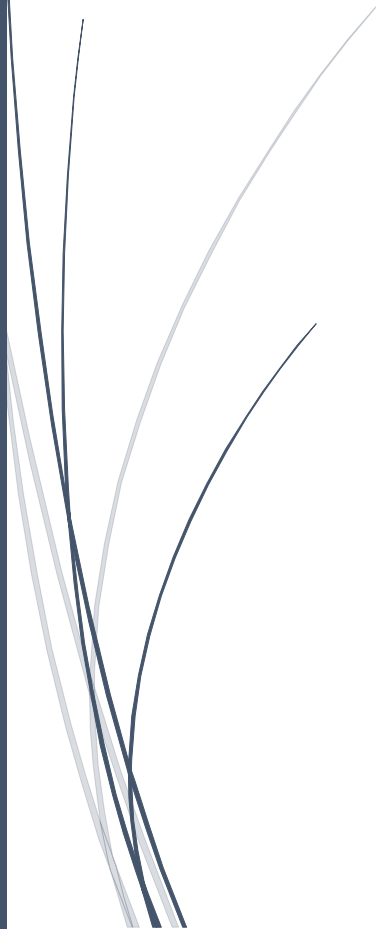
11/29/2016

Project-Documentation

Opendriverslog



Content

1. Schema

1.1 Project status

This is the project documentation for Opendriverslog. The project is not finished so there are some limitations that you should keep in mind. The frontend is done by around 90% and still has some bugs. All in all, in works fine though. The PDF export for the drivers logbooks is not finished ether and still needs some work. The automatic upload is in a similar state. We had some working tests with automated upload of GPS-data though. Current tracks are based on GPS-data only, getting the mileage from cars differs from model to model and needs to be implemented for most models, also it is currently not used in the Web-App. With these limitations in mind we put our code to the test and it worked quite well.

1.2 How it works

First of all, you need to put your device in your car. Just connect it to you OBDII port. It should be located close to the steering wheel (in a 1meter radius). These ports are standardized since 2008. After you built your device correctly (take a look at 2.2 Scripts and 3.1 Hardware) and is connected to your OBDII port in your car, it starts working and listens to your car data. While driving it saves your position and your odometer values. Next you just need to load the data to the server via Wi-Fi. Just log into the webapp and download your drivers logbook.

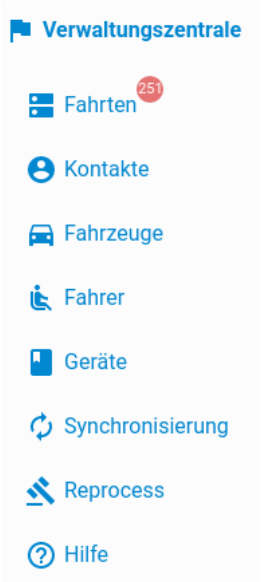
1.3 First time with the device

As already mentioned you need to connect the device to your OBDII port in your car. Normaly it is located under the dashboard but it also can be inside the armrest. In case you have trouble finding the port just use google and look for the OBDII location for your car.

After you put it in it starts working. In case you have got enough data just drive into your Wi-Fi (3.1 WRTNode explains how to setup you Wi-Fi device) and load the data to your server. The data will be loaded automatically by using curl.

1.4 Webapp

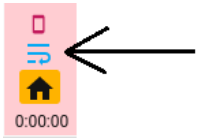
Set up the server as explained in the project readme and call your ODL website via your browser. Register yourself and configure your account. Click on “Verwaltungszentrale” (administration central). This opens this side panel:



- Fahrten: Shows your trips and lets you edit them.
- Kontakte: Here are your contacts. They will be shown on your map.
- Fahrzeuge: Shows your vehicles and lets you edit them.
- Fahrer: Shows your drivers and lets you edit them.
- Geräte: Shows your OBDII devices and lets you edit them.
- Synchronisierung: Synchronizes smartphone contacts with webapp.
- Reprocess: Only visible as admin. Reprocesses the trips.
- Hilfe: Shows helpful tips

You can fuse some trips together to one by clicking on the blue arrow that shows to another trip.

Also you can fuse several trips by marking them and combining them. Just klick on the trips you like to fuse (they get a darker grey background) and then click on the blue symbol on the right.



2. Software

2.1 Webapp

How to setup the server and install the webapp is explained in the project readme.

2.2 Scripts

To understand the scripts you need some knowledge about CAN-Bus and OBDII. You can use Wikipedia for a basic understanding of the OBDII port. There are alot of information you can get out of your car just by sending the right PID. A list of those is here: https://en.wikipedia.org/wiki/OBD-II_PIDs

To get the cardata we used a STN1110. For an explanation take a look at 3.2 Car reading chip. Note that the STN1110 first has a baud rate of 9600. The baud rate is the rate how fast it sends data to you Linux system (important for communication). The problem is that this baudrate is far slower than your CAN-Bus. With the command “ATMA” you get the whole CAN-Bus print. With this low baudrate

soon you will get a buffer overflow error. So if you want to use this bus you need to raise the baudrate. Unfortunately, this does not work with lua so far. This is why we use python here. Connect the STN1110-Board to your Linux system and change the baudrate with the /Prototype-scripts/sendCommands.py script. (Note that you can use a normal laptop for doing this by using a USB UART Adapter)

- Cd to the /Prototype-scripts/ folder
- Use this command: python sendCommands.py 9600 ATZ STSBR500000 STWBR ATZ

The sendCommands.py script send all parameters as command to your car. So this is what it does:

- Python sendCommands.py => start the script as python script
- 9600 => do it with a baudrate of 9600
- ATZ => This command resets the STN1110 (to test if the connection works)
- STSBR500000 => Set as new baudrate 500000 (the highest that the STN1110 can use. Also this command changes the communication baudrate between Linux system to the STN1110-Board to 500000)
- STWBR => Save the new budrate
- ATZ: Reset the STN1110. Also this shows that the new baudrate works

After the new baudrate is set you can use the scripts. Those are located under /Prototype-scripts/install/prod/.

Example: Atma.lua: It sends continuously the ATMA command to your STN1110-Board with the "libFunc.sendCommand"-command. This command is loaded into the script via the libODL.lua (the library). It listens to the CAN-Bus until it finds a known identifier. Known identifiers as well as there calculations are located under /Prototype-scripts/install/prod/canCalc/<CARTYPE>/<ID>.lua

There are already example calculations for a Ford and the Toyota ImieV.

3. Hardware

3.1 WRTNode

You can use any Linux system you want like a laptop for example. We used the WRTNode though (which is the reason why we used lua as programming language since it doesn't need much space on the WRTNode). It is small, has integrated Wi-Fi, a USB socket, UART connectors and a good documentation. You can take a look here: http://wiki.wrtnode.com/index.php?title=Main_Page (specially note the **starting guide**). In case you want to use the WRTNode you will need our firmware (located under /Prototype-scripts/Firmware/…sysupgrade.bin). Use this documentation for the upgrade: http://wiki.wrtnode.com/index.php?title=Refresh_the_firmware

Next you need to restore our defaults. In your browser call 192.168.8.1. Now go to "System => Backup / Flash Firmware" as before. This time though you don't use "Flash Image" but "Upload archive" and upload the tar.gz file under Prototype-scripts/Defaults/backup-OpenWrt-2016-07-26.tar.gz.

After running through the starting guide you need to execute the install shell script under /Prototype-scripts/install.sh. It should copy all necessary files to your WRTNode. Lastly install these needed packages:

- opkg update
- opkg install coreutils-stty usbutils screen luasocket curl

A small explanation for the packages:

- corteutils-stty: has all the needed adjustments for UART connection
- usbutils: helpful for usb devices like memory sticks (since WRTNode doesn't have much memory)
- screen: we use screen sessions for our programs that we can call any time to look at them without disturbing the process
- luasocket: helpful for better timestamp and brings some good extensions for lua like sleep
- curl: for sending a file automatically to a server

3.2 GPS-Device

We tried two working ways of gathering GPS-data.

In case you want your WRTNode to do everything you just need to connect a GPS-device to the USB port. Simple as that. Then start the script /Prototype-scripts/startGpsCat.sh. Note that you need to set the right conditions (like baudrate) for your device.

Secondly you can just use your phone and download your GPS-data with google maps.

3.3 Car reading chip

As hardware we created a device using the STN1110 chip. Follow the STN1110 guide to build you own. Due to legal reasons we cannot show our device in this documentation. Detailed guides for the STN1110 use these links:

Multiprotocol OBD to UART Interpreter Datasheet:
https://www.scantool.net/scantool/downloads/97/stn1110-ds.pdf

Family Reference and Programming Manual:
https://www.scantool.net/scantool/downloads/98/stn1100-frpm.pdf

STN1110 vs ELM327 Comparison:
https://www.scantool.net/scantool/downloads/102/stn1110_vs_elm327.pdf

PowerSave Functionality: https://www.scantool.net/scantool/downloads/79/stn11xx-powersave.pdf