# Rogers + CMI AV Research Project Phase 2B report

## Introduction

### Work Term Scope

The scope for this work term pertains to understanding the underlying technologies used in vehicular communications, and how these technologies can be used in the development of a diagnostic data collection and retrieval platform. This includes an understanding of the Controller Area Network (CAN) bus, used for internal vehicle data communications, and the On-board Diagnostics II (OBD2) protocol, a protocol which dictates the required diagnostic data available on a vehicle’s CAN bus. Investigation includes active research, the investigation of and creation of software and hardware tools for interfacing with these protocols, and active development on a final deliverable demonstrating research findings into the prospect of a Data Platform Service for Distributed and Autonomous Vehicles.

## Controller Area Network and On-board Diagnostics II Overview

### Controller Area Network

CAN bus is a vehicle bus standard for transmitting data between a vehicle’s Electronic Control Units (ECUs) without the use of a central microcontroller. In a CAN bus, all devices are capable of transmitting messages which will be received by all other devices on the bus. Because any device can transmit at any time, CAN messages use an identifying ID, which is used to identify the message’s decoding rules, and determines the priority of the message. If two messages attempt to broadcast at the same time, the message with the higher priority is allowed to transmit, while the lower priority message must wait until the channel is clear.

A standard CAN frame may have a maximum of 8 to 64 bits of data as a message, which is contained in the data portion of the CAN frame. Other fields of a CAN frame include the 1-bit start frame, the 11-bit standard identifier ID field, the 1-bit remote transmission request field, the 16-bit cyclic redundancy check field, the 2-bit acknowledgment field, and the 7-bit end of frame field. These fields are only responsible for functions related to the transmission of the message itself. All data the message contains is stored in the data field of the frame. The data field may contain one or more signal values, which must be decoded to be understood. A visual explanation of the structure of a CAN frame can be seen in figure 1.

Diagram

Description automatically generated with low confidence

Figure 1: The structure of a CAN frame. The CAN ID and data fields are highlighted in orange and blue respectively.

### Decoding and Encoding with DBC Files

Within the data field of a CAN frame are the encoded values of the CAN message’s signals. A signal is a value which may be contained within an OBD2 message. These values are initially encoded and must be decoded to be meaningfully understood. The decoding rules for these values are usually proprietary to the vehicle manufacturer and are thus not commonly known. If obtained, the decoding rules for these messages are contained within a special file called a DBC file. A DBC file contains a list of known message IDs and the decoding rules for the signals matching that ID. If a frame has been successfully received, the message can be passed through the DBC file to produce the parameter values sent in that message.

A DBC file follows a specific structure. DBC files decode messages by matching the message ID to the signal decoding rules for the signals contained within the message. A DBC must have two separate syntax rules for matching messages and signals. A message is identified by the string, “BO\_”, followed by the message ID, the name of the message, and finally by the length of the data bytes. A signal is identified by the string, “SG\_”, followed by the signal name, the bit start, length, endianness, the scale and offset, minimum and maximum values for error checking, and finally the unit of the decoded signal.

Figure 2 displayed below demonstrates the decoding rules for a message of ID 2364540158, named EEC1, with a length of 8 bytes. This message can be decoded into a signal named EngineSpeed, which starts at bit 24, is 16 bits long, and uses the little-endian rule for ordering. To transform this signal into a readable value, it uses a scale of 0.125, and an offset of 0. When this message is passed through the DBC file, it will be run through these decoding rules, producing a numerical value between 0 and 8031.975, with a unit type of rpm. A decoded value which does not fall within this range indicates that the signal is invalid and is discarded.

Graphical user interface, application

Description automatically generated

Figure 2: The structure of a CAN message in a DBC file. The blue highlighted portions represent fields for message identification, including the CAN ID. The orange highlighted fields represented the decoding rules for specific signals within the data field of the CAN message.

Messages may also be multiplexed. In such a case, a message may contain more than one signal or signal set. This is the case for On-Board Diagnostics II (OBD2) messages. The below lines from a DBC file describe the decoding message for an OBD2 message with an ID of 2024, the message ID for an OBD2 response message. The length and response fields are the first decoded, followed the service field. The decoded service field indicates which of the two ParameterID\_Service signals, if any, are contained in the message. These decoded signals indicate which signal decoding rule to use to decode the OBD2 parameter contained in the message, which may be one of one-hundred ninety-six supported OBD2 parameters. A portion of an OBD2 DBC file can be seen in figure 3.

Text

Description automatically generated

Figure 3: Example portion of an On-board Diagnostics II DBC file. Unlike the CAN protocols which are proprietary to each vehicle and not known outside of the Original Equipment Manufacturer, OBD2 decoding rules are standardized and common knowledge.

### Accessing the CAN Bus

On a standard road vehicle, the CAN bus may be accessed through the OBD2 diagnostic connector port, usually located below the dashboard. This port may have up to 16 pins used to connect to the CAN bus. Ports 6 and 14 allow this port access to the vehicle’s interior CAN bus through CAN’s two communication wires, known as CAN high and CAN low. While intended for the transmission of OBD2 messages and diagnostic data, this port can be used to access the entirety of the CAN bus and to view the communications between all the vehicle ECUs. The available pins in this port may be seen in figure 4.

Timeline

Description automatically generated

Figure 4: Connections available in an OBD2 diagnostic connector port, located below the dashboard for most vehicles. Access to the CAN bus is provided through pins 6 and 14.

### On-board Diagnostics II (OBD2)

OBD2 is the CAN standard for the transmission of data related to the vehicle engine health and maintenance status. OBD2 was made mandatory in all road vehicles in North America starting in 1998 for the purposes of emissions tracking and has become more standardized around the world since. Almost all gas operated vehicles on the road today are compatible with OBD2. OBD2 is considered a higher layer protocol, using the CAN bus to handle lower layer logic. In this paradigm, the CAN bus is responsible for transmitting the OBD2 message within a CAN frame, with the data field of this frame containing the encoded OBD2 signals in hexadecimal format. The relationship between CAN and OBD2 can be seen in figure 6.

**Diagram

Description automatically generated**

Figure 5: The structure of an OBD2 message contained with a CAN frame. The CAN IDs of 7E8 and 7DF signifies that the CAN frame contains an OBD2 message.

Today, OBD2 is widely used to assess vehicle diagnostics using Diagnostic Trouble Codes (DTCs). When a vehicle detects a malfunction, it sends a DTC through the CAN bus, usually triggering the vehicle’s engine health indication light to activate. This also causes the vehicle to store the DTC as well as a freeze frame of the engine health parameters which detected the malfunction. These values can later be accessed by a technician to assist in the diagnosing of engine malfunctions.

OBD2 supports a list of 196 parameters which it can return in response to an OBD2 request, such as Engine RPM, Calculated Engine Load, Vehicle Speed, and Intake Air Temperature. These parameters each have a unique parameter ID, as well as associated decoding rules in the OBD2 DBC file. To receive an OBD2 response containing these parameters, an OBD2 request message must first be sent. An OBD2 request message has a message ID of 2015 in decimal or 7DF in hexadecimal, and contains fields describing the service mode, as well as the parameter value to be returned. An OBD2 response messages has a message ID of 2024 decimal or 7E8 hexadecimal and contains the signals which when decoded describe the service mode, length, response, parameter name, and parameter value which has been requested.

**Graphical user interface

Description automatically generated with medium confidence**

Figure 6: Example messages for OBD2 request and responses frames, with the CAN ID highlighted in orange and the encoded data highlighted in blue. The CAN ID 7DF represents an OBD2 request message, and the CAN ID 7E8 signifies an OBD2 response message.

OBD2 uses 10 diagnostic service modes by default. These modes are used to perform various functions related to sending and receiving diagnostic data. Service mode 1 is used to request and receive live data from the vehicle now the request is received. Service modes 2 and 3 are used to show freeze frame parameters and the DTC which the vehicle has stored after detecting a malfunction. Service mode 4 is used to clear these stored freeze frame data and DTCs. Other relevant service mode includes service mode 7, to detected recently detected DTCs, and service mode 9, to request vehicle information such as vehicle identification number (VIN). A list of OBD2 diagnostic service modes can be seen in the image below. A more thorough list of OBD2 service modes can be seen in figure 6.

Timeline

Description automatically generated

Figure 7: The ten standard service modes of ODB2. Vehicles may support more service modes than this standard list, and not all service modes will be supported by every vehicle.

The CAN message decoding rules for a vehicle’s internal communications are proprietary, and the DBC files required to decode these CAN messages must be either reverse-engineered or provided by the Original Equipment Manufacturer (OEM). In comparison, OBD2 is a standardized protocol which is required for be adopted by all gas-powered vehicles for emissions tracking. The DBC file and decoding rules for OBD2 data are not proprietary, and all OBD2 messages can be decoded with commonly available information. It is important to be aware that not all vehicles will support all OBD2 parameters and service modes, and that vehicles may also have support for OBD2 parameters and service modes which are not part of the standard OBD2 protocol.

## CAN Hardware Tools

### CSS CANedge2

The CANedge2 is remote logging device capable of automatically receiving and uploading timestamped raw CAN data to a virtual filesystem in a .MF4 file format. The CANedge2 can also be configured to automatically send OBD2 requests at scheduled intervals, allowing for automatic recording of diagnostic data from returned OBD2 responses. As an independent and contained device with no user interface, the CANedge2 is incapable of using a DBC file to decode CAN data, requiring a separate script to load the logged data from the virtual filesystem for decoding.

A picture containing electronics, remote, black, projector

Description automatically generated

Figure 8: The CSS Electronics CANedge2. A remote logger capable of reading a vehicle's CAN bus and automatically uploading the received CAN frames to a virtual filesystem in the cloud.

### PEAK PCAN-USB Adapter

A CAN adapter is required for a machine or computer to connect to the CAN bus through the OBD2 connector port. The PEAK PCAN-USB Adapter was used for this purpose, allowing for a host machine with CAN compatible software to be enabled to receive and send CAN messages directly through the OBD2 connector port. The PEAK PCAN-USB Adapter provides no logic or function beyond allowing connection with the CAN bus, and software capable of reading and sending CAN frames as well as decoding and encoding CAN messages must be present on the connected machine.

A picture containing connector, cable, white, adapter

Description automatically generated

Figure 9: The PEAK-System's PCAN USB CAN Adapter. Capable of providing stream access to the CAN bus through the vehicle's OBD2 connector port.

### Raspberry Pi

Raspberry Pi is a credit-card sized computer running Raspbian, a Linux derived operating system based on the Debian Linux distribution. Raspbian comes with all the standard suite of Linux packages installed by default, including the Linux SocketCAN Package required to interact with the CAN bus. A Raspberry Pi was used to allow for continual interaction with the CAN bus for in-vehicle data collection, processing, and logging.

A close-up of a circuit board

Description automatically generated with medium confidence

Figure 10: The Raspberry Pi, a credit card sized, microcomputer capable of running Linux operating systems and distributions. Allows for remote, computational operations and edge computing.

### MiFi 5G Mobile Hotspot

A mobile hotspot was used for Internet connection during live data collection. A MiFi 5G Mobile Hotspot provided reliable and fast internet connection on the road, allowing the CANedge2 and Raspberry Pi to receive and upload data when away from Wi-Fi connectivity.

A picture containing text, electronics

Description automatically generated

Figure 11: A 5G Mobile Hotspot. Provides fast, reliable internet access even when away from a Wireless Access Point.

### VFAN USB GPS Receiver

A USB GPS receiver was used to collected locational data in addition to diagnostic data. GPS receivers read satellite data to obtain GPS coordinate data structured into sentences following the NMEA protocol. When a collection of OBD2 parameters are obtained by the logger application, the current coordinates for latitude and longitude are appended to the document. By doing so, vehicle diagnostics can be tracked through location as well as by timestamp.

A pair of black headphones

Description automatically generated with medium confidence

Figure 12: A USB GPS adapter capable of satellite reading. Can provides GPS coordinate data through the reading of NMEA sentences.

## CAN Software Tools

### CSS Electronics Software API

Several software tools provided by CSS for decoding and processing data from the CANedge2 were used in the project. These include the CSS Electronics Python API, a set of modules in the programming language Python for decoding, processing, and iterating through the .MF4 file created and uploaded by the CANedge2. It also includes an online DBC file editor and viewer, and an online configuration editor to configure the CANedge2 device.

### Linux SocketCAN Package

SocketCAN is a set of open-source CAN drivers which are preinstalled on most major Linux distributions. SocketCAN allows for the creation of virtual CAN channels as well as connections to a physical CAN bus using a CAN adapter. SocketCAN handles all connection logic and allows for any Linux installed machine or computer to receive and send CAN messages. SocketCAN can only handle CAN messages in raw data, it provides no higher layer logic and has no decoding or encoding functionality. This functionality must be provided through a separate script.

### Python Libraries

Several libraries exist for the programming language Python for interacting with the CAN bus. Two of these used include the cantools library and the python-can library. The cantools library provides the lower-level interaction with the CAN bus, for direct receiving and sending of CAN messages and frames. These messages must be either decoded if received or encoded if sent. This is handled by the python-can library, which provides programming functions for interacting with CAN messages using a DBC file. By implementing both libraries together using the Socketcan Linux package, programmatic manipulation of the CAN bus is possible by a Linux installed computer or machine.

### Grafana

For visual demonstration of recorded diagnostic data, the data visualization and dashboard platform Grafana was used to create dashboards for displaying decoded OBD2 parameters in time stamped charts and displays. This allowed for the live display of received data and for the tracking of key diagnostic information, such as the average value and variance for recorded parameters. An example of a Grafana dashboard can be seen in figure 13.

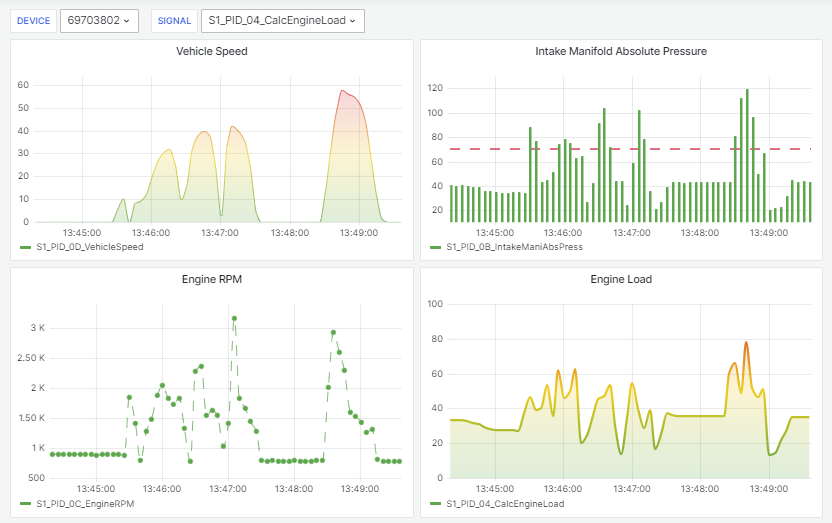


Figure 13: Example Grafana dashboard for displaying timestamped OBD2 parameters. The parameters shown are vehicle speed, intake manifold absolute pressure, engine RPM, and engine load.

## Developed Applications and Platforms

### Backend API Server

A backend web server was created to provide URL endpoints for the OBD2 Logger running on a configured Raspberry Pi device in the vehicle. In addition to providing the endpoints for receiving collected data, the API server also makes available numerous endpoints for requesting and retrieving collected data for the use in client applications. A practical use for this service includes machine learning, predictive maintenance, and fleet management. This server also provides the web facing interface for adding and configuring new vehicles for the data collection platform through a server-side browser accessible web interface.

### OBD2 In-vehicle Data Logger

The OBD2 Data Logger script was developed to manage the logic of recording and transmitting data directly from the vehicle. The data flow and application logic for the in-vehicle data logger can be seen in figure 14.

When booted, the logger periodically reads the configuration settings from the backend API. If these settings indicate that that the device has not been configured, the logger scans the connected vehicle for a list of supported OBD2 parameters, which it updates to the database. The web interface of the API server can then be used to configure the logger to scan for some or all the supported parameters, to deactivate or activate the logger, or to determine the time intervals for reading and uploading data. Whenever the logger updates its configuration, it will check whether it is activated or deactivated for logging. If it is, the logger will scan the vehicle to obtain the list of indicated parameters, which it will repeat until it has a list of timestamped OBD2 readings. After a timer has elapsed, the logger saves all collected data in a local file and uploads the collected data to the data platform through the Backend API endpoints.

Diagram

Description automatically generated

Figure 14: Data flow diagram for in-vehicle OBD2 logic application. Configuration settings are changed through web interface available on the backend API server.

### CAN MF4 File Decoder

CAN data collected by the CSS CANedge2 is initially saved in the .MF4 file format, a binary file format which requires special software to read. To read and eventually decode this data, an application was developed which could locate an MF4 file, retrieve recorded CAN messages, and then decode the CAN messages using a DBC file. The programming used to develop this application was later used to decode data collected by the CANedge2 as part of the CSS Electronics Data Collection Process.

Graphical user interface, text, application, email

Description automatically generated

Figure 15: Graphical user interface for MF4 file decoder used for decoding CAN messages contained with MF4 database files. The backend logic of this application was later used in the investigation into the CSS Electronics Data Collection Process.

### CAN Message Analyzer

Initial investigation into the CAN bus required an understanding of both the CAN bus and the available CAN compatible Python libraries. To investigate how these libraries can be used, a CAN Message Analyzer was developed capable of receiving CAN messages from the CAN bus in a continual stream, decoding these messages using a provided DBC file, and then displaying these messages and their decoded signals in tabular format. This application was developed on a Linux virtual machine, and used the SocketCAN Linux package and the cantools and python-can libraries for all CAN bus interaction. The analyzer application was successfully developed and used for testing CAN bus connectivity throughout the workterm. It allowed for the analysis of scanned CAN messages from test vehicles and for troubleshooting CAN connectivity errors.

Table

Description automatically generated

Figure 16: Graphical user interface for CAN analyser application. Capable of reading CAN bus and displaying frame data as well as decoding signals with a provided DBC file.

### MongoDB Data Platform Database

To store collected vehicle data, a MongoDB cloud database was utilized. This database was comprised of numerous collections to store timestamped vehicle data and logger configuration settings. The database is composed of four main collections: measurements, parameters, services, and vehicles. The measurements collection is a timeseries collection for storing collected vehicle data. The parameters collection stores fields related to the one-hundred ninety-six OBD2 parameters, such as the min and max values, the parameter ID, parameter name, and the structure of the parameter’s OBD2 request code. The service mode contains stored fields related to the ten default service modes of OBD2. The vehicles collections contain data related to vehicles configured for logging and is used to update the in-vehicle logger. The schema for this database can be seen in in figure 17 below.

### Diagram Description automatically generated

Figure 17: Schema diagram for MongoDB Data Platform database. Signal set collections contained recorded OBD2 parameters, while other collections are required for configuration and data relations purposes.

### Client Data Display Application

Once data has been collected and stored into the data platform database, it is made available to for external use by clients to use in the external software development and data analytics projects. An example of this is provided by a Python web application developed using Flask, which can request data from the data platform and display this data in time sorted line graph format. For an example of graphical data which can be displayed, see figure 18.

Chart, line chart

Description automatically generated

Figure 18: Line charts made with external data display application. Line charts are created by obtaining logged data through HTTP requests using the Backend API Server.

## Maintenance Architectures

### CSS Electronics Data Collection Process

As a proof of concept for vehicle diagnostic data collection through a 5G hotspot, a pathway was developed for logging OBD2 data from a connected vehicle using the CANedge2 and uploading this data to a S3 Bucket on the AWS cloud in .MF4 file format. This data was then loaded, decoded, and uploaded by a Python script using the CSS Electronics Python API to a cloud time series database. From there, the data was loaded using the data visualization platform Grafana to create timestamped graphs and displays.

This process demonstrated the viability of in-vehicle data collection for creating responsive and near-live displays of collected vehicle data through the CAN bus using an integrated device and a vehicle integrated hotspot.

Diagram

Description automatically generated

Figure 19: Architecture diagram for CSS Electronics Data Collection process. Consists of a linear sequence lacking a central platform for the collection and processing of data.

For further research into the prospect of building a real-time and live vehicle data collection and processing platform, additional development and customization was required, necessitating the design of an additional data collection platform.

### Vehicle Data Collection Platform

To further research into the potential of live data processing and collection, a more robust and customized process was developed by integrating together previous research into CAN, OBD2, and the software and hardware tools available for interfacing with CAN bus. This development culminated into the creation of a second, more centralized and programmatic iteration of the data collection process. A visualization of this process can be seen in figure 20 below:

Diagram

Description automatically generated

Figure 20: Architecture Diagram for Vehicle Data Collection Platform. Backend API server collects data from in-vehicle logger and provides database access to external clients and applications.

Communications from the vehicle are read by a Raspberry Pi, connected to the CAN bus through the OBD2 connector port by the PEAK PCAN-USB adapter. The Raspberry Pi receives the CAN bus communications through a SocketCAN channel from a Python script using the cantools library. Through this channel, the script periodically sends and receive OBD2 requests and responses. These responses are then decoded using the python-can library, and the decoded engine health values are checked for errors, preprocessed, and prepared to transmission. The values are then uploaded through the MiFi 5G Mobile to the Backend API server, where they are further processed, and stored in the MongoDB database in time series collections for later retrieval.

After the data has been stored in the data platform database, it is made available for use by external clients through the API endpoints. This data can then be used for data processing, machine learning, or other purposes.

## Conclusion

### Further Steps

The research completed over the course of this work term has contributed to investigating the CAN bus and the format of internal vehicle data transmissions for the purpose of diagnostic data collection. Early research focused on understanding the On-board Diagnostics II (OBD2) protocol, an upper layer protocol which determines the required format of diagnostic data communications within the CAN bus. After investigating the structure of OBD2, research was pursued to understand available hardware and software tools which support interfacing with CAN.

After investigating the available tools for CAN bus, the research findings were integrated into a vehicle data platform for collecting and retrieving vehicle diagnostic data from a time series database. Data collection was accomplished using a proof-of-concept integrated device capable of repeatedly scanning the CAN bus, processing the received data, and uploading collected data to the platform database through endpoints made available by the data platform’s backend API. The result of this research demonstrates a blueprint for a model for data collection for the purposes of predictive maintenance using 5G connectivity.