Missing Miles: Detecting mileage fraud via remote diagnostics

Bachelor of Science Graduation thesis in Automotive Engineering

ROHAAN GHOSH

Vehicle Electronics and Control
Faculty of Automotive Engineering
HAN UNIVERSITY OF APPLIED SCIENCES
Arnhem, Netherlands 2020

Bachelor's Graduation Thesis Report, 2020

Missing Miles: Detecting mileage fraud via remote diagnostics

Bachelor of Science Graduation thesis in Automotive Engineering

Final report presented for the Graduation thesis at

JIFELINE B.V.

Rohaan Ghosh

Arnhem, Netherlands, 2020.

University Supervisor:

Ellen Wesselingh

Faculty of Automotive Engineering

HAN University of Applied Sciences

Company Supervisor:

Frank Bouman

Jifeline B.V.

June 8^{th}, 2020.

Preface

This is the final report written during my graduation internship at Jifeline B.V. as my Bachelor's thesis. Mileage fraud commonly known as **'clocking'** (in the United Kingdom), is prevalent in the second-hand car market within European Union (EU), especially when cars are imported and exported to and from member states. It is difficult or near impossible to detect unless using diagnostic tools from Original Equipment Manufacturer (OEM) service centres or dealerships. Another means of detect clocking, is to have a system to keep a vehicle's mileage values in a database recorded over time. The above mentioned system is only in place for a couple of member states within the EU. If no check is carried out, a vehicle that has been clocked can be traded in the second-hand market without the buyer realising that the mileage value in his newly bought second-hand vehicle has been falsified. This report presents an alternative by means of an automated function via remote vehicle diagnostics to detect clocking.

Acknowledgement

I would like to express my appreciation and gratitude to **Jifeline B.V.** for providing me with the opportunity to perform my Graduation Thesis Assignment at their company in an innovative and professional environment. A special thanks to my company supervisor Frank Bouman for his guidance throughout this project. I would like to take this opportunity to thank all persons concerned at Jifeline, FactorIT and Car Lock Systems who have helped and acknowledged my work during this project. A mentioned to my supervising lecturer Ellen Wesselingh for her supervision and help during the course of this project.

Rohaan Ghosh
Arnhem, 08-06-2020

Summary

Problem The problem addressed in this report is the **mileage manipulation** of cars in the second-hand car market. This issue occurs within the EU, typically in vehicles being imported and exported. It is caused by lack of information on accuracy of odometer reading which leads to a negative impact on consumer confidence in this market. It also causes losses to consumers and other third party organisation in terms of product quality and economic value.

At Jifeline remote diagnostics, operators are required to perform diagnostic services on multiple vehicles in a single working day. This phenomenon can be identified using an OEM diagnostic tool by checking the different Electronic Control Unit (ECU)s within the vehicle. But this is a long process requiring time and a dedicated operator for a single vehicle to check all the modules. Jifeline as a company cannot dedicate such a high amount of resources on a single car considering the operators receive over a hundred service requests a day from third party garages that partner with Jifeline.

Project Objectives Jifeline possessing the means and knowledge to take appropriate action against the phenomenon of mileage fraud have created this project. The **objective** of which is to deliver a script that is able to detect the mileage stored in different modules of vehicles connected to the Jifeline Network. These vehicles referred to are Volkswagen (VW) Group, Modular Transverse Matrix (MQB) and Modular Longitudinal Matrix (MLB) vehicles. They are chosen for their shared functionality and their presence in the automotive market in terms of quantity. This report reflects on the project's proceedings.

Method Initially, an OEM diagnostic tool is used to scan a vehicle to recognise the ECUs containing mileage. This Controller Area Network (CAN) communication between tool and car is logged by the data logging system created in-house by the specialists at Jifeline. Following this, the data is analysed and reverse engineered to emulate the functionality of the tool by means of a script.

Initially, to simulate the tool intensive testing is required to develop and optimise this process. Testing on vehicles requires a hefty amount of time and resources. This becomes cumbersome. A secondary script is built to simulate the behaviour of the vehicle for the required diagnostic procedure. This is realised by the scripting environment present at Jifeline. The two scripts are tested against each other until successful execution. Now, the diagnostic tool simulating script can be tested against vehicles on a production environment.

The script simulating the diagnostic tool extracts the mileage stored and compares these values against each other to check for irregularities within the vehicle. The functionality of the script is extended to cater to vehicles that fall within the scope of this project. Finally, the script is tested and validated.

Principal Conclusions The process of performing a diagnostic procedure remotely by means of a python script has major advantages over the conventional procedure of performing vehicle diagnostics. The advantages gained in terms of time utilization, cost efficiency and effectiveness outweigh the disadvantages in the realm of diagnostic services.

The script created recreates the functionality carried out by the diagnostic tool, not only carries out it execution in a faster manner but also completely automates this process. Therefore, no operator input is required and the script knows exactly what to do to extract the mileage. With the information obtained from the script appropriate action can be taken therefore, helping to fight against mileage fraud.

Contents

List of Figures

List of Tables

Listings

List of Acronyms

4WD	Four wheel drive
AA	Android Application
ASAM	Association for Standardisation of Automation and Measuring Systems
ASDU	Application layer Service Data Unit
ABS	Anti-lock Braking Module
API	Application Programming Interface
AIDC	Automatic Identification and Data Capture
ASCII	American Standard Code for Information Interchange
BCM	Body Control Module
CP	Car-Pass
CAN	Controller Area Network
CEM	Central Electronics Module
COVID-19	Corona Virus Disease, 2019
DDM	Driver Door Module
DLC	Data Link Connector
DLL	Data Link Layer
DTC	Diagnostic Trouble Codes
DSCS	DiagnosticSessionContol service
EU	European Union
EC	European Commission
ECM	Engine Control Module
ECU	Electronic Control Unit
EOBD	European On-Board Diagnostics
FWD	Front wheel drive
HAN	Hogeschool van Arnhem en Nijmegen
HVAC	Heating Ventilation and Air-Conditioning Module
IEC	International Electrotechnical Commission
IPC	Instrument Panel Cluster Module
ISO	International Standards Organisation
JD	Jifeline Dashboard
JN	Jifeline Network
JL	Jifeline Local
JR	Jifeline Remote

JNW	Jifeline Networks
JRD	Jifeline Remote Diagnostics
Kbps	Kilobits per second
MS	Member States
MLB	Modular Longitudinal Matrix
MQB	Modular Transverse Matrix
MSI	Manufacturer Specific Identifier
NAP	Nationale AutoPas
NRSI	Negative Response Service Identifier
OBD	On-Board Diagnostics
OEM	Original Equipment Manufacturer
OSI	Open Systems Interconnection
PAM	Parking Assist Module
PDM	Passenger Door Module
PMP	Project Management Plan
PTI	Periodic Technical Inspection
RA	Remote Address
RDBI	ReadDataByIdentifiers
SA	Source Address
SI	Service Identifier
SR	Service Request
SAE	Society of Automotive Engineers
STID	Diagnostic Session Type Identifier
TA	Target Address
TP	TesterPresent
TRT	Transport e Territorio
TAtype	Target Address type
UDS	Unified Diagnostic Services
VW	Volkswagen
VAG	Volkswagen Aktiengesellschaft
VCDS	VAG-COM Diagnostic System
VIN	Vehicle Identification Number

CHAPTER 1

Introduction

1.1 Background

The process of changing/adjusting the mileage reading on a vehicle's odometer is known as mileage recalibration. When this process is carried out without disclosing it, it is known as *Odometer manipulation* or '*clocking*' (Clocking definition, 2020). The largest portion of the automotive market by value and share is the second-hand car market. It is two to three times larger than the market of new cars. Within the EU there is a massive import and export of second-hand cars. Approximately 2.4 million second-hand cars were traded between the Member States (MS) of the EU in 2014 (Borkowski, 2017). A study conducted by Transport e Territorio (TRT) reveals that the European Commission (EC) addressed the issue of odometer tampering for the first time within the EU in the "*Roadworthiness Package*". It was adopted on 11 March, 2014 (TRT (2017), 2017). It calls for Member States to adopt measure to record odometer readings of vehicles at every Periodic Technical Inspection (PTI). There exist such systems like Belgian Car-Pass (CP) and Dutch Nationale AutoPas (NAP) (RDW, 2014). These systems act as a database holding mileage information of vehicles, but on a national level. Therefore, they could be used as a basis to form an EU wide database.

An aspect of preventing odometer manipulation is detecting it. This can be identified by using OEM diagnostic tools. The problem here is these tools are expensive or require a continuous subscription and available mostly at dealerships and OEM certified workshops. This makes identification of this phenomenon difficult. Therefore, lack of correct tools to identify this phenomenon adds to this problem. All the factors added together make this a issue that is required to be addressed.

1.2 Problem

The two main problems that are caused due to mileage manipulation is the negative impact it has in terms of money being lost and the dissatisfaction it causes to the consumer base of the second-hand car market. Due to causes mentioned in section **??**, 30% - 50% of second-hand cars traded within the EU have their odometer manipulated (RPC, 2012). This has a detrimental impact on the second-hand car market both economically and in terms of consumer satisfaction. Also, the lack of information about accurate odometer readings leads to lower levels of trust amongst the consumers in this market when compared to other markets.

Jifeline Remote Diagnostics is a company that focuses on the after-sales market of the automotive industry servicing most types of passenger vehicles. Due to their network it is possible to access and collect data from vehicles when connected to it. Section 1.3 elaborates more on Jifeline's coverage area and vehicle's serviced. If a vehicle is detected having fraudulent mileage, Jifeline can notify the relevant authorities. These authorities can take the correct measures to deal with such a situation. Due to these reasons, Jifeline feels obligated to carry out such a project within the company as they have the means to prevent and take action against such a phenomenon.

1.3 Project Objective

Jifeline has the means and the knowledge to fight against this phenomenon. It's network has a coverage area spanning across nineteen countries (Jifeline Dashboard, n.d.) in the EU. This allows Jifeline to maintain a mileage database of vehicles serviced within this Network. Given the reasons and the infrastructure present, it creates a possibility to investigate into where a vehicle stores the mileage. Some questions come to minds when starting such an investigation. They are as follows. How many **ECUs** hold mileage information of the vehicle? How can this information be analysed and recorded? What are the similarities and differences between vehicles when carrying out such an investigation? How can they be categorised? Such questions form the basis of this investigation.

This report's objective is to reflect on the tasks carried out. It acts as a reference document to the project's proceedings. Information and method of how a vehicle's mileage is investigated are elaborated upon. Vehicles that come under the scope of this project are VW Group vehicles that fall under the MQB and MLB platforms. These platforms created by VW where vehicles share modular design. MQB vehicle are those which are constructed with transverse, front-engine, Front wheel drive (FWD)(or Four wheel drive (4WD)) layout. MLB vehicle are constructed with longitudinal, front-engined layout. They may or may not have 4WD. These vehicles are chosen because of the common functionality that they share and cover a vast portion of vehicles serviced by Jifeline Remote Diagnostics.

1.4 Method

This section elaborates on the method used for reaching the goals and providing the deliverables. To understand where the mileage is stored in a vehicle, it is required to know which ECUs in a vehicle contain this information. To obtain this information, a **diagnostic tool** is used to perform a vehicle scan. A diagnostic tool is used to obtain general information about a vehicle. It can also be used to program new modules or to re-calibrate existing modules. The diagnostic tool used for this project is **VAG-COM Diagnostic System (VCDS)**. It is a software package for Windows that emulates the functions VW OEM diagnostic tools. It is developed by Ross-Tech (VCDS, n.d.).

Once the scan is performed, the communication trace between the tool and the vehicle is logged in a log file. Using this log file as a base, work is started to develop an **automated function (script)** that can read the vehicle's mileage values from different ECUs emulating the role of a diagnostic tool. By observing these logs and reverse engineering them the objective can be achieved.

After this process is carried out for an in-scope vehicle, the same working methodology can be applied for the remainder of the vehicles to draw parallels and create the final product catering to all vehicles within this scope.

1.5 Report Structure

This document is organised in six chapters. Firstly, chapter 2 defines the research questions. Chapter 3, shows how mileage is extracted from a vehicle and categorisation of the vehicles serviced. Chapter 4, shows the design of the final product (script). It is broken down into four major parts, i.e., the function specifications required, high level design, low level design and lastly source code. Chapter 5, explains the approach taken to solve the problem at hand and verification of the final product. It is broken into two major parts; Testing and Validation. Chapter 6 summarises the most relevant aspects that have emerged and in previous chapters and makes recommendations about what more could be done to improve the script but could not be carried out due to limitations faced during the course of this project. The final part of this document contains the appendices and list of references. It contains reference information for this document as well as documents required as per requirements stated in the graduation guide provided by the Hogeschool van Arnhem en Nijmegen (HAN).

CHAPTER 2

Research Questions

2.1 Premise

This section elaborates on the premise that the main and sub-questions are formulated. It is common knowledge that the answers to sub-questions gives the answer to the main research question.

These question are formulated during the research phase of the project. They help to outline **critical tasks** required to carry out this investigation and achieve the goal of this project. Preliminary research is carried out by means of desk research and field to analysis the problem **mileage manipulation**. Desk research entails of analysis of the problem of mileage manipulation through existing literature. This can be seen in sections 1.1 and 1.2 as well as the **??** formulated during the initial phase of this project. Field research is carried out by scanning vehicle and vehicle test-setups (created in-house at Jifeline) with diagnostic tools and On-Board Diagnostics (OBD) fault code readers. From this research it is found that a vehicle's mileage can be obtained via an OEM diagnostic tool. With this method the different ECUs containing the vehicle's mileage can be identified. Knowledge obtained from this preliminary research helps to identify these critical task which in-turn form the research questions.

2.2 Main Question

How to design and build a reliable, automated and extensible function tailored towards investigating kilometres travelled for MQB and MLB vehicles via remote diagnostics?

2.3 Sub-questions

- How to learn and perform mileage checks on a vehicle?
- What are some basic requirements for performing diagnostic services on MQB and MLB vehicles?
- What components are required to build an automated function?
- How to verify extracted mileage data from the vehicle?
- How to make the function extensible for future use?

CHAPTER 3

Mileage Extraction

This chapter elaborates on procedure and analysis of mileage extraction. This is done to obtain the mileage of a vehicle; one of the primary objectives of this chapter. It also aims to draws parallels and differentiates in the procedure required to reach the aforementioned goal for MQB and MLB vehicles. The primary method used is observing the CAN communication between vehicle and diagnostic tool.

3.1 Investigation

This section shows what methods were used to obtain mileage values from the vehicle. The mileage value indicated on the odometer of the vehicle is not considered sufficient as the goal is to obtain where the mileage value is stored in a vehicle's ECUs, at what address, what is the format of the data and how to convert it into a readable format.

3.1.1 Preliminary Investigation

At Jifeline Remote Diagnostics, certain test-setups containing mission critical ECUs of vehicles exist. They are namely, the Engine Control Module (ECM), Instrument Panel Cluster Module (IPC), Central Electronics Module (CEM) and finally, the immobilizer. These setups were used for preliminary investigation to find which ECUs contain the mileage

Initially, the first method carried out was to obtain the mileage of a vehicle by an after-market **OBD-II/European On-Board Diagnostics (EOBD) fault code reader (reader)**. The idea was to get the mileage reading of the vehicle through this reader. After, exploring the options available on this reader it was found that the mileage value could not be read out. Although, what was noticed was when a fault code was read out there was a mileage stamp indicating at what mileage the fault code showed up in the vehicle. The disadvantage of this method is the current mileage is not shown. This means, if a vehicle has covered more kilometres after the fault code has showed up, the mileage reading obtained from the fault code is not the current mileage. Therefore, this method of investigation was dropped and another method of investigation was required to be adopted.

The second method, performing a scan on a vehicle using a **EOBD scan tool (diagnostic tool)**. Initially, this was tested on one of the test setups at Jifeline. The setup represented a VW Up. The diagnostic software used was **VCDS**. With this method the current mileage was retrieved from the ECM and IPC in the setup. Thus, this investigation conducted was successful and is the

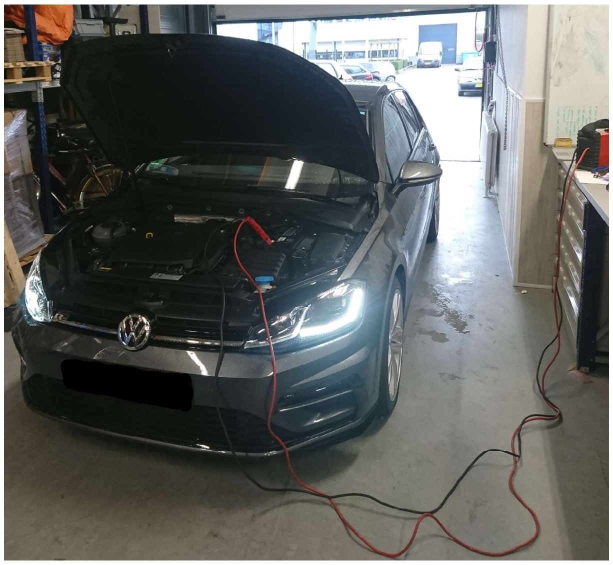


Figure 3.1: VW Golf Mk VII used for investigation

chosen methodology to further analyse the problem.

3.1.2 Diagnostic Tool Scan

This section shows the finalised methodology used to extract vehicle mileage from different ECUs. Following the investigation carried out in section 3.1.1 and the conclusion reached from it, scanning a vehicle using a **diagnostic tool** is the most suitable method. An applicable vehicle was chosen and an applicable diagnostic tool/software is required.

The chosen vehicle is a VW Golf Mark VII. This vehicle is part of the MQB platform. The software chosen is VCDS. Figure 3.1 shows the vehicle used to to carry out the investigation.

Using VCDS, a scan is performed to check how many ECUs are present in the vehicle. Table 3.1 shows all the ECUs present in the vehicle and their address. There are **seventeen** ECUs present. The next step is to check how many ECUs contain the mileage of the vehicle.

A scan on every ECU was done to find if mileage is present. Figure 3.2 shows how mileage was readout for every ECU using VCDS. Table 3.1 also shows the ECUs of the test vehicle and which of them contain mileage. From seventeen ECUs readout, **ten** of them contain the mileage of vehicle.

No	Address	ECU Name	Mileage value present
1	0x01	Engine Control Module	Yes
2	0x03	Brake Electronics Module	No
3	0x08	Auto HVAC Module	Yes
4	0x09	Central Electronics Module	Yes
5	0x10	Park Steer Assist Module	Yes
6	0x13	Adaptive Cruise Control Module	No
7	0x15	Airbag Module	No
8	0x16	Steering Wheel Module	No

Table 3.1 continued from previous page

No	Address	ECU Name	Mileage value present
9	0x17	Instruments Module	Yes
10	0x19	Gateway Module	Yes
11	0x42	Door Electronics Driver Module	Yes
12	0x44	Steering Assist Module	Yes
13	0x4B	Multifunctional Module	No
14	0x52	Door Electronics Passenger Module	Yes
15	0x5F	Information Electronics Module	Yes
16	0xD6	Light Control Left 2	No
17	0xD7	Light Control Right 2	No

Table 3.1: VW Golf Mk VII ECUs encountered and which contain vehicle mileage

3.2 Mileage Analysis

This section shows how results from the investigation carried out in section 3.1.2 are analysed. It emphasizes on the **CAN communication** between vehicle and tool. The CAN communication is logged by the in-house data logging system of Jifeline. This allows to analyse and reverse engineer the data being transferred between the two entities; vehicle and diagnostic tool, respectively. It helps to understanding, recognise and realise the behaviour of both entities when trying to re-create the functionality of either. The CAN protocol used is **Unified Diagnostic Services (UDS)**

To understand how to extract the mileage, certain concepts and workings within the CAN communication framework are required to be understood. They include but are not limited to translating mileage to readable format, **Service Identifier (SI)s** and **Manufacturer Specific Identifier (MSI)s** used by diagnostic tools to obtain mileage response. Also, ECUs containing unique and similar SIs and MSIs, respectively.

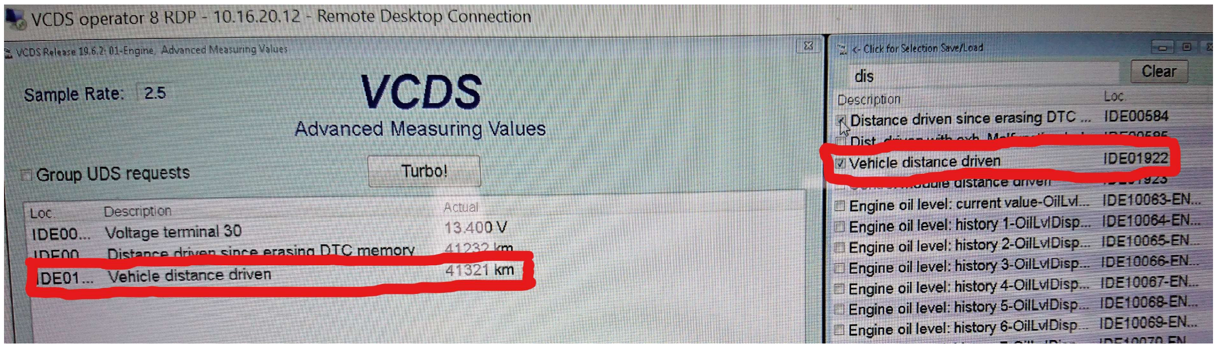
Detailed description of what is a SI is provided in section 4.1.3. It also shows all SIs found and used. Section 4.1.3 shows the MSIs and table 4.1 gives a list of these identifiers found and used.

Listing 3.1 shows what a request and response CAN communication message looks like. This communication trace is in the format that Jifeline creates in-house making it easier to understand. Aspects important to current topic are discussed.

- $11:41:51.621$ is the **timestamp** of the frame.
- CAN1 is the **CAN bus** being communicated too.
- $\rightarrow$ is the **request** message being sent to the vehicle from the tool.
- $\leftarrow$ is the **response** message from the vehicle to the tool.
- args[0x07E0 and 0x07E8] are the request and response **addresses** of the ECU, respectively.
- data[0x1902AE] contains **data** that can either be a request or response to and from the vehicle, respectively.

Listing 3.1: CAN communication frame

```
11:41:51.621 | CAN1 | -> | 93 ms | [ISOTP VAG] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x1902AE]
11:41:51.690 | CAN1 | <- | 30 ms | [ISOTP VAG] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x5902FF]
```



VCDS operator 8 RDP - 10.16.20.12 - Remote Desktop Connection

VCDS Release 19.6.2: 01-Engine, Advanced Measuring Values

Sample Rate: 2.5

VCDS

Advanced Measuring Values

Group UDS requests

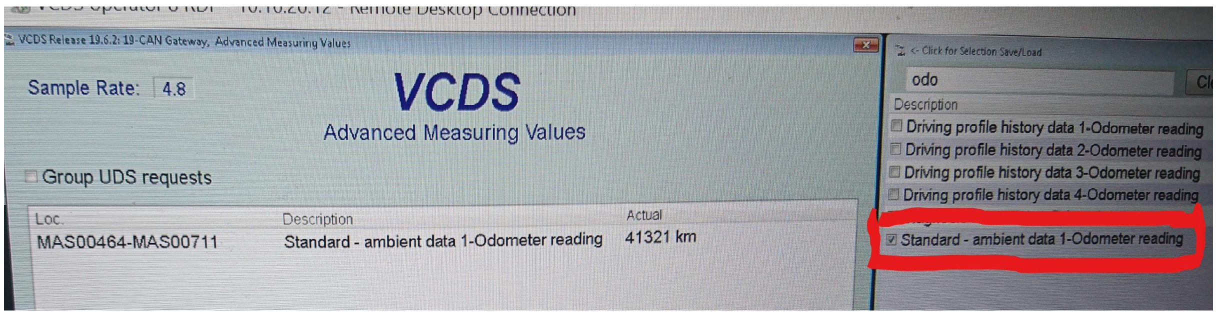
Loc	Description	Actual
IDE00...	Voltage terminal 30	13.400 V
IDE00...	Distance driven since erasing DTC memory	41232 km
IDE01...	Vehicle distance driven	41321 km

<- Click for Selection Save/Load

dis

Description	Loc
Distance driven since erasing DTC ...	IDE00584
Vehicle distance driven	IDE01922
Engine oil level: current value-OilLvl...	IDE10063-EN...
Engine oil level: history 1-OilLvlDisp...	IDE10064-EN...
Engine oil level: history 2-OilLvlDisp...	IDE10065-EN...
Engine oil level: history 3-OilLvlDisp...	IDE10066-EN...
Engine oil level: history 4-OilLvlDisp...	IDE10067-EN...
Engine oil level: history 5-OilLvlDisp...	IDE10068-EN...
Engine oil level: history 6-OilLvlDisp...	IDE10069-EN...

(a) ECM mileage: group, description and value



VCDS Release 19.6.2: 19-CAN Gateway, Advanced Measuring Values

Sample Rate: 4.8

VCDS

Advanced Measuring Values

Group UDS requests

Loc	Description	Actual
MAS00464-MAS00711	Standard - ambient data 1-Odometer reading	41321 km

<- Click for Selection Save/Load

odo

Description
- Driving profile history data 1-Odometer reading
- Driving profile history data 2-Odometer reading
- Driving profile history data 3-Odometer reading
- Driving profile history data 4-Odometer reading
- Standard - ambient data 1-Odometer reading

(b) CANGateway mileage: group, description and value

Figure 3.2: VW Golf VII ECU scans example

3.2.1 VW Golf Analysis

This is a **continuation** of the investigation carried out in section 3.1.2. The vehicle and diagnostic tool remain the same; VW Golf and VCDS, respectively. To read out the vehicle mileage, the diagnostic tool uses the SI **ReadDataByIdentifiers (RDBI)**. Following this, the MSIs varies depending upon the ECU being communicated to, although they may be similar at times. Firstly, breakdown and analysis of ECUs containing unique MSI to obtain mileage are discussed. Following this, ECUs containing same MSIs are discussed.

Unique MSIs

From the analysis of the CAN trace, two ECUs have unique MSIs. They are the ECM and IPC. Where the mileage value exists within response frame, how long it is and its description by the diagnostic tool are shown in this sections.

Engine Control Module The MSI which reads out mileage in this module is named **vehicle distance driven**; `0x10E0` is the hex service. If the vehicle sends a positive response to this request, the mileage is stored in this response. Listing 3.2 shows the request to read out the mileage and the positive response containing mileage. In this listing, 0x07E0 and 0x07E8 are the request and response addresses, respectively. The last **four bytes**; that is the last eight digits, of the positive response contain the mileage; denoted by `0x0000A169`.

Listing 3.2: ECM mileage request and response

```
13:54:32.301 | CAN1 | −> | 104 ms | [ISOTP VAG] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x22 10E0 ]
```

```
13:54:32.327 | CAN1 | <− | 26 ms | [ISOTP VAG] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x6210E0 0000A169 ]
```

Instruments Module The MSI which reads out mileage in this module is named **distance**; `0x2203` is the hex service. Listing 3.3 shows the request to read out the mileage and its positive response. In this listing, `0x0714` and `0x0774` are the request and response addresses respectively. The last **three bytes** of the positive response contains the mileage; denoted by `0x00A169`.

Listing 3.3: IPC mileage request and response

```
13:45:40.383 | CAN1 | −> | 46 ms | [ISOTP VAG] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x22 2203 ]
13:45:40.417 | CAN1 | <− | 34 ms | [ISOTP VAG] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0x622203 00A169 ]
```

Similar MSIs

Mentioned in section 3.1.2, ten ECUs store the vehicle's mileage. From section 3.2.1, two have unique MSIs. The remaining **eight** share the same MSI. This is denoted by **Standard - ambient data 1-Odometer reading**; `0x02BD` is the hex service. Although the remaining eight share the same MSI, the position of the mileage value within the positive response frame varies for certain ECUs. The ECUs where mileage position varies are the Passenger Door Module (PDM) and Driver Door Module (DDM). Those having the same mileage position are labelled **Type-I** and those that differ are labelled **Type-II** for this report only.

Type-I Listing 3.4 shows the request to read out the mileage and the positive response containing the mileage. In this listing, `0x0746` represents the Heating Ventilation and Air-Conditioning Module (HVAC)'s (referred to as "AutoHVAC" in this report) request address and `0x07B0` is the response address. For this ECU **bytes five to seven** of the positive response message contain the mileage of the vehicle; denoted by `0x00A169`.

Listing 3.4: Example AutoHVAC mileage request and response for comman MSI

```
14:21:11.436 | CAN1 | −> | 18 ms | [ISOTP VAG] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x22 02BD ]
14:21:11.540 | CAN1 | <− | 104 ms | [ISOTP VAG] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0x6202BD16 00A169 000050
    ↪ B0E54A]
```

Type-II Listing 3.5 shows the request to read out the mileage and the positive response containing the mileage. In this listing, `0x074A` represents the DDM's request address and `0x07B4` is the response address. For this ECU the mileage is embedded in **two and a half bytes**. Starting from position eleven to position fifteen; denoted by `0x0A169`.

Listing 3.5: Example DDM mileage request and response for common MSI

```
15:05:38.657 | CAN1 | −> | 57 ms | [ISOTP VAG] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x2202BD]
15:05:38.769 | CAN1 | <− | 112 ms | [ISOTP VAG] cmd[0x5000] args[0x07B4,0x00,0x00,0x00] data[0x6202BD0016 0A169 000285878
    ↪ B280]
```

3.2.2 MQB and MLB platforms

This section draws conclusions from the case study carried out in section 3.2.1. Findings from investigating other vehicles sharing these platforms are shown. The Vehicle Identification Number (VIN) is consider during investigations as it plays a crucial role in identifying the vehicle. This section elaborates on what basis these vehicles are categorised as well as the similarities and differences found within them.

VIN

It is a **17-digit** unique code used to identify vehicle containing the serial number. With the VIN and the correct Application Programming Interface (API), all details of a vehicle can be accessed. Listing 3.6 shows the type of details that can be accessed with the knowledge of the VIN. Therefore, the VIN is considered an extremely important aspect of identifying a vehicle.

Listing 3.6: Example details of vehicle got using the VIN

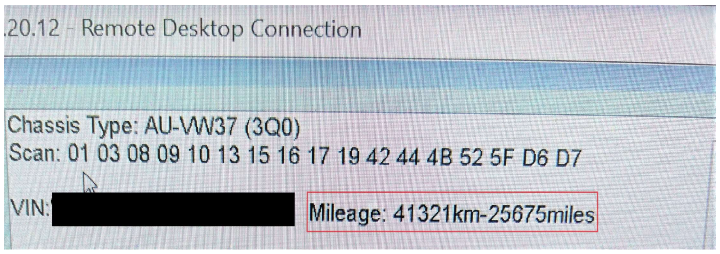
```
Chassis number:       WVGZZZ***********
Model:          Touareg BlueMotion
Technology:        ****
Production date:      **.**.2016
Model year:        2017
Sales type:        7P****
Engine code:        CV**
Acceleration code:       ***
Shaft drive characteristic:    **
Equipment:         **
Roof color:        **
Color-coded carpet
Exterior color / Paint number:  ** / ***

Note: The details of the vehicle have been redacted to keep the vehicle owner's privacy
```

Similarities

This section describes the similarities that are found with these platforms. The topics discussed are how the **mileage is converted**, how the vehicles belonging to these platforms are **categorised** and what common MSIs are used to **classify the ECUs**.

Mileage conversion Section 3.2.1 shows the mileage value that is retrieved. Although this value that is retrieved is in **hexadecimal**. This is converted into **decimal** to get the mileage in units of **kilometres** [km]. Vehicles belonging to this platform, all require the same conversion. From figure 3.3, the mileage of the vehicle is **41321 [km]**. The value obtained from the trace shown section 3.2.1 is `0xA169`. When this values is converted it corresponds to the value mentioned above in kilometres [km].


.20.12 - Remote Desktop Connection

Chassis Type: AU-VW37 (3Q0)
Scan: 01 03 08 09 10 13 15 16 17 19 42 44 4B 52 5F D6 D7

VIN: [redacted] Mileage: 41321km-25675miles

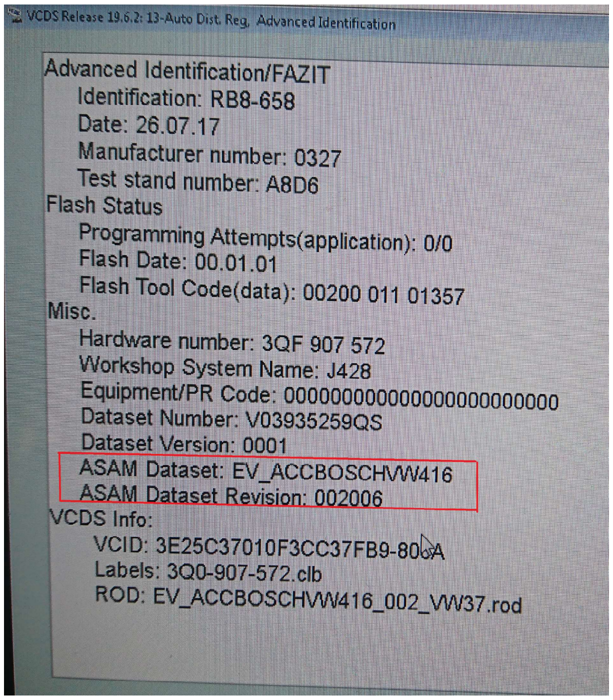
Figure 3.3: VW Golf Mk VII mileage

Vehicle categorisation Here, how the vehicles are classified is discussed. During this project, vehicles belonging to this platform may contain certain ECUs that do not follow the CAN protocol under the scope of this project. Thus, it is easier and more sensible to classify these vehicles not by the vehicle model, but the *ECUs present within these vehicles*.
Listing 3.7 shows the communication trace between the vehicle and tool from section 3.2.1. Here, the communication to the ECM (address: `0x07E8`) is in readable format. The communication to the Multifunctional Module (address: `0x17F00010`) is not in the same format therefore, it is consider out of scope.

Listing 3.7: example of ECUs within the Golf following different protocol

```
13:40:26.686 | CAN1 | <– | 149 ms | [ISOTP EOBD] cmd[0x5000] args[ 0x07E8 ,0x00,0x00,0x00] data[0x4100BE3EA813]
13:40:26.717 | CAN1 | [ 17F00010 ]<– | 31 ms | 2010000000000000 | .......
13:40:26.836 | CAN1 | <– | 119 ms | [ISOTP EOBD] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x4100BE3EA813]
13:40:26.986 | CAN1 | <– | 150 ms | [ISOTP EOBD] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x41208007B011]
13:40:27.137 | CAN1 | <– | 151 ms | [ISOTP EOBD] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x4140FED08401]
13:40:27.212 | CAN1 | [17F00010]<– | 75 ms | 2010000000000000 | .......
13:40:27.286 | CAN1 | <– | 74 ms | [ISOTP EOBD] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x4140FED08401]
```

ECU identification In paragraph **Vehicle categorisation** categorisation is made based on vehicle ECUs. To be able to categorise them in a unique manner, it is important to identify them. Therefore, **two** MSIs are used. They are described as **Association for Standardisation of Automation and Measuring Systems (ASAM) dataset** (commonly known as ODX Filename within Jifeline) represented by `0xF19E` and **ASAM dataset revision** represented by `0xF1A2`. Figure 3.4 shows the ECU details used to identify and categorise vehicles belonging to this platform.


VCDS Release 19.6.2: 13-Auto Dist. Reg. Advanced Identification

Advanced Identification/FAZIT
Identification: RB8-658
Date: 26.07.17
Manufacturer number: 0327
Test stand number: A8D6
Flash Status
Programming Attempts(application): 0/0
Flash Date: 00.01.01
Flash Tool Code(data): 00200 011 01357
Misc.
Hardware number: 3QF 907 572
Workshop System Name: J428
Equipment/PR Code: 000000000000000000000000
Dataset Number: V03935259QS
Dataset Version: 0001
ASAM Dataset: EV_ACCBOSCHVW416
ASAM Dataset Revision: 002006
VCDS Info:
VCID: 3E25C37010F3CC37FB9-806A
Labels: 3Q0-907-572.clb
ROD: EV_ACCBOSCHVW416_002_VW37.rod

Figure 3.4: Details of the Adaptive Cruise Control Module showing the ASAM dataset and ASAM dataset revision

Differences

This section describes the difference found within the platforms. The main differences identified are differences in position of where the mileage value is embedded. The method used to distinguish and identify is using a single byte hex value in the positive response message termed as **mileage identifier**.

Mileage identifier The mileage identifier method is *applicable when the Standard - ambient data 1-Odometer reading* service is requested. From the findings of the investigation and continuously researching other vehicles during the testing phase, the mileage identifier tells in which position of the positive response the mileage is stored. Varying positions of the mileage identifier indicate different positions of where the mileage is stored. During this project **three** variants of where the mileage value is stored have been discovered.

Listing 3.8 shows messages received and sent by an **Audi A4**. This vehicle is part of the VW Group MLB platform. Using this vehicle as an example a demonstration shall be made on how the position of the mileage identifier helps to identify mileage position. These messages are exchanged between three separate ECUs; they are the Information Electronics module (request address: `0x0773`, response address: `0x07DD`), DDM (request address: `0x074A`, response address: `0x07B4`) and Gateway module (request address: `0x0710`, response address: `0x077A`). Here, the same mileage identifier (hex value: `0x0D`) appears in three different positions. Following the mileage identifier, the mileage value (hex value: `0x919B`) is stored in these messages. From this it is seen the starting and ending positions where the mileage is present differs for all three variants.

Listing 3.8: Mileage identifier used to distinguish position of mileage value stored within positive response message

```
11:05:44.407 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x2202BD]
11:05:44.437 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0x6202BD 0D00919B 000050E8B16A]
11:05:44.417 | CAN1 | −> | 10 ms | [ISOTP] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x2202BD]
11:05:44.450 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x07B4,0x00,0x00,0x00] data[0x6202BD00 0D0919B 000287458B500]
11:05:44.439 | CAN1 | −> | 2 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x2202BD]
11:05:44.476 | CAN1 | <− | 6 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0x6202BD0050E8B16B 0D00919B 00]
```

Conclusion This chapter explains how the initial mileage was obtained from a vehicle using a diagnostic tools. It shows the procedure carried out in this project. The ECUs which contain the mileage. Figure D.5 shows all the ECUs investigated during this project and which of these contain the mileage. It also shows similarities and differences found between vehicles regarding the topic of how to obtain the mileage. Lastly, it elaborates on the different variations of how to obtain mileage, how they are categorised and what are the import aspects are to be kept in mind when using this information to design a automated function.

CHAPTER 4

Script Design

This chapter explains the design of the **final product (automated function/script)**. The script is written in the Python Programming language. The version of this language used is **Python 2.7**. This is the standardised scripting language used throughout the company. The vehicles implement the **International Standards Organisation (ISO) - 14229 Road vehicles - Unified Diagnostic Services - Specification and requirements** protocol as the CAN communication standard. This protocol is commonly known as UDS. As a reference document the 2^{nd} edition of the ISO - 14229:2006 published on 1^{st} December, 2006 is used for research and development purposes (ISO, 2006). It is broken into sub-chapters namely; **functional specifications, high level design, low level design and source code**.

4.1 Functional Specifications

This section mainly focuses on workings of the script with the outside environment. It shows the requirement for user interactions and script output. This section also, elaborates briefly on the functionality required to carry out communications with vehicles falling within the scope; namely VW Group MQB and MLB platforms. Configuring the script as such allows for it to be implemented on such vehicles. Functionality which falls out of scope will not be elaborated upon.

4.1.1 User Interactions

This section emphasises on the functional specifications that are required for the script to run on a vehicle. These specification include but are not limited to the initial state of the vehicle, the hardware and software required for the script to run successfully and its configurations.

Vehicle State The vehicle state required for running the script is its battery should be connected by an external power source. Following this the vehicle's key should be in the immobilizer at the **ignition 'on'** position. The rest of the vehicle's system should not be turned on and lastly, the doors should be closed.

Requirements Before being able to run the script on the vehicle, there are a few requirements that are must be met. These are:
• The vehicle should be connected with a pass-through device; namely the Jifeline Remote (JR) pass-through device.

• The remote should be connected to the an android device with proper internet connection and this device should have the Jifeline Android Application (AA) installed.
• The script user's device must be connected to Jifeline Network (JN) using a Jifeline Local (JL).
• The user must have the access to either the Jifeline Dashboard (JD) or the scripting environment to be able to start communications with the vehicle.

Configuration Figure 4.1 illustrates the the various components present in the Jifeline Remote Diagnostics (JRD) infrastructure. It also shows the flow of data from these components. Using this figure as a reference it can be seen the ① *JR* is connected to the JN via ② *AA* over the internet. This shows the configuration on the vehicle's side. ⑤ and ⑥ show the *JL* and the diagnostic tool, respectively. The latter's role is replaced by the script. They show the configuration requirements on the side of the user/operator. The script can be run from the *JD* ④ by creating a ticket with the connected vehicle or can be run by a *virtual robot* ⑦ in the scripting environment.

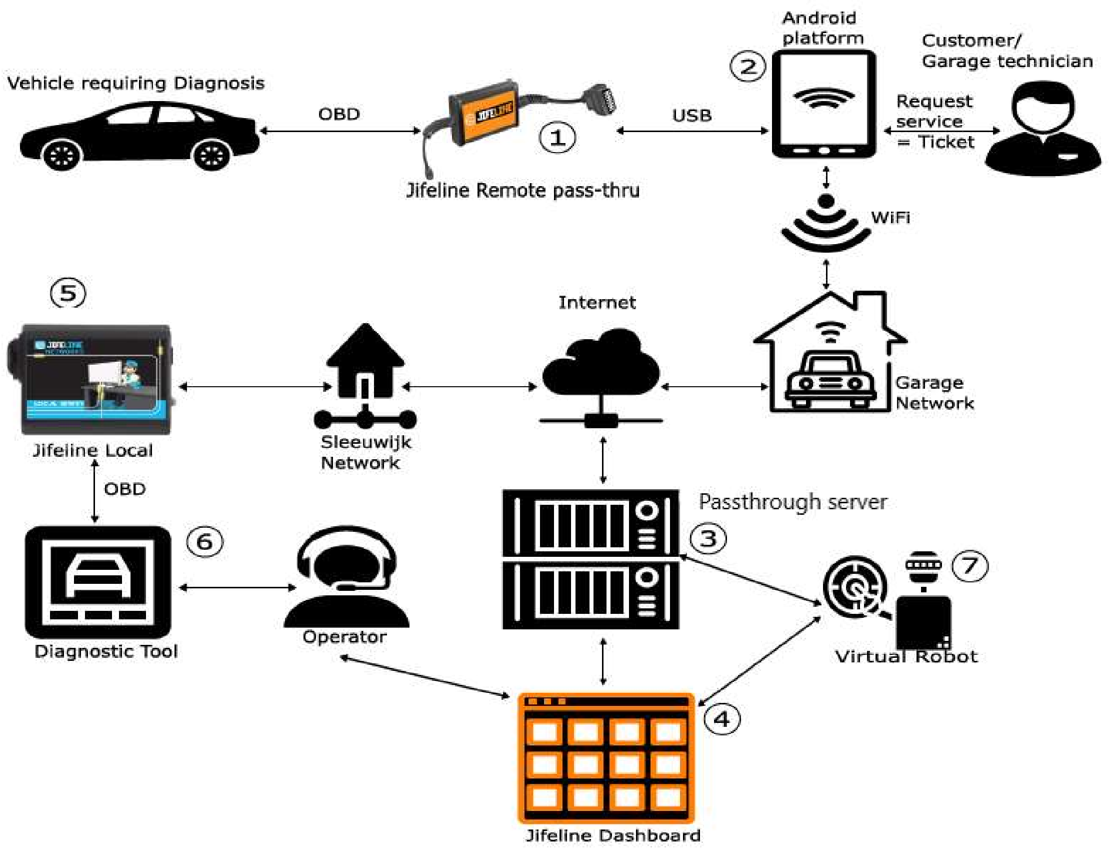


Figure 4.1: Jifeline Infrastructure[1]

4.1.2 Script Output

This section elaborates over the output the script produces. The output of the script can be observed by multiple methods. Currently, as an end product the most important output is the report that is

[1]Icons have been taken from `https://www.flaticon.com/`

send via the **Telegram Messenger** application. This is done by creating Telegram bots and these bots send the report over Telegram in the group where these exists.

Another important output is via the **terminal output** seen in the Jifeline scripting environment. This helps in understanding the behaviour of the script . The output is the same as the Telegram report, but it is easier and more accessible for research and development of the script. It is important to understand that the terminal output is a **live feed** of the script's execution. Listing 4.1 shows the reporting procedure via the terminal output.

Finally, when the script is run on a vehicle the CAN communication trace is logged and store in a **database** created in-house by Jifeline. This database can be **accessed any time** to refer to the trace to understand the response of the vehicle with respect to the script's execution. The format is the same as that seen on the terminal output.

Listing 4.1: Terminal Output

```
2020−05−26 16:14:09.252 | PRINT : [Script] [+] Vehicle mileage check successful!

2020−05−26 16:14:09.267 | PRINT : [Script] ======== Mileage Report =========
2020−05−26 16:14:09.268 | PRINT : [Script] [i] Total number of Mileage values found: 2
2020−05−26 16:14:09.272 | PRINT : [Script] [i] Connection id: "1590479545692"
2020−05−26 16:14:09.272 | PRINT : [Script] [i] Script description: "MQB/MLB Read VIN and Mileage"
2020−05−26 16:14:09.273 | PRINT : [Script] [i] Script uuid: "177ebb30−5866−42a4−b96b−d6953ceb5577"
2020−05−26 16:14:09.273 | PRINT : [Script] [i] Date & Time: Tue May 26 14:13:28 2020
2020−05−26 16:14:09.273 | PRINT : [Script] [i] Mileage: 41321
2020−05−26 16:14:09.274 | PRINT : [Script] [i] Number of ECUs found with mileage 41321[km]: 8
2020−05−26 16:14:09.274 | PRINT : [Script] [i] List of ECUs ['Engine Control Module', 'Instruments', 'Gateway', 'Information
    ↪ Electronics', 'Central Electronics', 'Steering Assist', 'Auto HVAC Module', 'Park Steer Assist']
2020−05−26 16:14:09.275 | PRINT : [Script] [i] Mileage: 41328
2020−05−26 16:14:09.277 | PRINT : [Script] [i] Number of ECUs found with mileage 41328[km]: 2
2020−05−26 16:14:09.278 | PRINT : [Script] [i] List of ECUs ['Door electronics Passenger', 'Door Electronics Driver']
2020−05−26 16:14:09.278 | PRINT : [Script]
2020−05−26 16:14:09.279 | PRINT : [Script] ======== End of Report ===========

2020−05−26 16:14:09.289 | PRINT : [DEBUG] Mileage: 41321
2020−05−26 16:14:09.290 | PRINT : [DEBUG] ECU details: [['Engine Control Module', 'EV_ECM15TFS01105E906018A', '003006'], ['
    ↪ Instruments', 'EV_DashBoardVDDMQBAB', '009058'], ['Gateway', 'EV_GatewNF', '013023'], ['Information Electronics', '
    ↪ EV_MUStd4CDELP', '001001'], ['Central Electronics', 'EV_BCMMQB', '018001'], ['Steering Assist', 'EV_SteerAssisMQB', '
    ↪ 013144'], ['Auto HVAC Module', 'EV_ACClimaBHBVW37X', '006145'], ['Park Steer Assist', 'EV_EPHVA18AU3700000', '
    ↪ 009035']]
2020−05−26 16:14:09.290 | PRINT : [DEBUG] Mileage: 41328
2020−05−26 16:14:09.291 | PRINT : [DEBUG] ECU details: [['Door electronics Passenger', 'EV_DCUPasseSideEWMAXKLO', '006003
    ↪ '], ['Door Electronics Driver', 'EV_DCUDriveSideEWMAXKLO', '006003']]
2020−05−26 16:14:09.291 | PRINT : [Script] [+] Main task finished!

2020−05−26 16:14:09.292 | PRINT : [Script] ======== Script finished! ===========
```

4.1.3 Software Specifications

The CAN protocol being used is UDS. This standard has been formulated for the globalisation of requirements for diagnostic systems irrespective of the serial data bus. It is based on the Open Systems Interconnection (OSI) Basic Reference Model. It is layer 7 or the application layer of this model. This model is applied on the MQB and MLB vehicles investigated in this project. Table D.2 in Appendix D shows the 7 layers of the OSI model. The vehicle diagnostic architecture helps to understand the structural implementation of this protocol on the investigated vehicles. The SI help to learn how to request for services from the ECUs within the vehicles for OBD purposes. The MSI show the manufacturer specific functionality implemented on these vehicles. Lastly, CAN Bus helps to understand from a software point of view what are the configuration requirements for this script to be able to run on these vehicles.

Vehicle Diagnostic Architecture The vehicle diagnostic architecture come across during this project is shown in figure 4.2; for this project the diagnostic tester can be understood as either the diagnostic tool or the script/automated function. The ECUs are connected over an internal data link and indirectly connected to the diagnostic data link through a gateway. ISO 14229 applies to the the diagnostic communications over the diagnostic data link (ISO, 2006); most vehicles encountered during this course of this project implement UDS for diagnostic communications over the internal data link.

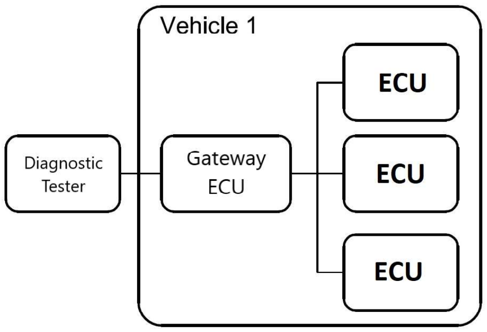


Figure 4.2: vehicle diagnostic architecture (ISO, 2006)

Service Identifiers

This section gives a concise understanding of SI. It also shows and describes the SI used during the course of this project. A SI is a single byte unsigned integer value ranging from 00-FF hex. It is used to encode specific values called in a request from the tool to the ECU. Table D.1 in Appendix D shows the SI values with respect to the service type and where it is defined. Table D.1 in Appendix D illustrates the request, positive and negative response codes for the SI used during the course of this project.

Diagnostic Session Control Service To identify a DiagnosticSessionContol service (DSCS), the SI is `0x10`. The diagnostic session allows for a specific set of services or functionality to be performed. A ECU starts up in the default diagnostic session of session type `0x01`. Figure D.3 in Appendix D gives a description of the DSCS request message sub-function parameter definition depending on the hex bit.

Read Data by Indetifier Service To identify a RDBI service, the SI is `0x22`. This allows for data records to be requested from the ECU. The request message may contain a single-byte or multi-byte dataIdentifier values to identify data records within the ECU. The number of *dataIdentifers* that can be requested at a single instance is limited by the vehicle manufacturer and ECU supplier.

Tester Present Service To identify a TesterPresent (TP) service, the SI is `0x3E`. This service is used for keeping the diagnostic session alive which the vehicle is currently in and prevents the vehicle from reverting back to the default session. It applies for diagnostic sessions other than the default session. It can also be used to do the same but over multiple ECUs at the same time.

Manufacturer specific functionality identifier (hex)	Description
`0xF19E`	ODX Filename (ASAM dataset)
`0xF190`	VIN identifier
`0xF1A2`	ASAM dataset revision
`0x03`	Default Volkswagen Diagnostic Session
`0x2203`	distance
`0x10E0`	Vehicle distance driven
`0x02BD`	Standard - ambient data 1-Odometer reading

Table 4.1: Manufacturer specific request message sub-function parameter definition

Manufacturer Specific Identifiers

UDS reserves certain bits within the CAN SI for the manufacturer. This section elaborates the manufacturer specific identifiers used or worked with during this project. Table 4.1 gives an overview of the identifiers used and their description.

CAN Bus

It is very important when trying to establish communication with a vehicle's ECUs that correct CAN Bus pins are connected on the vehicle's OBD sockets. Other technical details that are required are the bit-rate and the transceiver type. The MQB and MLB vehicles serviced have a **single CAN Bus network**. Listing 4.2 shows how these requirements are translated and applied in code. They are shown below:

- the CAN Bus pin connections used are pins **6** and **14** respectively.
- the CAN Bus bit-rate speed for this bus is **500** Kilobits per second (Kbps).
- A **high-speed** transceiver is used to maintain smooth communication between the vehicle's ECUs and client tester.

4.2 High Level

The created **automated function (script)** is divide into *two* parts. One to extract the data from the vehicle. The second to manipulate the extracted data. This section elaborates on the concepts used to extract and manipulate the data from the vehicle. It is divided into two parts. The first explains the concept used for extracting the data. The second, explains the concept behind manipulating the extracted data.

4.2.1 Data Extraction

This section elaborates on concepts applied for extracting the data from the vehicle. It represents the first part of the automated function. To extract the data from the vehicle the concept of **multi-threading** is used. Threads are lightweight processes or tasks (Thread, 2020). It allows for communication with multiple ECUs simultaneously. This greatly reduces the total run-time of the function in this implementation. Architecting the function to use threading also provides gain in design clarity (Threading, n.d.). It allows for a cleaner design which is easier to reason about. In listing 4.3, *class ECUScan(Task):* behaves as an individual thread extracting the data. Listing 4.4 shows creation of threads used for data extraction.

4.2.2 Data Manipulation

This section elaborates on concepts applied for manipulating data to obtain and compare the mileage values extracted. The methodology implemented takes effect after the data extraction process is completed; when all the running threads have stopped. The first step is to find where the mileage is embedded within the CAN frame. This knowledge is taken from section 3.2.2. When the mileage is found it is parsed into readable format Following this, the values are now compared against each other to check for equality. The values are then add to a data structure known as a dictionary in Python. A python dictionary works on the basis of a **key:value** pair. Here, the **mileage value** is assigned as the key. The list of **ECUs** which corresponding to the correct key are assigned as the value. In listing 4.3, *class MainTask(Task):* is where the data manipulation is carried out.

4.3 Low Level

From the script breakdown in section 4.2, this section will discuss over the method of how the two parts are executed. Firstly, methods used for data extraction are described. Secondly, methods used for data manipulation are explained. Figure 4.3 shows the execution procedure for the script. It is important to notice that irrespective of the results, this script gives an output. This means that even if no mileage is found within any of the scanned ECUs the script will execute completely and exit successfully. The only time this scripts stops pre-maturely is when an error occurs which causes to break the execution procedure.

4.3.1 Data Extraction Methods

This section shows methods used for data extraction in the class ECUScan(Task). All methods used during this process are seen in listing 4.5. These methods are executed in the method task(self): in chronological order. A description of critical methods are given:

__init__(self, ecu): It is used to initialise variable used throughout the execution of the parent class. This method passes the parameter **ecu**. This parameter contains the attributes of individuals ECUs to which the script communicates too. These attributes are defined in listing 4.7.

tester present(self): It is used to keep the vehicle in the current diagnostic session using the TP service. The purpose of this service is shown in section 4.1.3.

set_diagnostic_session(self): It is used to start the vehicle in the **default VW diagnostic session**. This enables the script to read out vehicle data using a RDBI service after this service is invoked. The method uses the DSCS service in combination with MSI `0x03`.

get_vin(self): It is used to obtain the VIN of a vehicle. The SI used is RDBI in conjunction with VIN identifier from table 4.1. The reason for obtaining a VIN of a vehicle is mentioned in section 3.2.2.

get_timestamp It is used to obtain a time stamp during script execution as a reference. This helps to understand the mileage of a vehicle with respect to time and date.

get_mileage_reading(self): It is used to obtain the mileage of a vehicle. The SI used is RDBI. There are three MSIs available to be used. They are described in table 4.1. This depends upon the ECU being communicated too as described in section 3.2.1.

get_module_details(self): It is used to obtain the ODX filename and ASAM dataset revision details of the ECU. The SI used is RDBI in conjunction with the MSIs mentioned in section 3.2.2. The reason for calling this method has also been discussed in the above mentioned section.

task(self): It is the execution method calling all methods listed in its parent class. This method is purely for running this script and is the programming method structure followed at Jifeline.

4.3.2 Data Manipulation Methods

This section shows methods used for data manipulation in the class MainTask(task). All methods used in this class are listed in listing 4.6. All methods in this section are invoked using method task(self). A description of what these method do is given:

count_most_common_vin(self, tasks): It checks for the most commonly occuring VIN. This is check is carried on data collected from ECUs containing a VIN. Ideally there should be only one VIN value.

update_vin(self, vin_task, vin_reading, retrieved_vins): It adds the valid VINs found to a dictionary containing the ECU name and VIN value. The dictionary is passed as the parameter **retrieved_vins**. Every VIN value/variation is saved a separate entry in this dictionary with the respective ECU.

get_mileage_identifier(self): It is used to find if a mileage identifier is present within the applicable collected mileage data. Details about mileage identifier are mentioned in section 3.2.2. If no mileage identifier is found it returns a message stating so.

hex_to_decimal(data_hex): It is used to convert the mileage value from hexadecimal to decimal. The hexadecimal value is passed through the parameter data_hex. This method is invoked in the method parse_mileage(self):

parse_mileage(self): It is used to extracted the mileage value from the mileage positive response frame. This depending upon the ECU and where this value lies within the frame. The information of where the mileage lies is taken from the dictionary ecu_mileage_position shown in listing 4.8. This method is one of the critical methods to find the mileage embedded in the response of respective ECUs.

unknown_mileage_identifier(self, engine_task, current_task): It is invoked when an unknown mileage identifier is present in the response frame. By passing the parameters engine_task and current_task, it uses the former as a reference to identify the hexadecimal mileage value against the later. The later is the ECU where the unknown identifier is being checked for. This method is invoked in parse_mileage(self) when mileage is found in a ECU and it does not match the current mileage identifier.

compare_mileage(self): It is used for validation of the mileage value found in the ECUs. It is used to compare mileage in the thousands keeping that as a margin for error. This is done because at times the mileage found in ECUs may not be exactly equal. It is invoked in update_mileage().

update_mileage(self, mileage_task, mileage_reading): It is uses the **same principle** as update_-vin(). It also contains the functionality to compare values with a margin for error as in invokes compare_mileage() as well as using the data obtain from get_module_details() as a reference for ECU identification.

print_report(self, retrieved_information, report_key, key_count, timestamp): It is used to print the findings after execution of this script. It gives an idea of the number of mileage values found with their respective ECUs and a time stamp as a reference for when this value was found. The mileage values and ECUs are passed through the parameter retrieved_information. The number of values found are passed through parameter key_count and the time stamp through parameter timestamp.

task(self): It is used to invoke the methods used to manipulate the value and printing the output of this script. It is similar to the methods with the same name mention in section 4.3.1 but with the functionality of methods mentioned here. It is this method which executes entire functionality of the script.

4.4 Source Code

This section contains listing of code used as a reference in sections 4.1, 4.2 and 4.3. It helps in describing the how and what the script does.

Listing 4.2: CAN Bus requirements

```
    def communication(self):
        self.context.registerBus(
            CanBus("bus1", 6, 14, Protocol.CAN.ISOTP, Bitrate.CAN.BITRATE_500, Transceiver.HIGH_SPEED))
```

Listing 4.3: Skeleton of script showing the classes used

```
# jifeline libraries
import ...

# class to configure, start and communicate the script and bus requirements
class Main(Main):...

# class which performs the data extraction
class ECUScan(Task):...

# class to collect extracted data and perform the required manipulation
class MainTask(Task):...
```

Listing 4.4: Creating threads for data extraction

```
        """
        For loop to iterate through every ECU object in a list: ecus.
        These objects are passed through the class: ECUScan().
        Finally, the class object is append to a list: task.
        Now, data extraction process is carried out via threads using objects in task.
        """
    for ecu in ecus:
        task = ECUScan(ecu)
        self.tasks.append(task)
        self.start_sub_task(task)

    # wait for tasks to finish
    self.wait_for_sub_tasks()
```

Listing 4.5: Methods for data extraction

```
class ECUScan(Task):

    def __init__(self, ecu):...

    def tester_present(self):...

    def set_diagnostic_session(self):...

    def get_vin(self):...

    @staticmethod
    def get_timestamp():...

    def get_mileage_reading(self):...

    def get_module_details(self):...

    def task(self):...
```

Listing 4.6: Methods for data manipulation

```
class MainTask(Task):

    def count_most_common_vin(self, tasks):...

    def update_vin(self, vin_task, vin_reading, retrieved_vins):...

    def get_mileage_identifier(self):...

    @staticmethod
  # this method is called in parse_mileage()
  def hex_to_decimal(data_hex):...

  def parse_mileage(self):...

  def unknown_mileage_identifier(self, engine_task, current_task):...

  @staticmethod
  def compare_mileage(mileage_reading, task_mileage):...

  def update_mileage(self, mileage_task, mileage_reading):...

  def print_report(self, retrieved_information, report_key, key_count, timestamp):...

  def task(self):...
```

Listing 4.7: ECU attributes list

```
"""
Ecus is a list of lists, each list contains a VAGEcu object. Every object starts it's own thread to communicate with the ECU,
but currently it is limited to twenty threads. VAGEcu object stores all the DiagnosticSocket objects for every known
VAG ECU along with request and response IDs. It is to be looked up in its parent class for more details.
```

```
"""
        ecus = [
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x7E0, 0x7E8, padding=True),
                    label="Engine")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x714, 0x77E, padding=True),
                    label="InstrumentCluster")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x710, 0x77A, padding=True),
                    label="CANGateway")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x773, 0x7DD, padding=True),
                    label="InformationElectronics")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x713, 0x77D, padding=True),
                    label="BrakeElectronics")], # A.K.A: ABS Brakes/ESP
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x74B, 0x7B5, padding=True),
                    label="PassengerDoorElectronics")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x74A, 0x7B4, padding=True),
                    label="DriverDoorElectronics")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x723, 0x78D, padding=True),
                    label="TrunkElectronics")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x74F, 0x7B9, padding=True),
                    label="FrontSensorDriveAssist")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x70E, 0x778, padding=True),
                    label="CentralElectronics")], # A.K.A: BCM
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x757, 0x7C1, padding=True),
                    label="AdaptiveCruiseControl")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x712, 0x77C, padding=True),
                    label="SteeringAssist")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x746, 0x7B0, padding=True),
                    label="AutoHVAC")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x70A, 0x774, padding=True),
                    label="ParkSteerAssist")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x70C, 0x776, padding=True),
                    label="SteeringWheel")],
            [VagEcu(DiagnosticSocket(self.con.bus1, DSProtocol.UDS, 0x715, 0x77F, padding=True),
                    label="Airbags")]
        ]
```

Listing 4.8: Information where the mileage lies in their respective ECUs

```
# these are dictionaries in a dictionary "ecu_mileage_position" containing different variants for getting mileage bytes
        self.ecu_mileage_position = {
            "Engine": { # ecu: 0x01
                "10E0": {
                    "Variant_I": {"mileage_start_offset": 0, "mileage_end_offset": 8},
                    "Variant_II": {"mileage_start_offset": 0, "mileage_end_offset": 4, "conversion_factor": 10},
                }
            },
            "AutoHVAC": { # ecu: 0x08
                "02BD": {
                    "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
                    "Variant_II": {"mileage_start_offset": 12, "mileage_end_offset": 18},
                }
            },
            "CentralElectronics": { # ecu: 0x09
                "02BD": {
                    "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
                }
            },
            "ParkSteerAssist": { # ecu: 0x10
                "02BD": {
                    "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
                }
            },
            "InstrumentCluster": { # ecu: 0x17
                "2203": {
                    "Variant_I": {"mileage_start_offset": 0, "mileage_end_offset": 6},
                    "Variant_II": {"mileage_start_offset": 0, "mileage_end_offset": 4, "conversion_factor": 10},
                }
            },
            "CANGateway": { # ecu: 0x19
                "02BD": {
                    "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
                "Variant_II": {"mileage_start_offset": 12, "mileage_end_offset": 18}
            }
        },
        "DriverDoorElectronics": { # ecu: 0x42
            "02BD": {
                "Variant_I": {"mileage_start_offset": 4, "mileage_end_offset": 9},
            }
        },
        "SteeringAssist": { # ecu: 0x44
            "02BD": {
                "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
                "Variant_II": {"mileage_start_offset": 4, "mileage_end_offset": 10}
            }
        },
        "PassengerDoorElectronics": { # ecu: 0x52
            "02BD": {
                "Variant_I": {"mileage_start_offset": 4, "mileage_end_offset": 9},
            }
        },
        "InformationElectronics": { # ecu: 0x5F
            "02BD": {
                "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
            }
        },
        "FrontSensorDriveAssist": { # ecu: 0xA5
            "02BD": {
                "Variant_I": {"mileage_start_offset": 2, "mileage_end_offset": 8},
            }
        }
    }
```

Conclusion This chapter describes the design of the script created. It puts into perspective how the knowledge gained from chapter 3 is applied in the script to replicate the functionality of a diagnostic tool. It shows what steps are taken to achieve the goal of the project using this script.

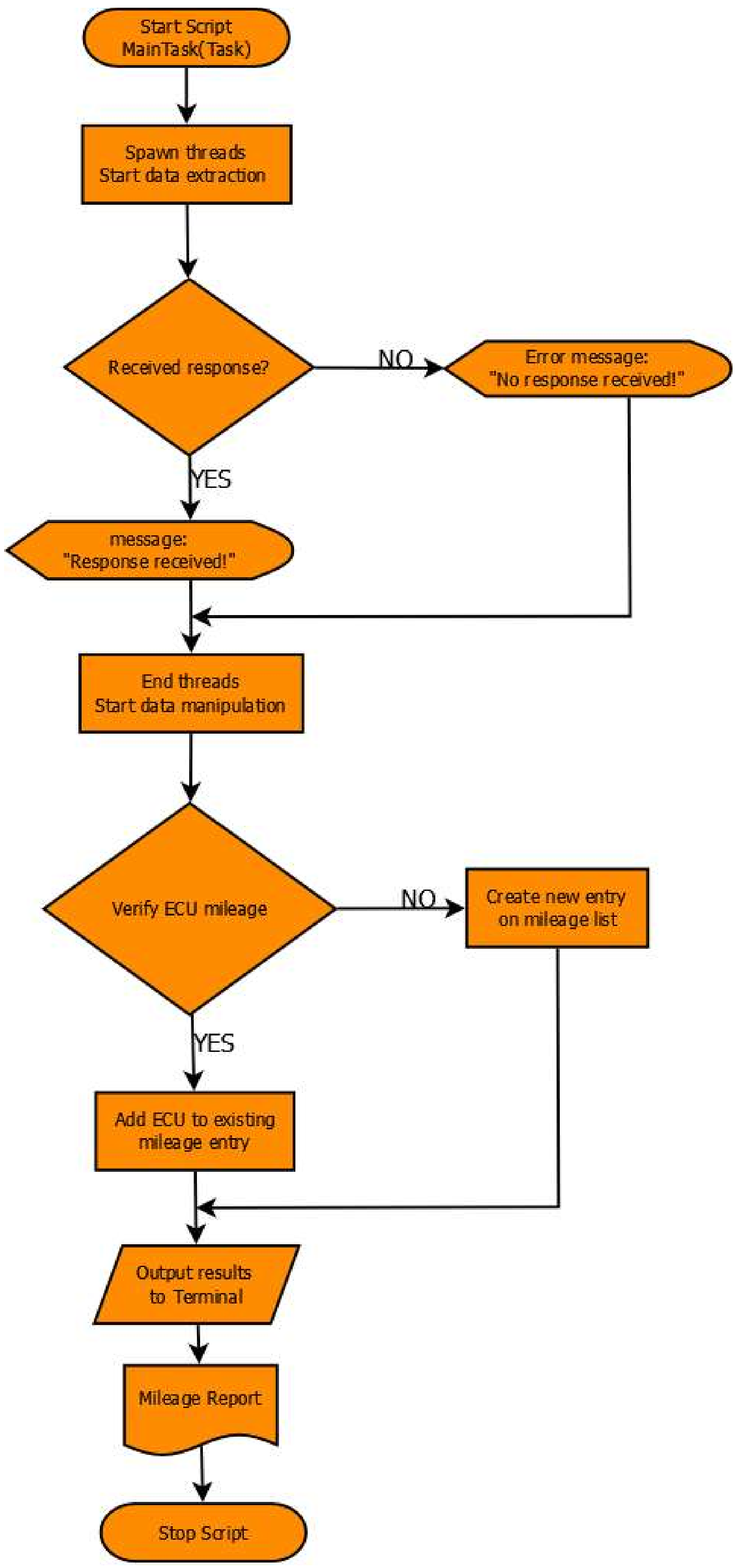


Figure 4.3: Script Execution procedure

CHAPTER 5

Verification

This chapters describes the testing and validation methods. Elaborations are made over the testing process carried out and how the script was validated. The project problem and the testing procedure were carried out simultaneously. It is because of the project methodology followed is the **Agile**. This methodology suits the project as it is required to continuously develop and test the software throughout the project life-cycle. More details about this methodology is mentioned in the Project Management Plan (PMP). This helps to include specific vehicle characteristics to the script noticed during this phase.

5.1 Testing

This section explains the testing methodology carried out during the project and also justifies it. Table 5.1 shows a conceptual understandings of the testing methodology followed mentioning the requirements and goal for every task carried out. The requirements (requirements column) indicate the knowledge or research required for performing mileage checks. The goal (goal column) elaborates on what is achieved when the task is performed successfully. Additionally, research and analysis is carried out if a vehicle is encountered requiring a different mileage extraction procedure. This includes small variations in already existing methods. Table D.4 in appendix D shows an in-depth mileage test plan followed as defined in the PMP. This is observed for a detailed overview of the testing procedure.

Before testing can be carried out certain aspects of the mileage extraction process should be known. They are labelled as Pre-requisites. These aspects are discussed over in this section. The Pre-requisites are formulated to carry out research as well as to obtain mileage details of vehicle. This helps to identify the ECUs containing mileage namely, the request and response frames.

5.1.1 Pre-requisites

This sections elaborates on knowledge required before the testing process. Asking certain questions give the answer to what is required for the pre-requisites. Some of these questions are as follows:

- What is the maximum value that can be held in the memory of an ECUs as mileage data?
- What is the request identifier required to get a positive response contain the mileage data?
- What is the size of the mileage frame within the positive response?

- What variations exists between ECUs of the same as well as different vehicles?

Maximum mileage value The maximum mileage value theoretically possible depends on the number of bytes present in the mileage frame. For example if the mileage frame is 3 bytes; the maximum value that it can store is `0xFFFFFF`. When this is converted to decimal it comes to 16,777,215 [km]. Such a value is deemed very large and can be considered as infinite. This is because it is almost impossible to reach such a mileage value practically.

Request identifier The initial byte of the request identifier is called a service identifier or SI The SI used is RDBI. The remainder of the request identifier depends on the ECU being communicated too. This information is elaborated in detail in Unique and Similar MSIs of section 3.2.1.

Mileage frame size The mileage frame size is the number of bytes allocated to store the mileage within the positive response frame. Part of chapter 3.2 sheds light on the different mileage frame sizes encountered during this project. They are also described in Unique and Similar MSIs of sections 3.2.1.

Existing variations There exists similarities and differences in the position of the mileage value within the response. These variations depend on the ECU being communicated too. This information is provided for in detail in section 3.2.2 in which elaborations are made on the similarities and differences found.

5.1.2 Key Tasks

This section elaborates on the key tasks required for execution for every testing cycle. These aspects are looked at when a new vehicle is tested by the script or by a diagnostic tool. During the initial phase of this project, testing of the vehicles were carried out using the diagnostic tool to gather information about vehicle mileage information. Detail explanation about analysing a vehicle using a diagnostic tool has been elaborated upon in section 3.1.2.

Step	Task	Requirements	Goal
1	Planning the test	when new findings or development has been noticed when executing existing script on a different vehicle	To add new functionality to the existing script
2	Developing the test case	In depth analysis of the new finding is carried out and how to implement by script	Knowledge gained on what new features are required
3	Setting up the test environment	Implementing new functionality on script by knowledge gained from test case	Script contains new functionality required for testing
4	Execution of the test	Script contains new section required for test	To check if the added section behaves as expected
5	Validation	testing of script is performed and a result is obtained	To check if test is successful or not, if not repeat steps 2-4

Table 5.1: Script Testing Cycle

In the later phases of the project majority of testing was carried out using the script. Preparations for this testing procedure was done by observing the log file containing the CAN communication between the script and vehicle. By observing these logs, the knowledge gained was analysed and the functionality required to execute the new procedure was implemented on the script.

Another key task is debugging. This task is carried out before the script is tested on a vehicle. This includes getting rid of syntax errors, formatting error and logical errors in any present. Preparation carried out for this process is done by creating a **vehicle simulator script** which simulates the behaviour of a vehicle. This is made possible due to the scripting environment created in-house at Jifeline. It is a virtual environment that allows to re-create the behaviour of vehicles when a new cases are observed. An advantage of this technique is that these vehicle simulator scripts are made to fit the testing requirements and are able to test what is required. This allows for testing of specific aspects of the script. They can also be altered to fit the requirements of most test.

Finally, this paragraph closely describes the steps described in table 5.1. Analysing the log file and understanding the new findings are carried out in step 2. Step 3 is equivalent to updating script tasks to test the new cases observed and testing it is step 4. Step 5 of the of this testing cycle takes place when the script is tested on the production environment

5.2 Validation

This section provides a general understanding of how the scripts are validated. This is an important aspect as it gives an idea if the created script works as expected. It also helps to recognise errors and realise if more functionality is required to be added when new findings are discovered. There are two main aspects of validating a script. They are internal and external, respectively.

5.2.1 Internal

This section does the validation carried out in two fold. One is the validation of the information extracted by the script. The second is the validation of the script itself. The later is carried out to ensure correct behaviour of the script and so that a exact understanding is achieved of what the script is doing when is it run.

Information validation Table 5.2 shows how the obtained results were logged acting as a database. This shows the structure used to log key information required to identify vehicle ECUs and categorise them. It helps to differentiate the variations and categorise similarities found within the same type of ECUs. Whenever the script is tested on a vehicle and a new development is found, it is added to this table. This helps to act as a reference document to validate which ECUs have already been tested and their characteristics. This knowledge then is implemented on the script to include new variations found from testing. Listing 4.8 shows how the knowledge gained is implemented on the script.

The columns labelled as **ODX** and **revision** stand for the ODX filename and ASAM dataset revision, respectively. This helps to identify the ECUs encountered and acts a means to categorise and distinguish them from other ECUs of the same type. The position column shows which position the mileage lies within the response frame. This helps to categorise them in terms of similarity and differences. These are some of the key aspects used to validate the results of the script.

Script validation This is another method used for validation. It tries to validate the mileage found from the script. It serves two purposes.

Firstly, to validate the mileage found in the vehicle. This is done within the script during runtime. This function is carried out in methods *update_mileage()* and *compare__mileage()*. Explanation of this functionality implemented on the script is elaborated in section 4.3.2.

Secondly, it also helps to realise if the script is working as expected. When a script is run and the result is not as expected, this method helps to realise that. This suggests that the script requires to be updated accordingly.

ECU	Address	mileage	SI	MSI	position	ODX	revision	Conversion	extra	comments
Gateway	0x19	Yes	RDBI	02BD	[8:14]	EV_GatewNF	01302*	hex to decimal	None	None
						EV_GatewA0	01302*			
						EV_GatewLear	0*****			
				02BD	[18:24]	EV_CGateCONTIAU491	00****			

Table 5.2: Test Result

5.2.2 External

This section shows the validation carried out from outside the perspective of the company. No field research is carried out for the external validation. This was due to the pandemic Corona Virus Disease, 2019 (COVID-19). Therefore, the method of external validation carried out is desk research. This is done by obtaining information about the yearly mileage that vehicles covers. Within the Netherlands vehicles drive an average of approximately 13000 [km] in 2017 and an average of 12,000 km/year for the EU as a whole (Sectoral Profile-Transport, 2020). Keeping in mind the scope, the MQB and MLB platforms were launched in 2012 and 2007, respectively (MQB - MLB, 2011).

Using the average values mentioned in conjunction with the manufacturing date of the investigated vehicle, a general consensus can be made of what is the total mileage. This can be used as a means of external validation. Although it is only a secondary measure if no conclusive answer can be made after carrying out the means of validation mentioned in section 5.2.1.

Conclusion This chapter explains how testing and validating the mileage found. It underlines important aspects required before testing as well as the method of testing implemented. The key tasks for testing are analysing log files, updating script and testing the later.

Validation is carried out internally and externally. Internally, the main tasks of validation are information and script validation. It helps to keep a record of what is already tested which can be used as reference for future cases. This is possible once the script works as expected. Externally, a benchmark mileage value of vehicles is considered. Although this value varies with the age of the vehicle. Therefore, covering both criterion.

CHAPTER 6

Discussions and Conclusion

6.1 Discussion

The set of main and sub-questions are required to be answered for designing and building a reliable, automated and extensible function investigating the mileage of MQB and MLB vehicles via remote diagnostics. From the chapters in this report, the basic ideology, method and knowledge required for building such a function applicable over the current remote diagnostics platform are elaborated upon. Not only, have these requirements been elaborated over but also how to test and validate the built function. This is done to verify the work carried out during this project.

Initially, a research study is carried out to investigate possibilities and come to conclusions on which path to take for such a project thereby defining a fixed scope. They help in defining the research questions mention in chapter 2. The research questions form the basis of this investigation. They provided structure on topics such as how and what is required to be done to reach the goal of this project.

The contents of chapter 3 answers the research question about learning and **performing mileage checks** on a vehicle. This chapter not only does so but also shows how to categorise and differentiate similarities and difference amongst vehicles. It also sheds light on knowledge required for building different components of an automated function. Lastly, it highlights the investigation phase of this project and also shows the results of the investigation.

Chapter 4 discusses over topics such as **basic requirements** required to carry out diagnostic services over in-scope vehicles, the **components required** to build such a function and finally, how this function is made extensible for future use. These requirements are discussed in detail in section 4.1.
The chapter also describes the breakdown of the script conceptually to understand it's working and the ideology implemented when designing it. Figure 4.3 shows the concepts followed in the script chronologically. It helps to bridge the gap between the concepts and the execution method.
Lastly, the script performs a verification of the information extracted using the knowledge gained by the investigation carried out and conclusions reached in sections 3.2.1 and 3.2.2, respectively. This helps to verify mileage extracted without requiring an external source.

Finally, chapter 5 gives the answers on how to **verify mileage** extracted from a vehicle. This is done in conjunction with testing of the script. Testing is considered a part of this because it helps to clear out errors from the script and optimise the process. Following this, validation is carried out.

This helping to verify the information obtained from the script. It is done internally; with resources present within the company, as well as externally, by method of desk research. These measures help to add credibility for the investigation carried out.

6.2 Conclusion

In conclusion, section 6.1 elaborates on how chapters provide the answers to sub-questions mentioned in section 2.3. This leads to answering of the main question to design and build a reliable, automated function for performing mileage checks on vehicles belonging to VW Group's MQB and MLB platform.

The advantage of having such a product is that it can deliver the same result as a diagnostic tool but within a **smaller time span** and allows for customisation tailored to customer demands. The total run-time of this script is approximately $60 - 90$ seconds. When performing the same procedure using the diagnostic tool it may take upto $10 - 15$ minutes provided the operator knows exactly which ECUs contain the mileage. Also, the scan must be performed manually using the tool and know exactly which request is required to obtain the vehicle's mileage from their respective ECUs. In the case of the script, the operator need not know this information as it is embedded within the script, but only required to run it on the vehicle. Therefore, the script makes the entire process of obtaining a vehicles mileage from different ECUs completed **automated**.

From the investigations carried out in this project certain conclusion are reach between the similarities and differences in characteristics of ECUs present within these vehicles. Firstly, from the vehicles serviced during this project it is found that approximately eleven types of ECUs contain the mileage of the vehicle. From them, two of them possess individual methods to request the mileage. The remaining follow the same method to request the mileage. Secondly, it is found that ECUs following the same method generally have the mileage embedded in 3 bytes of the response. The exceptions to this rule are the DDM and PDM with have the mileage response in two and half bytes of the entire frame. The IPC, which possesses its own individual method to request the mileage also has the mileage embedded in 3 bytes of the response. Only, the ECM has its mileage embedded in 4 bytes of the response. This knowledge is substantiated during testing and validation of the script to affirm its **reliability**.

With this script in place Jifeline can now use it to check mileage of vehicles giving a warning label if an unexpected situation occurs. This can be followed up a more detailed investigation of the vehicle tested, by Jifeline or the relevant authorities depending upon the severity of the situation. Therefore, this script helps to screen out irregularities in mileage between different ECUs thereby acting as a first line of defence against mileage manipulation.

6.3 Future Studies

For future study topics more detailed research is required on the CAN response frame containing the mileage as it contains data other than just the mileage of the vehicle. This is known by observing the logs of certain vehicles.

A lot of the data required to run this script is embedded within it. This can be exported to external databases or libraries. This helps to make the script concise and behave as a general function which can service a larger range of vehicle. This can be done by exporting manufacturer specific data to libraries. Al though to do this investigations must be made into the manufacturer specific functionality of vehicles and it is easier to do this when categorising similar vehicles. This can even helps reduce the run time of a script.

A drawback of the script is that it can only communicate to a certain number of ECUs at a time. This is considered so because newer vehicles have increasing number of modules installed within them. To overcome this, communication to ECUs can be carried out sequentially. This has a drawback of its own, that is it would take longer thereby increasing the run-time of the script. Therefore, it is a compromise between speed and volume.

APPENDIX A

Personal tutor Approval

The form including the Personal Tutor's approval is attached in the following page

HAN_UNIVERSITY OF APPLIED SCIENCES

N University of Applied Sciences
stitute of Automotive Engineering

Date received : . . . .

Application Form Graduation Phase

Details student:

Student number: . 583353. . . . .

Last name : . . Ghosh. . . . .

Initials: . .R G . . . . First name : . .Rohaan. . . . .

E-mailaddress : .rohaan10@gmail.com. . . . . Telephone number: . +31 682724192. . . . . .

Study program : Development Engineer (CB) / ~~Business Management Engineer (TC)~~*

Specialism : ~~AD (Automotive Development)~~ / ~~AM (Automotive Management)~~ / ~~AS (Automotive Services)~~
AT (Automotive Testing) *
* Delete what is not applicable.
Minor : Erasmus Exchange at CTU. . . . .

Scheduled start of graduation: . 05.02.2020 . . . .

Company details: . . .- . . . .

* Delete what is not applicable.
Do you have contact with this company? : Yes / No *

No company yet, but I am interested in the following companies : .Jifeline BV . . . .

. . . .

Area of interest : . Automotive programming of Electronic Control Units . . . .

. . . .

. . . .

To check which Expert Team applies to me (depends on current CB or TC and followed minor) is:
(Note, mention a maximum of one Experts team!)

☐ Business & Management	☐Manufacturing	☑Electronics& Control Systems
☐Structural Design	☐Power train	☐Vehicle Technologies

Date Meeting Study Coach : ....

Name Study Coach : . . Arthur Appelman. . . . . Signature Study Coach: [illegible]

Opinion Study Coach on Start Graduation: Positive~~/ Negative~~ *

Remarks and / or explanations:

....

....

....

....

Hand in the complete form at the Practical Office.

APPENDIX B

Final Project Management Plan

The Final Project Management Plan is in the attachments provided with this report.

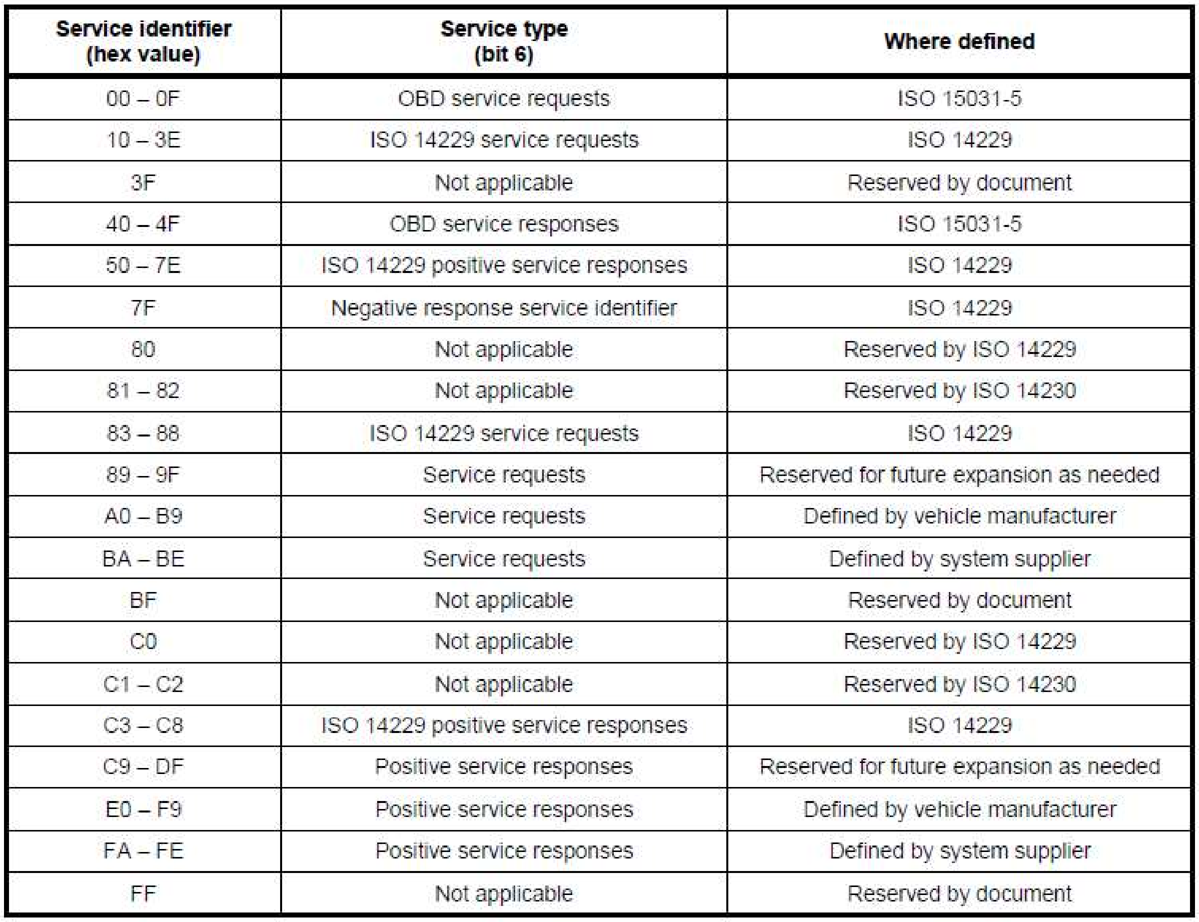
APPENDIX C

Reflection Report

The reflection report is in the attachments provided with this report.

APPENDIX D

Reference Tables



Service identifier (hex value)	Service type (bit 6)	Where defined
00 – 0F	OBD service requests	ISO 15031-5
10 – 3E	ISO 14229 service requests	ISO 14229
3F	Not applicable	Reserved by document
40 – 4F	OBD service responses	ISO 15031-5
50 – 7E	ISO 14229 positive service responses	ISO 14229
7F	Negative response service identifier	ISO 14229
80	Not applicable	Reserved by ISO 14229
81 – 82	Not applicable	Reserved by ISO 14230
83 – 88	ISO 14229 service requests	ISO 14229
89 – 9F	Service requests	Reserved for future expansion as needed
A0 – B9	Service requests	Defined by vehicle manufacturer
BA – BE	Service requests	Defined by system supplier
BF	Not applicable	Reserved by document
C0	Not applicable	Reserved by ISO 14229
C1 – C2	Not applicable	Reserved by ISO 14230
C3 – C8	ISO 14229 positive service responses	ISO 14229
C9 – DF	Positive service responses	Reserved for future expansion as needed
E0 – F9	Positive service responses	Defined by vehicle manufacturer
FA – FE	Positive service responses	Defined by system supplier
FF	Not applicable	Reserved by document

Figure D.1: SI Table[1]

[1]This table has been taken from ISO 14229-1
[2]The information has been taken from ISO-14229-1
[3]This information has been taken from ISO 14229-1

Description	Service Identifier		
-	Request	Positive response	Negative response
DiagnosticSessionControl	0x10	0x50	0x7F
TesterPresent	0x3E	0x7E	0x7F
ReadDataByIdentifier	0x22	0x62	0x7F

Table D.1: Example of SIs with their respective positive and negative response codes

OSI layer	Enhanced diagnostic services (non-emissions-related)	
Application (layer 7)	ISO 14229	ISO 14229
Presentation (layer 6)	-	-
Session (layer 5)	ISO 14229-3:2012	further standards
Transport (layer 4)	ISO 14229-3:2012	further standards
Network (layer 3)	ISO 14229-3:2012	further standards
Data link (layer 2)	ISO 11898-1	further standards
Physical (layer 1)	ISO 11898-1	further standards

Table D.2: Example of diagnostic/programming specifications applicable to the OSI layers[2]

Hex (bit 6-0)	Description
0x00	ISOSAEReserved*
0x01	defaultSession
0x02	programmingSession
0x03	extendedDiagnosticSession
0x04	safetySystemDiagnosticSession
0x05−0x3F	ISOSAEReserved
0x40−0x5F	vehicleManufacturerSpecific
0x60−0x7E	systemSupplierSpecific
0x7F	ISOSAEReserved

Table D.3: DSCS request message sub-function parameter definition[3]
*Society of Automotive Engineers (SAE)

Step	Task	Requirements	Goal
1	Selecting vehicle manufacturer	The vehicle manufacturer implements UDS protocol on its vehicles, it is supported by the system (Jifeline) and is widely traded in the second hand market and is suseptible to mileage fraud	To be able to cover multiple vehicles from this manufacturer (better if it uses a modular vehicle platform, e.g. MQB from VAG)
2	Choose a vehicle	It must fulfil the goal of Task 1	Use this vehicle as a sample to start investigation of the problem
3	Perform scan with a diagnostic tool	It should be compatible with the vehicle, be able to read out mileage values from all control modules is possible and provide an overview of all present modules (e.g. VCDS for VAG)	Gain knowledge on all modules possessing mileage information and obtain a log file containing this information
4	Analyse log file	Must be in readable format for the user and containing enough information to be able to emulate a vehicle by means of a script	Obtain knowledge on the procedure of extracting mileage data and how to covert it to readable format. Also, identifying and categorising the modules that are actual relevant by means of a knowledge base
5	build vehicle simulator	Enough knowledge gathered by the tool to simulate vehicle conditions at a standstill with ignition on and read mileage and module information	To help validate the mileage extraction script when it is run against the vehicle simulator script
6	Testing vehicle simulator	The vehicle simulator script should be free of all syntax and logical errors so that it doesn't hinder testing against diagnostic tool	To validate the vehicle simulator script if it behaves like a real vehicle
7	Build Script	Must have sufficient knowledge to cover almost every possibility that is experienced by the user and have some measures for unseen cases or give an indication for situations that is not taken into account.	Show modules containing similar mileages and raise a warning if any of the values do not match. It should be able to show if there is something wrong or not
8	Testing Phase I	Have a working scripts for mileage extraction and vehicle simulation	To test the scripts against each other to debug them of any syntax and logic errors when building the script
9	Testing Phase II	The mileage extraction script should know how the procedure works and must have achieved the goal of testing phase I	To verify and validate if the built script is able to detect any faults if present.

Table D.4: Mileage Test Plan

Address	ECU Name	Mileage value present
0x01	Engine Control Module	Yes
0x02	AutoTrans Module	No
0x03	Brake Electronics Module	No
0x05	Acc Start Auth Module	No
0x08	Auto HVAC Module	Yes
0x09	Central Electronics Module	Yes
0x10	Park Steer Assist Module	Yes
0x13	Adaptive Cruise Control Module	No
0x15	Airbag Module	No
0x16	Steering Wheel Module	No
0x17	Instruments Module	Yes
0x19	Gateway Module	Yes
0x22	All Wheel Drive Module	No
0x2B	Steering Column Lock Module	No
0x36	Seat Memory Driver Module	No
0x3C	Lane Change Module	No
0x42	Door Electronics Driver Module	Yes
0x44	Steering Assist Module	Yes
0x46	Comfort Convenience Module	No
0x4B	Multifunctional Module	No
0x51	Electric Drive Module	No
0x52	Door Electronics Passenger Module	Yes
0x5F	Information Electronics Module	Yes
0x61	Battery Regulation Module	No
0x69	Trailer Module	No
0x6C	Backup Camera Module	No
0x6D	Trunk Electronics Module	No
0x74	Chassis Control Module	No
0x75	Telematics Module	No
0x81	Gear Shift Module	No
0xA5	Front Sensor Drive Assist Module	Yes
0xAC	Reductant Control Module	No

Table D.5 continued from previous page

Address	ECU Name	Mileage value present
0xC4	DC/DC Converter	No
0xC5	Thermal Management Module	No
0xC6	Battery Charger Module	No
0xCA	Sunroof Module	No
0xCF	Lane Change Assist (Additional) Module	No
0xD6	Light Control Left 2	No
0xD7	Light Control Right 2	No

Table D.5: ECUs encountered

APPENDIX E

Scripts Log

Listing E.1: Example of script communication trace with an Audi A6

```
14:47:13.060 | ALL | <– | 2 ms | [Debug]: ======== Starting Script! =============
14:47:13.061 | ALL | <– | 1 ms | [Debug]: _ _ ---- --- / _ _ _ --- ---- ---- ---- --- _ _ _ _ _ ---- _ _ --- _ _ _ _ ---- ---- ---- ----
14:47:13.062 | ALL | <– | 1 ms | [Debug]: |\/| | | |--] / |\/| | |--] |--/ |--- |--| | \ | | | |\ | |--| |\ | | \ |\/| | | |--- |--| | -- |---
14:47:13.063 | ALL | <– | 1 ms | [Debug]: | | |-\| |--] / | | |--- |--] | \ |--- | | |--/ \/ | | \| | | | \| |--/ | | | |--- |--- | | |--] |---
14:47:13.066 | ALL | <– | 3 ms | [Debug]:
14:47:13.178 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting Engine
14:47:13.178 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting InstrumentCluster
14:47:13.178 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting CANGateway
14:47:13.180 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting InformationElectronics
14:47:13.184 | ALL | <– | 4 ms | [Debug]: [DEBUG] [+] Starting scan on InstrumentCluster
14:47:13.184 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting BrakeElectronics
14:47:13.184 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting scan on Engine
14:47:13.186 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting PassengerDoorElectronics
14:47:13.188 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting DriverDoorElectronics
14:47:13.193 | ALL | <– | 5 ms | [Debug]: [DEBUG] [+] Starting scan on CANGateway
14:47:13.194 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting TrunkElectronics
14:47:13.195 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on InformationElectronics
14:47:13.196 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on BrakeElectronics
14:47:13.196 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x1003]
14:47:13.197 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x1003]
14:47:13.199 | CAN1 | –> | 2 ms | [ISOTP] cmd[0x5000] args[0x0713,0x00,0x00,0x00] data[0x1003]
14:47:13.200 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on PassengerDoorElectronics
14:47:13.208 | ALL | <– | 8 ms | [Debug]: [DEBUG] [+] Starting FrontSensorDriveAssist
14:47:13.209 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x1003]
14:47:13.209 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x1003]
14:47:13.209 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074B,0x00,0x00,0x00] data[0x1003]
14:47:13.210 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on DriverDoorElectronics
14:47:13.211 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x1003]
14:47:13.212 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on TrunkElectronics
14:47:13.214 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting CentralElectronics
14:47:13.215 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x0723,0x00,0x00,0x00] data[0x1003]
14:47:13.217 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting AdaptiveCruiseControl
14:47:13.221 | ALL | <– | 4 ms | [Debug]: [DEBUG] [+] Starting scan on FrontSensorDriveAssist
14:47:13.221 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074F,0x00,0x00,0x00] data[0x1003]
14:47:13.223 | ALL | <– | 2 ms | [Debug]: [DEBUG] [+] Starting SteeringAssist
14:47:13.227 | ALL | <– | 4 ms | [Debug]: [DEBUG] [+] Starting AutoHVAC
14:47:13.231 | ALL | <– | 4 ms | [Debug]: [DEBUG] [+] Starting ParkSteerAssist
14:47:13.238 | ALL | <– | 7 ms | [Debug]: [DEBUG] [+] Starting SteeringWheel
14:47:13.243 | ALL | <– | 5 ms | [Debug]: [DEBUG] [+] Starting Airbags
14:47:13.250 | ALL | <– | 7 ms | [Debug]: [DEBUG] [+] Starting scan on AdaptiveCruiseControl
14:47:13.250 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting scan on AutoHVAC
14:47:13.250 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting scan on SteeringAssist
14:47:13.250 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting scan on ParkSteerAssist
14:47:13.251 | ALL | <– | 1 ms | [Debug]: [DEBUG] [+] Starting scan on SteeringWheel
14:47:13.252 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x0757,0x00,0x00,0x00] data[0x1003]
14:47:13.253 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x1003]
14:47:13.253 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x0712,0x00,0x00,0x00] data[0x1003]
14:47:13.253 | ALL | <– | 0 ms | [Debug]: [DEBUG] [+] Starting scan on CentralElectronics
14:47:13.253 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070A,0x00,0x00,0x00] data[0x1003]
14:47:13.254 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x070C,0x00,0x00,0x00] data[0x1003]
14:47:13.254 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070E,0x00,0x00,0x00] data[0x1003]
```

```
14:47:13.254 | ALL | <− | 0 ms | [Debug]: [DEBUG] [+] Starting scan on Airbags
14:47:13.256 | CAN1 | −> | 2 ms | [ISOTP] cmd[0x5000] args[0x0715,0x00,0x00,0x00] data[0x1003]
14:47:13.361 | CAN1 | <− | 105 ms | [ISOTP] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.366 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.370 | CAN1 | −> | 4 ms | [ISOTP] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x22F19E]
14:47:13.375 | CAN1 | −> | 5 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x22F19E]
14:47:13.376 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.380 | CAN1 | <− | 4 ms | [ISOTP] cmd[0x5000] args[0x077D,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.384 | CAN1 | −> | 4 ms | [ISOTP] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x22F19E]
14:47:13.386 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.389 | CAN1 | −> | 3 ms | [ISOTP] cmd[0x5000] args[0x0713,0x00,0x00,0x00] data[0x22F19E]
14:47:13.392 | CAN1 | <− | 3 ms | [ISOTP] cmd[0x5000] args[0x078D,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.394 | CAN1 | −> | 2 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x22F19E]
14:47:13.401 | CAN1 | −> | 7 ms | [ISOTP] cmd[0x5000] args[0x0723,0x00,0x00,0x00] data[0x22F19E]
14:47:13.409 | CAN1 | <− | 8 ms | [ISOTP] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.418 | CAN1 | −> | 9 ms | [ISOTP] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x22F19E]
14:47:13.433 | CAN1 | <− | 15 ms | [ISOTP] cmd[0x5000] args[0x077C,0x00,0x00,0x00] data[0x5003003201F4]
14:47:13.442 | CAN1 | −> | 9 ms | [ISOTP] cmd[0x5000] args[0x0712,0x00,0x00,0x00] data[0x22F19E]
14:47:13.455 | CAN1 | <− | 13 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F4761746577506B6F55445300]
14:47:13.459 | CAN1 | <− | 4 ms | [ISOTP] cmd[0x7F78] args[0x077E,0x00,0x00,0x00] data[0x]
14:47:13.464 | CAN1 | −> | 5 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.466 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F45434D3230544449303131130344C393036303231484100]
14:47:13.466 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x7F78] args[0x07DD,0x00,0x00,0x00] data[0x]
14:47:13.474 | CAN1 | <− | 8 ms | [ISOTP] cmd[0x5000] args[0x077F,0x00,0x00,0x00] data[0x5003004B012C]
14:47:13.475 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.475 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0715,0x00,0x00,0x00] data[0x22F19E]
14:47:13.483 | CAN1 | <− | 8 ms | [ISOTP] cmd[0x5000] args[0x078D,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F484453474175353732000000000000000000000000]
14:47:13.489 | CAN1 | <− | 6 ms | [ISOTP] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F416972436F6E6469436F6D666F55445300]
14:47:13.492 | CAN1 | −> | 3 ms | [ISOTP] cmd[0x5000] args[0x0723,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.494 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x077D,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F45535039424F534348415535375800]
14:47:13.498 | CAN1 | −> | 4 ms | [ISOTP] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.502 | CAN1 | −> | 4 ms | [ISOTP] cmd[0x5000] args[0x0713,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.508 | CAN1 | <− | 6 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0x62F1A2303032303131]
14:47:13.513 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x5000] args[0x077C,0x00,0x00,0x00] data[0x62F19E45565F524345505300]
14:47:13.516 | CAN1 | −> | 3 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x22F190]
14:47:13.521 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x62F1A2303034303039]
14:47:13.522 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x0712,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.529 | CAN1 | −> | 7 ms | [ISOTP] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x22F190]
14:47:13.531 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x078D,0x00,0x00,0x00] data[0x62F1A2303031303232]
14:47:13.539 | CAN1 | −> | 8 ms | [ISOTP] cmd[0x5000] args[0x0723,0x00,0x00,0x00] data[0x22F190]
14:47:13.543 | CAN1 | <− | 4 ms | [ISOTP] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0x62F1A2303033303038]
14:47:13.553 | CAN1 | <− | 10 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F4D554869673643347656E324842415300]
14:47:13.554 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0x7F2231]
14:47:13.554 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F524244344B00000000000000000000000000]
14:47:13.554 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x22F190]
14:47:13.558 | CAN1 | −> | 4 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.559 | ALL | <− | 1 ms | [Debug]: [DEBUG] [−] Failed getting VIN details from CANGateway Module. Received:
    ↪ RequestOutOfRangeException (710 −> 77A) request: 22F190 response: 7F2231 error code: 31
14:47:13.561 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x62F190∗redacted∗]
14:47:13.563 | CAN1 | −> | 2 ms | [ISOTP] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.563 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0710,0x00,0x00,0x00] data[0x2202BD]
14:47:13.571 | ALL | <− | 8 ms | [Debug]: [DEBUG] [+] VIN: ∗redacted∗ of vehicle Engine module
14:47:13.572 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x07E0,0x00,0x00,0x00] data[0x2210E0]
14:47:13.573 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x077D,0x00,0x00,0x00] data[0x62F1A2303034303138]
14:47:13.579 | CAN1 | <− | 6 ms | [ISOTP] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0x7F2231]
14:47:13.581 | CAN1 | −> | 2 ms | [ISOTP] cmd[0x5000] args[0x0713,0x00,0x00,0x00] data[0x22F190]
14:47:13.584 | CAN1 | <− | 3 ms | [ISOTP] cmd[0x5000] args[0x078D,0x00,0x00,0x00] data[0
    ↪ x62F1902D2D2D2D2D2D2D2D2D2D2D2D2D2D2D2D2D]
14:47:13.589 | ALL | <− | 5 ms | [Debug]: [DEBUG] [−] Failed getting VIN details from AutoHVAC Module. Received:
    ↪ RequestOutOfRangeException (746 −> 7B0) request: 22F190 response: 7F2231 error code: 31
14:47:13.589 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x5000] args[0x077F,0x00,0x00,0x00] data[0
    ↪ x62F19E45565F41697262614155313042504141553634580]
14:47:13.589 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0746,0x00,0x00,0x00] data[0x2202BD]
14:47:13.593 | ALL | <− | 4 ms | [Debug]: [DEBUG] [+] VIN: −−−−−−−−−−−−−−−−− of vehicle TrunkElectronics module
```

```
14:47:13.593 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0723,0x00,0x00,0x00] data[0x2202BD]
14:47:13.594 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x077C,0x00,0x00,0x00] data[0x62F1A2313036303037]
14:47:13.594 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x5000] args[0x077A,0x00,0x00,0x00] data[0x7F2231]
14:47:13.597 | CAN1 | −> | 3 ms | [ISOTP] cmd[0x5000] args[0x0715,0x00,0x00,0x00] data[0x22F1A2]
14:47:13.603 | CAN1 | −> | 6 ms | [ISOTP] cmd[0x5000] args[0x0712,0x00,0x00,0x00] data[0x22F190]
14:47:13.606 | ALL | <− | 3 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from CANGateway Module. Received
    ↪ RequestOutOfRangeException (710 −> 77A) request: 2202BD response: 7F2231 error code: 31
14:47:13.606 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x7F78] args[0x07DD,0x00,0x00,0x00] data[0x]
14:47:13.623 | CAN1 | <− | 17 ms | [ISOTP] cmd[0x5000] args[0x07B0,0x00,0x00,0x00] data[0x7F2231]
14:47:13.624 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x7F78] args[0x077E,0x00,0x00,0x00] data[0x]
14:47:13.624 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x5000] args[0x077D,0x00,0x00,0x00] data[0x7F2231]
14:47:13.624 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x5000] args[0x078D,0x00,0x00,0x00] data[0x7F2231]
14:47:13.625 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x07E8,0x00,0x00,0x00] data[0x7F2231]
14:47:13.635 | ALL | <− | 10 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from AutoHVAC Module. Received
    ↪ RequestOutOfRangeException (746 −> 7B0) request: 2202BD response: 7F2231 error code: 31
14:47:13.635 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] Failed getting VIN details from BrakeElectronics Module. Received:
    ↪ RequestOutOfRangeException (713 −> 77D) request: 22F190 response: 7F2231 error code: 31
14:47:13.635 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from TrunkElectronics Module. Received
    ↪ RequestOutOfRangeException (723 −> 78D) request: 2202BD response: 7F2231 error code: 31
14:47:13.639 | ALL | <− | 4 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from Engine Module. Received
    ↪ RequestOutOfRangeException (7E0 −> 7E8) request: 2210E0 response: 7F2231 error code: 31
14:47:13.639 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0713,0x00,0x00,0x00] data[0x2202BD]
14:47:13.641 | CAN1 | <− | 2 ms | [ISOTP] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0x62F1A2303034303630]
14:47:13.651 | CAN1 | <− | 10 ms | [ISOTP] cmd[0x5000] args[0x077C,0x00,0x00,0x00] data[0x7F2231]
14:47:13.651 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x22F190]
14:47:13.660 | ALL | <− | 9 ms | [Debug]: [DEBUG] [−] Failed getting VIN details from SteeringAssist Module. Received:
    ↪ RequestOutOfRangeException (712 −> 77C) request: 22F190 response: 7F2231 error code: 31
14:47:13.661 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x0712,0x00,0x00,0x00] data[0x2202BD]
14:47:13.666 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x5000] args[0x077F,0x00,0x00,0x00] data[0x62F1A2303031303134]
14:47:13.675 | CAN1 | −> | 9 ms | [ISOTP] cmd[0x5000] args[0x0715,0x00,0x00,0x00] data[0x22F190]
14:47:13.682 | CAN1 | <− | 7 ms | [ISOTP] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0x7F2231]
14:47:13.683 | CAN1 | <− | 1 ms | [ISOTP] cmd[0x5000] args[0x077D,0x00,0x00,0x00] data[0x7F2231]
14:47:13.692 | ALL | <− | 9 ms | [Debug]: [DEBUG] [−] Failed getting VIN details from InstrumentCluster Module. Received:
    ↪ RequestOutOfRangeException (714 −> 77E) request: 22F190 response: 7F2231 error code: 31
14:47:13.693 | CAN1 | −> | 1 ms | [ISOTP] cmd[0x5000] args[0x0714,0x00,0x00,0x00] data[0x222203]
14:47:13.693 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from BrakeElectronics Module. Received
    ↪ RequestOutOfRangeException (713 −> 77D) request: 2202BD response: 7F2231 error code: 31
14:47:13.718 | CAN1 | <− | 25 ms | [ISOTP] cmd[0x5000] args[0x077C,0x00,0x00,0x00] data[0x7F2231]
14:47:13.723 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x5000] args[0x077E,0x00,0x00,0x00] data[0x62220300BBBF]
14:47:13.727 | ALL | <− | 4 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from SteeringAssist Module. Received
    ↪ RequestOutOfRangeException (712 −> 77C) request: 2202BD response: 7F2231 error code: 31
14:47:13.730 | ALL | −> | 3 ms | [FRAMEWORK] cmd[0xF004] args[0x03,0x64,0x01,0x01] data[0x]
14:47:13.732 | ALL | <− | 2 ms | [Debug]: [DEBUG] [+] Request sent 2203 −> Received response 00BBBF. In InstrumentCluster
    ↪ Module
14:47:13.755 | CAN1 | <− | 23 ms | [ISOTP] cmd[0x5000] args[0x077F,0x00,0x00,0x00] data[0x62F190∗redacted∗]
14:47:13.765 | ALL | <− | 10 ms | [Debug]: [DEBUG] [+] VIN: ∗redacted∗ of vehicle Airbags module
14:47:13.765 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0715,0x00,0x00,0x00] data[0x2202BD]
14:47:13.768 | ALL | <− | 3 ms | [FRAMEWORK] cmd[0xF004] args[0x03,0x64,0x01,0x01] data[0x]
14:47:13.800 | CAN1 | <− | 32 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0x62F1A2303031313135]
14:47:13.809 | CAN1 | −> | 9 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x22F190]
14:47:13.826 | CAN1 | <− | 17 ms | [ISOTP] cmd[0x5000] args[0x077F,0x00,0x00,0x00] data[0x7F2231]
14:47:13.837 | ALL | <− | 11 ms | [Debug]: [DEBUG] [−] Failed getting vehicle mileage from Airbags Module. Received
    ↪ RequestOutOfRangeException (715 −> 77F) request: 2202BD response: 7F2231 error code: 31
14:47:13.857 | CAN1 | <− | 20 ms | [ISOTP] cmd[0x7F78] args[0x07DD,0x00,0x00,0x00] data[0x]
14:47:13.910 | CAN1 | <− | 53 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0x62F190∗redacted∗]
14:47:13.920 | ALL | <− | 10 ms | [Debug]: [DEBUG] [+] VIN: ∗redacted∗ of vehicle InformationElectronics module
14:47:13.920 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x0773,0x00,0x00,0x00] data[0x2202BD]
14:47:13.967 | CAN1 | <− | 47 ms | [ISOTP] cmd[0x7F78] args[0x07DD,0x00,0x00,0x00] data[0x]
14:47:13.985 | CAN1 | <− | 18 ms | [ISOTP] cmd[0x5000] args[0x07DD,0x00,0x00,0x00] data[0x6202BDE900BBBF0000514CEBD5]
14:47:13.995 | ALL | <− | 10 ms | [Debug]: [DEBUG] [+] Request sent 02BD −> Received response E900BBBF0000514CEBD5. In
    ↪ InformationElectronics Module
14:47:14.881 | CAN1 | <− | 886 ms | [ISOTP] cmd[0x4100] args[0x074B,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.884 | CAN1 | <− | 3 ms | [ISOTP] cmd[0x4100] args[0x074A,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.889 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x4100] args[0x074F,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.894 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x4100] args[0x0757,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.912 | CAN1 | <− | 18 ms | [ISOTP] cmd[0x4100] args[0x070A,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.912 | CAN1 | <− | 0 ms | [ISOTP] cmd[0x4100] args[0x070C,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:14.917 | CAN1 | <− | 5 ms | [ISOTP] cmd[0x4100] args[0x070E,0xE28B,0x00,0x00] data[0x0210035555555555]
14:47:23.211 | ALL | <− | 8294 ms | [Debug]: [DEBUG] [−] No response received on PassengerDoorElectronics Module, could not set
    ↪ diagnostic session. Is the ignition on?
14:47:23.211 | CAN1 | −> | 0 ms | [ISOTP] cmd[0x5000] args[0x074B,0x00,0x00,0x00] data[0x22F19E]
```

```
14:47:23.212 | ALL | <– | 1 ms | [Debug]: [DEBUG] [–] No response received on DriverDoorElectronics Module, could not set
    ↪ diagnostic session. Is the ignition on?
14:47:23.212 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x22F19E]
14:47:23.223 | ALL | <– | 11 ms | [Debug]: [DEBUG] [–] No response received on FrontSensorDriveAssist Module, could not set
    ↪ diagnostic session. Is the ignition on?
14:47:23.223 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074F,0x00,0x00,0x00] data[0x22F19E]
14:47:23.255 | ALL | <– | 32 ms | [Debug]: [DEBUG] [–] No response received on AdaptiveCruiseControl Module, could not set
    ↪ diagnostic session. Is the ignition on?
14:47:23.255 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on ParkSteerAssist Module, could not set diagnostic
    ↪ session. Is the ignition on?
14:47:23.255 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on CentralElectronics Module, could not set diagnostic
    ↪ session. Is the ignition on?
14:47:23.255 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on SteeringWheel Module, could not set diagnostic
    ↪ session. Is the ignition on?
14:47:23.255 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x0757,0x00,0x00,0x00] data[0x22F19E]
14:47:23.255 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070A,0x00,0x00,0x00] data[0x22F19E]
14:47:23.256 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x070E,0x00,0x00,0x00] data[0x22F19E]
14:47:23.256 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070C,0x00,0x00,0x00] data[0x22F19E]
14:47:33.212 | ALL | <– | 9956 ms | [Debug]: [DEBUG] [–] No response received in PassengerDoorElectronics Module, unable to
    ↪ retrieve module details.
14:47:33.212 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074B,0x00,0x00,0x00] data[0x22F190]
14:47:33.214 | ALL | <– | 2 ms | [Debug]: [DEBUG] [–] No response received in DriverDoorElectronics Module, unable to retrieve
    ↪ module details.
14:47:33.214 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x22F190]
14:47:33.223 | ALL | <– | 9 ms | [Debug]: [DEBUG] [–] No response received in FrontSensorDriveAssist Module, unable to retrieve
    ↪ module details.
14:47:33.224 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x074F,0x00,0x00,0x00] data[0x22F190]
14:47:33.256 | ALL | <– | 32 ms | [Debug]: [DEBUG] [–] No response received in ParkSteerAssist Module, unable to retrieve module
    ↪ details.
14:47:33.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received in AdaptiveCruiseControl Module, unable to retrieve
    ↪ module details.
14:47:33.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received in CentralElectronics Module, unable to retrieve module
    ↪  details.
14:47:33.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received in SteeringWheel Module, unable to retrieve module
    ↪ details.
14:47:33.256 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070A,0x00,0x00,0x00] data[0x22F190]
14:47:33.256 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x0757,0x00,0x00,0x00] data[0x22F190]
14:47:33.257 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x070E,0x00,0x00,0x00] data[0x22F190]
14:47:33.257 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070C,0x00,0x00,0x00] data[0x22F190]
14:47:43.217 | ALL | <– | 9960 ms | [Debug]: [DEBUG] [–] No response received on DriverDoorElectronics Module, unable to retrieve
    ↪  VIN.
14:47:43.217 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074A,0x00,0x00,0x00] data[0x2202BD]
14:47:43.218 | ALL | <– | 1 ms | [Debug]: [DEBUG] [–] No response received on PassengerDoorElectronics Module, unable to retrieve
    ↪  VIN.
14:47:43.218 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074B,0x00,0x00,0x00] data[0x2202BD]
14:47:43.224 | ALL | <– | 6 ms | [Debug]: [DEBUG] [–] No response received on FrontSensorDriveAssist Module, unable to retrieve
    ↪ VIN.
14:47:43.224 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x074F,0x00,0x00,0x00] data[0x2202BD]
14:47:43.256 | ALL | <– | 32 ms | [Debug]: [DEBUG] [–] No response received on ParkSteerAssist Module, unable to retrieve VIN.
14:47:43.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on AdaptiveCruiseControl Module, unable to retrieve
    ↪ VIN.
14:47:43.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on CentralElectronics Module, unable to retrieve VIN.
14:47:43.256 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on SteeringWheel Module, unable to retrieve VIN.
14:47:43.256 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070A,0x00,0x00,0x00] data[0x2202BD]
14:47:43.257 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x0757,0x00,0x00,0x00] data[0x2202BD]
14:47:43.257 | CAN1 | –> | 0 ms | [ISOTP] cmd[0x5000] args[0x070E,0x00,0x00,0x00] data[0x2202BD]
14:47:43.258 | CAN1 | –> | 1 ms | [ISOTP] cmd[0x5000] args[0x070C,0x00,0x00,0x00] data[0x2202BD]
14:47:44.740 | CAN1 | <– | 1482 ms | [ISOTP] cmd[0x4100] args[0x074A,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.745 | CAN1 | <– | 5 ms | [ISOTP] cmd[0x4100] args[0x074B,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.746 | CAN1 | <– | 1 ms | [ISOTP] cmd[0x4100] args[0x074F,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.777 | CAN1 | <– | 31 ms | [ISOTP] cmd[0x4100] args[0x070A,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.783 | CAN1 | <– | 6 ms | [ISOTP] cmd[0x4100] args[0x0757,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.787 | CAN1 | <– | 4 ms | [ISOTP] cmd[0x4100] args[0x070E,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:44.788 | CAN1 | <– | 1 ms | [ISOTP] cmd[0x4100] args[0x070C,0xE28B,0x00,0x00] data[0x032202BD55555555]
14:47:53.219 | ALL | <– | 8431 ms | [Debug]: [DEBUG] [–] No response received on DriverDoorElectronics Module, unable to retrieve
    ↪  mileage information.
14:47:53.219 | ALL | <– | 0 ms | [Debug]: [DEBUG] [–] No response received on PassengerDoorElectronics Module, unable to retrieve
    ↪  mileage information.
14:47:53.225 | ALL | <– | 6 ms | [Debug]: [DEBUG] [–] No response received on FrontSensorDriveAssist Module, unable to retrieve
    ↪ mileage information.
```

```
14:47:53.258 | ALL | <− | 33 ms | [Debug]: [DEBUG] [−] No response received on ParkSteerAssist Module, unable to retrieve mileage
    ↪ information.
14:47:53.258 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No response received on AdaptiveCruiseControl Module, unable to retrieve
    ↪ mileage information.
14:47:53.258 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No response received on CentralElectronics Module, unable to retrieve
    ↪ mileage information.
14:47:53.258 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No response received on SteeringWheel Module, unable to retrieve mileage
    ↪ information.
14:47:54.256 | ALL | <− | 998 ms | [Debug]: [DEBUG] [+] Found common mileage identifier −> E9
14:47:54.257 | ALL | <− | 1 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: Engine
14:47:54.260 | ALL | <− | 3 ms | [Debug]: [DEBUG] [+] Mileage in InstrumentCluster Module: 48063
14:47:54.260 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: CANGateway
14:47:54.268 | ALL | <− | 8 ms | [Debug]: [DEBUG] [+] Mileage Identifier Position: 0
14:47:54.272 | ALL | <− | 4 ms | [Debug]: [DEBUG] [+] Mileage in InformationElectronics Module: 48063
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: BrakeElectronics
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: PassengerDoorElectronics
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: DriverDoorElectronics
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: TrunkElectronics
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: FrontSensorDriveAssist
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: CentralElectronics
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: AdaptiveCruiseControl
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: SteeringAssist
14:47:54.272 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: AutoHVAC
14:47:54.273 | ALL | <− | 1 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: ParkSteerAssist
14:47:54.273 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: SteeringWheel
14:47:54.273 | ALL | <− | 0 ms | [Debug]: [DEBUG] [−] No mileage response found for ecu: Airbags
14:47:54.277 | ALL | <− | 4 ms | [Debug]: [+] Vehicle mileage check successful!
14:47:54.277 | ALL | <− | 0 ms | [Debug]: [!] Empty vin ['−−−−−−−−−−−−−−−−−'] found in ['TrunkElectronics'] Module(s)!
14:47:54.278 | ALL | <− | 1 ms | [Debug]:
14:47:54.278 | ALL | <− | 0 ms | [Debug]:
14:47:54.279 | ALL | <− | 1 ms | [Debug]: ======== VIN Report ==========
14:47:54.279 | ALL | <− | 0 ms | [Debug]: [i] Total number of VIN values found: 2
14:47:54.279 | ALL | <− | 0 ms | [Debug]: [i] Connection id: "1588769015010"
14:47:54.279 | ALL | <− | 0 ms | [Debug]: [i] Script description: "MQB/MLB Read VIN and Mileage"
14:47:54.280 | ALL | <− | 1 ms | [Debug]: [i] Script uuid: "177ebb30−5866−42a4−b96b−d6953ceb5577"
14:47:54.281 | ALL | <− | 1 ms | [Debug]: [i] Date & Time: Wed May 06 12:47:13 2020
14:47:54.281 | ALL | <− | 0 ms | [Debug]: [i] VIN: −−−−−−−−−−−−−−−−−
14:47:54.281 | ALL | <− | 0 ms | [Debug]: [i] List of ECUs ['TrunkElectronics']
14:47:54.282 | ALL | <− | 1 ms | [Debug]: [i] VIN: *****************
14:47:54.285 | ALL | <− | 3 ms | [Debug]: [i] List of ECUs ['Engine', 'InformationElectronics', 'Airbags']
14:47:54.285 | ALL | <− | 0 ms | [Debug]:
14:47:54.285 | ALL | <− | 0 ms | [Debug]: ======== End of Report ========
14:47:54.285 | ALL | <− | 0 ms | [Debug]:
14:47:54.285 | ALL | <− | 0 ms | [Debug]:
14:47:54.285 | ALL | <− | 0 ms | [Debug]:
14:47:54.285 | ALL | <− | 0 ms | [Debug]: ======== Mileage Report =======
14:47:54.285 | ALL | <− | 0 ms | [Debug]: [i] Total number of Mileage values found: 1
14:47:54.285 | ALL | <− | 0 ms | [Debug]: [i] Connection id: "1588769015010"
14:47:54.285 | ALL | <− | 0 ms | [Debug]: [i] Script description: "MQB/MLB Read VIN and Mileage"
14:47:54.304 | ALL | <− | 19 ms | [Debug]: [i] Script uuid: "177ebb30−5866−42a4−b96b−d6953ceb5577"
14:47:54.304 | ALL | <− | 0 ms | [Debug]: [i] Date & Time: Wed May 06 12:47:13 2020
14:47:54.304 | ALL | <− | 0 ms | [Debug]: [i] Mileage: 48063
14:47:54.304 | ALL | <− | 0 ms | [Debug]: [i] Number of ECUs found with mileage 48063[km]: 2
14:47:54.304 | ALL | <− | 0 ms | [Debug]: [i] List of ECUs ['InstrumentCluster', 'InformationElectronics']
14:47:54.304 | ALL | <− | 0 ms | [Debug]:
14:47:54.305 | ALL | <− | 1 ms | [Debug]: ======== End of Report ========
14:47:54.305 | ALL | <− | 0 ms | [Debug]:
14:47:54.305 | ALL | <− | 0 ms | [Debug]: [DEBUG] Mileage: 48063
14:47:54.305 | ALL | <− | 0 ms | [Debug]: [DEBUG] ECU details: [['InstrumentCluster', 'EV_RBD4K', '004060'], ['
    ↪ InformationElectronics', 'EV_MUHig6C3Gen2HBAS', '001115']]
14:47:55.207 | ALL | <− | 902 ms | [Debug]: [+] Main task finished!
14:47:55.207 | ALL | <− | 0 ms | [Debug]:
14:47:55.208 | ALL | <− | 1 ms | [Debug]: ======== Script finished! ========
```

References

Borkowski, P. (2017, November). *Study on the added value of further eu level measures addressing odometer manipulation in motor vehicles traded across the eu: economic analysis* (Tech. Rep.). EU Commission. (Accessed 09-02-2020)

Commission, E. (2012, July). *Roadworthiness package commission staff working paper impact assessment.* https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/road_worthiness_package/impact_assessment_en.pdf. (Accessed 6-02-2020)

Enerdata. (2020, February). *Change in distance travelled by car.* https://www.odyssee-mure.eu/publications/efficiency-by-sector/transport/distance-travelled-by-car.html.

Iso 14229 road vehicles – unified diagnostic services (uds) – specification and requirements. (2006). (International Organization for Standardization (ISO))

Jifeline. (n.d.). https://admin.jifeline.com/statistics/google/maps. (Accessed on 03-05-2020)

Jim, A. (n.d.). *An intro to threading in python: What is a thread?* https://realpython.com/intro-to-python-threading/. (accessed on 27-05-2020)

LLC, R.-T. (n.d.). *Vcds: Product information.* https://www.ross-tech.com/vag-com/product.html. (Accessed on 07-04-2020)

Pötsch, H. P. (2011, May). *Volkswagen - driving forward.* https://www.webcitation.org/6CMzXwGlU?url=http://www.volkswagenag.com/content/vwcorp/info_center/de/talks_and_presentations/2011/05/PPT_FFM.-bin.acq/qual-BinaryStorageItem.Single.File/Deutsche%20Bank%20Presentation%20Handout.pdf.

Python Software, F. (2020, April). *thread-multiple threads of control.* https://docs.python.org/2/library/thread.html.

RDW. (2014). *Tellerstanden en de rdw.* https://www.rdw.nl/particulier/voertuigen/auto/tellerstanden/tellerstanden-en-rdw.

Register, N. M. (2020). *What is clocking?* https://nmr.ie/car-clocking/. (Accessed on 08-02-2020)

Research for tran committee - odometer tampering: measures to prevent it, european parliament, policy department for structural and cohesion policies, brussel [Computer software manual]. (2017).