OVMS

Open Vehicle Monitoring System

Model: OVMS-31-5360A

www.openvehicles.com

OVMS Advanced User Guide
v3.2.0 (18th January 2019)

For OVMS Hardware Module v3.1 and Firmware v3.2.xxx (or later)

History

Version	Date	Who	Summary
v3.1.0	15th March 2018	Mark Webb-Johnson	Initial version written
v3.1.1	6th August 2018	Robert Sharpe	Simplified terminology/layout to make easier to read/use Identified potential TODOs Added vehicle commands section Created standard, vehicle specific template
v3.2.0	18th January 2019	Mark Webb-Johnson	Document javascript functionality for v3.2.

Table of Contents

Welcome **7**

OVMS System Components **9**

The OVMS Module 9

Cellular Modem Option and GSM Antenna 10

GPS/GNSS Antenna 10

Vehicle Connection 10

OVMS Server v2 10

OVMS Server v3 11

Upgrading from OVMS v1/v2 11

Installation **12**

Pre-Installation Steps 12

OVMS v2 Server Registration 12

Hologram Activation 13

OVMS Module Installation 13

Powering the module 13

Server account 13

Initial Connection (Wifi and Browser) 14

Setup wizard 15

Manual configuration 16

Vehicle Configuration 16

OVMS Server v2 Configuration 16

Auto Start Configuration 17
Networking Options 18
1. Wifi Client 19
2. Wifi Access Point 19
3. Cellular Data 19
Firmware Update 20
Flash from Web 20
Flash from File 20
12V Calibration 21
Module Configuration 22
Cell phone App Configuration 22
Advanced Console Usage 23
Console Connections 23
USB Console 23
SSH Console 24
Console Basics 26
Logging 27
Logging to the console 28
Logging to SD CARD 29
Logging configuration [3.1.005] 30
Virtual File System 32
Over the Air Updates 33
Boot Status & Crash Report 35
Configuration 36
Metrics 36
Time 38
Geofenced Locations 40
Events 40
Notifications 45
Command Scripts 46
JavaScripting 47
Persistent JavaScript 47
JavaScript Modules 48
Internal Modules 49
Internal Objects and Functions/Methods 49
assert(condition,message) 49
print(string) 49
OvmsCommand 49

OvmsLocation 50
OvmsMetrics 50
OvmsVehicle 50
Module Factory Reset 52
Module Factory Firmware 54
Install esptool.py 54
Vehicle Documentation 55
DEMO Demonstration vehicle 55
KS Kia Soul EV 56
OBD-II cable 56
Configuration 56
Battery size 56
Real life ideal range 57
Open charge port using key fob 57
Security pin code for remote commands 57
Soul specific metrics 57
Soul specific shell commands 59
Read only commands 59
xks trip 59
xks tpms 59
xks aux 59
xks vin 59
Commands 59
xks trunk <pin code> 60
xks chargeport <pin code> 60
xks ParkBreakService <on/off> 60
xks IGN1 <on/off><pin> 60
xks IGN2 <on/off><pin> 60
xks ACC <on/off><pin> 60
xks START <on/off><pin> 60
xks headlightdelay <on/off> 60
xks onetouchturnsignal <0=Off, 1=3 blinks, 2=5 blinks, 3=7 blinks> 60
xks autodoorunlock <1=Off, 2=On vehicle off, 3=On shift to P, 4=On driver door unlock> 60
ECU-write commands 60
NL Nissan Leaf 62
Configuration 62
2011-2015 models 62
2016-2017 models 62

30 kwh models 62
Custom Metrics 62
Custom Commands 62
Range Calculation: 63
O2 OBDII 64
RT Renault Twizy 65
Configuration 65
Custom Metrics 65
Custom Commands 65
TR Tesla Roadster 66
Hardware Requirements 66
Module Installation in the vehicle 66
Antenna Installation 67
Antenna beside rear passenger headrest 67
Antenna on windshield/windscreen 68
Configuration Options 68
Tesla Roadster Notes 69
Thanks 69
TS Tesla Model S 70
Configuration 70
Custom Metrics 70
Custom Commands 70
Template 70
Configuration 71
Custom Metrics 71
Custom Commands 71
XX TRACK 72
OBDII ECU 73
Purpose 73
Cabling 73
Setup 73
Operation 74
Defaults and customization 75
Special handling 76
Metric Scripts 77
External Power Control 78

Appendices **79**
Appendix 1: OVMS Metrics Table 79

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 optionally positioning a cellular antenna. Once connected, the OVMS module enables remote control and monitoring of your car.

This user guide is for the version v3 of OVMS module.

There are four ways for you to communicate with the OVMS module:

1. If you have the cellular option, and a cellular plan that supports SMS messages, you can send text messages from a cell phone to the OVMS module's phone number. The module will respond back via text messaging. If you want, the OVMS module can also send text messages to you when the car reaches certain states, such as if charging is interrupted.

2. Using either a cellular data connection, or WiFi, you can use a smartphone App. Both the OVMS module and the App communicate with an OVMS server via TCP/IP over the Internet. The smartphone Apps provide a richer experience and more functionality than SMS, but they do require a data plan on the SIM card you purchase and install in the OVMS module.

3. Use a laptop/workstation with a USB port to communicate using a serial terminal.

4. Use WiFi and a web browser / telnet / ssh console client.

This Guide will help you setup and configure your OVMS module. Initial configuration of the OVMS v3 module is usually done over WiFi using a Web Browser, telnet or ssh. Once configured, you can use SMS, cell phone Apps, laptops, or WiFi to communicate with the OVMS module.

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.

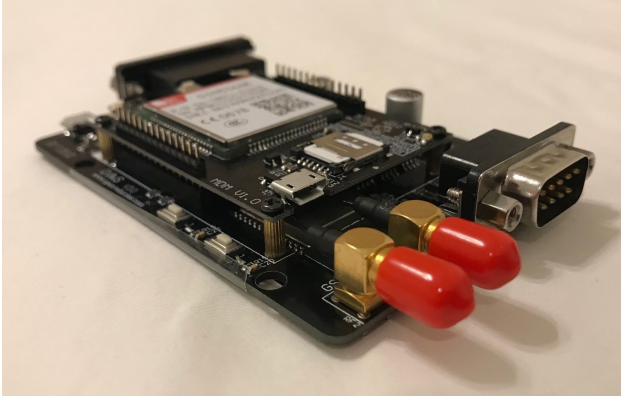
The OVMS module is continuously powered by the car, even when the car is off.

Warning!
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.

OVMS System Components

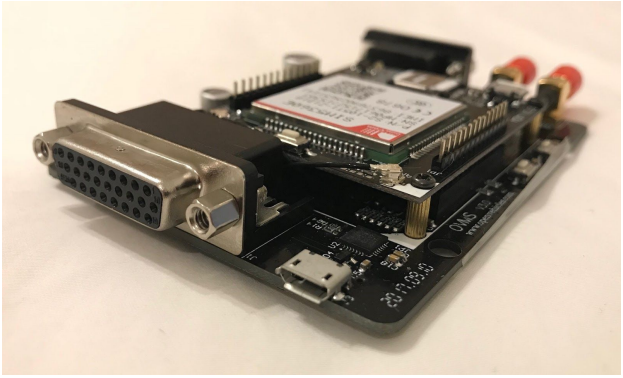
The OVMS Module

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.



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.

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.

Each OVMS modem module is 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.

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”.

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”.

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.

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.

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.

Upgrading from OVMS v1/v2

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.

Installation

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 small screwdriver, for opening the module (if necessary).
3. A micro-usb cable suitable for connecting to your computer.
4. A laptop or desktop computer (if necessary).
5. A cable suitable for connecting to your vehicle.
6. A GSM antenna (if you are using the cellular option).
7. 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.

OVMS v2 Server Registration

Prior to installation, you should create an account, and register your vehicle on one of the public OVMS v2 servers:

1. www.openvehicles.com

   This OVMS v2 server is run by the project. It is hosted in Hong Kong.

2. https://dexters-web.de/

   This OVMS v2 server is hosted in Europe.

During the registration process you will need to decide a VehicleID (unique ID to be identify your vehicle), and a Server Password (password to be used on cellphone apps, servers, and vehicle

modules). The server itself will also have a specific hostname and port it uses for OVMS protocol v2 communications.

Hologram Activation

OVMS has partnered with Hologram and is providing a Hologram SIM pre-installed in every OVMS modem board. In addition, Hologram have provided our community a coupon code valid for US$5 off data usage:

Hologram Coupon Code: OVMS

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 package) from the web or terminal shells 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.

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 sufficient power. We recommend using a USB hub with a separate power supply or a direct port of your laptop / PC.

Server account

If you want to use the OVMS App and/or server based telemetry services, you'll need a 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 OVMS servers:

- Asia-Pacific: https://www.openvehicles.com/
- 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`

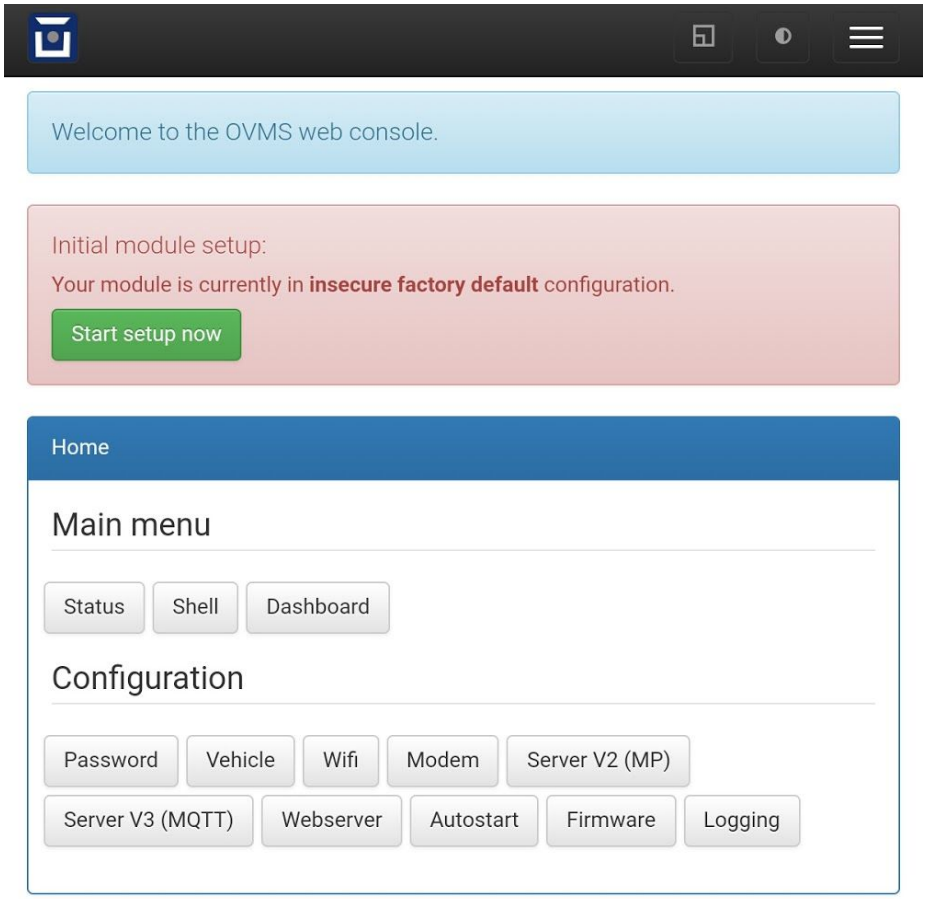
As this is insecure, you should take care not to leave the module running unconfigured.

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.

Launch your web browser, and connect as follows:

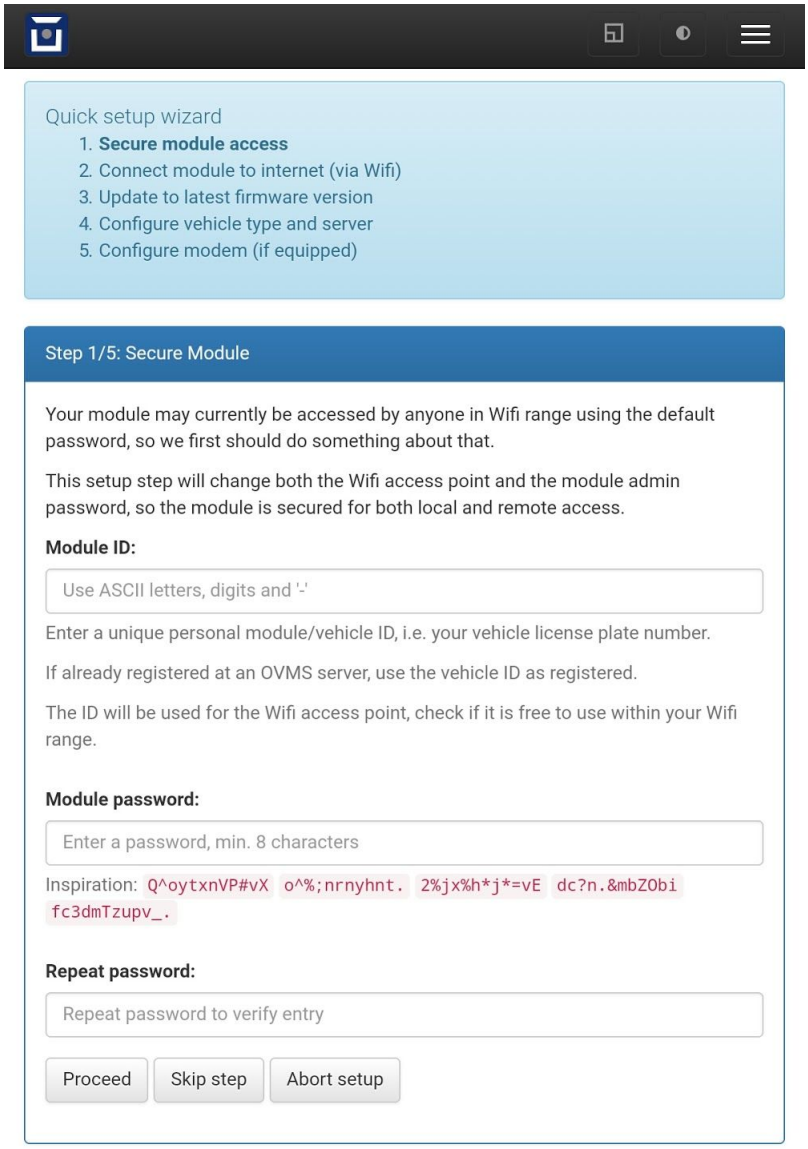
URL: http://192.168.4.1/

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.



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”.

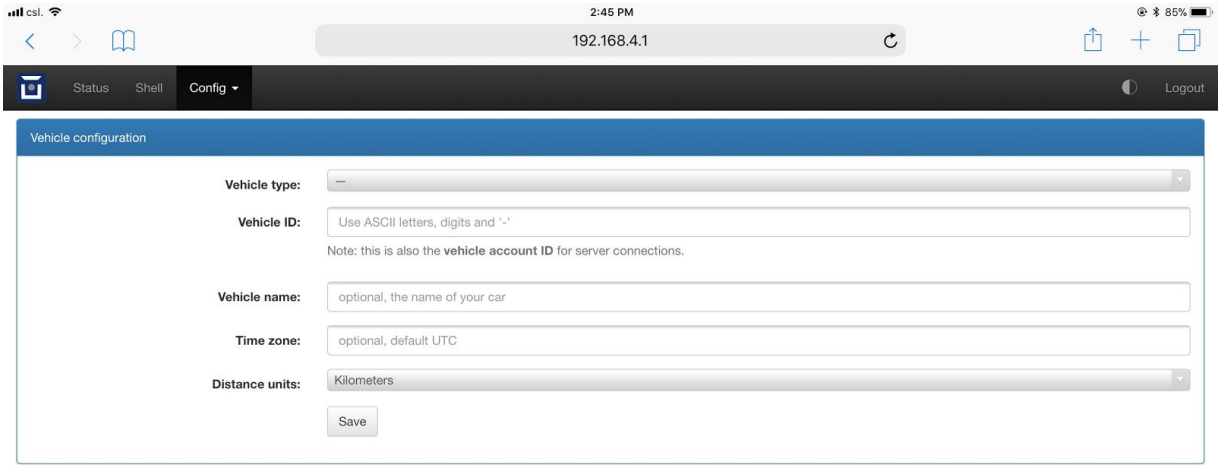
The wizard should be able to restore access after problems occurring in the process. As a last resort if it fails to recover at some point, you can always do a factory reset and start over again.

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

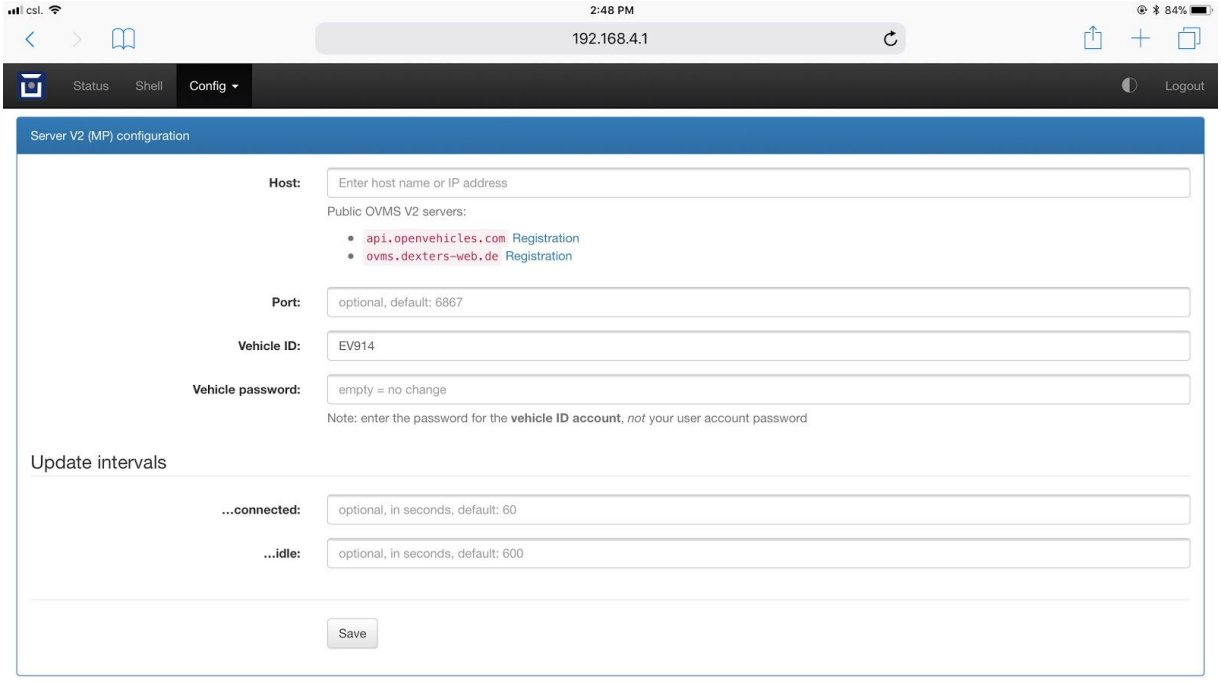
Go to Config / Vehicle:



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 this 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

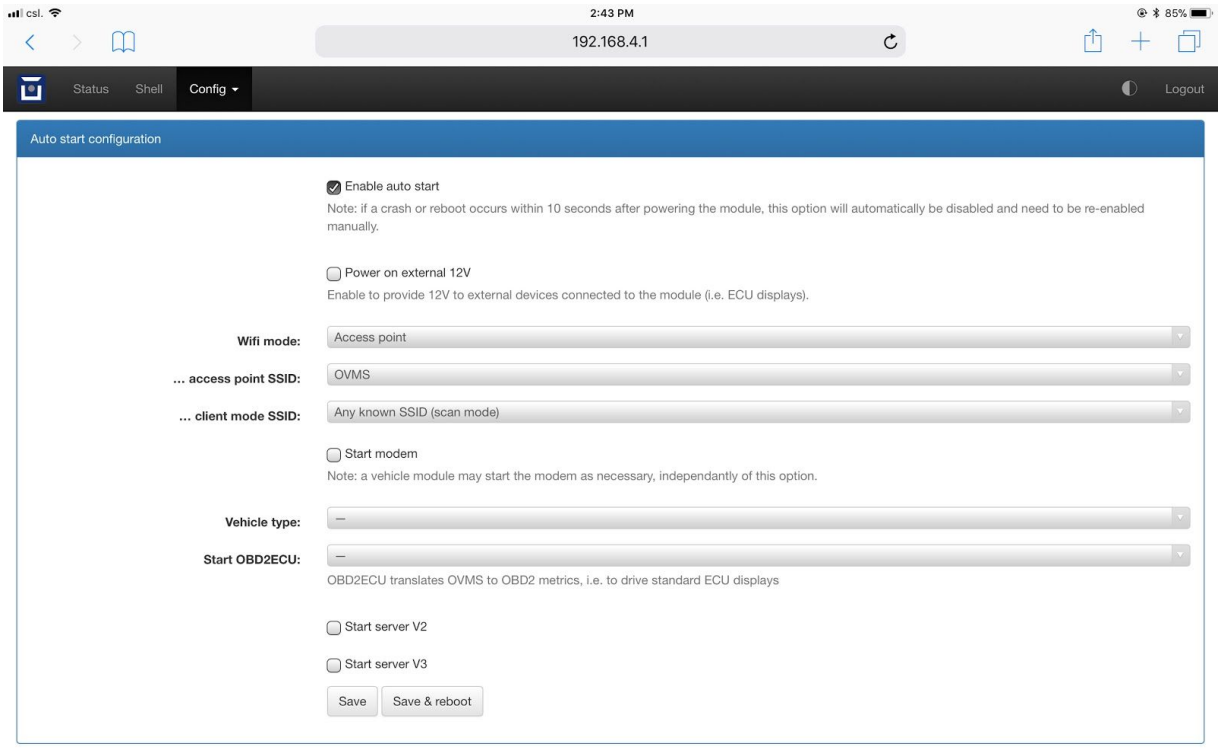
Go to Config / Server V2 to configure the connection to the OVMS v2 server you will be using:



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.



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. Also, **do not use** client scan mode with “AP+Client”, as this is not supported! The web interface will prevent these combinations.

If you have configured this combination manually, the Wifi network will not start automatically. You need to 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

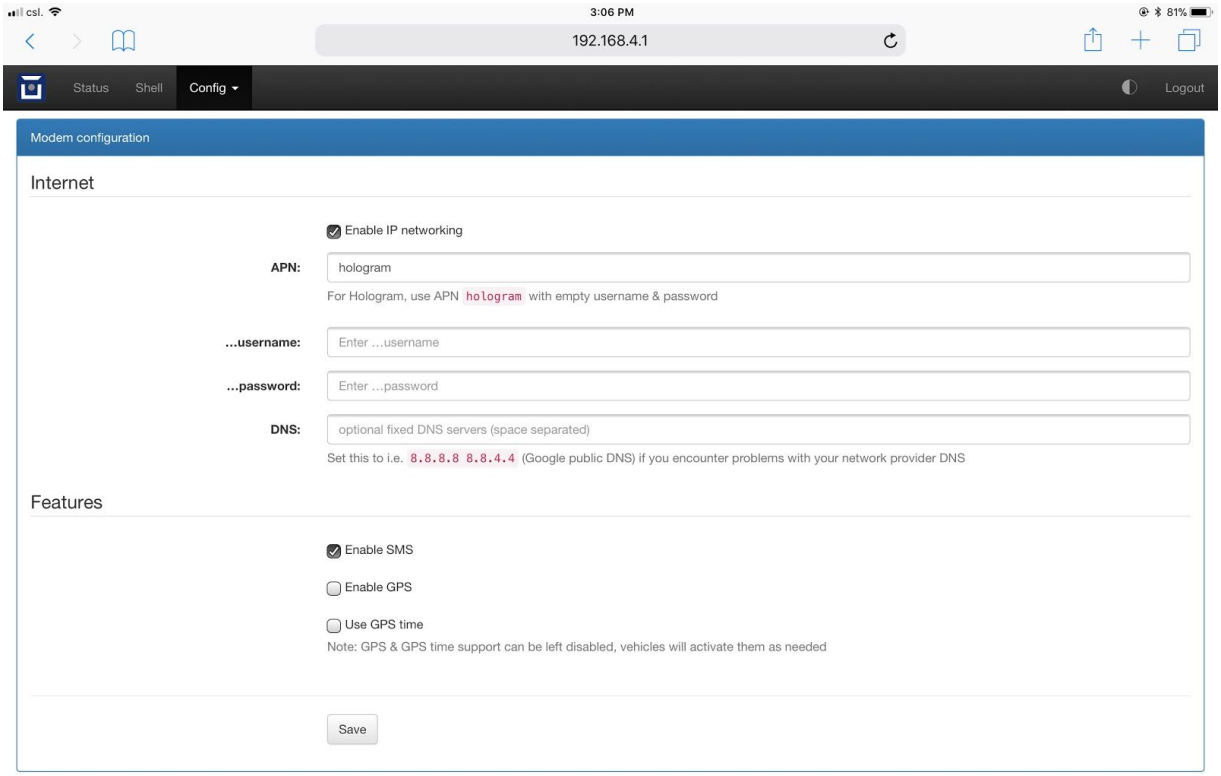
OVMS can connect to a WiFi Access Point, using standard WiFi (802.11 b/g/n) protocols. To connect the SSID (Access Point name) and associated password. In simple client mode, you can connect either to a specific SSID. Alternatively, you can use a scanning client mode to connect to any known WiFi Access 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 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.

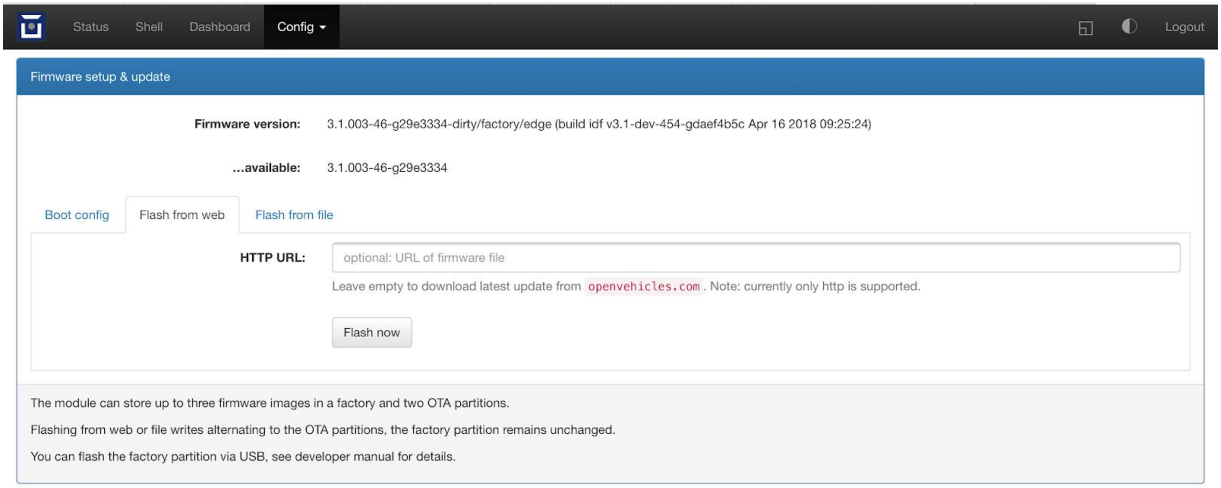


Firmware Update

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.

12V Calibration

The 12V voltage is measured using the incoming voltage that powers the OVMS. You can calibrate it using…

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

The `<factor>` has to be calculated using:

```
oldFactor * (displayedVoltage / 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.

The initial **12V reference voltage** (= fully charged & calmed down voltage level) on startup & after reset can be set by

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

The default reference voltage is `12.6`. The value will be updated automatically if your vehicle supports the `v.e.charging12v` flag.

The **12V alert threshold** can be set by

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

The 12V alert threshold is defined by a relative value to the **12v reference voltage** with a default value of `1.6`. If the actual 12V reading drops below `12v.ref - 12v.alert`, the 12V alert is raised.

Related metrics:

Metric	Example value	Meaning
`v.b.12v.current`	`0.6A`	Momentary current level at the 12V battery
`v.b.12v.voltage`	`13.28V`	Momentary voltage level at the 12V battery

`v.b.12v.voltage.ref`	`12.51V`	Reference voltage of the fully charged & calmed down 12V battery
`v.b.12v.voltage.alert`	`no`	If the 12V critical alert is active (yes/no).
`v.e.charging12v`	`yes`	If the 12V battery is charging or not (yes/no)

Module Configuration

Cell phone App Configuration

Advanced Console Usage

Console Connections

OVMS v3 includes a full 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, a 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 from SILABS website.

   Linux note: if 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.

   a. 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>”.

   b. 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).

   c. 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.

   d. 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.

A note about OpenSSH: with version 6.6, 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`.

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
```

The TAB key can also be used to expand on commands or parameter options:

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

Logging

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 async (USB) console will have log monitoring ‘yes’, but the others ‘no’. Note: the web shell does not support log monitoring yet.

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

The syntax of this command is ‘log level <level> [<component>]’. If the component is not specified, it applies to all components that haven’t had a level set explicitly. 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)
```

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 file” without a file name or “log close”. 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, you can use a script bound to the “sd.mounted” event to re-enable file logging automatically or configure automatic logging.

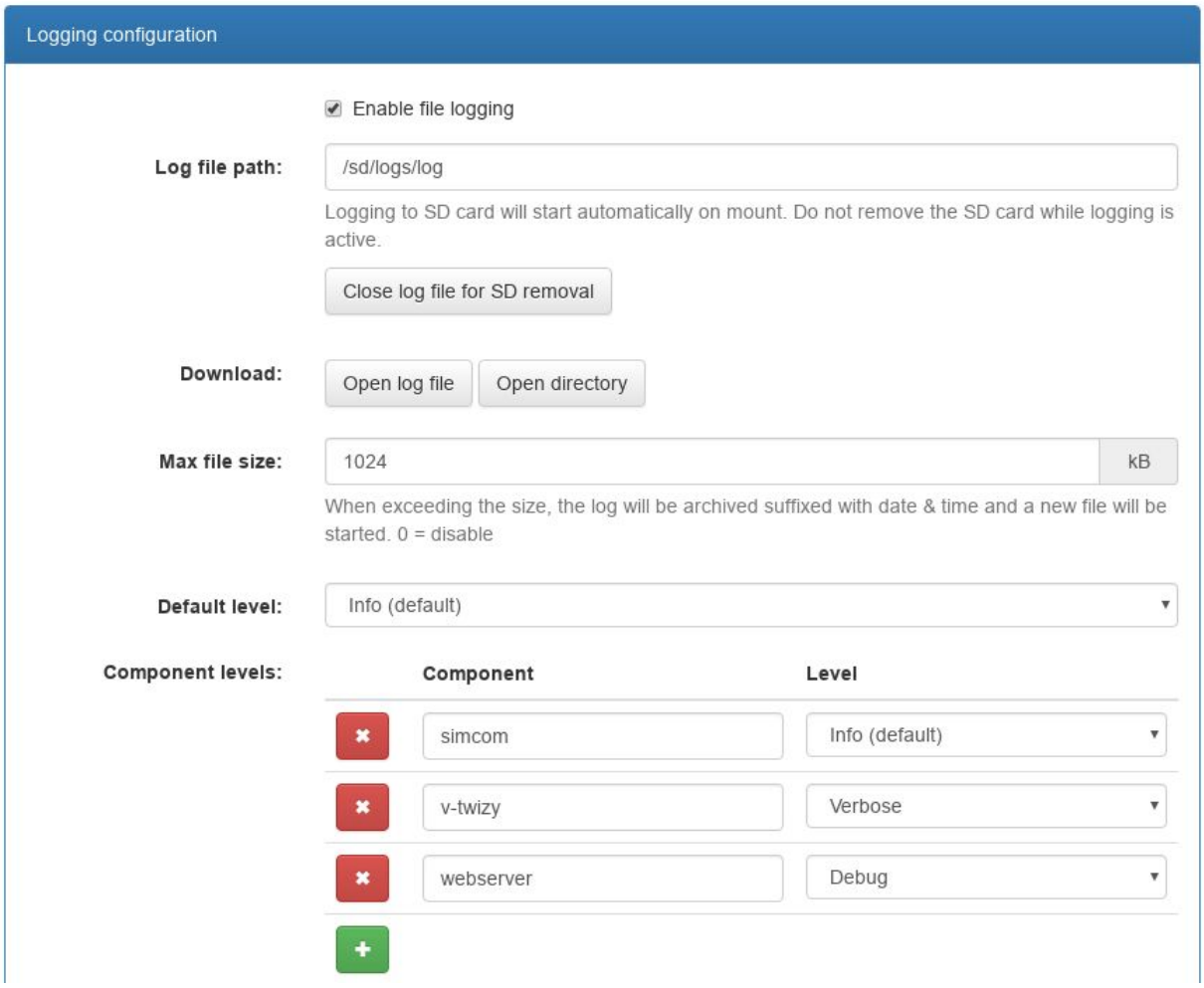
You can use the webserver to view and download the files. The webserver default configuration enables directory listings and access to files located under the document root directory, which is “/sd” by default. Any path not bound to an internal webserver function is served from the document root. So 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.

Logging configuration [3.1.005]

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.maxsize: 1024
  file.path: /sd/logs/log
  level: info
  level.simcom: info
  level.v-twizy: verbose
  level.webserver: debug
```

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.

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.

Virtual File System

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 flexibly and can be used for storing of configuration logs, firmware images, etc.

Over the Air 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)
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
```

Boot Status & Crash Report

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`".

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
```

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.tp.fl.p                                206.843kPa
v.tp.fl.t                                33°C
v.tp.fr.p                                206.843kPa
v.tp.fr.t                                33°C
v.tp.rl.p                                275.79kPa
v.tp.rl.t                                34°C
v.tp.rr.p                                275.79kPa
v.tp.rr.t                                34°C
v.type                                   DEMO
```

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

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
set                 Set the value of a metric
trace               METRIC trace framework
```

Time

Geofenced Locations

Events

Internally, OVMS raises events whenever significant events occur. An event is simply a name, along with associated data. Individual vehicle types may also issue their own events, and custom user events are also possible. Here is the list of standard system events in the OVMS v3 firmware:

Event	Data	Purpose
sd.mounted		The SD card is mounted and ready to use
sd.unmounting		The SD card is currently unmounting
sd.unmounted		The SD card has completed unmounting
sd.insert		The SD card has just been inserted
sd.remove		The SD card has just been removed
location.enter.<name>	<name>	The specified geolocation has been entered
location.leave.<name>	<name>	The specified geolcation has been left
gps.lock.acquired		GPS lock has been acquired
gps.lock.lost		GPS lock has been lost
system.wifi.down		WiFi connection has been lost
app.connected		One or more remote Apps have connected
app.disconnected		No remote Apps are currently connected
server.v2.waitnetwork		V2 server connection is waiting for network
server.v2.connectwait		V2 server is pausing before connection
server.v2.connecting		V2 server connection in progress
server.v2.authenticating		V2 server connection is authenticating
server.v2.connected		V2 server connection established online
server.v2.disconnected		V2 server connection has been lost
server.v2.waitreconnect		V2 server is pausing before re-connection
server.v2.stopped		V2 server has been stopped

server.v3.waitnetwork		V3 server connection is waiting for network
server.v3.connectwait		V3 server is pausing before connection
server.v3.connecting		V3 server connection in progress
server.v3.authenticating		V3 server connection is authenticating
server.v3.connected		V3 server connection established online
server.v3.disconnected		V3 server connection has been lost
server.v3.waitreconnect		V3 server is pausing before re-connection
server.v3.stopped		V3 server has been stopped
vehicle.require.gps		A vehicle has indicated it requires GPS
vehicle.require.gpstime		A vehicle has indicated it requires GPS time
vehicle.type.set	<type>	Vehicle module has been loaded
vehicle.type.cleared		Vehicle module has been unloaded
vehicle.alert.12v.on		12V system voltage is below alert threshold
vehicle.alert.12v.off		12V system voltage has recovered
vehicle.alert.bms		VMS volts/temps exceeded thresholds
vehicle.on		Vehicle has been switched on
vehicle.off		Vehicle has been switched off
vehicle.awake		Vehicle systems are awake
vehicle.asleep		Vehicle systems are asleep
vehicle.charge.start		Vehicle has started to charge
vehicle.charge.stop		Vehicle has stopped charging
vehicle.charge.prepare		Vehicle is preparing to charge
vehicle.charge.finished		Vehicle charge has completed normally
vehicle.charge.pilot.on		Vehicle charge pilot signal is on
vehicle.charge.pilot.off		Vehicle charge pilot signal is off
vehicle.charge.12v.start		Vehicle 12V battery is charging

vehicle.charge.12v.stop		Vehicle 12V battery has stopped charging
vehicle.locked		Vehicle has been locked
vehicle.unlocked		Vehicle has been unlocked
vehicle.valet.on		Vehicle valet mode activated
vehicle.valet.off		Vehicle valet mode deactivated
vehicle.headlights.on		Vehicle headlights are on
vehicle.headlights.off		Vehicle headlights are off
vehicle.alarm.on		Vehicle alarm has been armed
vehicle.alarm.off		Vehicle alarm has been disarmed
vehicle.charge.mode	<mode>	Vehicle charge mode has been set
vehicle.charge.state	<state>	Vehicle charge state has changed
system.modem.gotgps		Modem GPS has obtained lock
system.modem.lostgps		Modem GPS has lost lock
system.modem.poweringon		Modem is powering on
system.modem.poweredon		Modem is powered on
system.modem.muxstart		Modem MUX has started
system.modem.netwait		Modem is pausing before starting DATA
system.modem.netstart		Modem is starting DATA network
system.modem.netloss		Modem has lost DATA network
system.modem.nethold		Modem is pausing DATA network
system.modem.netsleep		Modem is sleeping DATA network
system.modem.netdeepsleep		Modem is deep sleeping DATA network
system.modem.stop		Modem has been shut down
system.modem.received.ussd	<ussd>	A USSD message has been received
network.interface.up		Network connection is established
system.modem.gotip		Modem received IP address from DATA

system.modem.down		Modem has been disconnected
system.shuttingdown		System is shutting down
system.shutdown		System has been shut down
network.wifi.up		WIFI network is up
network.reconfigured		Networking has been reconfigured
network.up		One or more networks are up
network.interface.change		Network interface change detected
network.wifi.down		WIFI network is down
network.down		All networks are down
network.modem.up		Modem network is up
network.modem.down		Modem network is down
network.mgr.init		Network manager has initialised
network.mgr.stop		Network managed has been stopped
config.mounted		Configuration is mounted and available
config.unmounted		Configuration is unmounted and unavailable
config.changed		Configuration has changed
housekeeping.init		Housekeeping has initialised
system.start		System is starting
ticker.1		One second has passed since last ticker
ticker.10		Ten seconds has passed
ticker.60		One minute has passed
ticker.300		Five minutes have passed
ticker.600		Ten minutes have passed
ticker.3600		One hour has passed

Notifications

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.

Note that the developer building firmware can optionally set the OVMS_DEV_SDCARDSCRIPTS build flag. If that is set, then the system will also check /sd/scripts and /sd/events for scripts.

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.

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.

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/script/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
```

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(',');
        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);
```

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) work directly).

Internal Modules

The JSON module is provided with a single ‘print’ method, to print out a given javascript object in JSON format.

The PubSub module is provided to provide 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.

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. These include:

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.

OvmsCommand

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
```

OvmsLocation

The OvmsLocation object exposes one method “Status”. This method is passed a single parameter as the location name. It returns true if the vehicle is currently in that location’s geofence, false if not, or undefined if the location name passed is not valid.

OvmsMetrics

The OvmsMetrics object exposes the Value() and AsFloat() methods. Both are passed a single string as the metric name to lookup, and return the metric value as strings or floats appropriately.

OvmsVehicle

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

- Type() to return the type of the currently loaded vehicle module
- Wakeup() to wakeup the vehicle (return TRUE if successful)
- Homelink(button,durationms) to fire the given homelink button
- ClimateControl(onoff) to turn on/off climate control
- Lock(pin) to lock the vehicle
- Unlock(pin) to unlock the vehicle
- Valet(pin) to activate valet mode
- Unvalet(pin) to deactivate valet mode
- SetChargeMode(mode) to set the charge mode
- SetChargeCurrent(limit) to set the charge current limit
- SetChargeTimer(onoff, start) to set the charge timer
- StartCharge() to start the charge
- StopCharge() to stop the charge
- StartCooldown() to start a cooldown charge
- StopCooldown() to stop the cooldown charge

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

```
OVMS# script eval 'JSON.print(this)'
{
  "assert": function () { [native code] },
  "print": function () { [native code] },
  "OvmsCommand": {
    "Exec": function Exec() { [native code] }
  },
  "OvmsLocation": {
    "Status": function Status() { [native code] }
  },
  "OvmsMetrics": {
    "AsFloat": function AsFloat() { [native code] },
    "Value": function Value() { [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] },
    "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": {
    "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] }
  }
}
```

Module Factory Reset

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.

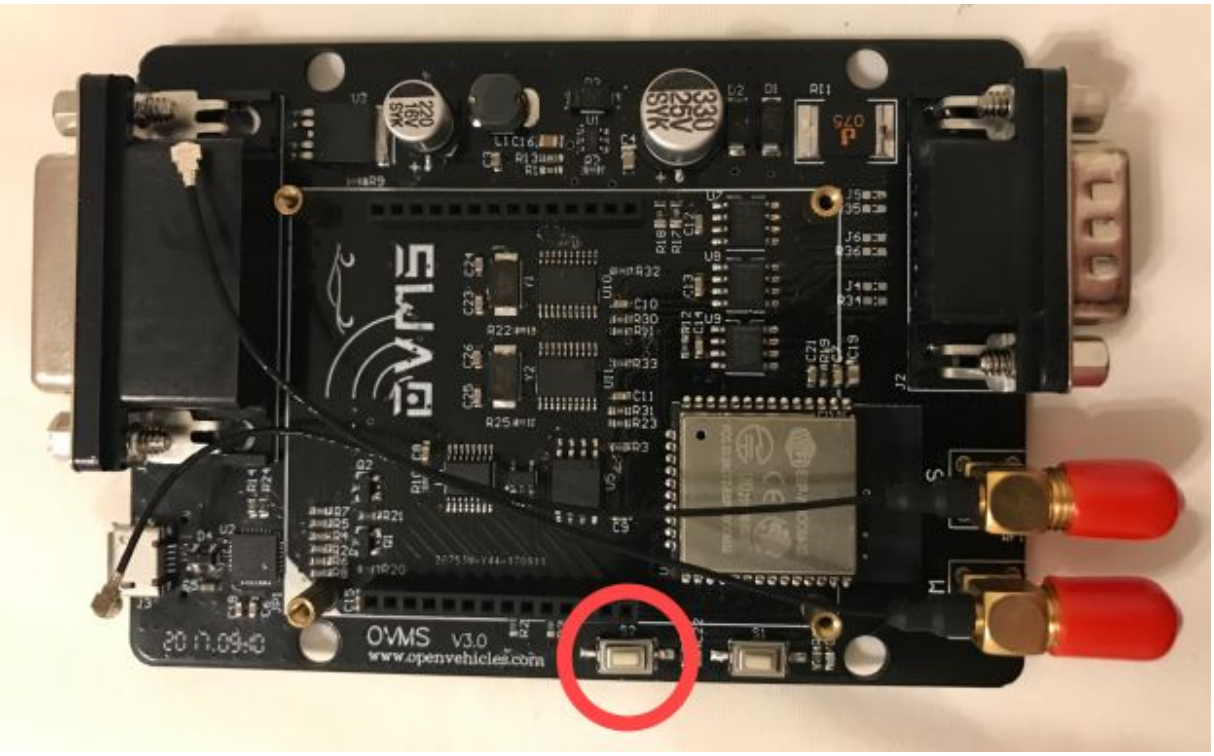
If you don't have console access, you can perform a factory reset by placing an empty file named "`factoryreset.txt`" in the root directory of an SD card and insert that SD into the (running) module. The file will be deleted and the module will reboot within about 30 seconds.

If you don't have console access (lost module password) and don't have an SD card, you can perform a factory reset of the configuration store using the esptool.py tool from the Espressif ESP-IDF toolkit:

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

Note: the device needs to be changed to the one assigned by your system, i.e. /dev/ttyUSB0 on a Linux system. After using esptool.py to manually erase_region, you should go into the console and do the 'module factory reset' step to properly factory reset.

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



After either of these methods, 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 as soon as possible, as the module is accessable by anyone knowing the initial password.

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.

Install 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
```

Vehicle Documentation

DEMO Demonstration vehicle

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.

KS Kia Soul EV

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.

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.

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.

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.

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
```

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.

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:

- 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
- 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.

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.

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.

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 of or off IGN1-relay. Can be used to wake up part of the car.

xks IGN2 <on/off><pin>

Turn of or off IGN2-relay. Can be used to wake up part of the car.

xks ACC <on/off><pin>

Turn of or off ACC-relay. Can be used to wake up part of the car.

xks START <on/off><pin>

Turn of 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.

ECU-write commands

These commands are for the extra playful people. Use with caution.

xks sjb <b1><b2><b3>
Send command to smart junction box.

xks bcm <b1><b2><b3>
Send command to body control module.

NL Nissan Leaf

Configuration

2011-2015 models

To enable remote commands, either

1. Unplug any CARWINGS, Nissan Connection or TCU units[1] or
2. on Generation 1 Cars, wire the RC3 to TCU pin 11 (see MyNissanLeaf post)

2016-2017 models

Set the model year as follows[2] and if necessary configure 30 kwhr model

```
config set xnl modelyear 2016
or
config set xnl modelyear 2017
```

30 kwh models

For models with a 30 kwhr battery pack, set the capacity manually as follows

```
config set xnl maxGids 356
config set xnl newCarAh 79
```

Custom Metrics

TBC

Custom Commands

TBC

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.

[1] If a CARWINGS, Nissan Connect or the TCU hardware is fitted, the OVMS messages will be overridden by the TCU and when the OVMS tries to wake the car, it will wake up and then go back to sleep.
[2] In model year 2016 Nissan changed how remote commands are handled, switching from the EV bus to the Car bus and changing the messages. The OVMS defaults to pre-2016 behaviour.

Range Calculation:

The OVMS uses 2 configuration options to calculate remaining range, whPerGid (default 80Wh/gid) and kmPerKWh (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.

O2 OBDII

RT Renault Twizy

Configuration

Custom Metrics

Custom Commands

TR Tesla Roadster

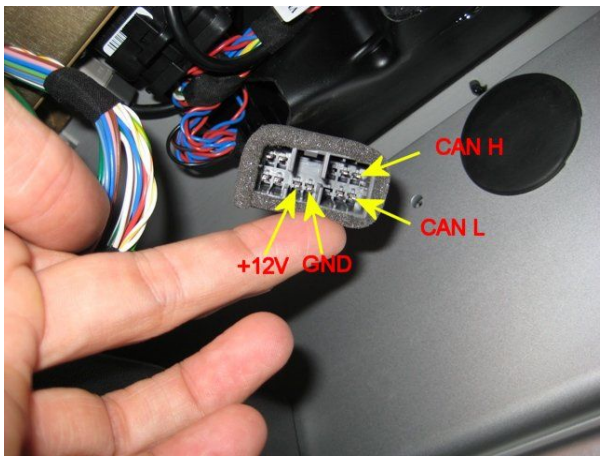
The Tesla Roadster support in OVMS is perhaps the most mature in the project. All versions (1.x, 2.x, and 3.x) are supported, for both North American and other variants.

Hardware Requirements

The following hardware is required:

- OVMS v3 module (and optional modem, if required)
- OVMS Data Cable for Early Teslas
- OVMS GSM Antenna (or other similar antenna)
- Adhesive velcro strips (or other similar)

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.

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).

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.

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).

Configuration Options

The Tesla Roadster specific configuration options are in configuration parameter ‘xtr’:

Instance	Default	Description
digital.speedo	no	Set to ‘yes’ to enable digital speedometer
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

Tesla Roadster Notes

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:

1. 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.

2. 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.

3. 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.

4. 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.

5. The digital speedometer function replaces the AMPS display in the dashboard instrument cluster with the vehicle speed. 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.

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.

TS Tesla Model S

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.

Configuration

Custom Metrics

Custom Commands

Template

Configuration

Custom Metrics

Custom Commands

XX TRACK

OBDII ECU

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 the 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.

Cabling

The cable and OBDII connector are not provided with the OVMSv3 module, and can be built using the diagram below. In future, a standard cable will be available for this, and this guide updated when that happens.

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

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

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

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.

Defaults and customization

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

PID		Requested Parameter	Mapped Metric value	Metric Name
Dec	Hex			
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

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

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*')

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=OvmsMetricFloat("v.p.speed");
ret2=OvmsMetricFloat("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.

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
```

Appendices

Appendix 1: OVMS Metrics Table

The following is a table of the various metrics that the OVMSv3 knows about, and the cars which support each of the metrics. The Metric Name is used by the OBDII ECU task for identifying metrics that are to be reported to an attached OBDII Device (HUD, dongle, etc.).

		Metric is Mappable to OBDII	Vehicles Supported			
Metric Value	**Metric Name**		**Tesla Roadster**	**Nissan Leaf**	**Renault Twizy**	**Kia Soul**
Software version string	m.version		✓	✓	✓	✓
Hardware version string	m.hardware		✓	✓	✓	✓
WiFi MAC Address	m.serial		✓	✓	✓	✓
# of active tasks	m.tasks	✓	✓	✓	✓	✓
Free memory (bytes)	m.freeram	✓	✓	✓	✓	✓
Seconds since Boot	m.monotonic	✓	✓	✓	✓	✓
Current time (UTC)	m.time.utc		✓	✓	✓	✓
Network type - none, wifi, gsm	m.net.type		✓	✓	✓	✓
Network signal quality [%?]	m.net.sq		✓	✓	✓	✓
Network provider name	m.net.provider		✓	✓	✓	✓
Modem ICCID	m.net.mdm.iccid		✓	✓	✓	✓
Modem model string	m.net.mdm.model		✓	✓	✓	✓
True = V2 server connected [1]	s.v2.connected		✓	✓	✓	✓
V2 clients connected [1]	s.v2.peers		✓	✓	✓	✓
True = V3 server connected [1]	s.v3.connected		✓	✓	✓	✓
V3 clients connected [1]	s.v3.peers		✓	✓	✓	✓
Vehicle type code	v.type		✓		✓	✓
		Metric is Mappable	**Vehicles Supported**			

Metric Value	Metric Name	to OBDII	Tesla Roadster	Nissan Leaf	Renault Twizy	Kia Soul
DMV Vehicle identification number	v.vin	See Privacy in OBDII ECU	✓	✓	✓	✓
State of charge [%]	v.b.soc	✓	✓	✓	✓	✓
State of health [%]	v.b.soh	✓		✓	✓	✓
Calculated capacity [Ah]	v.b.cac	✓	✓	✓	✓	✓
Main Battery momentary consumption [Wh/km]	v.b.consumption	✓	✓	✓	✓	✓
Main battery momentary voltage [V]	v.b.voltage	✓		✓	✓	✓
Main battery momentary current [A]	v.b.current	✓	✓	✓	✓	✓
Main battery coulomb used on trip [Ah]	v.b.coulomb.used	✓			✓	
Main battery coulomb recovered on trip [Ah]	v.b.coulomb.recd	✓			✓	
Main battery momentary power [kW]	v.b.power	✓	✓		✓	✓
Main battery energy used on trip [kWh]	v.b.energy.used	✓			✓	✓
Main battery energy recovered on trip [kWh]	v.b.energy.recd	✓			✓	✓
Ideal range at 100% SOC & current conditions [km]	v.b.range.full	✓			✓	✓
Ideal range [km]	v.b.range.ideal	✓	✓	✓	✓	✓
Estimated range [km]	v.b.range.est	✓	✓		✓	✓
Auxiliary 12V battery momentary voltage [V]	v.b.12v.voltage	✓			✓	✓
Auxiliary 12V battery reference voltage [V]	v.b.12v.voltage.ref	?			✓	✓
Auxiliary 12V battery voltage alert status [Bool]	v.b.12v.voltage.alert	?			✓	✓

		Metric is Mappable to OBDII	Vehicles Supported			
Metric Value	**Metric Name**		**Tesla Roadster**	**Nissan Leaf**	**Renault Twizy**	**Kia Soul**
Auxiliary 12V battery momentary current [A]	v.b.12v.current	✓			✓	
Battery temperature [°C]	v.b.temp	✓	✓	✓	✓	✓
Momentary charger supply voltage [V]	v.c.voltage	✓	✓	✓ battery side	✓	✓
Momentary charger output current [A]	v.c.current	✓	✓	✓ Battery side	✓	✓
Maximum charger output current [A]	v.c.climit	✓	✓	✓	✓	✓
Duration of running charge [sec]	v.c.time	✓			✓	✓
Energy sum for running charge [kWh]	v.c.kwh	✓	✓		✓	✓
standard, range, performance, storage	v.c.mode				✓	✓
True if timer enabled	v.c.timermode		✓			✓
Time timer is due to start	v.c.timerstart		✓			✓
charging, topoff, done, prepare, timerwait, heating, stopped	v.c.state		✓		✓	✓
scheduledstop, scheduledstart, onrequest, timerwait, powerwait, stopped, interrupted	v.c.substate		✓		✓	✓
undefined, type1, type2, chademo, roadster, teslaus, supercharger, ccs	v.c.type			✓		✓
Pilot signal present	v.c.pilot		✓		✓	✓
True = currently charging	v.c.charging		✓	✓	✓	✓
Sufficient range limit for current charge [km]	v.c.limit.range				✓	✓
Sufficient SOC limit for current charge [%]	v.c.limit.soc				✓	✓

		Metric is Mappable to OBDII	Vehicles Supported			
Metric Value	**Metric Name**		**Tesla Roadster**	**Nissan Leaf**	**Renault Twizy**	**Kia Soul**
Estimated time remaining for full charge [min]	v.c.duration.full	✓			✓	✓
… for sufficient range [min]	v.c.duration.range	✓			✓	✓
… for sufficient SOC [min]	v.c.duration.soc	✓			✓	✓
Charger temperature [°C]	v.c.temp	✓	✓		✓	✓
Inverter temperature [°C]	v.i.temp	✓	✓	✓	✓	✓
Motor speed (RPM)	v.m.rpm	✓	✓		✓	✓
Motor temperature [°C]	v.m.temp	✓	✓		✓	✓
Door open flag: Front Left	v.d.fl		✓	✓		✓
Door open flag: Front Right	v.d.fr		✓	✓		✓
Door open flag: Rear Left	v.d.rl			✓		✓
Door open flag: Rear Right	v.d.rr			✓		✓
Charge port open flag	v.d.cp		✓		✓	✓
Hood open flag	v.d.hood		✓			
Trunk open flag	v.d.trunk		✓	✓		✓
Active drive profile number [1]	v.e.drivemode				✓	
Gear	v.e.gear			✓	✓	
Drive pedal state [%]	v.e.throttle	✓		✓	✓	
Brake pedal state [%]	v.e.footbrake	✓		✓	✓	
Handbrake Set flag	v.e.handbrake		✓	✓	✓	✓
Car is Awake flag	v.e.awake		✓	✓	✓	✓
Car is charging 12v battery flag	v.e.charging12v				✓	✓
Car cooling flag	v.e.cooling		✓	✓		✓
Car heating flag	v.e.heating			✓		✓
Car HVAC is on flag	v.e.hvac	✓	✓	✓		✓
Car is On flag	v.e.on		✓	✓	✓	✓

Car is Locked flag	v.e.locked		✓	✓	✓	✓
Car in Valet Mode flag	v.e.valet		✓		✓	
Car headlights are on flag	v.e.headlights			✓		✓
Car alarm is set flag	v.e.alarm		✓			
Seconds since car was parked	v.e.parktime	✓			✓	✓
ms_v_env_ctrl_login	v.e.c.login				✓	
ms_v_env_ctrl_config	v.e.c.config				✓	
Ambient temperature [°C]	v.e.temp	✓	✓	✓		✓
Cabin temperature [°C]	v.e.cabintemp	✓		✓		
GPS has satellite lock flag	v.p.gpslock		✓		✓	✓
	v.p.gpsstale					✓
GPS Mode N/A/D/E	v.p.gpsmode				✓	✓
Horizontal dilution of precision (smaller=better)	v.p.gpshdop	✓			✓	✓
# of GPS Satellites	v.p.satcount	✓			✓	✓
Position latitude	v.p.latitude	✓	✓		✓	✓
Position Longitude	v.p.longitude	✓	✓		✓	✓
Direction of travel	v.p.direction	✓	✓		✓	✓
GPS Altitude	v.p.altitude	✓	✓		✓	✓
Vehicle speed [kph]	v.p.speed	✓	✓	✓	✓	✓
GPS speed over ground [kph]	v.p.gpsspeed	✓			✓	✓
Vehicle odometer [km]	v.p.odometer	✓	✓	✓	✓	✓
Vehicle trip odometer [km]	v.p.trip	✓	✓		✓	✓
Tire Front Left temperature	v.tp.fl.t		✓			✓
Tire Front Right temperature	v.tp.fr.t		✓			✓
Tire Rear Right temperature	v.tp.rr.t		✓			✓
Tire Rear Left temperature	v.tp.rl.t		✓			✓
		Metric is Mappable to OBDII	**Vehicles Supported**			
Metric Value	**Metric Name**		**Tesla**	**Nissan**	**Renault**	**Kia**

			Roadster	Leaf	Twizy	Soul
Tire Front Left pressure	v.tp.fl.p		✓	✓		✓
Tire Front Right pressure	v.tp.fr.p		✓	✓		✓
Tire Rear Right pressure	v.tp.rr.p		✓	✓		✓
Tire Rear Left pressure	v.tp.rl.p		✓	✓		✓
Vehicle Specific: “gids”	xnl.v.bat.gids			✓		
Vehicle Specific: “hx”	xnl.v.bat.hx			✓		

COMMANDS

Command Summary & number	Command parameters	Vehicles Supported			
		Tesla Roadster	Nissan Leaf	Renault Twizy	Kia Soul
CMD_QueryFeatures - 1					
CMD_SetFeature - 2	(feature number, value)				
CMD_QueryParams - 3					
CMD_SetParam - 4	(param number, value)				
CMD_Reboot - 5					
CMD_Alert - 6					
CMD_Execute - 7	(text command with arguments)				
CMD_SetChargeMode - 10					
CMD_StartCharge - 11			✓		
CMD_StopCharge - 12			X		
CMD_SetChargeCurrent - 15	(amps)		X		
CMD_SetChargeModeAndCurrent - 16	(mode, amps)		X		
CMD_SetChargeTimer - 17	(mode, start time)				
CMD_WakeupCar - 18			(2013-) ✓ (2011/2) X [3]		
CMD_WakeupTempSystem - 19					
CMD_Lock - 20					
CMD_ValetOn - 21			X		
CMD_UnLock - 22					
CMD_ValetOff -23			X		
CMD_Homelink - 24	(button_nr)		X		

[3] Needs addition of wake-up hardware

CMD_CoolDown - 25			X		
CMD_ClimateControl - 26	(mode)		Partial see config		

FCC Warning

This device complies with part 15 of the FCC rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any inte rference received, including interference that may cause undesired operation.

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interferenceto radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Reorient or relocate the receiving antenna.
- Reorient or relocate the receiving antenna.
- Consult the dealer or an experienced radio/TV technician for help important announcemen

Radiation Exposure Statement

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment. This equipment should be installed and operated with minimum distance 40cm between the radiator and your body.