>
122
123    <p>Note: to stop a monitor, set <code>data-updcnt</code> to 0:
124    <button type="button" class="btn btn-default" onclick="$('#out4').data('updcnt',
↪ 0)">

```

(continues on next page)

(continued from previous page)

```

125     Stop updates
126     </button></p>
127
128
129     <h3>Execute Javascript</h3>
130
131     <p>To execute Javascript code directly (i.e. without calling <code>script eval</
132     <code>),
133     simply exchange the <code>data-cmd / data-updcmd</code> attribute by
134     <code>data-js / data-updjs</code>. Example:</p>
135
136     <p>
137         <button type="button" class="btn btn-default" data-target="#out5"
138             data-js="JSON.print(OvmsConfig.GetValues('vehicle'))">
139             Show vehicle config
140         </button>
141         <pre id="out5" />
142     </p>
143
144 </div>
145
146
147 <script>
148 (function() {
149     /* Show page source: */
150     var pagesrc = $('#main').html();
151     $('.panel-heading').prepend('<button type="button" class="btn btn-sm btn-info_
152     <action-showsrc"' +
153     ' style="float:right; position:relative; top:-5px;">Show page source</button>');
154     $('.action-showsrc').on('click', function() {
155         $('<div/>').dialog({
156             title: 'Source Code',
157             body: '<pre style="font-size:85%; height:calc(100vh - 230px);">'
158                 + encode_html(pagesrc) + '</pre>',
159             size: 'lg',
160         });
161     });
162 </script>

```

### 31.1.3 Notifications

notifications.htm (hint: right click, save as)

```

1 <!--
2     Test/Development/Documentation page
3     - enable web server file access
4     - upload to web file path, e.g. /sd/dev/notifications.htm
5     - open in framework by e.g. http://test1.local/#/dev/notifications.htm
6 -->
7
8 <div class="panel panel-primary">
9     <div class="panel-heading">Notification Test/Demo</div>
10    <div class="panel-body">

```

(continues on next page)



(continued from previous page)

```

11 <h4>Receiver</h4>
12 <!--
13     You can use data-subscriptions to preconfigure subscriptions.
14     Note: types "info", "error" and "alert" get sent to all receivers.
15 -->
16 <pre id="log" class="receiver" data-subscriptions="notify/stream/myapp/#">I'm
↪preconfigured to receive notify/stream/myapp/#</pre>
17 </div>
18 <div class="panel-footer">
19     <div>
20         <label for="topics">Test command:</label>
21         <input type="text" class="form-control font-monospace" id="cmd" value="notify
↪raise text stream myapp.input 'my first stream'">
22         <p>JSON example: <code class="autoselect">notify raise text stream myapp.input '
↪{"my": "JSON stream", "pi": 3.141, "fib": [1, 2, 3, 5, 8, 13]}'</code></p>
23         <button type="button" class="btn btn-default" id="action-exec">Execute</button>
24         <samp id="cmdres" />
25     </div>
26     <div>
27         <label for="topics">Topic subscription (separate topics by space):</label>
28         <input type="text" class="form-control font-monospace" id="topics" value=
↪"notify/stream/myapp/#">
29         <button type="button" class="btn btn-default" id="action-sub">Sub</button>
30         <button type="button" class="btn btn-default" id="action-unsub">Unsub</button>
31         <button type="button" class="btn btn-default" id="action-unsuball">Unsub all</
↪button>
32     </div>
33 </div>
34 </div>
35
36 <script>
37
38 // Receiver event handling:
39 $(' #log').on('msg:notify', function(ev, msg) {
40     // You normally should filter by type/subtype here, e.g.:
41     // if (subtype.startsWith('myapp')) { ... }
42     // msg has type, subtype and value
43     // Dump the msg into the receiver to show the structure:
44     $(this).text(JSON.stringify(msg, null, 2));
45
46     // A convenient way to transport complex data is JSON.
47     // To decode a JSON string, use JSON.parse().
48     // Note: JSON.parse() needs strict JSON syntax, i.e. quoted names.
49     var payload;
50     try {
51         payload = JSON.parse(msg.value);
52         if (payload) {
53             $(this).append("<div>Found JSON payload:</div>")
54                 .append($("<div />").text(JSON.stringify(payload, null, 2)).html());
55             console.log(payload);
56         }
57     } catch(e) {
58         // no JSON
59         console.log(e);
60     }
61 });
62

```

(continues on next page)

(continued from previous page)

```

63 // Topic subscription can be done on page load using the data-subscriptions attribute
64 // and/or on demand using the subscribe & unsubscribe calls:
65 $('#action-sub').on('click', function(ev) {
66     var topics = $('#topics').val();
67     $('#log').subscribe(topics).text("added " + topics);
68 });
69 // The receiver remembers the subscriptions and does an auto unsubscribe on unload.
70 // You can also unsubscribe topics dynamically. Your data producer can check for
71 // active subscriptions using the MyNotify.HasReader() method.
72 $('#action-unsub').on('click', function(ev) {
73     var topics = $('#topics').val();
74     $('#log').unsubscribe(topics).text("removed " + topics);
75 });
76 $('#action-unsuball').on('click', function(ev) {
77     $('#log').unsubscribe().text("removed all subscriptions");
78 });
79
80 // Command handler:
81 $('#action-exec').on('click', function(ev) {
82     var cmd = $('#cmd').val();
83     if (cmd) loadcmd(cmd, '#cmdres');
84 });
85 $('#cmd').on('keydown', function(ev) {
86     if (ev.which == 13) $('#action-exec').trigger('click');
87 });
88
89 </script>

```

### 31.1.4 Hook Plugins

hooks.htm (hint: right click, save as)

```

1 <!--
2   Test/Development/Documentation page
3 -->
4
5 <div class="panel panel-primary">
6   <div class="panel-heading">Page Hook Plugins</div>
7   <div class="panel-body">
8
9     <p>Plugins can provide pages on their own or extend existing pages. To extend an_
    ↳existing
10     page, the page needs to support this by offering hook points.</p>
11
12     <p>You are not limited to the hook points for page modifications. To insert your_
    ↳extensions
13     at arbitrary places:</p>
14     <ul>
15       <li>insert your extensions dynamically or with <code>display:none</code>,</li>
16       <li>register a onetime handler for the <code>load</code> event on <code>#main</
    ↳code>,</li>
17       <li>move your extensions into place and show them from the event handler.</li>
18     </ul>
19
20     <p>...and yes, you're also not limited to adding stuff.</p>

```

(continues on next page)

(continued from previous page)

```

21  <h3>Example</h3>
22
23  <p>The following hook plugin adds a custom menu and some color to <code>/home</code>
24  ↳: </p>
25
26  <pre id="plugindisplay" style="font-size:85%"></pre>
27
28  <h3>Available Hooks</h3>
29
30  <p>...as of January 2019. This is work in progress, if you miss a hook somewhere, ↳
31  ↳contact us.</p>
32
33  <table class="table table-condensed font-monospace">
34    <tr><th colspan="2">Framework</th></tr>
35    <tr><td>/</td><td>html.pre, head.post, body.post</td></tr>
36    ↳</td></tr>
37    <tr><td>/home</td><td>body.pre, body.post</td></tr>
38    ↳</td></tr>
39    <tr><td>/dashboard</td><td>body.pre</td></tr>
40    ↳</td></tr>
41    <tr><td>/status</td><td>body.pre, body.post</td></tr>
42    ↳</td></tr>
43    <tr><td>/shell</td><td>body.pre, body.post</td></tr>
44    ↳</td></tr>
45    <tr><th colspan="2">Renault Twizy</th></tr>
46    <tr><td>/xrt/drivemode</td><td>body.pre, body.post</td></tr>
47    ↳</td></tr>
48    <tr><td>/xrt/scmon</td><td>body.pre, body.post</td></tr>
49    ↳</td></tr>
50  </table>
51
52  </div>
53
54  <script>
55  (function() {
56    var pluginsrc = $('#pluginsrc').html();
57    $('#plugindisplay').html(encode_html(pluginsrc.substr(1)));
58  })();
59  </script>
60
61  <div id="pluginsrc" style="display:none">
62  <!--
63    Hook plugin for /home:body.pre
64    - custom coloring of menu titles
65    - add custom menu "Test / Demo" after "Main"
66  -->
67
68  <style>
69  .menu legend {
70    color: brown;
71  }
72  </style>
73
74  <fieldset class="menu" id="fieldset-menu-demo1" style="display:none">
75    <legend>Test / Demo</legend>

```

(continues on next page)

(continued from previous page)

```

69     <ul class="list-inline">
70         <li><a class="btn btn-default" href="/usr/demo/metrics" target="#main">Metrics </
→a></li>
71         <li><a class="btn btn-info" href="/logs/log" target="_blank">Open Log File </a></
→li>
72     </ul>
73 </fieldset>
74
75 <script>
76 $( '#main' ).one( 'load', function( ev ) {
77     $( '#fieldset-menu-demo1' ).insertAfter( '#fieldset-menu-main' ).show();
78 });
79 </script>
80 </div>

```

### 31.1.5 Solid Gauges

solidgauge.htm (hint: right click, save as)

```

1  <!--
2      Test/Development/Documentation page; install as plugin to test
3  -->
4
5  <style>
6      .solidgauge .highcharts-yaxis-grid .highcharts-grid-line,
7      .solidgauge .highcharts-yaxis-grid .highcharts-minor-grid-line {
8          display: none;
9      }
10     .solidgauge .highcharts-pane {
11         fill: #eee;
12         fill-opacity: 1;
13         stroke: #aaa;
14         stroke-width: 2px;
15     }
16     .night .solidgauge .highcharts-pane {
17         fill: #b1b1b1;
18         stroke: #b1b1b1;
19     }
20     .solidgauge .highcharts-point {
21         transition: fill 500ms, stroke 500ms; /* see stops styles note */
22     }
23     #throttle-gauge .highcharts-color-0 {
24         fill: #eee; /* initial colors = pane for smooth fade in */
25         stroke: #aaa;
26         fill-opacity: 0.8;
27         stroke-width: 3px;
28         stroke-opacity: 0.5;
29     }
30     #rpm-gauge .highcharts-color-0 {
31         fill: #55e;
32         stroke: none;
33     }
34 </style>
35
36 <div class="panel panel-primary">

```

(continues on next page)

(continued from previous page)

```

37 <div class="panel-heading">Solid Gauge</div>
38 <div class="panel-body">
39
40   <div class="receiver">
41
42     <div class="row">
43       <div class="col-sm-6">
44         <div class="metric chart" data-metric="v.e.throttle" style="height:220px">
45           <div class="chart-box solidgauge" id="throttle-gauge"/>
46         </div>
47         <p>
48           Note: stops don't work by default in styled mode. This example includes a
49           ↪ simple redraw
50           event callback that applies styles from the stops. To get a smooth color
51           ↪ transition,
52           the CSS transition time needs to be equal to the chart animation time.
53         </p>
54       </div>
55       <div class="col-sm-6">
56         <div class="metric chart" data-metric="v.m.rpm" style="height:220px">
57           <div class="chart-box solidgauge gaugechart" id="rpm-gauge"/>
58         </div>
59         <p>
60           Note: the <code>gaugechart</code> CSS class pulls in the default styling
61           ↪ for the bands,
62           labels and ticks here. You can also copy those CSS rules from <code>ovms.
63           ↪ css</code>
64           and do your own styling.
65         </p>
66       </div>
67     </div>
68
69     <div>
70       <p><button type="button" class="btn btn-default action-gendata">Generate random
71       ↪ data</button></p>
72     </div>
73
74   </div>
75 </div>
76
77 <script>
78 (function() {
79
80   /* Show page source: */
81   var pagesrc = $('#main').html();
82   $('#.panel-heading').prepend('<button type="button" class="btn btn-sm btn-info
83   ↪ action-showsrc"' +
84     ' style="float:right; position:relative; top:-5px;">Show page source</button>');
85   $('#.action-showsrc').on('click', function() {
86     $('#<div/>').dialog({
87       title: 'Source Code',
88       body: '<pre style="font-size:85%; height:calc(100vh - 230px);">'
89         + encode_html(pagesrc) + '</pre>',
90       size: 'lg',
91     });
92   });
93 });

```

(continues on next page)

(continued from previous page)

```

88  /* Init throttle gauge: */
89  $("#throttle-gauge").chart({
90    chart: {
91      type: 'solidgauge',
92      spacing: [0, 0, 0, 0],
93      margin: [0, 0, 0, 0],
94      animation: { duration: 500, easing: 'easeOutExpo' },
95      events: {
96        // Apply stops styles on redraw:
97        redraw: function(ev) {
98          var y = this.series[0].yData[0], $pt = $(this.renderTo).find('.highcharts-
↪point');
99          $.map(this.yAxis[0].stops, function(stop) { if (y >= stop[0]) $pt.
↪css(stop[1]); });
100        },
101      },
102    },
103    title: { text: "Throttle", verticalAlign: "middle", y: 75 },
104    credits: { enabled: false },
105    tooltip: { enabled: false },
106    plotOptions: {
107      solidgauge: { dataLabels: { enabled: false }, overshoot: 1 }
108    },
109    pane: [{
110      startAngle: -90, endAngle: 90, size: '140%', center: ['50%', '85%'],
111      background: { innerRadius: '60%', outerRadius: '100%', shape: 'arc' },
112    }],
113    yAxis: [{
114      title: { text: '%' },
115      className: 'throttle',
116      reversed: false,
117      min: 0, max: 100,
118      stops: [
119        [0, { fill: '#afa', stroke: '#4f4' }],
120        [50, { fill: '#ffa', stroke: '#ff4' }],
121        [80, { fill: '#faa', stroke: '#f44' }],
122      ],
123      tickPixelInterval: 40, tickPosition: 'inside', tickLength: 13,
124      labels: { step: 2, distance: 10, x: 0, y: 0, zIndex: 2 },
125    }],
126    series: [{
127      name: 'Throttle', data: [40],
128      className: 'throttle',
129      animation: { duration: 0 },
130    }],
131    /* Update method: */
132    onUpdate: function(update) {
133      // Create gauge data set from metric:
134      var data = [ metrics["v.e.throttle"] ];
135      // Update chart:
136      this.series[0].setData(data);
137    },
138  });
139
140  /* Init rpm gauge: */
141  $("#rpm-gauge").chart({
142    chart: {

```

(continues on next page)

(continued from previous page)

```

143     type: 'solidgauge',
144     spacing: [0, 0, 0, 0],
145     margin: [0, 0, 0, 0],
146     animation: { duration: 500, easing: 'easeOutExpo' },
147   },
148   title: { text: "RPM", verticalAlign: "middle", y: 75 },
149   credits: { enabled: false },
150   tooltip: { enabled: false },
151   plotOptions: {
152     solidgauge: { dataLabels: { enabled: false }, overshoot: 1 }
153   },
154   pane: [{
155     startAngle: -90, endAngle: 90, size: '140%', center: ['50%', '85%'],
156     background: { innerRadius: '70%', outerRadius: '100%', shape: 'arc' },
157   }],
158   yAxis: [{
159     title: { text: 'rpm' },
160     className: 'rpm',
161     reversed: false,
162     min: 0, max: 11000,
163     plotBands: [
164       { from: 0, to: 7000, className: 'green-band' },
165       { from: 7000, to: 9000, className: 'yellow-band' },
166       { from: 9000, to: 11000, className: 'red-band' },
167     ],
168     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
169     tickPixelInterval: 40, tickPosition: 'inside', tickLength: 10,
170     labels: { step: 2, distance: 10, x: 0, y: 0, zIndex: 2 },
171   }],
172   series: [{
173     name: 'RPM', data: [4500],
174     className: 'rpm',
175     animation: { duration: 0 },
176     innerRadius: '72%',
177     radius: '92%'
178   }],
179   /* Update method: */
180   onUpdate: function(update) {
181     // Create gauge data set from metric:
182     var data = [ metrics["v.m.rpm" ] ];
183     // Update chart:
184     this.series[0].setData(data);
185   },
186 });
187
188 /* Test metrics generator: */
189 $('action-gendata').on('click', function() {
190   var td = {};
191   td["v.e.throttle"] = Math.random() * 100;
192   td["v.m.rpm"] = Math.random() * 11000;
193   $('receiver').trigger('msg:metrics', $.extend(metrics, td));
194 });
195
196 }) ();
197 </script>

```

### 31.1.6 Command Streaming

loadcmd.htm (hint: right click, save as)

```

1 <!--
2   Test/Development/Documentation page; install as plugin to test
3 -->
4
5 <div class="panel panel-primary">
6   <div class="panel-heading">Asynchronous Command Streaming</div>
7   <div class="panel-body">
8
9     <h3>Synopsis</h3>
10
11     <p class="lead"><code>[jqxhr=] loadcmd(command [,target] [,filter] [,timeout])</code></p>
12
13     <p>The <code>loadcmd()</code> function executes a shell command or evaluates_
14     ↪ javascript
15       code with asynchronous streaming of the output into a target element or onto a_
16     ↪ function.</p>
17
18     <h3>Basic Usage</h3>
19
20     <p>As this is the underlying function for command buttons, basic usage
21       is very similar, but fully scriptable:</p>
22
23     <p>
24       ↪ <button type="button" class="btn btn-default" onclick="loadcmd('boot status', '
25         ↪ #out1')">
26         Show boot status
27       </button>
28     </p>
29     <pre id="out1" />
30
31     <p>As with command buttons, you can append the output by prefixing the target
32     ↪ selector with "+". You can omit the output by passing null as the target. If_
33     ↪ target
34       is a DOM element, loadcmd automatically sets the <code>loading</code> class on_
35     ↪ the
36       target while the command is processed.</p>
37
38     <p>The target element's min-height is automatically fixed to the current height
39       of the target before the output is set. This avoids page layout jumps when
40       reusing a target for commands.</p>
41
42     <p>If the target is scrollable and the content exceeds the visible area, the
43       element is automatically scrolled to show the new content, unless the user has
44       done a manual scrolling on the element.</p>
45
46     <p>If the command output stops for <code>timeout</code> seconds, the request is
47       aborted and a timeout error is shown. Default timeout is 20 seconds for standard
48       commands, 300 seconds for known long running commands.</p>
49
50     <h3>Evaluate Javascript Code</h3>

```

(continues on next page)



(continued from previous page)

```

49  <p>To execute Javascript, either pass an object instead of the <code>command</
↳code> string,
50      containing the command as property "command" and a second property "type" with
51      value "js", or simply use the wrapper call <code>loadjs()</code>. Example:</p>
52
53  <p>
54      <button type="button" class="btn btn-default"
55          onclick="loadjs('print(OvmsVehicle.Type())', '#out1js')">
56          Show vehicle type
57      </button>
58  </p>
59  <pre id="out1js" />
60
61  <p>Javascript evaluation is not limited to a single command or line. Hint: to
↳avoid
62      encoding complex JS code in the onclick attribute, store the code in some hidden
63      DOM element and read it via <code>$(...).text()</code>.</p>
64
65
66  <h3>jqXHR Object</h3>
67
68  <p><code>loadcmd()</code> returns the jqXHR (XMLHttpRequest) object in charge for
69  the asynchronous execution of the request. This can be used to track the results
70  of the execution, check for errors or to abort the command.</p>
71
72  <p>
73      <button type="button" class="btn btn-default" id="startcg">Start chargen</
↳button>
74      <button type="button" class="btn btn-default" id="abortcg" disabled>Abort
↳chargen</button>
75  </p>
76  <pre id="out2" style="max-height:200px; overflow:auto;" />
77
78  <p>The jQuery XHR object is also a "thenable", so actions to be performed after
79  the command execution can simply be chained to the object. Example:</p>
80
81  <p><button type="button" class="btn btn-default" id="showstat">Show stat</button>
↳</p>
82
83  <p>See <a target="_blank" href="http://api.jquery.com/jquery.ajax/#jqXHR">jQuery
84  documentation</a> for full details and options.</p>
85
86
87  <h3>Filter / Process Output</h3>
88
89  <p>Supply a filter function to hook into the asynchronous output stream. Use
↳filters
90      to filter (ahem...) / preprocess / reformat the command output, scan the stream
↳for some
91      info you'd like to know as soon as possible, or completely take over the output
92      processing.</p>
93
94  <p>The filter function is called when a new chunk of output has arrived or when
95  a stream error has occurred. The function gets a message object containing
96  the <code>request</code> object and either a <code>text</code> for normal
↳outputs
97      or an <code>error</code>, which is a preformatted error output you can use or

```

(continues on next page)

(continued from previous page)

```

98     ignore.</p>
99
100     <p>If the filter function returns a string, that will be added to the output_
    ↪target.
101     If it returns <code>null</code>, the target will remain untouched.</p>
102
103     <p>Hint: if you just want to scan the text for some info, you can pass on the_
    ↪message
104     after your scan to the default <code>standardTextFilter()</code>.</p>
105
106     <p>Example: let's reformat a <code>can status</code> dump into a nice Bootstrap_
    ↪table:</p>
107
108     <p><button type="button" class="btn btn-default" id="canstatus">CAN1 status</
    ↪button></p>
109
110     <table class="table table-condensed table-border table-striped table-hover" id=
    ↪"canout">
111         <thead>
112             <tr><th class="col-xs-6">Key</th><th class="col-xs-6">Value</th></tr>
113         </thead>
114         <tbody/>
115     </table>
116
117 </div>
118 </div>
119
120 <script>
121 (function() {
122
123     /* Show page source: */
124     var pagesrc = $('#main').html();
125     $('.panel-heading').prepend('<button type="button" class="btn btn-sm btn-info_
    ↪action-showsrc"' +
126     ' style="float:right; position:relative; top:-5px;">Show page source</button>');
127     $('.action-showsrc').on('click', function() {
128         $('<div/>').dialog({
129             title: 'Source Code',
130             body: '<pre style="font-size:85%; height:calc(100vh - 230px);">'
131                 + encode_html(pagesrc) + '</pre>',
132             size: 'lg',
133         });
134     });
135
136     /* Example: jqXHR.abort() */
137     var xhr;
138     $('#startcg').on('click', function() {
139         xhr = loadcmd('test chargen 20 600', '#out2');
140         $('#abortcg').prop('disabled', false);
141     });
142     $('#abortcg').on('click', function() {
143         xhr.abort();
144     });
145
146     /* Example: jqXHR.then() */
147     $('#showstat').on('click', function() {
148         loadcmd('stat').then(function(output) {

```

(continues on next page)

(continued from previous page)

```

149     confirmdialog('STAT result', '<samp>' + output + '</samp>', ['Close']);
150   });
151 });
152
153 /* Example: filter / output processor */
154 $('#canstatus').on('click', function() {
155   var $table = $('#canout > tbody');
156   var buf = '';
157   $table.empty();
158   loadcmd('can can1 status', function(msg) {
159     if (msg.error) {
160       // Render error into table row:
161       $('<tr class="danger"><td colspan="2">' + msg.error + '</td></tr>').appendTo(
162         ↪ $table);
163     }
164     else if (msg.text) {
165       // As this is a stream, the text chunks received need not be single or
166       ↪ complete lines.
167       // We're interested in lines here, so we buffer the chunks and split the
168       ↪ buffer at '\n':
169       buf += msg.text;
170       var lines = buf.split('\n');
171       if (lines.length > 1) {
172         buf = lines[lines.length-1];
173         for (var i = 0; i < lines.length-1; i++) {
174           // Skip empty lines:
175           if (lines[i] == "") continue;
176           // Split line into columns:
177           var col = lines[i].split(/: +/);
178           // Create table row & append to table, add some color on error counters:
179           if (col[0].match("[Ee]rr"))
180             ↪ $('<tr class="warning"><th>' + col[0] + '</th><td>' + col[1] + '</td></tr>').appendTo($table);
181           else
182             ↪ $('<tr><th>' + col[0] + '</th><td>' + col[1] + '</td></tr>').appendTo(
183             ↪ $table);
184         }
185       }
186       // Filter has handled everything:
187       return null;
188     });
189   });
190 }
191 </script>

```

### 31.1.7 File API

The file API enables web clients to read and write files at arbitrary VFS locations.

- API URL: /api/file
- Methods:
  - GET – read file

- POST – write file
- Parameters:
  - path – the full path to the file
  - content – the new file content on POST
- Output:
  - HTTP status: 200 (OK) or 400 (Error)
  - HTTP body: file content on GET or error message

On writing, missing directories along the path will be created automatically.

### Usage Example

The API can be accessed easily in the web frontend with the jQuery AJAX methods:

```
// Write object JSON encoded into file:
var json = JSON.stringify(object);
$.post("/api/file", { "path": "/sd/mystore/file1", "content": json })
  .done(function() {
    confirmdialog('Saved', '<p>File has been saved.</p>', ["OK"], 2);
  })
  .fail(function(jqXHR) {
    confirmdialog('Save failed', jqXHR.responseText, ["OK"]);
  });

// Read JSON encoded object from file:
$.get("/api/file", { "path": "/sd/mystore/file1" })
  .done(function(responseText) {
    var object = JSON.parse(responseText);
    // ... process object ...
  })
  .fail(function(jqXHR) {
    confirmdialog('Load failed', jqXHR.responseText, ["OK"]);
  });
```

### External Access

From external clients, the API can be used by either registering a session cookie or by supplying the `apikey` parameter (as explained in the general authorization overview).

**Read example:**

```
curl "http://192.168.4.1/api/file?apikey=password&path=/sd/mystore/file1"
```

**Write example:**

```
curl "http://192.168.4.1/api/file" -d 'apikey=password' \
-d 'path=/sd/mystore/file1' --data-urlencode 'content@localfile'
```

### 31.1.8 Longtouch Buttons

`btn-longtouch.htm` (hint: right click, save as)

```

1 <!--
2   Test/Development/Documentation page
3   - enable web server file access
4   - upload to web dir, e.g. scp testpage.htm test1.local:/sd/dev/
5   - open in framework by e.g. http://test1.local/#/dev/testpage.htm
6 -->
7
8 <div class="panel panel-primary">
9   <div class="panel-heading">Longtouch Buttons</div>
10  <div class="panel-body">
11
12    <p>Note: open this test page with a mobile / touch screen device or simulate
13    a mobile device using the web debugger of your browser.</p>
14
15    <p>The buttons below should work normally on a non-touch device.</p>
16
17    <p>On a touch screen device, touching the buttons starts a countdown
18    of 1.5 seconds, showing an overlay with a progress bar and giving vibration
19    signals (on devices supporting this). The button action is triggered only
20    after the countdown has finished while holding the touch.</p>
21
22    <p>This feature is meant to a) secure certain buttons against unintentional_
23    ↪activation
24    and to b) allow secure usage of these buttons while driving, i.e. when the mobile_
25    ↪phone /
26    tablet is mounted to a car holder.</p>
27
28    <p>Standard candidates for this are buttons triggering deep modifications, i.e.
29    changing the car tuning or drive mode.</p>
30
31    <p>To activate the feature on a button, simply add the class <code>btn-longtouch</
32    ↪code> to the button.
33    To activate it on a group of buttons, add the class to the container of the_
34    ↪buttons.</p>
35
36    <p>The progress overlay shows the button title if set (HTML supported), else the_
37    ↪button text.</p>
38
39    <form action="#">
40      <div class="btn-group btn-group-justified btn-longtouch">
41        <div class="btn-group btn-group-lg">
42          <button type="submit" name="load" value="0" id="prof-0" class="btn btn-
43          ↪default"><strong>STD</strong></button>
44        </div>
45        <div class="btn-group btn-group-lg">
46          <button type="submit" name="load" value="1" id="prof-1" class="btn btn-
47          ↪default" title="Power Mode"><strong>PWR</strong></button>
48        </div>
49        <div class="btn-group btn-group-lg">
50          <button type="submit" name="load" value="2" id="prof-2" class="btn btn-
51          ↪success"><strong>ECO</strong></button>
52        </div>
53        <div class="btn-group btn-group-lg">
54          <button type="submit" name="load" value="3" id="prof-3" class="btn btn-
55          ↪default" title="Winter Mode"><strong>ICE</strong></button>
56        </div>
57      </div>
58    </form>
59  </div>
60 </div>

```

(continues on next page)

(continued from previous page)

```

49     </form>
50
51     </div>
52 </div>

```

### 31.1.9 Basic Dialogs

dialogtest.htm (hint: right click, save as)

```

1  <!--
2    Test/Development/Documentation page
3    - enable web server file access
4    - upload to web dir, e.g. scp testpage.htm test1.local:/sd/dev/
5    - open in framework by e.g. http://test1.local/#/dev/testpage.htm
6  -->
7
8  <div class="panel panel-primary">
9    <div class="panel-heading">Dialog Test/Demo</div>
10   <div class="panel-body">
11     <h4>Core Widget</h4>
12     <button class="btn btn-default" id="action-load">Dialog 1</button>
13     <button class="btn btn-default" id="action-save">Dialog 2</button>
14     <button class="btn btn-default" id="action-both">Both together</button>
15     <button class="btn btn-default" id="action-dyn">Dynamic custom</button>
16     <hr />
17     <h4>Utility Wrappers</h4>
18     <button class="btn btn-default" id="action-info">Alert</button>
19     <button class="btn btn-default" id="action-confirm">Confirm</button>
20     <button class="btn btn-default" id="action-choice">Choice</button>
21     <button class="btn btn-default" id="action-prompt">Prompt</button>
22     <button class="btn btn-default" id="action-password">Password</button>
23     <hr />
24     <h4>Log</h4>
25     <pre id="log" />
26   </div>
27 </div>
28
29 <div id="dialog1" />
30 <div id="dialog2" />
31 <div id="dialog3" />
32
33 <script>
34
35 //
36 // =====
37 // Dialog core widget:
38 //   options: {
39 //     title: '',
40 //     body: '',
41 //     show: false,           / true = open directly on init
42 //     remote: false,
43 //     backdrop: true,       background overlay
44 //     keyboard: true,       ESC = close
45 //     transition: 'fade',   / ''
46 //     size: '',             / 'sm' / 'lg'

```

(continues on next page)

(continued from previous page)

```

46 //     contentClass: '',          added to .modal-content
47 //     onShown: null,            function(input)
48 //     onHidden: null,           function(input)
49 //     onShow: null,             function(input)
50 //     onHide: null,            function(input)
51 //     onUpdate: null,          function(input)
52 //     buttons: [{}],           see example
53 //     timeout: 0,              in seconds
54 //     input: null,             predefined/shared input object to use
55 //   },
56
57 // Note: use static dialogs (attached to the document) for speed and to store reusable
58 // input, e.g. for a promptdialog.
59
60 // The default dialog will contain just a "Close" button:
61 $('#dialog1').dialog({
62   title: 'Please note',
63   body: '<p>This is just a test.</p>'
64 });
65 $('#dialog2').dialog({
66   title: 'Alert',
67   body: '<p>Danger, Will Robinson, danger!</p>',
68   contentClass: 'alert-danger', // note: looks strange, just an example -- don't use
69 });
70
71 // You can simply open preconfigured dialogs like this:
72 $('#action-load').on('click', function() {
73   $('#dialog1').dialog('show');
74 });
75 $('#action-save').on('click', function() {
76   $('#dialog2').dialog('show');
77 });
78
79 // ...or you can reconfigure dialogs on the fly like this:
80 // Note: the new configuration is stored in the dialog.
81 $('#action-both').on('click', function() {
82   $('#dialog1').dialog('show', { body: '<p>This is now another test.</p>' });
83   // This example also opens a second dialog on top of the first:
84   $('#dialog2').dialog('show', { body: '<p>This is a second layer dialog.</p>' });
85 });
86
87 // Use dynamic dialogs for single shot purposes:
88 $('#action-dyn').on('click', function() {
89   $('#log').text("");
90   $('<div />').dialog({
91     title: 'Dynamic...',
92     body: '<p>...dialog with custom buttons.</p>',
93     buttons: [
94       // Buttons default to auto hiding the dialog:
95       { label: "Nope" },
96       // You can get the button clicked in the onHide callback, or you can
97       // add specific action callbacks to each button.
98       { label: "Maybe...", btnClass: "warning", autoHide: false, action: ↪
↪function(input) {
99         // input is the data container of the widget, with only predefined
100         // member being "button" = the button clicked (or null).
101         // You can use this container for custom extensions:

```

(continues on next page)

(continued from previous page)

```

102     input.clicked_maybe = true;
103     $(this).find(".modal-body").append("<p>You're now a maybe person.</p>");
104     } },
105     // autoHide callbacks are called on the hidden event, so a fade out
106     // is guaranteed to be finished in the callback.
107     { label: "Done", btnClass: "primary", action: function(input) {
108         $("#log").text(JSON.stringify(input, null, 2));
109     } },
110     ],
111     });
112 });
113
114
115 //
116 // =====
117 // Dialog utility wrappers:
118 //   confirmdialog(title, body, buttons, [callback,] timeout)
119 //   promptdialog(type, title, body, buttons, callback)
120 // Both can be used plugin style or standalone for dynamic dialogs.
121
122 // Dynamic alert() style dialog with 5 seconds timeout:
123 $('#action-info').on('click', function(){
124     confirmdialog("Sorry...", "I'm afraid I can't do that, Dave.", ["OK"], 5);
125     // Note: you can also add a callback here to know when the dialog is dismissed.
126 });
127
128 // Dynamic confirm() style dialog:
129 $('#action-confirm').on('click', function(){
130     $("#log").text("");
131     confirmdialog("Load file", "Discard unsaved changes?", ["No", "Yes"],
132     function(confirmed){
133         if (confirmed)
134             $("#log").text("Loading now...");
135         else
136             $("#log").text("Load aborted.");
137     });
138 });
139
140 // The callback argument is the index of the button clicked (or null), so
141 // simple choice dialogs can be done like this:
142 $('#action-choice').on('click', function(){
143     $("#log").text("");
144     confirmdialog("Select", "Please select the slot to use:", ["1", "2", "3", "4"],
145     function(button){
146         if (button != null)
147             $("#log").text("Using slot " + (button+1));
148         else
149             $("#log").text("Abort.");
150     });
151 });
152
153 // Static prompt() style text input dialog (with #dialog3 keeping the dialog):
154 $('#action-prompt').on('click', function(){
155     $("#log").text("");
156     $('#dialog3').promptdialog("text", "Save data", "Please enter file name:", ["Cancel",
157     "Save"], function(button, input){
158         if (button && input)

```

(continues on next page)



(continued from previous page)

```

155     $("#log").text("Saving to file: " + input);
156     else
157         $("#log").text("Save aborted.");
158     });
159 });
160
161 // All single field HTML5 input types can be used:
162 $('#action-password').on('click', function() {
163     $("#log").text("");
164     promptdialog("password", "Authentication", "Please enter PIN:", ["Cancel", "Continue
165     ↪"], function(button, input) {
166         if (button == input)
167             $("#log").text("PIN entered: " + input);
168         else
169             $("#log").text("Auth aborted.");
170     });
171 });
172 </script>

```

### 31.1.10 File Dialogs

filedialog.htm (hint: right click, save as)

```

1  <!--
2      Test/Development/Documentation page
3      - enable web server file access
4      - upload to web file path, e.g. /sd/dev/filedialog.htm
5      - open in framework by e.g. http://test1.local/#/dev/filedialog.htm
6  -->
7
8  <div class="filedialog" id="fileselect"/>
9
10 <div class="panel panel-primary">
11     <div class="panel-heading">FileDialog Test</div>
12     <div class="panel-body">
13
14         <h2>Dynamic API</h2>
15
16         <p>
17             <button class="btn btn-default" id="action-load">Load</button>
18             <button class="btn btn-default" id="action-save">Save</button>
19         </p>
20         <pre id="log"/>
21
22         <h2>Data API</h2>
23
24         <div class="input-group">
25             <input type="text" class="form-control font-monospace" placeholder="Enter file_
26             ↪name"
27                 name="filename" id="input-filename" value="" autocapitalize="none"
28             ↪autocorrect="off"
29                 autocomplete="section-demo" spellcheck="false">
30             <div class="input-group-btn">
31                 <!-- use data-toggle="filedialog", data-target & data-input on the button: -->

```

(continues on next page)

(continued from previous page)

```

30     <button type="button" class="btn btn-default" data-toggle="filedialog" data-
    ↪target="#fd-api" data-input="#input-filename">Select</button>
31     </div>
32 </div>
33
34     <!-- This is the file dialog triggered by the button: -->
35     <div class="filedialog" id="fd-api" data-options='{
36         "title": "This is a data API dialog",
37         "path": "/sd/foo/",
38         "quicknav": ["/sd/", "/sd/foo/", "/sd/bar"]
39     }' />
40     <!-- ...that's it, no JS code necessary.
41         Note: JSON syntax needs to be strict here, see JS console for errors.
42     -->
43
44 </div>
45 </div>
46
47 <script>
48
49 // ↪
    ↪=====
50 // FileDialog is a Dialog with an embedded FileBrowser.
51 //
52 // The input object combines the dialog button and the file browser data.
53 //
54 // Options:
55 //     title: 'Select file',           -- dialog title
56 //     submit: 'Select',               -- label of submit button
57 //     select: 'f',                   -- 'f' = only files selectable (default), 'd
    ↪' = only directories
58 //     onSubmit: null,                -- callback function(input)
59 //     onCancel: null,                -- callback function(input)
60 //
61 // ...inherited from FileBrowser:
62 //     path: '',
63 //     quicknav: ['/sd/', '/store/'],
64 //     filter: null,
65 //     sortBy: null,
66 //     sortDir: 1,
67 //
68 // ...inherited from Dialog:
69 //     backdrop: true,
70 //     keyboard: true,
71 //     transition: 'fade',
72 //     size: 'lg',
73 //     onUpdate: null,
74 //
75
76 // Init dialog:
77 $('#fileselect').filedialog({
78     path: '/store/scripts/',
79     quicknav: ['/store/scripts/', '/store/events/', '/store/obd2ecu/'],
80 });
81
82 $('#action-load').on('click', function() {
83     $('#log').empty();

```

(continues on next page)

(continued from previous page)

```

84  $("#fileselect").filedialog('show', {
85      title: "Load Script",
86      submit: "Load",
87      onSubmit: function(input) {
88          // This is called when the user doubleclicks an entry or presses Enter in a
89          ↪directory.
90          // Note: you need to check if input.file is valid (empty = directory selected).
91          if (input.file)
92              $('#log').text('Loading: "' + input.path + '"\n');
93          else
94              $('#log').text('Directory selection: "' + input.path + '"\n');
95      },
96      onCancel: function(input) {
97          // This is called when the user cancels or closes the dialog.
98          $('#log').text('Load cancelled\n');
99      },
100  });
101
102  $('#action-save').on('click', function() {
103      $('#log').empty();
104      $("#fileselect").filedialog('show', {
105          title: "Save Script",
106          submit: "Save",
107          onSubmit: function(input) {
108              if (input.file)
109                  $('#log').text('Saving: "' + input.path + '"\n');
110              else
111                  $('#log').text('Directory selection: "' + input.path + '"\n');
112          },
113          onCancel: function(input) {
114              $('#log').text('Save cancelled\n');
115          },
116      });
117  });
118
119  </script>

```

### 31.1.11 File Browser Widget

filebrowser.htm (hint: right click, save as)

```

1  <!--
2      Test/Development/Documentation page
3      - enable web server file access
4      - upload to web file path, e.g. /sd/dev/filebrowser.htm
5      - open in framework by e.g. http://test1.local/#/dev/filebrowser.htm
6  -->
7
8  <style>
9      /* Note: set height or min-height on the filebrowser tbody.
10         Default height by class "filebrowser" is 310px (each file row has a height of 31px
11         ↪with the default font size) */
12  #myfilebrowser tbody {
13      height: 35vh;

```

(continues on next page)

(continued from previous page)

```

13 }
14 </style>
15
16 <div class="panel panel-primary">
17   <div class="panel-heading">FileBrowser Test/Demo</div>
18   <div class="panel-body">
19
20     <div class="filebrowser" id="myfilebrowser" />
21
22     <hr/>
23     <pre id="log"/>
24     <button type="button" class="btn btn-default" id="action-setpath">Set path to /sd/
↪ logs/log</button>
25     <button type="button" class="btn btn-default" id="action-stopload">Stop loading_
↪ dir</button>
26     <button type="button" class="btn btn-default" id="action-filteron">Filter *.zip</
↪ button>
27     <button type="button" class="btn btn-default" id="action-filteroff">Filter off</
↪ button>
28     <button type="button" class="btn btn-default" id="action-sortoff">Sort off</
↪ button>
29     <p>Note: the test widget is configured to stop/inhibit loading on selection of a
↪ ".txt" file or a directory matching "DCIM".</p>
30   </div>
31 </div>
32
33 <script>
34
35 //_
↪ =====
36 // FileBrowser Widget:
37 //
38 // Test/demo of init with all options.
39 // Note: you can init a filebrowser without any options.
40 // Defaults are path & quicknav as shown below, no sorting, no callbacks.
41 //
42 // The input object passed to all callbacks contains these fields:
43 // - path          -- the full path (dir + "/" + file)
44 // - file          -- the file part of the currently selected path
45 // - dir           -- the directory part of the currently selected path
46 // - noload        -- set to true to inhibit directory loading (see onPathChange)
47 //
48 // Note: path, file & dir are unvalidated user input.
49 //
50 // Note on sorting: disabling initial sorting improves user interaction while loading_
↪ the
51 // directory, i.e. to select files/dirs while the loader is running. This may
52 // be an option especially for large directories. The user can still sort manually.
53 //
54 // Methods:
55 // - getInput()      -- retrieve input object
56 // - setPath(newpath, reload) -- add trailing slash to enter dir w/o file_
↪ selection
57 // - sortList(by, dir) -- sort file list
58 // - loadDir()        -- trigger directory reload
59 // - stopLoad()       -- abort directory loading
60 // - newDir()         -- open create directory sub dialog

```

(continues on next page)

(continued from previous page)

```

61 //
62
63 $('#myfilebrowser').filebrowser({
64   path: '/sd/',
65   quicknav: ['/sd/', '/store/'],
66   filter: null,           // see Filter examples below
67   sortBy: "name",        // ...or "size" or "date" or null (disable)
68   sortDir: 1,            // ...or -1 for reverse
69   onUpdate: function(input) {
70     // This is called after any widget configuration update, use for custom
    ↪ extensions.
71     $('#log').append('onUpdate: ' + JSON.stringify(input) + '\n');
72   },
73   onPathChange: function(input) {
74     // Called whenever the path changes.
75     $('#log').append('onPathChange: ' + JSON.stringify(input) + '\n');
76     // To inhibit loading the directory for the new path, set input.noload to true:
77     if (input.dir.indexOf("DCIM") >= 0 || input.file.indexOf('.txt') >= 0) {
78       $('#log').append('inhibiting directory loading\n');
79       input.noload = true;
80     }
81   },
82   onAction: function(input) {
83     // This is called when the user doubleclicks an entry or presses Enter in a
    ↪ directory:
84     $('#log').append('onAction: ' + JSON.stringify(input) + '\n');
85   },
86 });
87
88 $('#action-setpath').on('click', function(ev) {
89   $('#myfilebrowser').filebrowser('setPath', '/sd/logs/log', true);
90 });
91 $('#action-stopload').on('click', function(ev) {
92   $('#myfilebrowser').filebrowser('stopLoad');
93 });
94 $('#action-sortoff').on('click', function(ev) {
95   $('#myfilebrowser').filebrowser('sortList', '');
96 });
97
98 // Filter example:
99 $('#action-filteron').on('click', function(ev) {
100   $('#myfilebrowser').filebrowser({
101     // The list filter may be given as a string (regular expression applied to file
    ↪ name):
102     filter: "\\\\.zip$",
103     // A string filter always lists directories.
104
105     // For advanced usage, you may alternatively specify a filter callback like this:
106     // filter: function(f) { return f.isdir || f.name.match("\\\\.zip$"); }
107     // The function is called per list entry object with...
108     // - isdir: true = is sub directory
109     // - name: the name part (file or directory, dir with trailing '/')
110     // - path: the full path of this entry
111     // - size: formatted size ('6.8k')
112     // - date: formatted date ('23-Nov-2018 17:42')
113     // - bytes: size in bytes, -1 for directories
114     // - isodate: ISO style date 'YYYY-mm-dd HH:MM'

```

(continues on next page)

(continued from previous page)

```

115 // - class: '', can be used to add a custom row class
116 // Return true to allow the entry to be added to the list.
117
118 // Note: list filters do not restrict the allowed path input. To do so,
119 // register onPathChange or onAction and check the input in your callback.
120 // See FileDialog widget "select" option handling for an example.
121 });
122 });
123 $('#action-filteroff').on('click', function(ev) {
124     $('#myfilebrowser').filebrowser({
125         filter: null
126     });
127 });
128
129 </script>

```

### 31.1.12 Slider Widget

input-slider.htm (hint: right click, save as)

```

1 <style>
2 /* Align multiple slider inputs by suitably fixing their value width: */
3 .form-inline .form-control.slider-value {
4     width: 80px;
5 }
6 </style>
7
8 <div class="panel panel-primary">
9     <div class="panel-heading">Slider Widget</div>
10    <div class="panel-body">
11
12        <h3>Standard Numerical Inputs</h3>
13
14        <p>The number input is just the bootstrap default:</p>
15
16        <input class="form-control" type="number" id="input-num1" name="num1" value="80"
17        ↪min="0" max="100" step="1">
18        <br/>
19
20        <p>The range input has some styling optimization for touch devices:</p>
21
22        <input class="form-control" type="range" id="input-rng1" name="rng1" value="80"
23        ↪min="0" max="100" step="1">
24        <br/>
25
26        <h3>Slider Widget</h3>
27
28        <p>
29            In many cases you'd like to give especially the touchscreen user a combination
30            ↪of these input
31            elements, and as sliding may be too imprecise you also need large buttons for
32            ↪single step changes.
33            That's what the slider widget does. It also adds the option of a checkbox and a
34            ↪default value to
35            reflect if the user control shall be applied.

```

(continues on next page)

(continued from previous page)

```

31     </p>
32
33     <div class="row">
34         <div class="col-md-9">
35
36             <p>Sliders can easily be created from minimal markup using the <code>.slider()
↪ </code> plugin,
37             with configuration given by data attributes and/or dynamic options:</p>
38
39             <div class="form-control slider" id="sld1" data-min="-5" data-max="50" data-
↪ step="0.5"
40                 data-default="40" data-unit="%" data-disabled="false" data-checked="true" ↪
↪ data-value="10" />
41             <br/>
42
43             <div class="form-control slider" id="sld2" />
44             <br/>
45
46             <p>...and slider state, limits and value can be controlled using the same ↪
↪ method:<br/>
47             <button type="button" class="btn btn-default" onclick="$('#sld1').slider({ ↪
↪ value:42 })">
48                 Set sld1 value to 42
49             </button>
50             <button type="button" class="btn btn-default" onclick="$('#sld1').slider({ ↪
↪ checked:true })">
51                 Check sld1
52             </button>
53             <button type="button" class="btn btn-default" onclick="$('#sld1').slider({ ↪
↪ disabled:true })">
54                 Disable sld1
55             </button>
56             <button type="button" class="btn btn-default" onclick="$('#sld1').slider({ ↪
↪ disabled:false })">
57                 Enable sld1
58             </button>
59         </p>
60         <br/>
61
62         <p>If you need even more control, you can create the slider markup yourself ↪
↪ as well:</p>
63
64         <div class="form-control slider" id="sld3" data-default="50" data-reset="false
↪ "
65             data-value="80" data-min="-10" data-max="100" data-step="1">
66             <div class="slider-control form-inline">
67                 <input class="slider-enable" type="checkbox" checked>
68                 <input class="form-control slider-value" type="number" id="input-sld3" ↪
↪ name="sld3">
69                 <span class="slider-unit">%</span>
70                 <input class="btn btn-default slider-down" type="button" value=" ">
71                 <input class="btn btn-default slider-set" type="button" value="Lo" data-
↪ set="25">
72                 <input class="btn btn-default slider-set" type="button" value="Hi" data-
↪ set="75">
73                 <input class="btn btn-default slider-up" type="button" value=" ">
74             </div>

```

(continues on next page)

(continued from previous page)

```

75     <input class="slider-input" type="range">
76   </div>
77   <br/>
78
79   </div>
80   <div class="col-md-3">
81
82     <p><u>Events &amp; values</u>:</p>
83     <pre id="show-sldev"></pre>
84
85   </div>
86 </div>
87
88 <br/>
89
90 <p>
91   Checkbox, buttons and unit are optional. You can reduce the widget to just a
92   ↪number or
93   just a range input, the input then needs to have the <code>slider-input</code>
94   ↪class.
95   </p>
96
97   <p>
98     The checkbox element defines the default value to be set on unchecking in
99     <code>data-default</code>. By default, the checkbox will restore the previous
100    ↪user
101    value when re-checked, to disable this, set <code>data-reset</code> to "true".
102    To reset the value to the default from a script, call the <code>.slider()</code>
103    method with <code>value: null</code> (this resets both the actual and the stored
104    user value).
105    </p>
106
107    <p>
108      Values and checkbox status need to be consistent on init, or be set by your
109      ↪script.
110      To hook into value changes, attach event handlers to events <code>input</code>
111      and/or <code>change</code> as usual. Read the <code>checked</code> property to
112      ↪get the
113      checkbox state.
114      </p>
115
116   </div>
117 </div>
118
119 <script>
120 (function() {
121
122   /* Show page source: */
123   var pagesrc = $('#main').html();
124   $('.panel-heading').prepend('<button type="button" class="btn btn-sm btn-info
125   ↪action-showsrc"' +
126     ' style="float:right; position:relative; top:-5px;">Show page source</button>');
127   $('.action-showsrc').on('click', function() {
128     $('#<div/>').dialog({
129       title: 'Source Code',
130       body: '<pre style="font-size:85%; height:calc(100vh - 230px);">'
131         + encode_html(pagesrc) + '</pre>',

```

(continues on next page)



(continued from previous page)

```

126     size: 'lg',
127   });
128 });
129
130 /* Init sliders: */
131 $('slider').slider();
132 $('#sld2').slider({ min:-10, max:10, step:0.1, default:2.5, unit:'kW',
↪checked:false, value:-3.8 });
133
134 /* Show slider events & values: */
135 var sldev = {};
136 $('#input-sld1, #input-sld2, #input-sld3').on('input change', function(ev) {
137   sldev[this.name] = $.extend(sldev[this.name], { checked: this.checked });
138   // Note: this.value is unvalidated here for a direct entry, but the validation is
↪simple:
139   sldev[this.name][ev.type] = Math.max(this.min, Math.min(this.max, 1*this.value));
140   $('#show-sldev').text(JSON.stringify(sldev, null, 2));
141 });
142
143 }) ();
144 </script>

```

### 31.1.13 Regen Brake Monitor



This plugin defines a page including an acceleration level gauge chart, number metrics for the acceleration, speed and

battery power and two large indicators for the regen brake light and the brake pedal state.

It's useful to fine tune the regen brake settings, as you can monitor the actual deceleration levels while driving and check the smoothing level.

**Install:** add the source as a page plugin, e.g. /dev/regenmon.

regenmon.htm (hint: right click, save as)

```
1 <!--
2   Test/Development/Documentation page; install as plugin to test
3 -->
4
5 <style>
6 .indicator > .label {
7   font-size: 150%;
8   line-height: 200%;
9   margin: 10px;
10  padding: 10px;
11  display: block;
12  border-radius: 50px;
13 }
14 .metric.number .label {
15   min-width: 8em;
16 }
17 .metric.number {
18   display: block;
19   float: none;
20   text-align: center;
21 }
22 </style>
23
24 <div class="panel panel-primary">
25   <div class="panel-heading">Regen Brake Monitor</div>
26   <div class="panel-body">
27
28     <div class="receiver" id="regenmon-receiver">
29
30       <div class="row">
31         <div class="col-sm-6">
32           <div class="metric chart" data-metric="v.p.acceleration" style="height:300px
33 ↪">
34
35             <div class="chart-box gaugechart" id="accel-gauge"/>
36           </div>
37           <div class="metric number" data-metric="v.p.acceleration" data-prec="2">
38             <span class="label">Acceleration:</span>
39             <span class="value">?</span>
40             <span class="unit">m/s2</span>
41           </div>
42           <div class="metric number" data-metric="v.p.speed" data-prec="1">
43             <span class="label">Speed:</span>
44             <span class="value">?</span>
45             <span class="unit">kph</span>
46           </div>
47           <div class="metric number" data-metric="v.b.power" data-prec="1">
48             <span class="label">Battery power:</span>
49             <span class="value">?</span>
50             <span class="unit">kW</span>
51           </div>
52         </div>
53       </div>
54     </div>
55   </div>
56 </div>
```

(continues on next page)

(continued from previous page)

```

50     <br class="clearfix"/>
51 </div>
52 <div class="col-sm-6">
53     <div class="indicator" data-metric="v.e.regenbrake">
54         <span class="label label-default">REGEN</span>
55     </div>
56     <div class="indicator" data-metric="v.e.footbrake">
57         <span class="label label-default">FOOT</span>
58     </div>
59 </div>
60 </div>
61
62 </div>
63
64 </div>
65 </div>
66
67 <script>
68 (function() {
69
70     /* Init acceleration gauge: */
71     $("#accel-gauge").chart({
72         chart: {
73             type: 'gauge',
74             spacing: [0, 0, 0, 0],
75             margin: [0, 0, 0, 0],
76             animation: { duration: 250, easing: 'easeOutExpo' },
77         },
78         title: { text: "Acceleration", verticalAlign: "middle", y: 75 },
79         credits: { enabled: false },
80         tooltip: { enabled: false },
81         plotOptions: {
82             gauge: { dataLabels: { enabled: false }, overshoot: 5 }
83         },
84         pane: [{
85             startAngle: -120, endAngle: 120, size: '100%', center: ['50%', '60%']
86         }],
87         yAxis: [{
88             title: { text: 'm/s2' },
89             className: 'accel',
90             reversed: false,
91             min: -2, max: 2,
92             minorTickInterval: 0.1, minorTickLength: 5, minorTickPosition: 'inside',
93             tickInterval: 0.5, tickPosition: 'inside', tickLength: 13,
94             labels: { step: 1, distance: -28, x: 0, y: 5, zIndex: 2 },
95         }],
96         series: [{
97             name: 'Acceleration', data: [0],
98             className: 'accel',
99             animation: { duration: 0 },
100            pivot: { radius: '10' },
101            dial: { radius: '88%', topWidth: 1, baseLength: '20%', baseWidth: 10,
102            ↪ rearLength: '20%' },
103        }],
104        /* Update method: */
105        onUpdate: function(update) {
106            // Create gauge data set from metric:

```

(continues on next page)

(continued from previous page)

```

106     var data = [ metrics["v.p.acceleration"] ];
107     // Update chart:
108     this.series[0].setData(data);
109   },
110   });
111
112   /* Init indicators: */
113   $('#regenmon-receiver').on('msg:metrics', function(e, update) {
114     $(this).find('.indicator').each(function() {
115       var $el = $(this), metric = $el.data("metric"), val = update[metric];
116       if (val == null)
117         return;
118       else if (val != 0)
119         $el.children().removeClass('label-default').addClass('label-danger');
120       else if (val == 0)
121         $el.children().removeClass('label-danger').addClass('label-default');
122     });
123   });
124
125   }) ();
126 </script>

```

### 31.1.14 Dashboard



A refined and configurable version of this plugin has been added to the standard OVMS web UI and gets automatically configured by the vehicle modules with their respective vehicle parameters.

It's a good example of the Highcharts gauge chart options and shows how to combine multiple gauge charts into a single container (div #gaugeset1) and how to style the charts using CSS.

It also includes a simple test data generator so can be tested without actual vehicle data.

To add a new vehicle parameter set for the standard dashboard, simply override the `GetDashboardConfig()` method in your vehicle class. Have a look at the existing overrides (for example Twizy, Kia Soul, Smart ED, ...).

**Install:** not necessary for the standard dashboard. If you're going to build your own dashboard from this, install it as a page type plugin.

dashboard.htm (hint: right click, save as)

```

1 <!-- Main -->
2
3 <style>
4 @media (max-width: 767px) {
5   .panel-single .panel-body {
6     padding: 2px;
7   }
8 }
9 .dashboard .highcharts-data-label text {
10   font-size: 2em;
11 }
12 .dashboard .overlay {
13   position: absolute;
14   z-index: 10;
15   top: 62%;
16   text-align: center;
17   left: 0%;
18   right: 0%;
19 }
20 .dashboard .overlay .value {
21   color: #04282b;
22   background: #97b597;
23   padding: 2px 5px;
24   margin-right: 0.2em;
25   text-align: center;
26   border: 1px inset #c3c3c380;
27   font-family: "Monaco", "Menlo", "Consolas", "QuickType Mono", "Lucida Console",
  ↳ "Roboto Mono", "Ubuntu Mono", "DejaVu Sans Mono", "Droid Sans Mono", monospace;
28   display: inline-block;
29 }
30 .night .dashboard .overlay .value {
31   color: #fff;
32   background: #252525;
33 }
34 .dashboard .overlay .unit {
35   color: #666;
36   font-size: 12px;
37 }
38 .dashboard .overlay .range-value .value {
39   margin-left: 10px;
40   width: 100px;
41 }
42 .dashboard .overlay .energy-value {
43   margin-top: 4px;
44 }
45 .dashboard .overlay .energy-value .value {
46   margin-left: 18px;
47   width: 100px;
48   font-size: 12px;
49 }
50 </style>
51
52 <div class="panel panel-primary" id="panel-dashboard">
53   <div class="panel-heading">Dashboard</div>
54   <div class="panel-body">
55
56     <div class="receiver get-window-resize" id="livestatus">

```

(continues on next page)

(continued from previous page)

```

57     <div class="dashboard" style="position: relative; width: 100%; height: 300px;
↪margin: 0 auto">
58         <div class="overlay">
59             <div class="range-value"><span class="value">0 0</span><span class="unit">km
↪</span></div>
60             <div class="energy-value"><span class="value">0.0 0.0</span><span class=
↪"unit">kWh</span></div>
61             </div>
62             <div id="gaugeset1" style="width: 100%; height: 100%;"></div>
63         </div>
64     </div>
65
66 </div>
67 </div>
68
69
70 <!-- Chart -->
71
72 <style>
73 .highcharts-plot-band, .highcharts-pane {
74     fill-opacity: 0;
75 }
76 .highcharts-plot-band.border {
77     stroke: #666666;
78     stroke-width: 1px;
79 }
80
81 .green-band {
82     fill: #55BF3B;
83     fill-opacity: 0.4;
84 }
85 .yellow-band {
86     fill: #DDDF0D;
87     fill-opacity: 0.5;
88 }
89 .red-band {
90     fill: #DF5353;
91     fill-opacity: 0.6;
92 }
93 .violet-band {
94     fill: #9622ff;
95     fill-opacity: 0.6;
96 }
97 .night .violet-band {
98     fill: #9622ff;
99     fill-opacity: 0.8;
100 }
101
102 .highcharts-gauge-series .highcharts-pivot {
103     stroke-width: 1px;
104     stroke: #757575;
105     fill-opacity: 1;
106     fill: black;
107 }
108 .highcharts-gauge-series.auxgauge .highcharts-pivot {
109     fill-opacity: 1;
110     fill: #fff;

```

(continues on next page)

(continued from previous page)

```

111     stroke-width: 0;
112 }
113 .night .highcharts-gauge-series.auxgauge .highcharts-pivot {
114     fill: #000;
115 }
116 .highcharts-gauge-series .highcharts-dial {
117     fill: #d80000;
118     stroke: #000;
119     stroke-width: 0.5px;
120 }
121 .highcharts-yaxis-grid .highcharts-grid-line,
122 .highcharts-yaxis-grid .highcharts-minor-grid-line {
123     display: none;
124 }
125 .highcharts-yaxis .highcharts-tick {
126     stroke-width: 2px;
127     stroke: #666666;
128 }
129 .night .highcharts-yaxis .highcharts-tick {
130     stroke: #e0e0e0;
131 }
132 .highcharts-yaxis .highcharts-minor-tick {
133     stroke-width: 1.8px;
134     stroke: #00000085;
135     stroke-dasharray: 6;
136     stroke-dashoffset: -4.8;
137     stroke-linecap: round;
138 }
139 .night .highcharts-yaxis .highcharts-minor-tick {
140     stroke: #ffffff85;
141 }
142 .highcharts-axis-labels {
143     fill: #000;
144     font-weight: bold;
145     font-size: 0.9em;
146 }
147 .night .highcharts-axis-labels {
148     fill: #ddd;
149 }
150 .highcharts-data-label text {
151     fill: #333333;
152 }
153 .night .highcharts-data-label text {
154     fill: #fff;
155 }
156
157 .highcharts-axis-labels.speed {
158     font-size: 1.2em;
159 }
160 </style>
161
162 <script type="text/javascript">
163
164 // Vehicle specific configuration:
165
166 var vehicle_config_twizy = {
167     yAxis: [{

```

(continues on next page)

(continued from previous page)

```

168 // Speed:
169 min: 0, max: 120,
170 plotBands: [
171   { from: 0, to: 70, className: 'green-band' },
172   { from: 70, to: 100, className: 'yellow-band' },
173   { from: 100, to: 120, className: 'red-band' }]
174 },{
175 // Voltage:
176 min: 45, max: 60,
177 plotBands: [
178   { from: 45, to: 47.5, className: 'red-band' },
179   { from: 47.5, to: 50, className: 'yellow-band' },
180   { from: 50, to: 60, className: 'green-band' }]
181 },{
182 // SOC:
183 min: 0, max: 100,
184 plotBands: [
185   { from: 0, to: 12.5, className: 'red-band' },
186   { from: 12.5, to: 25, className: 'yellow-band' },
187   { from: 25, to: 100, className: 'green-band' }]
188 },{
189 // Efficiency:
190 min: 0, max: 300,
191 plotBands: [
192   { from: 0, to: 150, className: 'green-band' },
193   { from: 150, to: 250, className: 'yellow-band' },
194   { from: 250, to: 300, className: 'red-band' }]
195 },{
196 // Power:
197 min: -10, max: 30,
198 plotBands: [
199   { from: -10, to: 0, className: 'violet-band' },
200   { from: 0, to: 15, className: 'green-band' },
201   { from: 15, to: 25, className: 'yellow-band' },
202   { from: 25, to: 30, className: 'red-band' }]
203 },{
204 // Charger temperature:
205 min: 20, max: 80, tickInterval: 20,
206 plotBands: [
207   { from: 20, to: 65, className: 'normal-band border' },
208   { from: 65, to: 80, className: 'red-band border' }]
209 },{
210 // Battery temperature:
211 min: -15, max: 65, tickInterval: 25,
212 plotBands: [
213   { from: -15, to: 0, className: 'red-band border' },
214   { from: 0, to: 50, className: 'normal-band border' },
215   { from: 50, to: 65, className: 'red-band border' }]
216 },{
217 // Inverter temperature:
218 min: 20, max: 80, tickInterval: 20,
219 plotBands: [
220   { from: 20, to: 70, className: 'normal-band border' },
221   { from: 70, to: 80, className: 'red-band border' }]
222 },{
223 // Motor temperature:
224 min: 50, max: 125, tickInterval: 25,

```

(continues on next page)



(continued from previous page)

```

225     plotBands: [
226       { from: 50, to: 110, className: 'normal-band border' },
227       { from: 110, to: 125, className: 'red-band border' }]
228   ]
229 };
230
231 var vehicle_config_default = {
232   yAxis: [{
233     // Speed:
234     min: 0, max: 120,
235     plotBands: [
236       { from: 0, to: 70, className: 'green-band' },
237       { from: 70, to: 100, className: 'yellow-band' },
238       { from: 100, to: 120, className: 'red-band' }]
239   }, {
240     // Voltage:
241     min: 310, max: 410,
242     plotBands: [
243       { from: 310, to: 325, className: 'red-band' },
244       { from: 325, to: 340, className: 'yellow-band' },
245       { from: 340, to: 410, className: 'green-band' }]
246   }, {
247     // SOC:
248     min: 0, max: 100,
249     plotBands: [
250       { from: 0, to: 12.5, className: 'red-band' },
251       { from: 12.5, to: 25, className: 'yellow-band' },
252       { from: 25, to: 100, className: 'green-band' }]
253   }, {
254     // Efficiency:
255     min: 0, max: 400,
256     plotBands: [
257       { from: 0, to: 200, className: 'green-band' },
258       { from: 200, to: 300, className: 'yellow-band' },
259       { from: 300, to: 400, className: 'red-band' }]
260   }, {
261     // Power:
262     min: -50, max: 200,
263     plotBands: [
264       { from: -50, to: 0, className: 'violet-band' },
265       { from: 0, to: 100, className: 'green-band' },
266       { from: 100, to: 150, className: 'yellow-band' },
267       { from: 150, to: 200, className: 'red-band' }]
268   }, {
269     // Charger temperature:
270     min: 20, max: 80, tickInterval: 20,
271     plotBands: [
272       { from: 20, to: 65, className: 'normal-band border' },
273       { from: 65, to: 80, className: 'red-band border' }]
274   }, {
275     // Battery temperature:
276     min: -15, max: 65, tickInterval: 25,
277     plotBands: [
278       { from: -15, to: 0, className: 'red-band border' },
279       { from: 0, to: 50, className: 'normal-band border' },
280       { from: 50, to: 65, className: 'red-band border' }]
281   }, {

```

(continues on next page)

(continued from previous page)

```

282 // Inverter temperature:
283 min: 20, max: 80, tickInterval: 20,
284 plotBands: [
285   { from: 20, to: 70, className: 'normal-band border' },
286   { from: 70, to: 80, className: 'red-band border' }]
287 }, {
288 // Motor temperature:
289 min: 50, max: 125, tickInterval: 25,
290 plotBands: [
291   { from: 50, to: 110, className: 'normal-band border' },
292   { from: 110, to: 125, className: 'red-band border' }]
293   ]
294 };
295
296 var vehicle_config = vehicle_config_twizy;
297
298 var gaugeset1;
299
300 function get_dashboard_data() {
301   var rmin = metrics["v.b.range.est"]||0, rmax = metrics["v.b.range.ideal"]||0;
302   var euse = metrics["v.b.energy.used"]||0, erec = metrics["v.b.energy.recd"]||0;
303   if (rmin > rmax) { var x = rmin; rmin = rmax; rmax = x; }
304   var md = {
305     range: { value: "" + rmax.toFixed(0) + " " + rmin.toFixed(0) },
306     energy: { value: "" + euse.toFixed(1) + " " + erec.toFixed(1) },
307     series: [
308       { data: [metrics["v.p.speed"]] },
309       { data: [metrics["v.b.voltage"]] },
310       { data: [metrics["v.b.soc"]] },
311       { data: [metrics["v.b.consumption"]] },
312       { data: [metrics["v.b.power"]] },
313       { data: [metrics["v.c.temp"]] },
314       { data: [metrics["v.b.temp"]] },
315       { data: [metrics["v.i.temp"]] },
316       { data: [metrics["v.m.temp"]] },
317     ];
318   };
319   return md;
320 }
321
322 function update_dashboard() {
323   var md = get_dashboard_data();
324   $(' .range-value .value').text(md.range.value);
325   $(' .energy-value .value').text(md.energy.value);
326   gaugeset1.update({ series: md.series });
327 }
328
329 function init_gaugeset1() {
330   var chart_config = {
331     chart: {
332       type: 'gauge',
333       spacing: [0, 0, 0, 0],
334       margin: [0, 0, 0, 0],
335       animation: { duration: 0, easing: 'swing' },
336     },
337     title: { text: null },
338     credits: { enabled: false },
339     tooltip: { enabled: false },

```

(continues on next page)

(continued from previous page)

```

339
340     pane: [
341         { startAngle: -125, endAngle: 125, center: ['50%', '45%'], size: '80%' }, //
↪Speed
342         { startAngle: 70, endAngle: 110, center: ['-20%', '20%'], size: '100%' }, //
↪Voltage
343         { startAngle: 70, endAngle: 110, center: ['-20%', '60%'], size: '100%' }, // SOC
344         { startAngle: -110, endAngle: -70, center: ['120%', '20%'], size: '100%' }, //
↪Efficiency
345         { startAngle: -110, endAngle: -70, center: ['120%', '60%'], size: '100%' }, //
↪Power
346         { startAngle: -45, endAngle: 45, center: ['20%', '100%'], size: '30%' }, //
↪Charger temperature
347         { startAngle: -45, endAngle: 45, center: ['40%', '100%'], size: '30%' }, //
↪Battery temperature
348         { startAngle: -45, endAngle: 45, center: ['60%', '100%'], size: '30%' }, //
↪Inverter temperature
349         { startAngle: -45, endAngle: 45, center: ['80%', '100%'], size: '30%' }], //
↪Motor temperature
350
351     responsive: {
352         rules: [{
353             condition: { minWidth: 0, maxWidth: 400 },
354             chartOptions: {
355                 pane: [
356                     { size: '60%' }, // Speed
357                     { center: ['-20%', '20%'] }, // Voltage
358                     { center: ['-20%', '60%'] }, // SOC
359                     { center: ['120%', '20%'] }, // Efficiency
360                     { center: ['120%', '60%'] }, // Power
361                     { center: ['15%', '100%'], size: '25%' }, // Charger temperature
362                     { center: ['38.33%', '100%'], size: '25%' }, // Battery temperature
363                     { center: ['61.66%', '100%'], size: '25%' }, // Inverter temperature
364                     { center: ['85%', '100%'], size: '25%' }], // Motor temperature
365                 yAxis: [{ labels: { step: 1 } }], // Speed
366             },
367         ], {
368             condition: { minWidth: 401, maxWidth: 450 },
369             chartOptions: {
370                 pane: [
371                     { size: '70%' }, // Speed
372                     { center: ['-15%', '20%'] }, // Voltage
373                     { center: ['-15%', '60%'] }, // SOC
374                     { center: ['115%', '20%'] }, // Efficiency
375                     { center: ['115%', '60%'] }, // Power
376                     { center: ['15%', '100%'], size: '27.5%' }, // Charger temperature
377                     { center: ['38.33%', '100%'], size: '27.5%' }, // Battery temperature
378                     { center: ['61.66%', '100%'], size: '27.5%' }, // Inverter temperature
379                     { center: ['85%', '100%'], size: '27.5%' }], // Motor temperature
380                 },
381             }, {
382                 condition: { minWidth: 451, maxWidth: 600 },
383                 chartOptions: {
384                     pane: [
385                         { size: '80%' }, // Speed
386                         { center: ['-10%', '20%'] }, // Voltage
387                         { center: ['-10%', '60%'] }, // SOC

```

(continues on next page)

(continued from previous page)

```

388         { center: ['110%', '20%'] }, // Efficiency
389         { center: ['110%', '60%'] }], // Power
390     },
391 }, {
392     condition: { minWidth: 601 },
393     chartOptions: {
394         pane: [
395             { size: '85%' }, // Speed
396             { center: ['0%', '20%'] }, // Voltage
397             { center: ['0%', '60%'] }, // SOC
398             { center: ['100%', '20%'] }, // Efficiency
399             { center: ['100%', '60%'] }], // Power
400         ],
401     }
402 },
403
404 yAxis: [{
405     // Speed axis:
406     pane: 0, className: 'speed', title: { text: 'km/h' },
407     reversed: false,
408     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
409     tickPixelInterval: 30, tickPosition: 'inside', tickLength: 13,
410     labels: { step: 2, distance: -28, x: 0, y: 5, zIndex: 2 },
411 }, {
412     // Voltage axis:
413     pane: 1, className: 'voltage', title: { text: 'Volt', align: 'low', x: 90, y:
414 ↪ 35 },
415     reversed: true,
416     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
417     tickPixelInterval: 30, tickPosition: 'inside', tickLength: 13,
418     labels: { step: 1, distance: -25, x: 0, y: 5, zIndex: 2 },
419 }, {
420     // SOC axis:
421     pane: 2, className: 'soc', title: { text: 'SOC', align: 'low', x: 85, y: 35 },
422     reversed: true,
423     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
424     tickPixelInterval: 30, tickPosition: 'inside', tickLength: 13,
425     labels: { step: 1, distance: -25, x: 0, y: 5, zIndex: 2 },
426 }, {
427     // Efficiency axis:
428     pane: 3, className: 'efficiency', title: { text: 'Wh/km', align: 'low', x: -125,
429 ↪ y: 35 },
430     reversed: false,
431     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
432     tickPixelInterval: 30, tickPosition: 'inside', tickLength: 13,
433     labels: { step: 1, distance: -25, x: 0, y: 5, zIndex: 2 },
434 }, {
435     // Power axis:
436     pane: 4, className: 'power', title: { text: 'kW', align: 'low', x: -115, y: 35 }
437 ↪ ,
438     reversed: false,
439     minorTickInterval: 'auto', minorTickLength: 5, minorTickPosition: 'inside',
440     tickPixelInterval: 30, tickPosition: 'inside', tickLength: 13,
441     labels: { step: 1, distance: -25, x: 0, y: 5, zIndex: 2 },
442 }, {
443     // Charger temperature axis:
444     pane: 5, className: 'temp-charger', title: { text: 'CHG °C', y: 10 },

```

(continues on next page)

(continued from previous page)

```

442     tickPosition: 'inside', tickLength: 10, minorTickInterval: null,
443     labels: { step: 1, distance: 3, x: 0, y: 0, zIndex: 2 },
444   }, {
445     // Battery temperature axis:
446     pane: 6, className: 'temp-battery', title: { text: 'BAT °C', y: 10 },
447     tickPosition: 'inside', tickLength: 10, minorTickInterval: null,
448     labels: { step: 1, distance: 3, x: 0, y: 0, zIndex: 2 },
449   }, {
450     // Inverter temperature axis:
451     pane: 7, className: 'temp-inverter', title: { text: 'PEM °C', y: 10 },
452     tickPosition: 'inside', tickLength: 10, minorTickInterval: null,
453     labels: { step: 1, distance: 3, x: 0, y: 0, zIndex: 2 },
454   }, {
455     // Motor temperature axis:
456     pane: 8, className: 'temp-motor', title: { text: 'MOT °C', y: 10 },
457     tickPosition: 'inside', tickLength: 10, minorTickInterval: null,
458     labels: { step: 1, distance: 3, x: 0, y: 0, zIndex: 2 },
459   } ],
460
461   plotOptions: {
462     gauge: {
463       dataLabels: { enabled: false },
464       overshoot: 1
465     }
466   },
467   series: [ {
468     // Speed value:
469     yAxis: 0, name: 'Speed', className: 'speed fullgauge', data: [0],
470     pivot: { radius: '10' },
471     dial: { radius: '88%', topWidth: 1, baseLength: '20%', baseWidth: 10, ↵
↵rearLength: '20%' },
472   }, {
473     // Voltage value:
474     yAxis: 1, name: 'Voltage', className: 'voltage auxgauge', data: [0],
475     pivot: { radius: '85' },
476     dial: { radius: '95%', baseWidth: 5, baseLength: '90%' },
477   }, {
478     // SOC value:
479     yAxis: 2, name: 'SOC', className: 'soc auxgauge', data: [0],
480     pivot: { radius: '85' },
481     dial: { radius: '95%', baseWidth: 5, baseLength: '90%' },
482   }, {
483     // Efficiency value:
484     yAxis: 3, name: 'Efficiency', className: 'efficiency auxgauge', data: [0],
485     pivot: { radius: '85' },
486     dial: { radius: '95%', baseWidth: 5, baseLength: '90%' },
487   }, {
488     // Power value:
489     yAxis: 4, name: 'Power', className: 'power auxgauge', data: [0],
490     pivot: { radius: '85' },
491     dial: { radius: '95%', baseWidth: 5, baseLength: '90%' },
492   }, {
493     // Charger temperature value:
494     yAxis: 5, name: 'Charger temperature', className: 'temp-charger tempgauge', ↵
↵data: [0],
495     dial: { radius: '90%', baseWidth: 3, baseLength: '90%' },
496   }, {

```

(continues on next page)

(continued from previous page)

```

497     // Battery temperature value:
498     yAxis: 6, name: 'Battery temperature', className: 'temp-battery tempgauge',
↪data: [0],
499     dial: { radius: '90%', baseWidth: 3, baseLength: '90%' },
500     }, {
501     // Inverter temperature value:
502     yAxis: 7, name: 'Inverter temperature', className: 'temp-inverter tempgauge',
↪data: [0],
503     dial: { radius: '90%', baseWidth: 3, baseLength: '90%' },
504     }, {
505     // Motor temperature value:
506     yAxis: 8, name: 'Motor temperature', className: 'temp-motor tempgauge', data:
↪[0],
507     dial: { radius: '90%', baseWidth: 3, baseLength: '90%' },
508     }}
509 };
510
511 // Inject vehicle config:
512 for (var i = 0; i < chart_config.yAxis.length; i++) {
513     $.extend(chart_config.yAxis[i], vehicle_config.yAxis[i]);
514 }
515
516 gaugeset1 = Highcharts.chart('gaugeset1', chart_config,
517     function (chart) {
518         chart.update({ chart: { animation: { duration: 1000, easing: 'swing' } } });
519         $('#livestatus').on("msg:metrics", function(e, update){
520             update_dashboard();
521         }).on("window-resize", function(e) {
522             chart.reflow();
523         });
524     }
525 );
526 }
527
528 function init_charts() {
529     init_gaugeset1();
530 }
531
532 if (window.Highcharts) {
533     init_charts();
534 } else {
535     $.ajax({
536         url: window.assets.charts_js,
537         dataType: "script",
538         cache: true,
539         success: function(){ init_charts(); }
540     });
541 }
542
543 </script>
544
545
546
547 <!-- ***** TESTDATENGGENERATOR ***** -->
548 <br class="clearfix">
549 <div id="viewportinfo">WxH</div>
550 <div id="debugfs"></div>

```

(continues on next page)

(continued from previous page)

```

551 <button class="btn btn-info" id="mkmetrics">Make test metrics</button>
552 <br class="clearfix">
553 <script type="text/javascript">
554 $("#mkmetrics").on("click", function() {
555     var msg = {
556         metrics: {
557             "v.b.range.est": Math.random()*80,
558             "v.b.range.ideal": Math.random()*80,
559             "v.b.energy.used": Math.random()*20,
560             "v.b.energy.recd": Math.random()*20,
561             "v.p.speed": Math.random()*120,
562             "v.b.voltage": 45 + Math.random()*15,
563             "v.b.soc": Math.random()*100,
564             "v.b.consumption": Math.random()*300,
565             "v.b.power": Math.random()*40 - 10,
566             "v.c.temp": Math.random()*80,
567             "v.b.temp": Math.random()*80,
568             "v.i.temp": Math.random()*80,
569             "v.m.temp": Math.random()*80,
570         }
571     };
572     $.extend(metrics, msg.metrics);
573     $(".receiver").trigger("msg:metrics", msg.metrics);
574 });
575 $(window).on("resize", function() {
576     $("#viewportinfo").text($(window).width() + " x " + $(window).height());
577 }).trigger("resize");
578 </script>

```

## 31.1.15 Twizy: Dashboard Tuning Plugin



This plugin adds two sliders to adjust neutral and braking recuperation levels to the dashboard.

The sliders listen to driving profile changes and update accordingly.

If not yet logged in, the sliders are disabled.

**Install:** add the plugin as a hook type to page /dashboard, hook body.pre.

dashboard-tuneslider.htm (hint: right click, save as)

```

1 <!--
2   Hook plugin for /dashboard:body.pre or :body.post
3   - add sliders to adjust recuperation power levels
4 -->
5
6 <style>
7 #tuneslider {
8   margin: 10px 8px 0;
9 }
10 .form-inline .form-control.slider-value {
11   width: 80px;

```

(continues on next page)



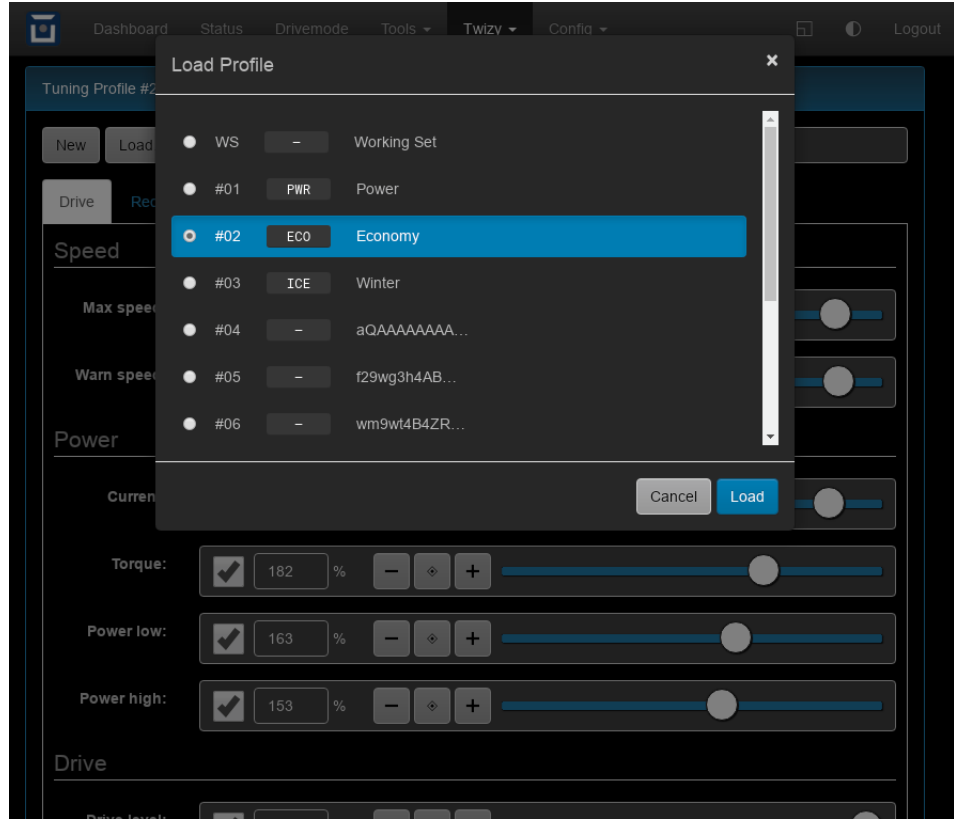
(continued from previous page)

```

12 }
13 </style>
14
15 <div class="receiver" id="tuneslider" style="display:none">
16   <div class="form-group">
17     <label class="control-label" for="input-neutral">Neutral recuperation:</label>
18     <div class="form-control slider" id="neutral" />
19   </div>
20   <div class="form-group">
21     <label class="control-label" for="input-brake">Brake recuperation:</label>
22     <div class="form-control slider" id="brake" />
23   </div>
24 </div>
25
26 <script>
27 (function() {
28
29   // Init sliders:
30   $('#neutral').slider({ min:0, max:100, step:1, unit:'%', default:18, value:18,
31   ↪checked:false });
32   $('#brake').slider({ min:0, max:100, step:1, unit:'%', default:18, value:18,
33   ↪checked:false });
34
35   // Update sliders on profile changes:
36   var profile;
37   $('#tuneslider').on('msg:metrics', function(ev, update) {
38     if (update["xrt.cfg.profile"] != null) {
39       profile = update["xrt.cfg.profile"];
40       var neutral = profile[7], brake = profile[8]; // see cfgconv.c tagnames for_
41   ↪profile structure
42       $('#neutral').slider({ checked: (neutral!=-1), value: (neutral!=-1) ? neutral :
43   ↪null });
44       $('#brake').slider({ checked: (brake!=-1), value: (brake!=-1) ? brake : null });
45     }
46   });
47
48   // Update profile on slider changes:
49   $('#tuneslider .slider-value').on('change', function(ev) {
50     var neutral = $('#input-neutral').prop('checked') ? $('#input-neutral').val() : -
51   ↪1,
52     brake = $('#input-brake').prop('checked') ? $('#input-brake').val() : -1,
53     autorecup_minprc = profile[43], autorecup_ref = profile[44];
54     var cmd = "xrt cfg recup " + neutral + " " + brake + " " + autorecup_ref + " " +
55   ↪autorecup_minprc;
56     loadcmd(cmd, '#loadres');
57   });
58
59   // Install into panel:
60   $('#main').one('load', function(ev) {
61     if (!loggedin) {
62       $('#tuneslider .slider').slider({ disabled: true });
63     }
64     $('#tuneslider').appendTo('#panel-dashboard .panel-body').show();
65   });
66 }) ();
67 </script>

```

### 31.1.16 Twizy: Tuning Profile Editor



This plugin has been added to the Twizy code. It's used here as a more complex example of what can be done by plugins.

It's a full featured SEVCON tuning profile editor including dialogs to load & save profiles from the OVMS configuration and to create & read base64 profile codes.

See Twizy documentation for details on tuning profiles and capabilities.

**Install:** not necessary if vehicle Twizy is configured (see "Twizy" menu). If you'd like to test this for another vehicle, add as a page plugin e.g. /test/profed.

profile-editor.htm (hint: right click, save as)

```

1 <!--
2   Twizy page plugin: Tuning Profile Editor
3   Note: included in firmware v3.2
4 -->
5
6 <style>
7 .form-inline .form-control.slider-value {
8   width: 70px;
9 }
10 .radio-list {
11   height: 313px;
12   overflow-y: auto;
13   overflow-x: hidden;
14   padding-right: 15px;
15 }
```

(continues on next page)

(continued from previous page)

```

16 .radio-list .radio {
17     overflow: hidden;
18 }
19 .radio-list .key {
20     min-width: 20px;
21     display: inline-block;
22     text-align: center;
23     margin: 0 10px;
24 }
25 .radio-list kbd {
26     min-width: 60px;
27     display: inline-block;
28     text-align: center;
29     margin: 0 20px 0 10px;
30 }
31 .radio-list .radio label {
32     width: 100%;
33     text-align: left;
34     padding: 8px 30px;
35 }
36 .radio-list .radio label.active {
37     background-color: #337ab7;
38     color: #fff;
39     outline: none;
40 }
41 .radio-list .radio label.active input {
42     outline: none;
43 }
44 .night .radio-list .radio label:hover {
45     color: #fff;
46 }
47 </style>
48
49 <div class="panel panel-primary">
50     <div class="panel-heading">Tuning Profile <span id="headkey">Editor</span></div>
51     <div class="panel-body">
52         <form action="#">
53
54             <div class="form-group">
55                 <div class="flex-group">
56                     <button type="button" class="btn btn-default action-new">New</button>
57                     <button type="button" class="btn btn-default action-load">Load...</button>
58                     <input type="hidden" id="input-key" name="key" value="">
59                     <input type="hidden" id="input-base64-reset" name="base64-reset" value="">
60                     <input type="text" class="form-control font-monospace" placeholder="Base64_
61 ↪profile code"
62                     name="base64" id="input-base64" value="" autocapitalize="none" autocorrect=
63 ↪"off"
64                     autocomplete="off" spellcheck="false">
65                 </div>
66             </div>
67
68             <ul class="nav nav-tabs">
69                 <li class="active"><a data-toggle="tab" href="#tab-drive" aria-expanded="true">
70 ↪Drive</a></li>
71                 <li class=""><a data-toggle="tab" href="#tab-recup" aria-expanded="false">Recup
72 ↪</a></li>

```

(continues on next page)

(continued from previous page)

```

69     <li class=""><a data-toggle="tab" href="#tab-ramps" aria-expanded="false">Ramps
    ↪ </a></li>
70 </ul>
71 <div class="tab-content">
72
73     <div id="tab-drive" class="tab-pane section-drive active in">
74
75         <fieldset id="part-speed">
76             <legend>Speed</legend>
77             <div class="form-horizontal">
78                 <div class="form-group">
79                     <label class="control-label col-sm-2" for="input-speed">Max speed:</
    ↪ label>
80                     <div class="col-sm-10"><div class="form-control slider" id="speed" /></
    ↪ div>
81                 </div>
82                 <div class="form-group">
83                     <label class="control-label col-sm-2" for="input-warn">Warn speed:</
    ↪ label>
84                     <div class="col-sm-10"><div class="form-control slider" id="warn" /></
    ↪ div>
85                 </div>
86             </div>
87         </fieldset>
88
89         <fieldset id="part-power">
90             <legend>Power</legend>
91             <div class="form-horizontal">
92                 <div class="form-group">
93                     <label class="control-label col-sm-2" for="input-current">Current:</
    ↪ label>
94                     <div class="col-sm-10"><div class="form-control slider" id="current" />
    ↪ </div>
95                 </div>
96                 <div class="form-group">
97                     <label class="control-label col-sm-2" for="input-torque">Torque:</label>
98                     <div class="col-sm-10"><div class="form-control slider" id="torque" /></
    ↪ div>
99                 </div>
100                <div class="form-group">
101                    <label class="control-label col-sm-2" for="input-power_low">Power low:</
    ↪ label>
102                    <div class="col-sm-10"><div class="form-control slider" id="power_low" /
    ↪ ></div>
103                </div>
104                <div class="form-group">
105                    <label class="control-label col-sm-2" for="input-power_high">Power high:
    ↪ </label>
106                    <div class="col-sm-10"><div class="form-control slider" id="power_high"
    ↪ ></div>
107                </div>
108            </div>
109        </fieldset>
110
111        <fieldset id="part-drive">
112            <legend>Drive</legend>
113            <div class="form-horizontal">

```

(continues on next page)

(continued from previous page)

```

114         <div class="form-group">
115             <label class="control-label col-sm-2" for="input-drive">Drive level:</
↪label>
116             <div class="col-sm-10"><div class="form-control slider" id="drive" /></
↪div>
117         </div>
118         <div class="form-group">
119             <label class="control-label col-sm-2" for="input-autodrive_ref">Auto 100
↪% ref:</label>
120             <div class="col-sm-10"><div class="form-control slider" id="autodrive_
↪ref" /></div>
121         </div>
122         <div class="form-group">
123             <label class="control-label col-sm-2" for="input-autodrive_minprc">Auto
↪min level:</label>
124             <div class="col-sm-10"><div class="form-control slider" id="autodrive_
↪minprc" /></div>
125         </div>
126     </div>
127 </fieldset>
128
129 <fieldset id="part-tsmagd">
130     <legend>Torque/Speed-Map: Drive</legend>
131     <div class="form-horizontal">
132         <div class="form-group">
133             <label class="control-label col-sm-2" for="input-tds_prc1">Trq level 1:
↪</label>
134             <div class="col-sm-10"><div class="form-control slider" id="tds_prc1" />
↪</div>
135         </div>
136         <div class="form-group">
137             <label class="control-label col-sm-2" for="input-tds_spd1">...at speed:
↪</label>
138             <div class="col-sm-10"><div class="form-control slider" id="tds_spd1" />
↪</div>
139         </div>
140         <div class="form-group">
141             <label class="control-label col-sm-2" for="input-tds_prc2">Trq level 2:
↪</label>
142             <div class="col-sm-10"><div class="form-control slider" id="tds_prc2" />
↪</div>
143         </div>
144         <div class="form-group">
145             <label class="control-label col-sm-2" for="input-tds_spd2">...at speed:
↪</label>
146             <div class="col-sm-10"><div class="form-control slider" id="tds_spd2" />
↪</div>
147         </div>
148         <div class="form-group">
149             <label class="control-label col-sm-2" for="input-tds_prc3">Trq level 3:
↪</label>
150             <div class="col-sm-10"><div class="form-control slider" id="tds_prc3" />
↪</div>
151         </div>
152         <div class="form-group">
153             <label class="control-label col-sm-2" for="input-tds_spd3">...at speed:
↪</label>

```

(continues on next page)

(continued from previous page)

```

154         <div class="col-sm-10"><div class="form-control slider" id="tsd_spd3" />
↪ </div>
155     </div>
156     <div class="form-group">
157         <label class="control-label col-sm-2" for="input-tsd_prc4">Trq level 4:
↪ </label>
158         <div class="col-sm-10"><div class="form-control slider" id="tsd_prc4" />
↪ </div>
159     </div>
160     <div class="form-group">
161         <label class="control-label col-sm-2" for="input-tsd_spd4">...at speed:
↪ </label>
162         <div class="col-sm-10"><div class="form-control slider" id="tsd_spd4" />
↪ </div>
163     </div>
164 </div>
165 </fieldset>
166
167 </div>
168
169 <div id="tab-recup" class="tab-pane section-recup">
170
171     <fieldset id="part-recup">
172         <legend>Recup</legend>
173         <div class="form-horizontal">
174             <div class="form-group">
175                 <label class="control-label col-sm-2" for="input-neutral">Neutral level:
↪ </label>
176                 <div class="col-sm-10"><div class="form-control slider" id="neutral" />
↪ </div>
177             </div>
178             <div class="form-group">
179                 <label class="control-label col-sm-2" for="input-brake">Brake level:</
↪ label>
180                 <div class="col-sm-10"><div class="form-control slider" id="brake" /></
↪ div>
181             </div>
182             <div class="form-group">
183                 <label class="control-label col-sm-2" for="input-autorecup_ref">Auto 100
↪ % ref:</label>
184                 <div class="col-sm-10"><div class="form-control slider" id="autorecup_
↪ ref" /></div>
185             </div>
186             <div class="form-group">
187                 <label class="control-label col-sm-2" for="input-autorecup_minprc">Auto
↪ min level:</label>
188                 <div class="col-sm-10"><div class="form-control slider" id="autorecup_
↪ minprc" /></div>
189             </div>
190         </div>
191     </fieldset>
192
193     <fieldset id="part-tsmapn">
194         <legend>Torque/Speed-Map: Neutral</legend>
195         <div class="form-horizontal">
196             <div class="form-group">
197                 <label class="control-label col-sm-2" for="input-tsn_prc1">Trq level 1:
↪ </label>

```

(continues on next page)

(continued from previous page)

```

198         <div class="col-sm-10"><div class="form-control slider" id="tsn_prc1" />
↪ </div>
199     </div>
200     <div class="form-group">
201         <label class="control-label col-sm-2" for="input-tsn_spd1">...at speed:
↪ </label>
202         <div class="col-sm-10"><div class="form-control slider" id="tsn_spd1" />
↪ </div>
203     </div>
204     <div class="form-group">
205         <label class="control-label col-sm-2" for="input-tsn_prc2">Trq level 2:
↪ </label>
206         <div class="col-sm-10"><div class="form-control slider" id="tsn_prc2" />
↪ </div>
207     </div>
208     <div class="form-group">
209         <label class="control-label col-sm-2" for="input-tsn_spd2">...at speed:
↪ </label>
210         <div class="col-sm-10"><div class="form-control slider" id="tsn_spd2" />
↪ </div>
211     </div>
212     <div class="form-group">
213         <label class="control-label col-sm-2" for="input-tsn_prc3">Trq level 3:
↪ </label>
214         <div class="col-sm-10"><div class="form-control slider" id="tsn_prc3" />
↪ </div>
215     </div>
216     <div class="form-group">
217         <label class="control-label col-sm-2" for="input-tsn_spd3">...at speed:
↪ </label>
218         <div class="col-sm-10"><div class="form-control slider" id="tsn_spd3" />
↪ </div>
219     </div>
220     <div class="form-group">
221         <label class="control-label col-sm-2" for="input-tsn_prc4">Trq level 4:
↪ </label>
222         <div class="col-sm-10"><div class="form-control slider" id="tsn_prc4" />
↪ </div>
223     </div>
224     <div class="form-group">
225         <label class="control-label col-sm-2" for="input-tsn_spd4">...at speed:
↪ </label>
226         <div class="col-sm-10"><div class="form-control slider" id="tsn_spd4" />
↪ </div>
227     </div>
228 </div>
229 </fieldset>
230
231 <fieldset id="part-tsmabp">
232     <legend>Torque/Speed-Map: Brake</legend>
233     <div class="form-horizontal">
234         <div class="form-group">
235             <label class="control-label col-sm-2" for="input-tsb_prc1">Trq level 1:
↪ </label>
236             <div class="col-sm-10"><div class="form-control slider" id="tsb_prc1" />
↪ </div>
237         </div>

```

(continues on next page)

(continued from previous page)

```

238     <div class="form-group">
239         <label class="control-label col-sm-2" for="input-tsb_spd1">...at speed:
↪ </label>
240         <div class="col-sm-10"><div class="form-control slider" id="tsb_spd1" />
↪ </div>
241     </div>
242     <div class="form-group">
243         <label class="control-label col-sm-2" for="input-tsb_prc2">Trq level 2:
↪ </label>
244         <div class="col-sm-10"><div class="form-control slider" id="tsb_prc2" />
↪ </div>
245     </div>
246     <div class="form-group">
247         <label class="control-label col-sm-2" for="input-tsb_spd2">...at speed:
↪ </label>
248         <div class="col-sm-10"><div class="form-control slider" id="tsb_spd2" />
↪ </div>
249     </div>
250     <div class="form-group">
251         <label class="control-label col-sm-2" for="input-tsb_prc3">Trq level 3:
↪ </label>
252         <div class="col-sm-10"><div class="form-control slider" id="tsb_prc3" />
↪ </div>
253     </div>
254     <div class="form-group">
255         <label class="control-label col-sm-2" for="input-tsb_spd3">...at speed:
↪ </label>
256         <div class="col-sm-10"><div class="form-control slider" id="tsb_spd3" />
↪ </div>
257     </div>
258     <div class="form-group">
259         <label class="control-label col-sm-2" for="input-tsb_prc4">Trq level 4:
↪ </label>
260         <div class="col-sm-10"><div class="form-control slider" id="tsb_prc4" />
↪ </div>
261     </div>
262     <div class="form-group">
263         <label class="control-label col-sm-2" for="input-tsb_spd4">...at speed:
↪ </label>
264         <div class="col-sm-10"><div class="form-control slider" id="tsb_spd4" />
↪ </div>
265     </div>
266     </div>
267     </fieldset>
268
269 </div>
270
271 <div id="tab-ramps" class="tab-pane section-ramps">
272
273     <fieldset id="part-rampsglobal">
274         <legend>General</legend>
275         <div class="form-horizontal">
276             <div class="form-group">
277                 <label class="control-label col-sm-2" for="input-ramplimit_accel">Limit_
↪ up:</label>
278                 <div class="col-sm-10"><div class="form-control slider" id="ramplimit_
↪ accel" /></div>

```

(continues on next page)



(continued from previous page)

```

279         </div>
280         <div class="form-group">
281             <label class="control-label col-sm-2" for="input-ramplimit_decel">Limit_
↪ down:</label>
282             <div class="col-sm-10"><div class="form-control slider" id="ramplimit_
↪ decel" /></div>
283             </div>
284             <div class="form-group">
285                 <label class="control-label col-sm-2" for="input-smooth">Smoothing:</
↪ label>
286                 <div class="col-sm-10"><div class="form-control slider" id="smooth" /></
↪ div>
287                 </div>
288             </div>
289         </fieldset>
290
291         <fieldset id="part-ramps">
292             <legend>Ramp Speeds</legend>
293             <div class="form-horizontal">
294                 <div class="form-group">
295                     <label class="control-label col-sm-2" for="input-ramp_start">Start/
↪ reverse:</label>
296                     <div class="col-sm-10"><div class="form-control slider" id="ramp_start"
↪ /></div>
297                     </div>
298                     <div class="form-group">
299                         <label class="control-label col-sm-2" for="input-ramp_accel">Accelerate:
↪ </label>
300                         <div class="col-sm-10"><div class="form-control slider" id="ramp_accel"
↪ /></div>
301                         </div>
302                         <div class="form-group">
303                             <label class="control-label col-sm-2" for="input-ramp_decel">Decelerate:
↪ </label>
304                             <div class="col-sm-10"><div class="form-control slider" id="ramp_decel"
↪ /></div>
305                             </div>
306                             <div class="form-group">
307                                 <label class="control-label col-sm-2" for="input-ramp_neutral">Neutral:
↪ </label>
308                                 <div class="col-sm-10"><div class="form-control slider" id="ramp_neutral
↪ " /></div>
309                                 </div>
310                                 <div class="form-group">
311                                     <label class="control-label col-sm-2" for="input-ramp_brake">Brake:</
↪ label>
312                                     <div class="col-sm-10"><div class="form-control slider" id="ramp_brake"
↪ /></div>
313                                     </div>
314                                 </div>
315                             </fieldset>
316
317                     </div>
318
319                 </div>
320
321                 <br>

```

(continues on next page)

(continued from previous page)

```

322 <div class="form-horizontal">
323   <div class="form-group">
324     <label class="control-label col-sm-2" for="input-label">Label:</label>
325     <div class="col-sm-10">
326       <input type="text" class="form-control" placeholder="Short button label"
327         name="label" id="input-label" value="" autocapitalize="none" autocorrect=
↪ "off"
328         autocomplete="off" spellcheck="false">
329     </div>
330   </div>
331   <div class="form-group">
332     <label class="control-label col-sm-2" for="input-title">Title:</label>
333     <div class="col-sm-10">
334       <input type="text" class="form-control" placeholder="optional title/name"
335         name="title" id="input-title" value="" autocapitalize="none" autocorrect=
↪ "off"
336         autocomplete="off" spellcheck="false">
337     </div>
338   </div>
339 </div>
340
341 <br>
342 <div class="text-center">
343   <button type="button" class="btn btn-default action-reset">Reset</button>
344   <button type="button" class="btn btn-default action-saveas">Save as...</button>
345   <button type="button" class="btn btn-primary action-save">Save</button>
346 </div>
347
348 </form>
349 </div>
350 </div>
351
352 <div id="key-dialog" />
353
354 <script>
355 (function() {
356
357   // init sliders:
358
359   $('#speed').slider({ unit:'kph', min:6, max:120, default:80, value:80,
↪ checked:false });
360   $('#warn').slider({ unit:'kph', min:6, max:120, default:89, value:89, checked:false,
↪ });
361
362   $('#current').slider({ unit:'%', min:10, max:123, default:100, value:100,
↪ checked:false });
363   $('#torque').slider({ unit:'%', min:10, max:130, default:100, value:100,
↪ checked:false });
364   $('#power_low').slider({ unit:'%', min:10, max:139, default:100, value:100,
↪ checked:false });
365   $('#power_high').slider({ unit:'%', min:10, max:130, default:100, value:100,
↪ checked:false });
366
367   $('#drive').slider({ unit:'%', min:10, max:100, default:100, value:100,
↪ checked:false });
368   $('#autodrive_ref').slider({ unit:'kW', min:0, max:25, step:0.1, default:0, value:0,
↪ checked:false });

```

(continues on next page)

(continued from previous page)

```

369  $('#autodrive_minprc').slider({ unit: '%', min:0, max:100, default:0, value:0,
↪checked:false });
370
371  $('#tsd_prc1').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
372  $('#tsd_spd1').slider({ unit: 'kph', min:0, max:120, default:33, value:33,
↪checked:false });
373  $('#tsd_prc2').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
374  $('#tsd_spd2').slider({ unit: 'kph', min:0, max:120, default:39, value:39,
↪checked:false });
375  $('#tsd_prc3').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
376  $('#tsd_spd3').slider({ unit: 'kph', min:0, max:120, default:50, value:50,
↪checked:false });
377  $('#tsd_prc4').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
378  $('#tsd_spd4').slider({ unit: 'kph', min:0, max:120, default:66, value:66,
↪checked:false });
379
380  $('#neutral').slider({ unit: '%', min:0, max:100, default:18, value:18,
↪checked:false });
381  $('#brake').slider({ unit: '%', min:0, max:100, default:18, value:18, checked:false }
↪);
382  $('#autorecup_ref').slider({ unit: 'kW', min:0, max:25, step:0.1, default:0, value:0,
↪checked:false });
383  $('#autorecup_minprc').slider({ unit: '%', min:0, max:100, default:0, value:0,
↪checked:false });
384
385  $('#tsn_prc1').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
386  $('#tsn_spd1').slider({ unit: 'kph', min:0, max:120, default:33, value:33,
↪checked:false });
387  $('#tsn_prc2').slider({ unit: '%', min:0, max:100, default:80, value:80,
↪checked:false });
388  $('#tsn_spd2').slider({ unit: 'kph', min:0, max:120, default:39, value:39,
↪checked:false });
389  $('#tsn_prc3').slider({ unit: '%', min:0, max:100, default:50, value:50,
↪checked:false });
390  $('#tsn_spd3').slider({ unit: 'kph', min:0, max:120, default:50, value:50,
↪checked:false });
391  $('#tsn_prc4').slider({ unit: '%', min:0, max:100, default:20, value:20,
↪checked:false });
392  $('#tsn_spd4').slider({ unit: 'kph', min:0, max:120, default:66, value:66,
↪checked:false });
393
394  $('#tsb_prc1').slider({ unit: '%', min:0, max:100, default:100, value:100,
↪checked:false });
395  $('#tsb_spd1').slider({ unit: 'kph', min:0, max:120, default:33, value:33,
↪checked:false });
396  $('#tsb_prc2').slider({ unit: '%', min:0, max:100, default:80, value:80,
↪checked:false });
397  $('#tsb_spd2').slider({ unit: 'kph', min:0, max:120, default:39, value:39,
↪checked:false });
398  $('#tsb_prc3').slider({ unit: '%', min:0, max:100, default:50, value:50,
↪checked:false });
399  $('#tsb_spd3').slider({ unit: 'kph', min:0, max:120, default:50, value:50,
↪checked:false });

```

(continues on next page)

(continued from previous page)

```

400   $('#tsb_prc4').slider({ unit: '%', min:0, max:100, default:20, value:20,
↪checked:false });
401   $('#tsb_spd4').slider({ unit: 'kph', min:0, max:120, default:66, value:66,
↪checked:false });
402
403
404   $('#ramplimit_accel').slider({ unit: '%', min:1, max:100, default:30, value:30,
↪checked:false });
405   $('#ramplimit_decel').slider({ unit: '%', min:0, max:100, default:30, value:30,
↪checked:false });
406   $('#smooth').slider({ unit: '%', min:0, max:100, default:70, value:70, checked:false,
↪});
407
408   $('#ramp_start').slider({ unit: '%', min:1, max:250, default:40, value:40,
↪checked:false });
409   $('#ramp_accel').slider({ unit: '%', min:1, max:100, default:25, value:25,
↪checked:false });
410   $('#ramp_decel').slider({ unit: '%', min:0, max:100, default:20, value:20,
↪checked:false });
411   $('#ramp_neutral').slider({ unit: '%', min:0, max:100, default:40, value:40,
↪checked:false });
412   $('#ramp_brake').slider({ unit: '%', min:0, max:100, default:40, value:40,
↪checked:false });
413
414   // profile handling:
415
416   const keys = [
417     "checksum",
418     "speed",
419     "warn",
420     "torque",
421     "power_low",
422     "power_high",
423     "drive",
424     "neutral",
425     "brake",
426     "tsd_spd1",
427     "tsd_spd2",
428     "tsd_spd3",
429     "tsd_spd4",
430     "tsd_prc1",
431     "tsd_prc2",
432     "tsd_prc3",
433     "tsd_prc4",
434     "tsn_spd1",
435     "tsn_spd2",
436     "tsn_spd3",
437     "tsn_spd4",
438     "tsn_prc1",
439     "tsn_prc2",
440     "tsn_prc3",
441     "tsn_prc4",
442     "tsb_spd1",
443     "tsb_spd2",
444     "tsb_spd3",
445     "tsb_spd4",
446     "tsb_prc1",

```

(continues on next page)

(continued from previous page)

```

447   "tsb_prc2",
448   "tsb_prc3",
449   "tsb_prc4",
450   "ramp_start",
451   "ramp_accel",
452   "ramp_decel",
453   "ramp_neutral",
454   "ramp_brake",
455   "smooth",
456   "brakelight_on",
457   "brakelight_off",
458   "ramplimit_accel",
459   "ramplimit_decel",
460   "autorecup_minprc",
461   "autorecup_ref",
462   "autodrive_minprc",
463   "autodrive_ref",
464   "current",
465 ];
466
467 const scale = {
468   "autodrive_ref": 0.1,
469   "autorecup_ref": 0.1,
470 };
471
472 var profile = {};
473
474 var plist = [
475   { label: "-", title: "Working Set" },
476   { label: "PWR", title: "Power" },
477   { label: "ECO", title: "Economy" },
478   { label: "ICE", title: "Winter" },
479 ];
480
481 // load profile list:
482 var plistloader = loadcmd('config list xrt').then(function(data) {
483   var lines = data.split('\n'), line, i, key;
484   for (i = 0; i < lines.length; i++) {
485     line = lines[i].match(/profile([0-9]{2})\.(?([^\:]*): (.*)/);
486     if (line && line.length == 4) {
487       key = Number(line[1]);
488       if (key < 1 || key > 99) continue;
489       if (!plist[key]) plist[key] = {};
490       plist[key][line[2]||"profile"] = line[3];
491     }
492   }
493 });
494
495 // current control → power ranges:
496 function currentControl(on, trigger) {
497   if (on) {
498     $("#input-torque, #input-power_low, #input-power_high").
↪ slider({ max: 254 });
499   } else {
500     $("#input-torque, #input-power_high").slider({ max: 130 });
501     $("#input-power_low").slider({ max: 139 });
502   }

```

(continues on next page)

(continued from previous page)

```

503     }
504     $('#input-current').on('change', function(ev) {
505         currentControl(this.checked);
506         $('#input-torque, #input-power_low, #input-power_high').trigger('change');
507     });
508
509     // load profile into sliders:
510     function loadProfile() {
511         currentControl(profile["current"] >= 0);
512         $.map(profile, function(val, key) {
513             $('#input-'+key).slider({ value: (val >= 0) ? (val * (scale[key]||1)) : null,
↪checked: (val >= 0) });
514         });
515     }
516
517     // calculate profile checksum:
518     function calcChecksum() {
519         var checksum, i;
520         checksum = 0x0101;
521         for (i=1; i<keys.length; i++)
522             checksum += profile[keys[i]] + 1;
523         if ((checksum & 0x0ff) == 0)
524             checksum >>= 8;
525         return (checksum & 0x0ff) - 1;
526     }
527
528     // load a base64 string into profile:
529     function loadBase64(base64) {
530         var bin = atob(base64);
531         $.map(keys, function(key, i) { profile[key] = (bin.charCodeAt(i)||0) - 1; });
532         if (profile["checksum"] == calcChecksum()) {
533             loadProfile();
534             return true;
535         } else {
536             confirmdialog("Profile Error", "Invalid base64 code (checksum mismatch)", [
↪"Close"]);
537             return false;
538         }
539     }
540
541     // make a base64 string from profile:
542     function makeBase64() {
543         profile["checksum"] = calcChecksum();
544         var u8 = new Uint8Array(keys.length);
545         $.map(keys, function(key, i) { u8[i] = profile[key] + 1; });
546         return btoa(String.fromCharCode.apply(null, u8));
547     }
548
549     // load a profile:
550     function loadKey(key) {
551         if (key < 0 || key > 99) return;
552         plistloader.then(function() {
553             $('#input-label').val(plist[key] && plist[key]["label"] || "");
554             $('#input-title').val(plist[key] && plist[key]["title"] || "");
555             loadcmd('xrt cfg get ' + key).fail(function(request, textStatus, errorThrown) {
556                 confirmdialog("Load Profile", xhrErrorInfo(request, textStatus, errorThrown),
↪["Close"]);

```

(continues on next page)

(continued from previous page)

```

557     }).done(function(res) {
558         var base64;
559         if (res.match(/error/i))
560             base64 = "AQ";
561         else
562             base64 = res.substr(res.lastIndexOf(' ') + 1);
563         if (loadBase64(base64)) {
564             $('#input-key').val(key);
565             $('#input-base64, #input-base64-reset').val(base64);
566             $('#headkey').text('#' + key);
567         }
568     });
569 });
570
571
572 // save profile:
573 function saveKey(key) {
574     if (key < 0 || key > 99) return;
575     var base64 = $('#input-base64').val(),
576         label = $('#input-label').val().replace(/"/g, '\\"'),
577         title = $('#input-title').val().replace(/"/g, '\\"'),
578         ckey = 'profile' + ((key < 10) ? '0': '') + key;
579     var cmd = 'xrt cfg set ' + key + ' ' + base64 + ' ' + label + ' ' + title + ' '
580     ↪';
581     loadcmd(cmd).fail(function(request, textStatus, errorThrown) {
582         confirmdialog("Save Profile", xhrErrorInfo(request, textStatus, errorThrown), [
583         ↪ "Close"]);
584     }).done(function(res) {
585         confirmdialog("Save Profile", res, ["Close"]);
586         if (!res.match(/error/i)) {
587             if (key > 0) {
588                 if (!plist[key]) plist[key] = {};
589                 plist[key]["label"] = label;
590                 plist[key]["title"] = title;
591             }
592             $('#input-key').val(key);
593             $('#input-base64-reset').val(base64);
594             $('#headkey').text('#' + key);
595         }
596     });
597 }
598
599 // base64 input:
600 $('#input-base64').on('change', function(ev) {
601     var base64 = $(this).val() || "AQ";
602     if (loadBase64(base64))
603         $('#input-base64-reset').val(base64);
604 });
605
606 // update profile & base64 on slider changes:
607 $('.slider-value').on('change', function(ev) {
608     var key = this.name;
609     var val = Math.max(this.min, Math.min(this.max, 1 * this.value));
610     profile[key] = this.checked ? (val / (scale[key] || 1)) : -1;
611     var base64 = makeBase64();
612     $('#input-base64').val(base64);
613 });

```

(continues on next page)

(continued from previous page)

```

612
613 // prep key dialog:
614 $('#key-dialog').dialog({
615   show: false,
616   onShow: function(input) {
617     var $this = $(this);
618     $this.addClass("loading");
619     plistloader.then(function(data) {
620       var curkey = $('#input-key').val() || 0, i, label, title;
621       $plist = $('<div class="radio-list" data-toggle="buttons" />');
622       for (i = 0; i <= Math.min(99, plist.length + 2); i++) {
623         if (plist[i] && (i==0 || plist[i].profile)) {
624           label = plist[i].label || "-";
625           title = plist[i].title || (plist[i].profile.substr(0, 10) + "...");
626         } else {
627           label = "-";
628           title = "-new-";
629         }
630         $plist.append('<div class="radio"><label class="btn">' +
631           '<input type="radio" name="key" value="' + i + '"><span class="key">' +
632           ((i==0) ? "WS" : ("#" + ((i<10)?'0':'') + i)) + '</span> <kbd>' +
633           encode_html(label) + '</kbd>' + encode_html(title) + '</label></div>');
634       }
635       $this.find('.modal-body').append($plist).find('input[value="'+curkey+'"]')
636         .prop("checked", true).parent().addClass("active");
637       $plist
638         .on('dblclick', 'label.btn', function(ev) { $this.dialog('triggerButton',
↪1); });
639         .on('keypress', function(ev) { if (ev.which==13) $this.dialog('triggerButton
↪', 1); });
640       $this.removeClass("loading");
641     });
642   },
643   onShown: function(input) {
644     $(this).find('.btn.active').focus();
645   },
646   onHide: function(input) {
647     var $this = $(this), dlg = $this.data("dialog");
648     var key = $this.find('input[name="key"]:checked').val();
649     if (key !== undefined && input.button && input.button.index)
650       dlg.options.onAction.call(this, key);
651   },
652 });
653
654 // buttons:
655 $('.action-new').on('click', function(ev) {
656   $('.slider').slider({ value: null });
657   $('#headkey').text('Editor');
658   $('#input-base64, #input-label, #input-title').val('').trigger('change');
659 });
660 $('.action-reset').on('click', function(ev) {
661   $('#input-base64').val($('#input-base64-reset').val()).trigger('change');
662 });
663 $('.action-load').on('click', function(ev) {
664   $('#key-dialog').dialog({
665     show: true, title: 'Load Profile', body: '',
666     buttons: [{ label: 'Cancel', btnClass: 'default' }, { label: 'Load', btnClass:
↪'primary' }],

```

(continues on next page)



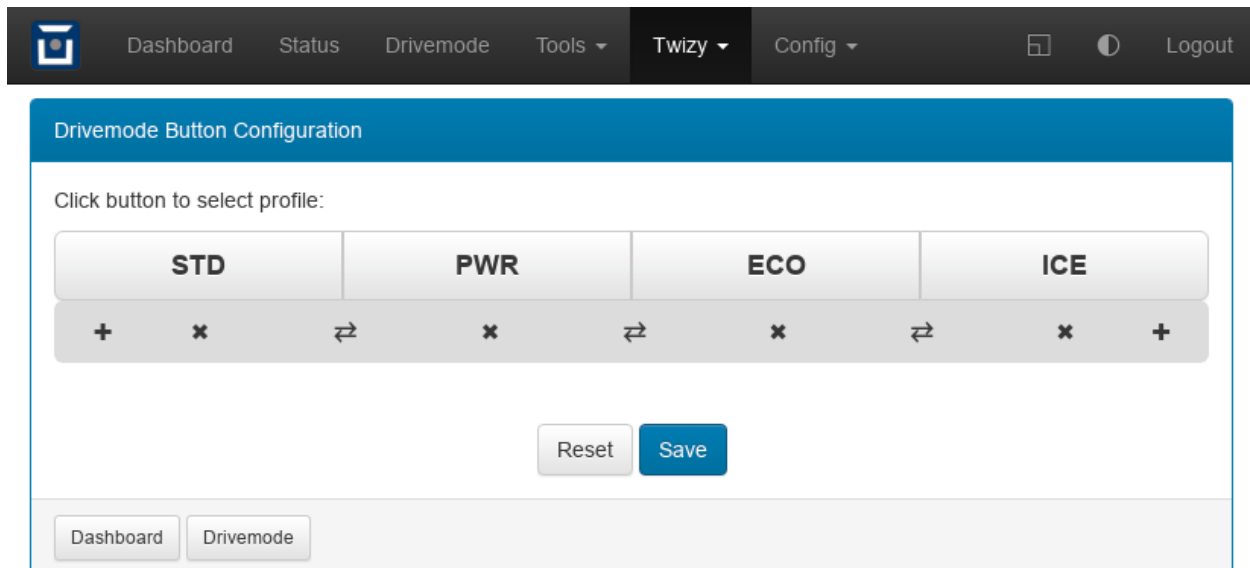
(continued from previous page)

```

667     onAction: function(key) { loadKey(key) },
668   });
669 });
670 $('.action-saveas').on('click', function(ev) {
671   $('#key-dialog').dialog({
672     show: true, title: 'Save Profile', body: '',
673     buttons: [{ label: 'Cancel', btnClass: 'default' }, { label: 'Save', btnClass:
674     ↪ 'primary' }],
675     onAction: function(key) { saveKey(key) },
676   });
677 });
678 $('.action-save').on('click', function(ev) {
679   var key = $('#input-key').val();
680   if (key === "")
681     $('.action-saveas').trigger('click');
682   else
683     saveKey(key);
684 });
685 // start: load profile / open load dialog:
686 if (page.params["key"] !== undefined)
687   loadKey(page.params["key"]);
688 else
689   $('.action-load').trigger('click');
690
691 }) ();
692 </script>

```

### 31.1.17 Twizy: Drivemode Button Editor



This plugin has been added to the Twizy code. It's used here as a more complex example of what can be done by plugins.

It's an editor for the drivemode buttons the Twizy adds to the dashboard for quick tuning profile changes. The editor

allows to change the layout (number and order of buttons) and the profiles to use.

It includes a profile selector built with the dialog widget, that retrieves the available profiles from the OVMS config store.

**Install:** not necessary if vehicle Twizy is configured (see “Twizy” menu). If you’d like to test this for another vehicle, add as a page plugin e.g. /test/dmconfig.

drivemode-config.htm (hint: right click, save as)

```
1 <!--
2   Twizy page plugin: Drivemode Button Configuration
3   Note: included in firmware v3.2
4 -->
5
6 <style>
7 .fullscreened .panel-single .panel-body {
8   padding: 10px;
9 }
10
11 .btn-group-lg>.btn,
12 .btn-lg {
13   padding: 10px 3px;
14   overflow: hidden;
15 }
16 #loadmenu .btn {
17   font-weight: bold;
18 }
19 .btn-group.btn-group-lg.exchange {
20   width: 2%;
21 }
22 #editor .btn:hover {
23   background-color: #e6e6e6;
24 }
25 #editor .btn-group-lg>.btn,
26 #editor .btn-lg {
27   padding: 10px 0px;
28   font-size: 15px;
29 }
30 #editor .exchange .btn {
31   font-weight: bold;
32 }
33 .night #editor .btn {
34   color: #000;
35   background: #888;
36 }
37 .night #editor .btn.focus,
38 .night #editor .btn:focus,
39 .night #editor .btn:hover {
40   background-color: #e0e0e0;
41 }
42
43 .radio-list {
44   height: 313px;
45   overflow-y: auto;
46   overflow-x: hidden;
47   padding-right: 15px;
48 }
49 .radio-list .radio {
```

(continues on next page)

(continued from previous page)

```

50     overflow: hidden;
51 }
52 .radio-list .key {
53     min-width: 20px;
54     display: inline-block;
55     text-align: center;
56     margin: 0 10px;
57 }
58 .radio-list kbd {
59     min-width: 60px;
60     display: inline-block;
61     text-align: center;
62     margin: 0 20px 0 10px;
63 }
64 .radio-list .radio label {
65     width: 100%;
66     text-align: left;
67     padding: 8px 30px;
68 }
69 .radio-list .radio label.active {
70     background-color: #337ab7;
71     color: #fff;
72     outline: none;
73 }
74 .radio-list .radio label.active input {
75     outline: none;
76 }
77 .night .radio-list .radio label:hover {
78     color: #fff;
79 }
80 </style>
81
82 <div class="panel panel-primary">
83     <div class="panel-heading">Drivemode Button Configuration</div>
84     <div class="panel-body">
85         <form action="#">
86
87             <p id="info">Loading button configuration...</p>
88
89             <div id="loadmenu" class="btn-group btn-group-justified"></div>
90             <div id="editor" class="btn-group btn-group-justified">
91                 <div class="btn-group btn-group-lg add back">
92                     <button type="button" class="btn" title="Add button"></button>
93                 </div>
94             </div>
95
96             <br>
97             <br>
98             <div class="text-center">
99                 <button type="button" class="btn btn-default" onclick="reloadpage()">Reset</
100 button>
101                 <button type="button" class="btn btn-primary action-save">Save</button>
102             </div>
103
104             </form>
105         </div>
106     <div class="panel-footer">

```

(continues on next page)

(continued from previous page)

```

106     <a class="btn btn-sm btn-default" target="#main" href="/dashboard">Dashboard</a>
107     <a class="btn btn-sm btn-default" target="#main" href="/xrt/drivemode">Drivemode</
↪a>
108     </div>
109 </div>
110
111 <div id="key-dialog" />
112
113 <script>
114 (function() {
115
116     var $menu = $('#loadmenu'), $edit = $('#editor'), $back = $edit.find('.back');
117
118     var pbuttons = [0,1,2,3];
119     var plist = [
120         { label: "STD", title: "Standard" },
121         { label: "PWR", title: "Power" },
122         { label: "ECO", title: "Economy" },
123         { label: "ICE", title: "Winter" },
124     ];
125
126     // load profile list & button config:
127     $('.panel').addClass("loading");
128     var plistloader = loadcmd('config list xrt').then(function(data) {
129         var lines = data.split('\n'), line, i, key;
130         for (i = 0; i < lines.length; i++) {
131             line = lines[i].match(/profile([0-9]{2})\.(?([^\:]*): (.*)/);
132             if (line && line.length == 4) {
133                 key = Number(line[1]);
134                 if (key < 1 || key > 99) continue;
135                 if (!plist[key]) plist[key] = {};
136                 plist[key][line[2]||"profile"] = line[3];
137                 continue;
138             }
139             line = lines[i].match(/profile_buttons: (.*)/);
140             if (line && line.length == 2) {
141                 try {
142                     pbuttons = JSON.parse "[" + line[1] + "]");
143                 } catch (e) {
144                     console.error(e);
145                 }
146             }
147         }
148     });
149
150     // prep key dialog:
151     $('#key-dialog').dialog({
152         show: false,
153         title: "Select Profile",
154         buttons: [{ label: 'Cancel', btnClass: 'default' }, { label: 'Select', btnClass:
↪'primary' }],
155         onShow: function(input) {
156             var $this = $(this), dlg = $this.data("dialog");
157             $this.addClass("loading");
158             plistloader.then(function(data) {
159                 var curkey = dlg.options.key || 0, i, label, title;
160                 $plist = $('<div class="radio-list" data-toggle="buttons" />');

```

(continues on next page)

(continued from previous page)

```

161     for (i = 0; i <= Math.min(99, plist.length-1); i++) {
162         if (plist[i] && (i==0 || plist[i].profile)) {
163             label = plist[i].label || "-";
164             title = plist[i].title || (plist[i].profile.substr(0, 10) + "...");
165         } else {
166             label = "-";
167             title = "-new-";
168         }
169         $plist.append('<div class="radio"><label class="btn">' +
170             '<input type="radio" name="key" value="' + i + '"><span class="key">' +
171             ((i==0) ? "STD" : ("#" + ((i<10)?'0':') + i)) + '</span> <kbd>' +
172             encode_html(label) + '</kbd>' + encode_html(title) + '</label></div>');
173     }
174     $this.find('.modal-body').html($plist).find('input[value="'+curkey+'"]')
175     .prop("checked", true).parent().addClass("active");
176     $plist
177     .on('dblclick', 'label.btn', function(ev) { $this.dialog('triggerButton',
↵1); });
178     .on('keypress', function(ev) { if (ev.which==13) $this.dialog('triggerButton
↵', 1); });
179     $this.removeClass("loading");
180     });
181 },
182 onShown: function(input) {
183     $(this).find('.btn.active').focus();
184 },
185 onHide: function(input) {
186     var $this = $(this), dlg = $this.data("dialog");
187     var key = $this.find('input[name="key"]:checked').val();
188     if (key !== undefined && input.button && input.button.index)
189         dlg.options.onAction.call(this, key);
190 },
191 });
192
193 // profile selection:
194 $('#loadmenu').on('click', 'button', function(ev) {
195     var $this = $(this);
196     $('#key-dialog').dialog({
197         show: true,
198         key: $this.val(),
199         onAction: function(key) {
200             var prof = plist[key] || {};
201             $this.val(key);
202             $this.attr("title", prof.title || "");
203             $this.text(prof.label || ("#" + ((key<10)?'0':') + key));
204         },
205     });
206 });
207
208 // create buttons:
209
210 function addButton(key, front) {
211     var prof = plist[key] || {};
212     if ($menu[0].childElementCount == 0) {
213         $back.before('\
214             <div class="btn-group btn-group-lg add front">\
215                 <button type="button" class="btn" title="Add button"></button>\

```

(continues on next page)

(continued from previous page)

```

216     </div>\
217     <div class="btn-group btn-group-lg remove">\
218         <button type="button" class="btn" title="Remove button"></button>\
219     </div>');
220 } else {
221     $back.before('\
222     <div class="btn-group btn-group-lg exchange">\
223         <button type="button" class="btn" title="Exchange buttons"></button>\
224     </div>\
225     <div class="btn-group btn-group-lg remove">\
226         <button type="button" class="btn" title="Remove button"></button>\
227     </div>');
228 }
229 var $btn = $('\'
230     <div class="btn-group btn-group-lg">\
231         <button type="button" value="{key}" class="btn btn-default" title="{title}">
↪{label}</button>\
232     </div>'
233     .replace("{key}", key)
234     .replace("{title}", encode_html(prof.title || ""))
235     .replace("{label}", encode_html(prof.label || "#" + ((key < 10) ? "0:" + key)));
236 if (front)
237     $menu.prepend($btn);
238 else
239     $menu.append($btn);
240 }
241
242 plistloader.then(function() {
243     var key, prof;
244     for (var i = 0; i < pbuttons.length; i++) {
245         addButton(pbuttons[i]);
246     }
247     $('.panel').removeClass("loading");
248     $('#info').text("Click button to select profile:");
249 });
250
251 // editor buttons:
252 $('#editor').on('click', '.add .btn', function(ev) {
253     addButton(0, $(this).parent().hasClass("front"));
254 }).on('click', '.remove .btn', function(ev) {
255     var $this = $(this), pos = $edit.find('.btn').index(this) >> 1;
256     $($menu.children().get(pos)).detach();
257     if (pos > 0 || $menu[0].childElementCount == 0)
258         $this.parent().prev().detach();
259     else if ($menu[0].childElementCount != 0)
260         $this.parent().next().detach();
261     $this.parent().detach();
262 }).on('click', '.exchange .btn', function(ev) {
263     var pos = $edit.find('.btn').index(this) >> 1;
264     if (pos > 0) $($menu.children().get(pos)).insertBefore($($menu.children().get(pos-
↪1)));
265 });
266
267 // save:
268 $('.action-save').on('click', function(ev) {
269     pbuttons = [];
270     $menu.find('.btn').each(function() { pbuttons.push(this.value) });

```

(continues on next page)

(continued from previous page)

```

271     loadcmd('config set xrt profile_buttons ' + pbuttons.toString())
272     .fail(function(request, textStatus, errorThrown) {
273         confirmDialog("Save Configuration", xhrErrorInfo(request, textStatus,
↪errorThrown), ["Close"]);
274     })
275     .done(function(res) {
276         confirmDialog("Save Configuration", res, ["Close"]);
277     });
278 });
279
280 }) ();
281 </script>

```

## 31.2 Installing Plugins

The framework supports installing user content as pages or extensions to pages. To install an example:

1. Menu Config → Web Plugins
2. Add plugin: type “Page”, name e.g. “dev.metrics” (used as the file name in /store/plugin)
3. Save → Edit
4. Set page to e.g. “/usr/dev/metrics”, label to e.g. “Dev: Metrics”
5. Set the menu to e.g. “Tools” and auth to e.g. “None”
6. Paste the page source into the content area
7. Save → the page is now accessible in the “Tools” menu.

Hint: use the standard editor (tools menu) in a second session to edit a plugin repeatedly during test / development.

## 31.3 Basics

The OVMS web framework is based on HTML 5, Bootstrap 3 and jQuery 3. Plenty documentation and guides on these basic web technologies is available on the web, a very good resource is [w3schools.com](http://w3schools.com).

For charts the framework includes Highcharts 6. Info and documentation on this is available on [Highcharts.com](http://Highcharts.com).

The framework is “AJAX” based. The index page / loads the framework assets and defines a default container structure including a #menu and a #main container. Content pages are loaded into the #main container. The window URL includes the page URL after the hash mark #:

- `http://ovms.local/#/status` – this loads page /status into #main
- `http://ovms.local/#/dashboard?nm=1` – this loads the dashboard and activates night mode

Links and forms having id targets # . . . are automatically converted to AJAX by the framework:

- `<a href="/edit?path=/sd/index.txt" target="#main">Edit index.txt</a>` – load the editor

Pages can be loaded outside the framework as well (e.g. `http://ovms.local/status`). See index source on framework scripts and styles to include if you’d like to design standalone pages using the framework methods.

If file system access is enabled, all URLs not handled by the system or a user plugin (see below) are mapped onto the file system under the configured web root. Of course, files can be loaded into the framework as well. For example, if the web root is /sd (default):

- `http://ovms.local/#/mypage.htm` – load file `/sd/mypage.htm` into `#main`
- `http://test1.local/#/logs` – load directory listing `/sd/logs` into `#main`

**Important Note:** the framework has a global shared context (i.e. the `window` object). To avoid polluting the global context with local variables, good practice is to wrap your local scripts into closures. Pattern:

```
<script>
(function(){
  ... insert your code here ...
})();
</script>
```

## 31.4 Authorization

Page access can be restricted to authorized users either session based or per access. File access can be restricted using digest authentication.

The module password is used for all authorizations. A user account or API key administration is not yet included, the main username is `admin`.

To create a session, call the `/login` page and store the resulting cookie:

1. `curl -c auth -b auth 'http://192.168.4.1/login' -d username=admin -d password=...`
2. `curl -c auth -b auth 'http://192.168.4.1/api/execute?command=xrt+cfg+info'`

To issue a single call, e.g. to execute a command from a Wifi button, supply the password as `apikey`:

- `curl 'http://192.168.4.1/api/execute?apikey=password&command=xrt+cfg+info'`
- `curl 'http://192.168.4.1/api/execute?apikey=password&type=js&command=print(Duktape.version)'`

## 31.5 Web UI Development Framework

To simplify and speed up UI and plugin development, you can simply run a local web server from the `dev` directory of the `ovms_webserver` component source.

Preparation: create your local git clone if you don't have one already:

```
> cd ~
> git clone https://github.com/openvehicles/Open-Vehicle-Monitoring-System-3.git
```

Local web server options: [List of static web servers](#)

Example:

```
> cd ~/Open-Vehicle-Monitoring-System-3/vehicle/OVMS.V3/components/ovms_webserver/dev
> python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```



Now open the development framework in your browser at URI `http://localhost:8000`. If that doesn't work, check your firewall settings for port 8000 on localhost.

You should see the examples home. Edit `index.htm` to add custom menus, edit `home.htm` to add custom home buttons.

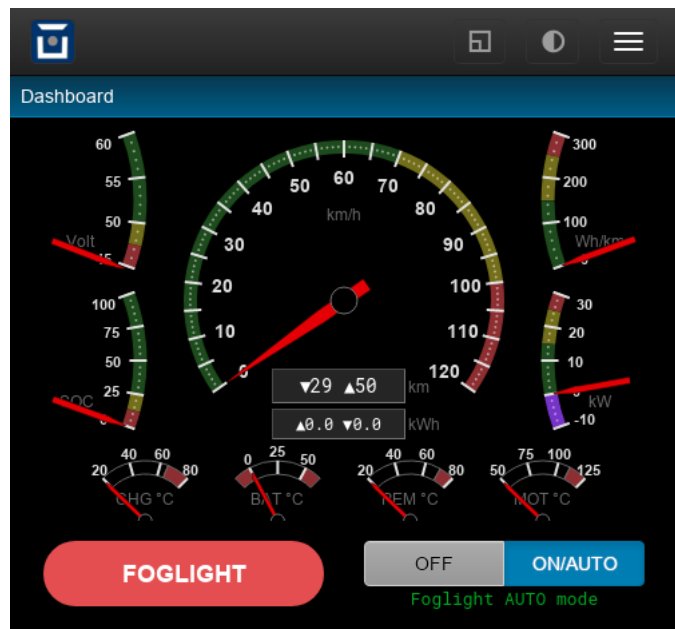
Hint: to test your plugin for mobile devices, use the mobile emulation mode of your browser's development toolkit. Mobile mode allows to test small screen resolutions, rotation and touch events.

A static local HTTP server allows to use all frontend features, but cannot emulate the backend API (command/script execution). If using a CGI capable HTTP server, you can also add a proxy handler for `/api/execute` that forwards the requests to your module by HTTP or SSH.

If you want to add your results to a C++ module, use the tools `mksrc` and `mksrcf` to convert your HTML files into C++ compatible strings. `mksrc` is meant for static strings, `mksrcf` for strings with `printf` style placeholders.



## 32.1 Example: Foglight



This is an example for combining a script module implementing some functional extension with a web plugin to provide a user interface.

The plugin realizes a fog light with automatic speed adaption, i.e. the light will be turned off above a configurable speed and on again when the vehicle slows down again. It also switches off automatically when the vehicle is parked.

The plugin shows how to...

- read custom config variables
- use module state variables

- react to events
- send custom events
- read metrics
- execute system commands
- provide new commands
- provide a web UI

### 32.1.1 Installation

- Save `foglight.js` as `/store/scripts/lib/foglight.js` (add the `lib` directory as necessary)
- Add `foglight.htm` as a web hook type plugin at page `/dashboard`, hook `body.pre`
- Execute script `eval 'foglight = require("lib/foglight")'`

Optionally:

- To test the module, execute script `eval foglight.info()` – it should print config and state
- To automatically load the module on boot, add the line `foglight = require("lib/foglight");` to `/store/scripts/ovmsmain.js`

### 32.1.2 Commands / Functions

The module provides two public API functions:

Function	Description
<code>foglight.set(onoff)</code>	...switch fog light on (1) / off (0)
<code>foglight.info()</code>	...output config & state in JSON format

To call these from a shell, use `script eval`. Example:

- `script eval foglight.set(1)`

### 32.1.3 Configuration

We'll add the configuration for this to the `vehicle` section:

Config	Default	Description
<code>foglight.port</code>	1	...EGPIO output port number
<code>foglight.auto</code>	no	...yes = speed automation
<code>foglight.speed.on</code>	45	...auto turn on below this speed
<code>foglight.speed.off</code>	55	...auto turn off above this speed

---

**Note:** You can add arbitrary config instances to defined sections simply by setting them: `config set vehicle foglight.auto yes`

*Update:* Beginning with firmware release 3.2.009, a general configuration section `usr` is provided for plugins. We recommend using this for all custom parameters now. Keep in mind to prefix all instances introduced by the plugin name, so your plugin can nicely coexist with others.

To store the config for simple & quick script access and implement the defaults, we introduce an **internal module member object** `cfg`:

Listing 1: Module Plugin

```

28 var cfg = {
29     "foglight.port":      "1",
30     "foglight.auto":      "no",
31     "foglight.speed.on":  "45",
32     "foglight.speed.off": "55",
33 };

```

By `foglight = require(...)`, the module is added to the global name space as a javascript object. This object can contain any internal standard javascript variables and functions. Internal members are hidden by default, if you would like to expose the `cfg` object, you would simply add a reference to it to the `exports` object as is done below for the API methods.

**Reading OVMS config variables from a script** currently needs to be done by executing `config list` and parsing the output. This is done by the `readconfig()` function:

```

42 function readconfig() {
43     var cmdres, lines, cols, i;
44     cmdres = OvmsCommand.Exec("config list vehicle");
45     lines = cmdres.split("\n");
46     for (i=0; i<lines.length; i++) {
47         if (lines[i].indexOf("foglight") >= 0) {
48             cols = lines[i].substr(2).split(": ");
49             cfg[cols[0]] = cols[1];
50         }
51     }

```

**Update:** OVMS release 3.2.009 adds the `OvmsConfig.GetValues()` API. To use this, we would now omit the “foglight.” prefix from our `cfg` properties. Reading the foglight configuration can then be reduced to a single line:

```
Object.assign(cfg, OvmsConfig.GetValues("vehicle", "foglight."));
```

### 32.1.4 Listen to Events

The module needs to listen to three events:

- `config.changed` triggers reloading the configuration
- `ticker.1` is used to check the speed once per second
- `vehicle.off` automatically also turns off the fog light

The per second ticker is only necessary when the speed adaption is enabled, so we can use this to show how to dynamically add and remove event handlers through the PubSub API:

Listing 2: Module Plugin

```

52 // update ticker subscription:
53 if (cfg["foglight.auto"] == "yes" && !state.ticker) {
54     state.ticker = PubSub.subscribe("ticker.1", checkspeed);
55 } else if (cfg["foglight.auto"] != "yes" && state.ticker) {
56     PubSub.unsubscribe(state.ticker);
57     state.ticker = false;
58 }

```

state is another internal object for our state variables.

### 32.1.5 Send Events

Sending custom events is a lightweight method to inform the web UI (or other plugins) about simple state changes. In this case we'd like to inform listeners when the fog light actually physically is switched on or off, so the web UI can give visual feedback to the driver on this.

Beginning with firmware release 3.2.006 there is a native API `OvmsEvents.Raise()` available to send events.

Before 3.2.006 we simply use the standard command event `raise`:

Listing 3: Module Plugin

```

61 // EGPIO port control:
62 function toggle(onoff) {
63     if (state.port != onoff) {
64         OvmsCommand.Exec("egpio output " + cfg["foglight.port"] + " " + onoff);
65         state.port = onoff;
66         OvmsCommand.Exec("event raise usr.foglight." + (onoff ? "on" : "off"));
67     }
68 }

```

The web plugin subscribes to the foglight events just as to any system event:

Listing 4: Web Plugin

```

64 // Listen to foglight events:
65 $('#foglight').on('msg:event', function(e, event) {
66     if (event == "usr.foglight.on")
67         update({ state: { port: 1 } });
68     else if (event == "usr.foglight.off")
69         update({ state: { port: 0 } });
70     else if (event == "vehicle.off") {
71         update({ state: { on: 0 } });
72         $('#action-foglight-output').empty();
73     }
74 });

```

**Note:** You can raise any event you like, but you shouldn't raise system events without good knowledge of their effects. Event codes are simply strings, so you're free to extend them. Use the prefix `usr.` for custom events to avoid potential conflicts with future system event additions.

### 32.1.6 Read Metrics

Reading metrics is straight forward through the `OvmsMetrics` API:

Listing 5: Module Plugin

```
74 var speed = OvmsMetrics.AsFloat("v.p.speed");
```

Use `Value()` instead of `AsFloat()` for non-numerical metrics.

**Note:** You cannot subscribe to metrics changes directly (yet). Metrics can change thousands of times per second, which would overload the scripting capabilities. To periodically check a metric, register for a ticker event (as shown here).

### 32.1.7 Provide Commands

To add commands, simply expose their handler functions through the `exports` object. By this, users will be able to call these functions using the `script eval` command from any shell, or from any script by referencing them via the global module variable, `foglight` in our case.

Listing 6: Module Plugin

```
99 // API method foglight.info():
100 exports.info = function() {
101   JSON.print({ "cfg": cfg, "state": state });
102 }
```

`JSON.print()` is a convenient way to communicate with a web plugin, as that won't need to parse some potentially ambiguous textual output but can simply use `JSON.parse()` to read it into a variable:

Listing 7: Web Plugin

```
82 // Init & install:
83 $('#main').one('load', function(ev) {
84   loadcmd('script eval foglight.info()').then(function(output) {
85     update(JSON.parse(output));
```

**Note:** Keep in mind commands should always output some textual response indicating their action and result. If a command does nothing, it should tell the user so. If a command is not intended for shell use, it should still provide some clue about this when called from the shell.

### 32.1.8 Module Plugin

`foglight.js` (hint: right click, save as)

```
1 /**
2  * /store/scripts/lib/foglight.js
3  *
4  * Module plugin:
5  * Foglight control with speed adaption and auto off on vehicle off.
```

(continues on next page)

(continued from previous page)

```

6  *
7  * Version 1.0   Michael Balzer <dexter@dexters-web.de>
8  *
9  * Enable:
10 *   - install at above path
11 *   - add to /store/scripts/ovmsmain.js:
12 *       foglight = require("lib/foglight");
13 *   - script reload
14 *
15 * Config:
16 *   - vehicle foglight.port           ...EGPIO output port number
17 *   - vehicle foglight.auto           ...yes = speed automation
18 *   - vehicle foglight.speed.on       ...auto turn on below this speed
19 *   - vehicle foglight.speed.off      ...auto turn off above this speed
20 *
21 * Usage:
22 *   - script eval foglight.set(1)     ...toggle foglight on
23 *   - script eval foglight.set(0)     ...toggle foglight off
24 *   - script eval foglight.info()     ...show config & state (JSON)
25 *
26 */
27
28 var cfg = {
29   "foglight.port":      "1",
30   "foglight.auto":      "no",
31   "foglight.speed.on":  "45",
32   "foglight.speed.off": "55",
33 };
34
35 var state = {
36   on: 0,           // foglight on/off
37   port: 0,         // current port output state
38   ticker: false,   // ticker subscription
39 };
40
41 // Read config:
42 function readconfig() {
43   var cmdres, lines, cols, i;
44   cmdres = OvmsCommand.Exec("config list vehicle");
45   lines = cmdres.split("\n");
46   for (i=0; i<lines.length; i++) {
47     if (lines[i].indexOf("foglight") >= 0) {
48       cols = lines[i].substr(2).split(": ");
49       cfg[cols[0]] = cols[1];
50     }
51   }
52   // update ticker subscription:
53   if (cfg["foglight.auto"] == "yes" && !state.ticker) {
54     state.ticker = PubSub.subscribe("ticker.1", checkspeed);
55   } else if (cfg["foglight.auto"] != "yes" && state.ticker) {
56     PubSub.unsubscribe(state.ticker);
57     state.ticker = false;
58   }
59 }
60
61 // EGPIO port control:
62 function toggle(onoff) {

```

(continues on next page)



(continued from previous page)

```

63   if (state.port != onoff) {
64       OvmsCommand.Exec("egpio output " + cfg["foglight.port"] + " " + onoff);
65       state.port = onoff;
66       OvmsCommand.Exec("event raise usr.foglight." + (onoff ? "on" : "off"));
67   }
68 }
69
70 // Check speed:
71 function checkspeed() {
72     if (!state.on)
73         return;
74     var speed = OvmsMetrics.AsFloat("v.p.speed");
75     if (speed <= cfg["foglight.speed.on"])
76         toggle(1);
77     else if (speed >= cfg["foglight.speed.off"])
78         toggle(0);
79 }
80
81 // API method foglight.set(onoff):
82 exports.set = function(onoff) {
83     if (onoff) {
84         state.on = 1;
85         if (cfg["foglight.auto"] == "yes") {
86             checkspeed();
87             print("Foglight AUTO mode\n");
88         } else {
89             toggle(1);
90             print("Foglight ON\n");
91         }
92     } else {
93         state.on = 0;
94         toggle(0);
95         print("Foglight OFF\n");
96     }
97 }
98
99 // API method foglight.info():
100 exports.info = function() {
101     JSON.print({ "cfg": cfg, "state": state });
102 }
103
104 // Init:
105 readconfig();
106 PubSub.subscribe("config.changed", readconfig);
107 PubSub.subscribe("vehicle.off", function(){ exports.set(0); });

```

### 32.1.9 Web Plugin

foglight.htm (hint: right click, save as)

```

1  <!--
2  foglight.htm: Web plugin for hook /dashboard:body.pre
3  - add button to activate/deactivate foglight
4  - add indicator to show current foglight state
5

```

(continues on next page)

(continued from previous page)

```

6   Requires module plugin: foglight.js
7
8   Version 1.0 Michael Balzer <dexter@dexters-web.de>
9   -->
10
11  <style>
12  #foglight {
13    margin: 10px 8px 0;
14  }
15  #foglight .indicator > .label {
16    font-size: 130%;
17    line-height: 160%;
18    margin: 0px;
19    padding: 10px;
20    display: block;
21    border-radius: 50px;
22  }
23  </style>
24
25  <div class="receiver" id="foglight" style="display:none">
26    <form>
27      <div class="form-group">
28        <div class="col-xs-6">
29          <div class="indicator indicator-foglight">
30            <span class="label label-default">FOGLIGHT</span>
31          </div>
32        </div>
33        <div class="col-xs-6">
34          <div class="btn-group btn-group-justified action-foglight-set" data-toggle=
35      ↪ "buttons">
36            <label class="btn btn-default action-foglight-0"><input type="radio" name=
37      ↪ "foglight" value="0">OFF</label>
38            <label class="btn btn-default action-foglight-1"><input type="radio" name=
39      ↪ "foglight" value="1">ON/AUTO</label>
40          </div>
41          <samp id="action-foglight-output" class="text-center"></samp>
42        </div>
43      </form>
44    </div>
45
46  <script>
47  (function() {
48
49    var foglight = { cfg: {}, state: { on: 0, port: 0 } };
50    var $indicator = $('#foglight .indicator-foglight > .label');
51    var $actionset = $('#foglight .action-foglight-set > label');
52
53    // State & UI update:
54    function update(data) {
55      $.extend(true, foglight, data);
56      // update indicator:
57      if (foglight.state.port)
58        $indicator.removeClass('label-default').addClass('label-danger');
59      else
60        $indicator.removeClass('label-danger').addClass('label-default');
61      // update buttons:

```

(continues on next page)

(continued from previous page)

```

60     $actionset.removeClass('active');
61     $actionset.find('input[value='+foglight.state.on+']').prop('checked', true).
↪parent().addClass('active');
62 }
63
64 // Listen to foglight events:
65 $('#foglight').on('msg:event', function(e, event) {
66     if (event == "usr.foglight.on")
67         update({ state: { port: 1 } });
68     else if (event == "usr.foglight.off")
69         update({ state: { port: 0 } });
70     else if (event == "vehicle.off") {
71         update({ state: { on: 0 } });
72         $('#action-foglight-output').empty();
73     }
74 });
75
76 // Button action:
77 $('#foglight .action-foglight-set input').on('change', function(e) {
78     foglight.state.on = $(this).val();
79     loadcmd('script eval foglight.set('+foglight.state.on+')', '#action-foglight-
↪output');
80 });
81
82 // Init & install:
83 $('#main').one('load', function(ev) {
84     loadcmd('script eval foglight.info()').then(function(output) {
85         update(JSON.parse(output));
86         $('#foglight').appendTo('#panel-dashboard .panel-body').show();
87     });
88 });
89
90 }) ();
91 </script>

```



## 33.1 CANopen Basics

### 33.1.1 Communication Objects

A CANopen network consists of “masters” and “slaves”, masters are clients, slaves are servers. At most one master may be active at a time.

CANopen supports addressing of up to 127 slaves on a bus using node IDs 1-127. Node address 0 is used for broadcasts to all nodes. Node addressing is simply mapped onto CAN IDs by adding the node id to a base ID.

The CANopen protocol mainly consists of...

- PDO (process data objects)
- SDO (service data objects)
- NMT (network management)
- SYNC (synchronisation)
- EMCY (emergency events)

**PDO** are regular, normally periodical, status update frames, for example sensor data. You can log them using the CAN monitor (`can log start monitor ...`). PDOs can be sent at any CAN ID except those reserved for other CANopen services.

**SDO** are memory registers of nodes that can be read and written by masters on request. SDO requests are normally sent at ID 0x600 + nodeid, responses at ID 0x580 + nodeid. SDOs are addressed by a 16 bit index + 8 bit subindex. Registers and data types for a given device are documented by the device specific object dictionary, normally represented as an EDS (electronic data sheet) file.

**NMT** are short datagrams to control node startup / shutdown. There's a special node state “pre-operational” allowing access to all operation and communication parameters of a node in a standardized way. NMT requests are sent at ID 0x000, responses and unsolicited updates (aka heartbeats) are sent at ID 0x700 + nodeid with length 1.

**SYNC** messages are datagrams of length 0, normally sent at ID 0x080.

**EMCY** messages are sent if a node encounters some kind of alert or warning condition, they are normally sent at ID 0x080 + nodeid with a length of 8 bytes.

CAN IDs	Communication objects
0x000	NMT requests
0x080	SYNC
0x081 - 0x0FF	EMCY
0x581 - 0x5FF	SDO responses
0x601 - 0x67F	SDO requests
0x701 - 0x77F	NMT responses / heartbeats

So if any of these IDs look familiar to you, chances are you've got a CANopen network.

---

**Note:** CANopen coexists nicely with OBD-II and often does in a vehicle (i.e. Renault Twizy). OBD-II devices normally are addressed at IDs > 0x780 so are outside the CANopen ID range. Even if they use non-standard IDs, the devices normally will detect and ignore frames not matching their protocol.

---

### 33.1.2 SDO Addresses

The SDO address space layout is standardized:

Index (hex)	Object
0000	not used
0001-001F	Static Data Types
0020-003F	Complex Data Types
0040-005F	Manufacturer Specific Complex Data Types
0060-007F	Device Profile Specific Static Data Types
0080-009F	Device Profile Specific Complex Data Types
00A0-0FFF	Reserved for further use
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardised Device Profile Area
A000-BFFF	Standardised Interface Profile Area
C000-FFFF	Reserved for further use

See [CiA DS301](#) for details on standard SDOs.

For example, a device shall tell about the PDOs it transmits or listens to, their IDs, update frequency and content structure at SDO registers 1400-1BFF. This is mandatory in theory but real devices may not fully comply to that rule.

CANopen compliant standard device types like motor controllers need to implement some standard profile registers. See [CiA DS401](#) for the generic I/O device class definition and [CiA DS402](#) for motor controllers.

Most devices will require some kind of login before allowing you to change operational parameters. This is also done using SDO writes, but there is no standard for this, so you'll need to dig into the device documentation.

Of course there's a lot more on CANopen, but this should get you going.

### 33.1.3 CAN in Automation

More info on the standard and specific device profiles can be found on the CiA website:

<https://www.can-cia.org/>

CAN in Automation (CiA) is the international users' and manufacturers' group for the CAN network (Controller Area Network), internationally standardized in the ISO 11898 series. The nonprofit association was established in 1992. The aim is to provide an unbiased platform for future developments of the CAN protocol and to promote the image of the CAN technology.

## 33.2 How to detect and identify CANopen nodes

So you've got a CAN bus with some devices, but you don't know which of them speaks "CANopen", if any? The OVMS v3 will help you to detect them and open their CANs ;)

### 33.2.1 Before you begin

...you need to activate the CAN bus(es) you're going to use. As a CANopen master needs to write to the network, you need to start the interfaces in active mode, i.e. ...:

```
OVMS# can can1 start active 500000
OVMS# can can2 start active 125000
```

... and then start the CANopen master for the bus(es), i.e.:

```
OVMS# copen can1 start
OVMS# copen can2 start
```

### 33.2.2 Detecting CANopen nodes

The "open" in "CANopen" means any implementation can decide how much of the standard it implements. There are some few mandatory features though, a CANopen slave has to implement, if it wants to comply with the standard.

The mandatory features helping to detect and identify CANopen nodes on a CAN bus are:

- NMT (network management), especially RESET and PREOP
- NMT bootup event messages
- Standard SDO access in pre-operational mode

If you've got CANopen nodes on a bus, even silent ones, issuing `copen ... nmt reset` will tell all of them to reboot, and as bootup messages are mandatory, you will see them in the OVMS log output like this:

```
I (162904) canopen: can1 node 1 new state: Booting
```

The OVMS CANopen master continuously monitors the network for NMT and EMCY messages. After bootup of all nodes, you can get a list of all nodes that have been detected by issuing `metrics list co.:`

```
OVMS# metrics list co.
co.can1.nd1.emcy.code
co.can1.nd1.emcy.type
co.can1.nd1.state           Operational
```

**Note:** if you request a reset, nodes may decide to boot into pre-operational state. That may produce some error messages. Don't worry, you can resolve this anytime by issuing `copen ... nmt start`.

### 33.2.3 Identifying CANopen nodes

In pre-operational state, a CANopen node must be accessible at the CANopen default IDs. That means if the node supports SDO access, we can query some standard attributes from it.

That's what `copen ... info` and `copen ... scan` do:

- `copen ... info` queries the standard device attributes from a specific node,
- `copen ... scan` queries all 127 node IDs.

**Caution:** There may be non-CANopen devices on the bus producing regular data frames at CANopen response IDs and/or reading and possibly misinterpreting CANopen requests sent to node IDs not planned by the manufacturer. Chances are low this triggers problems, but you should be ready to switch off the vehicle when doing a scan – just in case.

A complete scan takes about 20 seconds. A typical scan could look like this:

```
OVMS# level debug canopen
OVMS# copen can1 scan
Scan #1-127...
Node #1:
  Device type: 0x00000000
  Error state: 0x00
  Vendor ID: 0x0000001e
  Product: 0x0712302d
  Revision: 0x00010019
  S/N: 0x47c5.....
  Device name: Gen4 (Renault Twizy)11 November 2011 12
  HW version: 0x00000003
  SW version: 0712.0001
Node #23: SDO access failed
Node #25: SDO access failed
Node #27: SDO access failed
Node #30: SDO access failed
Node #87: SDO access failed
Done.
D (227994) canopen: ReadSDO #23 0x1000.00: InitUpload failed, CANopen error code_
↪0xffffffff
D (228194) canopen: ReadSDO #25 0x1000.00: InitUpload failed, CANopen error code_
↪0xffffffff
D (228444) canopen: ReadSDO #27 0x1000.00: InitUpload failed, CANopen error code_
↪0xffffffff
D (228844) canopen: ReadSDO #30 0x1000.00: InitUpload failed, CANopen error code_
↪0xffffffff
D (238384) canopen: ReadSDO #87 0x1000.00: InitUpload failed, CANopen error code_
↪0xffffffff
```

This means one CANopen node was found, and some non-CANopen frames were received on IDs 0x580 +23, +25, +27, +30 and +87.

### 33.2.4 Great! What now?

As you now know there's a CANopen node, you can look for documentation on it. You can also try to access more default SDOs to see if you can control and configure the node.



- If you're lucky, the device will provide its own EDS file at SDO 1021.00. You can check that by issuing...:

```
OVMS# copen <bus> readsdo <nodeid> 1021 0
```

- Check out the [CiA specifications](#), for CANopen standards, for example...
  - DS301 for details on general standard SDOs
  - DS401 for the generic I/O device class definition
  - DS402 for motor controller SDOs
- Look up the manufacturer of your device by its vendor ID:
   
<https://www.can-cia.org/services/canopen-vendor-id/>
- Contact the manufacturer of your device for specific documentation and EDS files.

## 33.3 CANopen Shell Commands

### Show status report

```
copen status
```

### Start / stop workers

```
copen <bus> start
copen <bus> stop
```

- stop will fail if clients are still running.

### Send NMT commands

```
copen <bus> nmt <start|stop|preop|reset|commreset> [id=0] [timeout_ms=0]
```

- id 0 = broadcast
- timeout > 0: wait for state change (heartbeat), 3 tries Note: state change response is not a mandatory CANopen feature.

### Read SDO

```
copen <bus> readsdo <id> <index_hex> <subindex_hex> [timeout_ms=50]
```

- index & subindex: hexadecimal without "0x" or "h"
- defaults to 3 tries on timeout

### Write SDO

```
copen <bus> writesdo <id> <index_hex> <subindex_hex> <value> [timeout_ms=50]
```

- index & subindex: hexadecimal without "0x" or "h"
- value: prefix "0x" = hex, else decimal, string if no decimal
- defaults to 3 tries on timeout

### Show node core attributes

```
copen <bus> info <id> [timeout_ms=50]
```

- prints device type, error register, device name etc.
- Note: some attributes read are optional and may be empty/zero.

### Scan bus for nodes

```
copen <bus> scan [[startid=1][-][endid=127]] [timeout_ms=50]
```

- loops “info” over multiple node ids (default: all)
- Note: a full scan with default timeout takes ~20 seconds

## 33.4 CANopen API Usage

### 33.4.1 Synchronous API

The synchronous use is very simple, all you need is a `CANopenClient` instance.

On creation the client automatically connects to the active CANopen worker or starts a new worker instance if necessary.

The `CANopenClient` methods will block until the job is done (or has failed). Job results and/or error details are returned in the caller provided job.

**Caution:** Do not make synchronous calls from code that may run in a restricted context, e.g. within a metric update handler. Avoid using synchronous calls from time critical code, e.g. a CAN or event handler.

### Example

```
#include "canopen.h"

// find CAN interface:
canbus* bus = (canbus*) MyPcpApp.FindDeviceByName("can1");
// ...or simply use m_can1 if you're a vehicle subclass

// create CANopen client:
CANopenClient client(bus);

// a CANopen job holds request and response data:
CANopenJob job;

// read value from node #1 SDO 0x1008.00:
uint32_t value;
if (client.ReadSDO(&job, 1, 0x1008, 0x00, (uint8_t*)&value, sizeof(value)) == COR_OK) {
    // read result is now in value
}

// start node #2, wait for presence:
if (client.SendNMT(&job, 2, CONC_Start, true) == COR_OK) {
    // node #2 is now started
}
```

(continues on next page)

(continued from previous page)

```
// write value into node #3 SDO 0x2345.18:
if (client.WriteSDO(&job, 3, 0x2345, 0x18, (uint8_t)&value, 0) == COR_OK) {
    // value has now been written into register 0x2345.18
}
```

## Main API methods

```
/**
 * SendNMT: send NMT request and optionally wait for NMT state change
 * a.k.a. heartbeat message.
 *
 * Note: NMT responses are not a part of the CANopen NMT protocol, and
 * sending "heartbeat" NMT state updates is optional for CANopen nodes.
 * If the node sends no state info, waiting for it will result in timeout
 * even though the state has in fact changed -- there's no way to know
 * if the node doesn't tell.
 */
CANopenResult_t SendNMT(CANopenJob& job,
    uint8_t nodeid, CANopenNMTCommand_t command,
    bool wait_for_state=false, int resp_timeout_ms=1000, int max_tries=3);

/**
 * ReceiveHB: wait for next heartbeat message of a node,
 * return state received.
 *
 * Use this to read the current state or synchronize to the heartbeat.
 * Note: heartbeats are optional in CANopen.
 */
CANopenResult_t ReceiveHB(CANopenJob& job,
    uint8_t nodeid, CANopenNMTState_t* statebuf=NULL,
    int rcv_timeout_ms=1000, int max_tries=1);

/**
 * ReadSDO: read bytes from SDO server into buffer
 * - reads data into buf (up to bufsize bytes)
 * - returns data length read in job.sdo.xfersize
 * - ... and data length available in job.sdo.contsize (if known)
 * - remaining buffer space will be zeroed
 * - on result COR_ERR_BufferTooSmall, the buffer has been filled up to bufsize
 * - on abort, the CANopen error code can be retrieved from job.sdo.error
 *
 * Note: result interpretation is up to caller (check device object dictionary
 * for data types & sizes). As CANopen is little endian as ESP32, we don't
 * need to check lengths on numerical results, i.e. anything from int8_t to
 * uint32_t can simply be read into a uint32_t buffer.
 */
CANopenResult_t ReadSDO(CANopenJob& job,
    uint8_t nodeid, uint16_t index, uint8_t subindex, uint8_t* buf, size_t bufsize,
    int resp_timeout_ms=50, int max_tries=3);

/**
 * WriteSDO: write bytes from buffer into SDO server
 * - sends bufsize bytes from buf
 * - ... or 4 bytes from buf if bufsize is 0 (use for integer SDOs of unknown type)
```

(continues on next page)

(continued from previous page)

```

*   - returns data length sent in job.sdo.xfersize
*   - on abort, the CANopen error code can be retrieved from job.sdo.error
*
* Note: the caller needs to know data type & size of the SDO register (check
*       device object dictionary). As CANopen servers normally are intelligent,
*       anything from int8_t to uint32_t can simply be sent as a uint32_t with
*       bufsize=0, the server will know how to convert it.
*/
CANopenResult_t WriteSDO(CANopenJob& job,
    uint8_t nodeid, uint16_t index, uint8_t subindex, uint8_t* buf, size_t bufsize,
    int resp_timeout_ms=50, int max_tries=3);

```

If you want to create custom jobs, use the low level method `ExecuteJob()` to execute them.

### 33.4.2 Asynchronous API

The `CANopenAsyncClient` class provides the asynchronous interface and the response queue.

To use the asynchronous API you need to handle asynchronous responses, which normally means adding a dedicated task for this. A minimal handling would be to simply discard the responses (just empty the queue), if you don't need to care about the results.

#### Example

Instantiate the async client for a CAN bus and a queue size like this:

```
CANopenAsyncClient m_async(m_can1, 50);
```

Example response handler task:

```

void MyAsyncTask()
{
    CANopenJob job;
    while (true) {
        if (m_async.ReceiveDone(job, portMAX_DELAY) != COR_ERR_QueueEmpty) {
            // ...process job results...
        }
    }
}

```

Sending requests is following the same scheme as with the synchronous API. Standard result code is `COR_WAIT`, an error may occur if the queue is full.

```

if (m_async.WriteSDO(m_nodeid, index, subindex, (uint8_t*)value, 0) != COR_WAIT) {
    // ...handle error...
}

```

#### Main API methods

The API methods are similar to the synchronous methods (see above).

```

CANOpenResult_t SendNMT(uint8_t nodeid, CANOpenNMTCommand_t command,
    bool wait_for_state=false, int resp_timeout_ms=1000, int max_tries=3);

CANOpenResult_t ReceiveHB(uint8_t nodeid, CANOpenNMTState_t* statebuf=NULL,
    int rcv_timeout_ms=1000, int max_tries=1);

CANOpenResult_t ReadSDO(uint8_t nodeid, uint16_t index, uint8_t subindex,
    uint8_t* buf, size_t bufsize,
    int resp_timeout_ms=100, int max_tries=3);

CANOpenResult_t WriteSDO(uint8_t nodeid, uint16_t index, uint8_t subindex,
    uint8_t* buf, size_t bufsize,
    int resp_timeout_ms=100, int max_tries=3);

```

CANOpenJob objects are created automatically by these methods. Jobs done need to be fetched by looping ReceiveDone() until it returns COR\_ERR\_QueueEmpty.

If you want to create custom jobs, use the low level method SubmitJob() to add them to the worker queue.

### 33.4.3 Error Handling

If an error occurs, it will be given as a CANOpenResult\_t other than COR\_OK or COR\_WAIT, either by a method result or by the CANOpenJob.result field.

Result codes are:

```

COR_OK = 0,

// API level:
COR_WAIT,                // job waiting to be processed
COR_ERR_UnknownJobType,
COR_ERR_QueueFull,
COR_ERR_QueueEmpty,
COR_ERR_NoCANWrite,
COR_ERR_ParamRange,
COR_ERR_BufferTooSmall,

// Protocol level:
COR_ERR_Timeout,
COR_ERR_SDO_Access,
COR_ERR_SDO_SegMismatch,

// General purpose application level:
COR_ERR_DeviceOffline = 0x80,
COR_ERR_UnknownDevice,
COR_ERR_LoginFailed,
COR_ERR_StateChangeFailed

```

Additionally, if an SDO read/write error occurs, an abortion error code may be given by the slave. These codes follow the CANopen standard and may be extended by device specific codes.

To translate a CANOpenResult\_t and/or a known SDO abort code into a string, use the CANopen class utility methods:

```

std::string GetAbortCodeName(const uint32_t abortcode);
std::string GetResultString(const CANOpenResult_t result);

```

(continues on next page)

(continued from previous page)

```
std::string GetResultString(const CANopenResult_t result, const uint32_t abortcode);  
std::string GetResultString(const CANopenJob& job);
```

### Example

```
if (job.result != COR_OK) {  
    ESP_LOGE(TAG, "Result for %s: %s",  
        CANopen::GetJobName(job).c_str(),  
        CANopen::GetResultString(job).c_str());  
}
```

### 33.4.4 Custom Address Schemes

The standard clients use the CiA DS301 default IDs for node addressing, i.e.:

```
NMT request      → 0x000  
NMT response     → 0x700 + nodeid  
SDO request      → 0x600 + nodeid  
SDO response     → 0x580 + nodeid
```

If you need another address scheme, create a sub class of `CANopenAsyncClient` or `CANopenClient` and override the `Init...` methods as necessary.

### 33.4.5 More Code Examples

- See shell commands in `canopen_shell.cpp`
- See classes `SevconClient` and `SevconJob` in the Twizy SEVCON module

### 34.1 Welcome

The OVMS (Open Vehicle Monitoring System) team is a group of enthusiasts who are developing a means to remotely communicate with our cars, and are having fun while doing it.

The OVMS module is a low-cost hardware device that you install in your car simply by installing a SIM card, connecting the module to your car's Diagnostic port connector, and positioning a cellular antenna. Once connected, the OVMS module enables remote control and monitoring of your car.

This guide documents the installation and operation of an OVMS server.

You should, however, start by asking yourself the question 'do I need this?'. You don't need to run your own OVMS server, as you can just use one of the public Open Vehicles OVMS servers. The choice is yours.

### 34.2 Installation

#### 34.2.1 Prerequisites

You'll need a Linux, OSX, BSD, or other flavour of Unix, server with a public IP address and a shell account able to setup a daemon listening on a port. You'll need a modern version of perl running on that server, and a bunch of perl modules (from cpan).

You'll need a MYSQL server running on the same machine, or another machine, and credentials to be able to create a database and accounts.

This could possibly run on a windows server, but you are on your own ;-) [ let us know instructions if you manage to get it running ]

#### 34.2.2 Clone from GitHub

Start the installation with a clone from GitHub:

`git clone https://github.com/openvehicles/Open-Vehicle-Server.git`

You can then change into the v3/server directory, and start the installation process.

### 34.2.3 Perl Module Installation

You will need, at a minimum, the following modules from CPAN (or from your distribution's package manager):

- Config::IniFiles
- Digest::MD5
- Digest::HMAC
- Crypt::RC4::XS
- MIME::Base64
- DBI
- DBD::mysql
- EV
- AnyEvent
- AnyEvent::HTTP
- AnyEvent::HTTPD
- [HTTP::Parser::XS](#)
- [Data::UUID](#)
- Email::MIME
- Email::Sender::Simple
- Net::SSLeay
- JSON::XS
- Protocol::WebSocket

### 34.2.4 MySQL Setup

Create a mysql database “openvehicles” and use the `ovms_server.sql` script to create the necessary `ovms_*` tables. Create a mysql username and password with access to the database.

Insert a record into the `ovms_cars` table, to give you access to your own car:

```
vehicleid: DEMO
owner: 1
carpass: DEMO
userpass: DEMO
cryptscheme: 0
v_ptoken:
v_lastupdate: 0000-00-00 00:00:00
```

Obviously, change the `vehicleid`, `carpass` and `userpass` as necessary. The only required fields are `vehicleid` and `carpass`.

If you are upgrading from server v2 to v3, you can instead source the `ovms_server_v2_to_v3.sql` code to migrate your database schema.



### 34.2.5 OVMS Server Configuration

Copy `conf/ovms_server.conf.default` to `conf/ovms_server.conf` to provide a template for your configuration. Then modify the configuration as appropriate.

### 34.2.6 OVMS Server Access to MySQL

In the `conf/ovms_server.conf` file, define the access to the mysql database:

```
[db]
path=DBI:mysql:database=openvehicles;host=127.0.0.1
user=<mysqlusername>
pass=<mysqlpassword>
```

If the host is remote, change the `host=` parameter to be the remote IP address. Use `127.0.0.1` for local.

Test the connection with:

```
$ mysql -h 127.0.0.1 -u <mysqlusername> -p
Enter password: <mysqlpassword>
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

If you get “ERROR 1044 (42000): Access denied”, fix it before proceeding. The most likely cause is a mistake in your mysql user grant rights.

### 34.2.7 Enable SSL (optional)

If you want to support SSL connections to your server (port 6869 for the REST API, port 6870 for OVMS MP), you need to supply a certificate. You can create a self-signed certificate or get a certificate signed by some root CA for your server.

In both cases you need to merge the private key PEM and the chain PEM into the file “`ovms_server.pem`” located in the same directory as the “`ovms_server.pl`”. Also take care to secure the file, as it now contains your private key.

To create a self-signed certificate, do:

```
$ openssl req -sha256 -newkey rsa:4096 -nodes -keyout privkey.pem -x509 -days 365 -
↳out fullchain.pem
$ cat privkey.pem fullchain.pem >conf/ovms_server.pem
$ chmod 0600 conf/ovms_server.pem
```

Or, if you want to reuse e.g. your Let’s Encrypt server certificate, do this as root:

```
# cat /etc/letsencrypt/live/yourhost/privkey.pem /etc/letsencrypt/live/yourhost/
↳fullchain.pem >conf/ovms_server.pem
# chmod 0600 conf/vms_server.pem
# chown youruid. conf/ovms_server.pem
```

... and add a cron job or certbot hook to check for renewals and redo these steps as necessary.

### 34.2.8 Configure the Plugins

The OVMS Server v3 is based on a pluggable architecture. The plugins themselves are stored in `plugins/system` and `plugins/local` directories. You must configure (in `conf/ovms_server.conf`) the plugins that you require.

We recommend the following:

```
[plugins]
load=<<EOT
VECE
DbDBI
AuthDrupal
ApiV2
Push
PushAPNS
PushGCM
PushMAIL
ApiHttp
ApiHttpCore
ApiHttpMqapi
EOT
```

### 34.2.9 Run The Server

You can run the server manually with:

```
$ ./ovms_server.pl
```

If your linux host is running systemd, you can also look at support/ovms\_server.service and support/ovms.logrotate as examples for how you can run this as a background daemon.

#### 7. ENJOY

Any questions/comments, please let us know.

Mark Webb-Johnson March 2020

## 34.3 Plugins

### 34.3.1 ApiHttp

This plugin provides an HTTP (and optional HTTPS) server and is used by subsequent API plugins to provide HTTP services. The HTTP server is on port tcp/6868. If you provide a conf/ovms\_server.pem file, it will also use that to launch a HTTPS server on port tcp/6869.

### 34.3.2 ApiHttpCore

This plugin provides support for the HTTP API in OVMS Server v3. It supports the /api HTTP endpoint.

If you want to use this plugin, you need to also include 'ApiHttp' before loading this.

### 34.3.3 ApiHttpFile

This plugin provides support serving static files from the httpfiles directory. It supports the /file HTTP endpoint.

If you want to use this plugin, you need to also include 'ApiHttp' before loading this.

### 34.3.4 ApiHttpMqapi

This plugin provides support for Mosquitto API functions for authentication and ACL access control. It supports the /mqapi HTTP endpoint.

If you want to use this plugin, you need to also include 'ApiHttp' before loading this.

You will also need to configure your mosquitto.conf to include:

```
auth_opt_superuser <your-superuser-username>
auth_opt_acl_cache_seconds 300
auth_opt_auth_cache_seconds 300
auth_opt_backends http
auth_opt_http_ip 127.0.0.1
auth_opt_http_port 6868
auth_opt_http_getuser_uri /mqapi/auth
auth_opt_http_superuser_uri /mqapi/superuser
auth_opt_http_aclcheck_uri /mqapi/acl
```

### 34.3.5 ApiV2

This plugin provides support for the v2 OVMS protocol, including crypto protocols 0x30 and 0x31. It provides a raw v2 protocol server on tcp/6867. If you provide a conf/ovms\_server.pem file, it will also launch an SSL enabled v2 protocol server on tcp/6870.

### 34.3.6 AuthDrupal

This plugin provides support for authentication via the Drupal database. It also provides a facility to periodically synchronise drupal users to OVMS owner table records.

Note that only one Auth\* plugin should be enabled.

### 34.3.7 AuthNone

This is a stub authentication plugin. It will always fail authentication, and is only provided as an example (or for use in systems where user authentication is not required).

Note that only one Auth\* plugin should be enabled.

### 34.3.8 DbDBI

This is the core database plugin for perl DBI style SQL databases. It is used, for example, to provide access to MySQL databases.

### 34.3.9 Push

This provides the core support for Push Notifications. The actual notifications themselves are issued by sub-plugins which register with this.

### 34.3.10 PushAPNS

This plugin supports the ‘apns’ notification type, for push notifications to Apple devices (iPhones, iPads, etc). It requires apple certificates, in PEM format, to be placed in `conf/ovms_apns_sandbox.pem` and `conf/ovms_apns_production.pem`.

### 34.3.11 PushGCM

This plugin supports the ‘gcm’ notification type, for push notifications to Android devices. It requires a google API key to be placed in the `conf/ovms_server.conf` file under [gcm] section, parameter ‘apikey’. You can obtain that key by:

# Log in to your Google account # Open <https://developers.google.com/mobile/add> # Select platform “Android” # Enter an arbitrary project name, e.g. “MyOvmsServer” # Package name should be “com.openvehicles.OVMS” # Activate “Cloud Messaging” and generate the key # Note the API key and project number (= GCM sender ID)

### 34.3.12 PushMAIL

This plugin supports the ‘mail’ notification type, for push notifications by eMail. It simply requires a sendmail style mailer installed on the server.

You can configure `conf/ovms_server.conf` [mail] section parameter ‘sender’ as the sender address to be used (otherwise defaulting to ‘[notifications@openvehicles.com](mailto:notifications@openvehicles.com)’).

### 34.3.13 VECE

This plugin extends the notification system to translate error codes into textual error messages. It uses files in the `vece` directory, named `<vehicletype>.vece` (for example, `tr.vece`, `va.vece`, etc).

The `vece` files themselves are ini style files.

### 35.1 Welcome

The OVMS (Open Vehicle Monitoring System) team is a group of enthusiasts who are developing a means to remotely communicate with our cars, and are having fun while doing it.

The OVMS module is a low-cost hardware device that you install in your car simply by installing a SIM card, connecting the module to your car's Diagnostic port connector, and positioning a cellular antenna. Once connected, the OVMS module enables remote control and monitoring of your car.

This developer's guide documents version 2 of the OVMS protocol.

### 35.2 Terms

**Server** The OVMS server

**Car** The OVMS car module

**App** An OVMS client application, one of:

- Active client: interactive user applications, i.e. mobile App (Note: the car module will raise the update frequency when active clients are connected)
- Batch client: a non-interactive application, i.e. shell script

### 35.3 Startup

On startup, there is no banner/welcome message from the server. The caller initiates the protocol by sending a welcome message to the server:

1. For cars

MP-C <protection scheme> <token> <digest> <car id>

2. For interactive user apps

MP-A <protection scheme> <token> <digest> <car id>

3. For noninteractive batch apps

MP-B <protection scheme> <token> <digest> <car id>

4. For servers

MP-S <protection scheme> <token> <digest> <car id>

The server responds with a welcome message to the caller:

MP-S <protection scheme> <token> <digest>

## 35.4 Encryption Scheme 0x30

This scheme is based on using shared secrets, hmac digest for authentication and encryption key negotiation, with RC4 stream cipher and base64 encoding.

Upon startup, both the server and callers generate random tokens (encoded as textual characters). Each party then hmac-md5s the token with the shared secret to create a digest, base64 encoded.

Note that the server will only issue it's welcome message after receiving and validating the caller's welcome message.

Validation of the welcome message is performed by:

1. Checking the received token to ensure that it is different from its own token, and aborting the connection if the same.
2. Hmac-md5s the received token with the shared secret and comparing the result to the received digest.
3. If the digest match, then the partner had authenticated itself (proven it knows the shared secret, or has listened to a previous conversation).
4. If the digests don't match, then abort the connection as the partner doesn't agree with the shared secret.

Once the partner has been authenticated:

1. Create a new hmac-md5 based on the client and server tokens concatenated, with the shared secret.
2. Use the new hmac digest as the key for symmetric rc4 encryption.
3. Generate, and discard, 1024 bytes of cipher text.

From this point on, messages are rc4 encrypted and base64 encoded before transmission. Lines are terminated with CR+LF.

## 35.5 Encryption Scheme 0x31

This scheme is based on server authentication, and is supported in OVMS server v3 and later. Typically, servers provide two methods of authentication:

- Username + Password: The usual username and associated password, as used when registering on the server.
- Username + ApiToken: An API token based authentication scheme, where each user may maintain one or more access tokens (which can be created or revoked at will).

The arguments to the MP-\* startup command are:

- <protection scheme>: Must be set to '1' (0x31)

- <token>: Must be the username pre-registered with the server
- <digest>: Must be the password, or api token, for that user
- <car id>: Must be the vehicle ID to connect to, or the special value '\*' to request connection to the first vehicle on that user's account

Once the partner has been authenticated, a response 'MP-S 1 <user> <vehicleid> <list-of-all-other-vehicles>' will be sent from the server to indicate a successful authentication.

From then onwards, the messages are in plaintext with no further encryption or encoding. Lines are terminated with CR+LF.

Note that this scheme is intended to be used with external encryption schemes, such as SSL/TLS.

## 35.6 Auto Provisioning

A caller can perform auto-provisioning at any time (authenticated or not). However, only one auto-provision can be performed for each connection.

Auto-Provisioning relies on two secrets known both to the server and client. The first is usually the VIN of the vehicle and the second is usually the ICCID of the SIM card in the vehicle module. The reason these two are chosen is that they can be auto-determined by the vehicle module, but also clearly seen by the user (for entry into the server).

The mechanism works by the client module first determining its VIN and ICCID secrets, then connecting to the server and sending a AP-C message to the server proving its VIN. The server will then lookup the auto-provisioning record, and reply with that to the client (via a AP-S or AP-X message).

The auto-provisioning record itself is platform dependent, but will typically be an ordered space separated list of parameter values. For OVMS hardware, and OVMS.X PIC firmware, these are merely parameters #0, #1, #2, etc, to be stored in the car module.

### 35.6.1 Auto Provisioning Protection Scheme 0x30

1. For cars

AP-C <protection scheme> <apkey>

2. For servers

AP-S <protection scheme> <server token> <server digest> <provisioning>

AP-X

When the provisioning record is created at the server, a random token is generated (encoded as textual characters). The server then hmac-md5s this token with the shared secret (usually the ICCID known by the server) to create a digest, base64 encoded. Using this hmac digest, the server generates and discards 1024 bytes of cipher text. The server then rc4 encrypts and base64 encodes the provisioning information and stores its token, digest and encoded provision record ready for the client to request.

The car knows its VIN and ICCID. With this information, it makes a connection to the OVMS service on the server, and provides the VIN as the <apkey> in a AP-C message to the server.

The server will ensure that it only responds to one AP-C message for any one connection. Once responded, all subsequent AP-C requests will always be replied with a AP-X message.

Upon receiving the AP-C message, the server looks up any provisioning records it has for the given <apkey>. If it has none, it replies with an AP-X message.

If the server does find a matching provisioning record, it replies with an AP-S message sending the previously saved server token, digest and encoded provisioning record to the car.

If the car receives an AP-X message, it knows that auto-provisioning was not successful.

If the car receives an AP-S message, it can first validate the server authenticity. By producing its own hmac-md5 of the server token and secret ICCID, the car can validate the server-provided digest is as expected. If this validation step does not succeed, the car should abort the auto-provisioning.

If acceptable, the car can decrypt the provisioning record by first generating and discarding 1024 bytes of cipher text, then decoding the provisioning record provided by the server.

## 35.7 Backwards Compatibility

Typically, comma-separated lists are used to transmit parameters. Applications, Servers and the Car firmware should in general ignore extra parameters not expected. In this way, the protocol messages can be extended by adding extra parameters, without breaking old Apps/Cars that don't expect the new parameters.

Similarly, unrecognized messages should be ignored. Unrecognised commands in the "C" (command) message should be responded to with a generic "unrecognized" response (in the "c" (command response) messages).

## 35.8 Messages

### 35.8.1 Car <-> Server <-> App

After discarding CR+LF line termination, and base64 decoding, the following protocol messages are defined.

```
<message> ::= <magic> <version> <space> <protmsg>
<magic> ::= MP-
<version> ::= 1 byte version number - this protocol is 0x30
<space> ::= ' ' (ascii 0x20)
<protmsg> ::= <servertocar> | <cartoserver> | <servertoapp> | <apptoserver>
<servertocar> ::= "S" <payload>
<cartoserver> ::= "C" <payload>
<servertoapp> ::= "s" <payload>
<apptoserver> ::= "c" <payload>
<payload> ::= <code> <data>
<code> ::= 1 byte instruction code
<data> ::= N bytes data (dependent on instruction code)
```

### 35.8.2 Ping message 0x41 "A"

This message may be sent by any party, to test the link. The expected response is a 0x61 ping acknowledgement. There is no expected payload to this message, and any given can be discarded.



### 35.8.3 Ping Acknowledgement message 0x61 “a”

This message is sent in response to a 0x41 ping message. There is no expected payload to this message, and any given can be discarded.

### 35.8.4 Command message 0x43 “C”

This message is sent <apptoserver> then <servertocar> and carries a command to be executed on the car. The message would normally be paranoid-encrypted.

<data> is a comma-separated list of:

- command (a command code 0..65535)
- parameters (dependent on the command code)

For further information on command codes and parameters, see the command section below.

### 35.8.5 Command response 0x63 “c”

This message is sent <cartoserver> then <servertoapp> and carries the response to a command executed on the car. The message would normally be paranoid-encrypted.

<data> is a comma-separated list of:

- command (a command code 0..65535)
- result (0=ok, 1=failed, 2=unsupported, 3=unimplemented)
- parameters (dependent on the command code and result)

For result=0, the parameters depend on the command being responded to (see the command section below for further information).

For result=1, the parameter is a textual string describing the fault.

For result=2 or 3, the parameter is not used.

### 35.8.6 Car Environment message 0x44 “D”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the environment settings of the vehicle.

<data> is comma-separated list of:

Door state #1

- bit0 = Left Door (open=1/closed=0)
- bit1 = Right Door (open=1/closed=0)
- bit2 = Charge port (open=1/closed=0)
- bit3 = Pilot present (true=1/false=0) (always 1 on my 2.5)
- bit4 = Charging (true=1/false=0)
- bit5 = always 1
- bit6 = Hand brake applied (true=1/false=0)
- bit7 = Car ON (“ignition”) (true=1/false=0)

Door state #2

- bit3 = Car Locked (locked=1/unlocked=0)
- bit4 = Valet Mode (active=1/inactive=0)
- bit6 = Bonnet (open=1/closed=0)
- bit7 = Trunk (open=1/closed=0)

Lock/Unlock state

- 4 = car is locked
- 5 = car is unlocked

Temperature of the PEM (celcius)

Temperature of the Motor (celcius)

Temperature of the Battery (celcius)

Car trip meter (in 1/10th of a distance unit)

Car odometer (in 1/10th of a distance unit)

Car speed (in distance units per hour)

Car parking timer (0 for not parked, or number of seconds car parked for)

Ambient Temperature (in Celcius)

Door state #3

- bit0 = Car awake (turned on=1 / off=0)
- bit1 = Cooling pump (on=1/off=0)
- bit6 = 1=Logged into motor controller
- bit7 = 1=Motor controller in configuration mode

Stale PEM, Motor, Battery temps indicator (-1=none, 0=stale, >0 ok)

Stale ambient temp indicator (-1=none, 0=stale, >0 ok)

Vehicle 12V line voltage

Door State #4

- bit2 = alarm sounds (on=1/off=0)

Reference voltage for 12v power

Door State #5

- bit0 = Rear left door (open=1/closed=0)
- bit1 = Rear right door (open=1/closed=0)
- bit2 = Frunk (open=1/closed=0)
- bit4 = 12V battery charging
- bit5 = Auxiliary 12V systems online
- bit7 = HVAC running

Temperature of the Charger (celsius)

Vehicle 12V current (i.e. DC converter output)

Cabin temperature (celsius)

### 35.8.7 Paranoid-mode encrypted message 0x45 “E”

This message is sent for any of the four message <protmsg> types, and represents an encrypted transmission that the server should just relay (or is relaying) without being able to interpret it. The encryption is based on a shared secret, between the car and the apps, to which the server is not privy.

<data> is:

- <paranoidtoken> | <paranoidcode>
- <paranoidtoken> ::= “T” <ptoken>
- <paranoidcode> ::= “M” <code> <data>

In the case of <paranoidtoken>, the <ptoken> is a random token that represent the encryption key. It can only be sent <cartoserver> or <servertoapp>. Upon receiving this token, the server discards all previously stored paranoid messages, sends it on to all connected apps, and then stores the token. Every time an app connects, the server also sends this token to the app.

In the case of <paranoidcode>, the <code> is a sub-message code, and can be any of the codes listed in this document (except for “A”, “a” and “E”). The <data> is the corresponding encrypted payload message. The encryption is performed on the <data> by:

- Create a new hmac-md5 based on the <ptoken>, with the shared secret.
- Use the new hmac digest as the key for symmetric rc4 encryption.
- Generate, and discard, 1024 bytes of cipher text.

and the data is base64 encoded. Upon receiving a paranoid message from the car, the server forwards it on the all connected apps, and then stores the message. Every time an app connects, the server sends all such stored messages. Upon receiving a paranoid message from an app, if the car is connected, the server merely forwards it on to the car, otherwise discarding it.

### 35.8.8 Car firmware message 0x46 “F”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the firmware versions of the vehicle.

<data> is comma-separated list of:

- Car firmware version
- Car VIN
- GSM signal level
- Write-enabled firmware (0=read-only, 1=write-enabled)
- Car type (TR=Tesla Roadster, others may follow)
- GSM lock

### 35.8.9 Server firmware message 0x66 “f”

This message is sent <servertocar> “S”, or <servertoapp> “s”, and transmits the firmware versions of the server.

<data> is comma-separated list of:

- Server firmware version

### 35.8.10 Car group subscription message 0x47 “G”

This message is sent <apptoserver> “A”, and requests subscription to the specified group.

<data> is comma-separated list of:

- Group name

### 35.8.11 Car group update message 0x67 “g”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits a group location message for the vehicle.

<data> is comma-separated list of:

- Vehicle ID (only <servertoapp>, not sent <cartoserver>)
- Group name
- Car SOC
- Car Speed
- Car direction
- Car altitude
- Car GPS lock (0=nogps, 1=goodgps)
- Stale GPS indicator (-1=none, 0=stale, >0 ok)
- Car latitude
- Car longitude

### 35.8.12 Historical Data update message 0x48 “H”

This message is sent <cartoserver> “C”, and transmits a historical data message for storage on the server.

<data> is comma-separated list of:

- type (unique storage class identification type)
- recordnumber (integer record number)
- lifetime (in seconds)
- data (a blob of data to be dealt with as the application requires)

The lifetime is specified in seconds, and indicates to the server the minimum time the vehicle expects the server to retain the historical data for. Consideration should be made as to server storage and bandwidth requirements.

The type is composed of <vehicletype> - <class> - <property>

<Vehicletype> is the usual vehicle type, or ‘\*’ to indicate generic storage suitable for all vehicles.

<Class> is one of:

- PWR (power)
- ENG (engine)
- TRX (transmission)
- CHS (chassis)
- BDY (body)
- ELC (electrics)
- SAF (safety)
- SEC (security)
- CMF (comfort)
- ENT (entertainment)
- COM (communications)
- X\*\* (unclassified and experimental, with \*\* replaced with 2 digits code)

<Property> is a property code, which the vehicle decides.

The server will timestamp the incoming historical records, and will set an expiry date of timestamp + <lifetime> seconds. The server will endeavor to retain the records for that time period, but may expend data earlier if necessary.

### 35.8.13 Historical Data update+ack message 0x68 “h”

This message is sent <cartoserver> “C”, or <severtocar> “c”, and transmits/acknowledges historical data message for storage on the server.

For <cartoserver>, the <data> is comma-separated list of:

- ackcode (an acknowledgement code)
- timediff (in seconds)
- type (unique storage class identification type)
- recordnumber (integer record number)
- lifetime (in seconds)
- data (a blob of data to be dealt with as the application requires)

The ackcode is a numeric acknowledgement code - if the server successfully receives the message, it will reply with “h” and this ackcode to acknowledge reception.

The timediff is the time difference, in seconds, to use when storing the record (e.g.; -3600 would indicate the record data is from one hour ago).

The lifetime is specified in seconds, and indicates to the server the minimum time the vehicle expects the server to retain the historical data for. Consideration should be made as to server storage and bandwidth requirements.

For <severtocar>, the <data> is:

- ackcode (an acknowledgement code)

The <cartoserver> message sends the data to the server. The <severtocar> message acknowledges the data.

### 35.8.14 Push notification message 0x50 “P”

This message is sent <cartoserver> “C”, or <servertoapp> “s”. When used by the car, it requests the server to send a textual push notification alert message to all apps registered for this car. The <data> is 1 byte alert type followed by N bytes of textual message. The server will use this message to send the notification to any connected apps, and can also send via external mobile frameworks for unconnected apps.

### 35.8.15 Push notification subscription 0x70 “p”

This message is sent <apptoserver> A”. It is used by app to register for push notifications, and is normally at the start of a connection. The <data> is made up of:

<appid>,<pushtype>,<pushkeytype>,<vehicleid>,<netpass>,<pushkeyvalue>

The server will verify the credentials for each vehicle, and store the required notification information.

Note: As of June 2020, only one vehicleid can be subscribed at a time. If multiple vehicles are required, then they should each be subscribed in individual messages.

### 35.8.16 Server -> Server Record message 0x52 “R”

This message is sent <servertoserver> “S”, and transmits an update to synchronized database table records.

Sub-type RV (Vehicle record): <data> is comma-separated list of:

- Vehicleid
- Owner
- Carpass
- v\_server
- deleted
- changed

Sub-type RO (Owner record): <data> is comma-separated list of:

- Ownerid
- OwnerName
- OwnerMail
- PasswordHash
- OwnerStatus
- deleted
- changed

### 35.8.17 Server -> Server Message Replication message 0x72 “r”

This message is sent <servertoserver> “S”, and replicates a message for a particular car.

<data> is comma-separated list of:

- vehicleid
- message code

- message data

### 35.8.18 Car state message 0x53 “S”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the last known status of the vehicle.

<data> is comma-separated list of:

- SOC
- Units (“M” for miles, “K” for kilometers)
- Line voltage
- Charge current (amps)
- Charge state (charging, topoff, done, prepare, heating, stopped)
- Charge mode (standard, storage, range, performance)
- Ideal range
- Estimated range
- Charge limit (amps)
- Charge duration (minutes)
- Charger B4 byte (tba)
- Charge energy consumed (1/10 kWh)
- Charge sub-state
- Charge state (as a number)
- Charge mode (as a number)
- Charge Timer mode (0=onplugin, 1=timer)
- Charge Timer start time
- Charge timer stale (-1=none, 0=stale, >0 ok)
- Vehicle CAC100 value (calculated amp hour capacity, in Ah)
- ACC: Mins remaining until car will be full
- ACC: Mins remaining until car reaches charge limit
- ACC: Configured range limit
- ACC: Configured SOC limit
- Cooldown: Car is cooling down (0=no, 1=yes)
- Cooldown: Lower limit for battery temperature
- Cooldown: Time limit (minutes) for cooldown
- ACC: charge time estimation for current charger capabilities (min.)
- Charge ETR for range limit (min.)
- Charge ETR for SOC limit (min.)
- Max ideal range

- Charge/plug type ID according to OpenChargeMaps.org connectiontypes (see <http://api.openchargemap.io/v2/referencedata/>)
- Charge power output (kW)
- Battery voltage (V)
- Battery SOH (state of health) (%)
- Charge power input (kW)
- Charger efficiency (%)

### 35.8.19 Car update time message 0x53 “T”

This message is sent <servertoapp> “s”, and transmits the last known update time of the vehicle.

<data> is the number of seconds since the car last sent an update message

### 35.8.20 Car location message 0x4C “L”

This message is sent <cartoserver> “C” and transmits the last known location of the vehicle.

<data> is comma-separated list of:

- Latitude
- Longitude
- Car direction
- Car altitude
- Car GPS lock (0=nogps, 1=goodgps)
- Stale GPS indicator (-1=none, 0=stale, >0 ok)
- Car speed (in distance units per hour)
- Car trip meter (in 1/10th of a distance unit)
- Drive mode (car specific encoding of current drive mode)
- Battery power level (in kW, negative = charging)
- Energy used (in Wh)
- Energy recovered (in Wh)
- Inverter motor power (kW) (positive = output)
- Inverter efficiency (%)
- GPS mode indicator (see below)
- GPS satellite count
- GPS HDOP (see below)
- GPS speed (in distance units per hour)

**GPS mode indicator:** this shows the NMEA receiver mode. If using the SIM5360 modem for GPS, this is a two character string. The first character represents the GPS receiver mode, the second the GLONASS receiver mode. Each mode character may be one of:

- *N* = No fix. Satellite system not used in position fix, or fix not valid



- *A* = Autonomous. Satellite system used in non-differential mode in position fix
- *D* = Differential (including all OmniSTAR services). Satellite system used in differential mode in position fix
- *P* = Precise. Satellite system used in precision mode. Precision mode is defined as: no deliberate degradation (such as Selective Availability) and higher resolution code (P-code) is used to compute position fix
- *R* = Real Time Kinematic. Satellite system used in RTK mode with fixed integers
- *F* = Float RTK. Satellite system used in real time kinematic mode with floating integers
- *E* = Estimated (dead reckoning) Mode
- *M* = Manual Input Mode
- *S* = Simulator Mode

**GPS HDOP:** HDOP = horizontal dilution of precision. This is a measure for the currently achievable precision of the horizontal coordinates (latitude & longitude), which depends on the momentary relative satellite positions and visibility to the device.

The lower the value, the higher the precision. Values up to 2 mean high precision, up to 5 is good. If the value is higher than 20, coordinates may be off by 300 meters from the actual position.

See [https://en.wikipedia.org/wiki/Dilution\\_of\\_precision\\_\(navigation\)](https://en.wikipedia.org/wiki/Dilution_of_precision_(navigation)) for further details.

### 35.8.21 Car Capabilities message 0x56 “V”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the vehicle capabilities. It was introduced with v2 of the protocol.

<data> is comma-separated list of vehicle capabilities of the form:

- C<cmd> indicates vehicle support command <cmd>
- C<cmdL>-<cmdH> indicates vehicle will support all commands in the specified range

### 35.8.22 Car TPMS message 0x57 “W” (old/obsolete)

**Note:** Message “W” has been replaced by “Y” (see below) for OVMS V3. The V3 module will still send “W” messages along with “Y” for old clients for some time. Clients shall adapt to using “Y” if available ASAP, “W” messages will be removed from V3 in the near future.

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the last known TPMS values of the vehicle.

<data> is comma-separated list of:

- front-right wheel pressure (psi)
- front-right wheel temperature (celcius)
- rear-right wheel pressure (psi)
- rear-right wheel temperature (celcius)
- front-left wheel pressure (psi)
- front-left wheel temperature (celcius)
- rear-left wheel pressure (psi)

- rear-left wheel temperature (celcius)
- Stale TPMS indicator (-1=none, 0=stale, >0 ok)

### 35.8.23 Car TPMS message 0x59 “Y”

This message is sent <cartoserver> “C”, or <servertoapp> “s”, and transmits the last known TPMS values of the vehicle.

<data> is comma-separated list of:

- number of defined wheel names
- list of defined wheel names
- number of defined pressures
- list of defined pressures (kPa)
- pressures validity indicator (-1=undefined, 0=stale, 1=valid)
- number of defined temperatures
- list of defined temperatures (Celcius)
- temperatures validity indicator (-1=undefined, 0=stale, 1=valid)
- number of defined health states
- list of defined health states (Percent)
- health states validity indicator (-1=undefined, 0=stale, 1=valid)
- number of defined alert levels
- list of defined alert levels (0=none, 1=warning, 2=alert)
- alert levels validity indicator (-1=undefined, 0=stale, 1=valid)

---

**Note:** Pressures are transported in kPa now instead of the former PSI. To convert to PSI, multiply by 0.14503773773020923.

---

### 35.8.24 Peer connection message 0x5A “Z”

This message is sent <servertocar> or <servertoapp> to indicate the connection status of the peer (car for <server-toapp>, interactive apps for <servertocar>). It indicates how many peers are currently connected.

It is suggested that the car should use this to immediately report on, and to increase the report frequency of, status - in the case that one or more interactive Apps are connected and watching the car.

Batch client connections do not trigger any peer count change for the car, but they still receive the car peer status from the server.

<data> is:

- Number of peers connected, expressed as a decimal string

## 35.9 Commands

### 35.9.1 Commands and Expected Responses

For message types “C” and “c”, the following commands and responses are expected to be supported.

#### 35.9.2 1 - Request feature list

Command parameters are unused and ignored by the car.

Response is a sequence of individual messages with each message containing the following parameters:

- feature number
- maximum number of features
- feature value

Registered features are:

- 0: Digital SPEEDO (experimental)
- 8: Location STREAM mode (consumes more bandwidth)
- 9: Minimum SOC
- 15: CAN bus can write-enabled

Note that features 0 through 7 are ‘volatile’ and will be lost (reset to zero value) if the power is lost to the car module, or module is reprogrammed. These features are considered extremely experimental and potentially dangerous.

Features 8 through 15 are ‘permanent’ and will be stored as parameters 23 through 31. These features are considered more stable, but optional.

#### 35.9.3 2 - Set feature

Command parameters are:

- feature number to set
- value to set

Response parameters are unused, and will merely indicate the success or not of the result.

#### 35.9.4 3 - Request parameter list

Command parameters are unused and ignored by the car.

Response is a sequence of individual messages with each message containing the following parameters:

- parameter number
- maximum number of parameters
- parameter value

Registered parameters are:

- 0: Registered telephone number
- 1: Registration Password

- 2: Miles / Kilometer flag
- 3: Notification method list
- 4: Server IP
- 5: GPRS APN
- 6: GPRS User
- 7: GPRS Password
- 8: Vehicle ID
- 9: Network Password
- 10: Paranoid Password

Note that some parameters (24 through 31) are tied directly to the features system (for permanent features) and are thus not directly maintained by the parameter system or shown by this command.

### 35.9.5 4 - Set parameter

Command parameters are:

- parameter number to set
- value to set

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.6 5 - Reboot

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result. Shortly after sending the response, the module will reboot.

### 35.9.7 6 - Charge Alert

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result. Shortly after sending the response, the module will issue a charge alert.

### 35.9.8 7 - Execute SMS command

Command parameter is:

- SMS command with parameters

Response is the output of the SMS command that would otherwise have been sent as the reply SMS, with LF characters converted to CR.

The caller id is set to the registered phone number. Return code 1 is used for all errors, i.e. authorization failure, command failure and unknown/unhandled commands.

Note: SMS commands with multiple replies are not yet supported, only the last reply will be returned.

### 35.9.9 10 - Set Charge Mode

Command parameters are:

- mode (0=standard, 1=storage, 3=range, 4=performance)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.10 11 - Start Charge

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.11 12 - Stop Charge

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.12 15 - Set Charge Current

Command parameters are:

- current (specified in Amps)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.13 16 - Set Charge Mode and Current

Command parameters are:

- mode (0=standard, 1=storage, 3=range, 4=performance)
- current (specified in Amps)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.14 17 - Set Charge Timer Mode and Start Time

Command parameters are:

- timermode (0=plugin, 1=timer)
- start time (0x059F for midnight GMT, 0x003B for 1am GMT, etc)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.15 18 - Wakeup car

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.16 19 - Wakeup temperature subsystem

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.17 20 - Lock Car

Command parameters are:

- pin (the car pin to use for locking)

Response parameters are unused, and will merely indicate the success or not of the result.

N.B. unlock/lock may not affect the immobilizer+alarm (when fitted)

### 35.9.18 21 - Activate Valet Mode

Command parameters are:

- pin (the car pin to activate valet mode)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.19 22 - Unlock Car

Command parameters are:

- pin (the car pin to use for unlocking)

Response parameters are unused, and will merely indicate the success or not of the result.

N.B. unlock/lock does not affect the immobilizer+alarm (when fitted)

### 35.9.20 23 - Deactivate Value Mode

Command parameters are:

- pin (the car pin to use for deactivating value mode)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.21 24 - Home Link

Command parameters are:

- button (home link button 0, 1 or 2)

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.22 25 - Cooldown

Command parameters are unused and ignored by the car.

Response parameters are unused, and will merely indicate the success or not of the result.

### 35.9.23 30 - Request GPRS utilization data

Command parameters are unused and ignored by the car.

Response is a sequence of individual messages with each message containing the following parameters:

- record number
- maximum number of records
- date
- car received bytes
- car transmitted bytes
- apps received bytes
- apps transmitted bytes

Note that this request is handled by the server, not the car, so must not be sent in paranoid mode. The response (from the server) will also not be sent in paranoid mode.

N.B. Dates (and GPRS utilization data) are in UTC.

### 35.9.24 31 - Request historical data summary

Command parameters are:

- since (optional timestamp condition)

Response is a sequence of individual messages with each message containing the following parameters:

- type number
- maximum number of types
- type value
- number of unique records (per type)
- total number of records (per type)
- storage usage (in bytes, per type)
- oldest data timestamp (per type)
- newest data timestamp (per type)

N.B. Timestamps are in UTC.

### 35.9.25 32 - Request historical data records

Command parameters are:

- type (the record type to retrieve)
- since (optional timestamp condition)

Response is a sequence of individual messages with each message containing the following parameters:

- response record number
- maximum number of response records
- data record type

- data record timestamp
- data record number
- data record value

### 35.9.26 40 - Send SMS

Command parameters are:

- number (telephone number to send sms to)
- message (sms message to be sent)

Response parameters are unused, and will merely indicate the success or not of the submission (not delivery) of the SMS.

### 35.9.27 41 - Send MMI/USSD Codes

Command parameters are:

- USSD\_CODE (the ussd code to send)

Response parameters are unused, and will merely indicate the success or not of the submission (not delivery) of the request.

### 35.9.28 49 - Send raw AT Command

Command parameters are:

- at (the AT command to send - including the AT prefix)

Response parameters are unused, and will merely indicate the success or not of the submission (not delivery) of the request.



### 36.1 Welcome

For some time now I've wanted to offer an HTTP API for OVMS. The existing car<->server and server<->app protocol is cool, but requires going through some hoops (perl, whatever) to handle the encryption and protocol level stuff. An HTTP API would make this much easier to externally script, and would be great for pulling down logs and other such things.

This document describes this HTTP API.

Mark, February 2013.

### 36.2 Format

The API is a RESTfull service running at:

- <http://api.openvehicles.com:6868/api/...>
- <https://api.openvehicles.com:6869/api/...>

Other OVMS servers should provide this API on the same ports, but for the rest of this manual we will simply refer to api.openvehicles.com as the server being used.

The return data is a json formatted array of hashes. Each record is one for one vehicle and shows you the vehicle id, as well as counts for the number of apps currently connected to that vehicle, and whether the vehicle is connected to the net (server) or not.

From the server point of view, we can treat an api session just like an app session. From the vehicle point of view there will be no difference - once an API connects to a vehicle, the server will send a "Z 1" message to tell the module it has a connection. If the session times out or is logged out, the server will inform the modules in the vehicles.

### 36.2.1 Example

For example, once authenticated you can request a list of vehicles on your account:

```
$ curl -v -X GET -b cookiejar http://api.openvehicles.com:6868/api/vehicles
* About to connect() to api.openvehicles.com port 6868 (#0)
*   Trying api.openvehicles.com....
* connected
* Connected to tmc.openvehicles.com (64.111.70.40) port 6868 (#0)
> GET /api/vehicles HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
OpenSSL/0.9.8r zlib/1.2.5
Host: api.openvehicles.com:6868
Accept: */*
Cookie: ovmsapisession=9ed66d0e-5768-414e-b06d-476f13be40ff

* HTTP 1.0, assume close after body
< HTTP/1.0 200 Logout ok
< Connection: close
< Content-Length: 280
< Cache-Control: max-age=0
< Content-Type: application/json
< Date: Fri, 22 Feb 2013 12:44:41 GMT
< Expires: Fri, 22 Feb 2013 12:44:41 GMT
<
[
  {"id":"DEMO","v_apps_connected":0,"v_net_connected":1},
  {"id":"MARKSCAR","v_apps_connected":1,"v_net_connected":1},
  {"id":"QCCAR","v_apps_connected":0,"v_net_connected":0},
  {"id":"RALLYCAR","v_apps_connected":0,"v_net_connected":0},
  {"id":"TESTCAR","v_apps_connected":0,"v_net_connected":0}
]
```

(note that the above is re-formatted slightly, to make it clearer to read).

## 36.3 Authentication

### 36.3.1 Alternatives

Each API call must be authenticated, and there are currently three options for this:

# Username+Password authentication: Specify parameters ‘username’ and ‘password’ in the URL, and they will be validated against registered users on the server. # Username+ApiToken authentication: Specify parameter ‘username’ as the registered user, and ‘password’ as a registered API token, in the URL, and they will be validated against registered API tokens for that user on the server. # Cookie authentication: A cookie may be obtained (by either of the above two authentication methods), and then used for subsequent API calls.

### 36.3.2 Cookie Based Authentication

Cookie based authentication avoids the requirement to authenticate each time. It is also a requirement for session based API calls. This is done with a http “GET /api/cookie” passing parameters ‘username’ and ‘password’ as your www.openvehicles.com username and password (or API token) respectively:

```
$ curl -v -X GET -c cookiejar
http://api.openvehicles.com:6868/api/cookie?username=USERNAME&password=PASSWORD
* About to connect() to tmc.openvehicles.com port 6868 (#0)
*   Trying 64.111.70.40...
*   connected
*   Connected to tmc.openvehicles.com (64.111.70.40) port 6868 (#0)
GET /api/cookie?username=USERNAME&password=PASSWORD HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
OpenSSL/0.9.8r zlib/1.2.5
Host: tmc.openvehicles.com:6868
Accept: */*

* HTTP 1.0, assume close after body
< HTTP/1.0 200 Authentication ok
< Connection: close
< Content-Length: 9
< Cache-Control: max-age=0
< Content-Type: text/plain
* Added cookie ovmsapisession="9ed66d0e-5768-414e-b06d-476f13be40ff" for domain tmc.
  ↪openvehicles.com, path /api/, expire 0
< Set-Cookie: ovmsapisession=9ed66d0e-5768-414e-b06d-476f13be40ff
< Date: Fri, 22 Feb 2013 12:43:56 GMT
< Expires: Fri, 22 Feb 2013 12:43:56 GMT
<
Login ok
```

Once logged in, all subsequent requests should pass the cookie (ovmsapisession). The session will expire after 3 minutes of no use, or you can specifically terminate / logout the session by calling “DELETE /api/cookie”.

The cookie can be destroyed (and session logged out) using the DELETE method (passing the original cookie):

```
$ curl -v -X DELETE -b cookiejar
http://api.openvehicles.com:6868/api/cookie
```

### 36.3.3 API Token Maintenance

An API token can be created with a POST to the /api/token API endpoint:

```
$ curl -v -X POST -F 'application=<app>' -F 'purpose=<purpose>' -F 'permit=<permit>'
http://api.openvehicles.com:6868/api/token
```

Note that the ‘purpose’ and ‘application’ fields are comments attached to the token and are intended to identify the application that created/uses the token and the purpose that token is used for.

The ‘permit’ field defines the list of rights granted to the user of this token.

Any of the three authentication mechanisms can be used for this, so long as the permissions include either ‘token.admin’ or ‘admin’ rights.

An API token can be deleted with a DELETE to the /api/token/<TOKEN> API endpoint:

```
$ curl -v -X DELETE http://api.openvehicles.com:6868/api/token/<TOKEN>
```

## 36.4 Requests

### 36.4.1 GET /api/cookie

Login and return a session cookie.

### 36.4.2 DELETE /api/cookie

Delete the session cookie and logout.

### 36.4.3 GET /api/token

Return a list of registered API tokens.

### 36.4.4 POST /api/token

Create an API token.

### 36.4.5 DELETE /api/token/<TOKEN>

Delete the specified API token.

### 36.4.6 GET /api/vehicles

Return a list of registered vehicles:

- id Vehicle ID
- v\_net\_connected Number of vehicles currently connected
- v\_apps\_connected Number of apps currently connected
- v\_btcs\_connected Number of batch clients currently connected

### 36.4.7 GET /api/protocol/<VEHICLEID>

Return raw protocol records (no vehicle connection):

- m\_msgtime Date/time message received
- m\_paranoid Paranoid mode flag
- m\_ptoken Paranoid mode token
- m\_code Message code
- m\_msg Message body

### 36.4.8 GET /api/status/<VEHICLEID>

Return vehicle status:

- soc
- units
- idealrange
- idealrange\_max
- estimatedrange
- mode
- chargestate
- cac100
- soh
- cooldown\_active
- fl\_dooropen
- fr\_dooropen
- cp\_dooropen
- pilotpresent
- charging
- caron
- carlocked
- valetmode
- bt\_open
- tr\_open
- temperature\_pem
- temperature\_motor
- temperature\_battery
- temperature\_charger
- tripmeter
- odometer
- speed
- parkingtimer
- temperature\_ambient
- carawake
- staletemps
- staleambient
- charging\_12v
- vehicle12v

- vehicle12v\_ref
- vehicle12v\_current
- alarmsounding

### 36.4.9 GET /api/tpms/<VEHICLEID>

Return tpms status:

- fr\_pressure
- fr\_temperature
- rr\_pressure
- rr\_temperature
- fl\_pressure
- fl\_temperature
- rl\_pressure
- rl\_temperature
- staletpms

### 36.4.10 GET /api/location/<VEHICLEID>

Return vehicle location:

- latitude
- longitude
- direction
- altitude
- gpslock
- stalegps
- speed
- tripmeter
- drivemode
- power
- energyused
- energyrecd

### 36.4.11 GET /api/charge/<VEHICLEID>

Return vehicle charge status:

- linevoltage
- battvoltage

- chargecurrent
- chargepower
- chargetype
- chargestate
- soc
- units
- idealrange
- estimatedrange
- mode
- chargelimit
- chargeduration
- chargeb4
- chargekwh
- chargesubstate
- chargetimermode
- chargestarttime
- chargetimerstale
- cac100
- soh
- charge\_etr\_full
- charge\_etr\_limit
- charge\_limit\_range
- charge\_limit\_soc
- cooldown\_active
- cooldown\_tbattery
- cooldown\_timelimit
- charge\_estimate
- charge\_etr\_range
- charge\_etr\_soc
- idealrange\_max
- cp\_dooropen
- pilotpresent
- charging
- caron
- temperature\_pem
- temperature\_motor

- temperature\_battery
- temperature\_charger
- temperature\_ambient
- carawake
- staletemps
- staleambient
- charging\_12v
- vehicle12v
- vehicle12v\_ref
- vehicle12v\_current

### 36.4.12 GET /api/historical/<VEHICLEID>

Request historical data summary (as array of):

- h\_recordtype
- distinctrecs
- totalrecs
- totalsize
- first
- last

### 36.4.13 GET /api/historical/<VEHICLEID>/<DATATYPE>

Request historical data records:

- h\_timestamp
- h\_recordnumber
- h\_data

### 36.4.14 Not Yet Implemented

- GET /api/vehicle/<VEHICLEID> Connect to, and return vehicle information
- DELETE /api/vehicle/<VEHICLEID> Disconnect from vehicle
- PUT /api/charge/<VEHICLEID> Set vehicle charge status
- DELETE /api/charge/<VEHICLEID> Abort a vehicle charge
- GET /api/lock/<VEHICLEID> Return vehicle lock status
- PUT /api/lock/<VEHICLEID> Lock a vehicle
- DELETE /api/lock/<VEHICLEID> Unlock a vehicle
- GET /api/valet/<VEHICLEID> Return valet status



- PUT /api/valet/<VEHICLEID> Enable valet mode
- DELETE /api/valet/<VEHICLEID> Disable valet mode
- GET /api/features/<VEHICLEID> Return vehicle features
- PUT /api/feature/<VEHICLEID> Set a vehicle feature
- GET /api/parameters/<VEHICLEID> Return vehicle parameters
- PUT /api/parameter/<VEHICLEID> Set a vehicle parameter
- PUT /api/reset/<VEHICLEID> Reset the module in a particular vehicle
- PUT /api/homelink/<VEHICLEID> Activate home link



## A

App, [337](#)

## C

Car, [337](#)

## S

Server, [337](#)