J1939 Tech Track Project Instructions

Introduction

The purpose of this document is to walk you through the course project that follows the J1939 and CAN tech tracks listed below.

* [CAN Basics Training A Practical Introduction to the CAN Bus](https://kvaser.com/course/can-basics-training-a-practical-introduction-to-the-can-bus/)
* [Introduction to J1939 (EN) – Learning Module J1939](https://elearning.vector.com/mod/page/view.php?id=406)
* [SAE J1939 Introduction](https://kvaser.com/about-can/higher-layer-protocols/j1939-introduction/)
* [The CAN Bus Protocol Tutorial](https://kvaser.com/can-protocol-tutorial/)

Included inside the project packet is an intercommunication description file (ICD) that will contain all the raw data and scaling values needed to interpret CAN messages.

***Note: This project cannot be completed without the J1939 tech track hardware. This should have been shipped to you upon the completion of the courses above.***

Hardware Setup

1. Plug in the USB from the controller into your computer. This will both power the unit, as well as letting you view certain data from the serial output.
2. Connect the high/low CAN wires to the controller. There is a DB9 adapter included that can be used for simply plugging in the PCAN adapter, although there is banana jack plugs if your setup breaks out the high/low wires.

Software Setup

* All of the software required for this project will be PCAN-View, the link for download is below.
  + <https://www.peak-system.com/PCAN-View.242.0.html?&L=1>
* This project does have a serial output, so if you download a virtual terminal emulator, you can easily view details about what is happening with the code. I have been using Tera Term, the download is listed below, although feel free to use any terminal emulator of your choice.
  + <https://tera-term.en.softonic.com/>
  + Port: COM6
  + Baud: 9600
  + ***Disclaimer:*** Tera Term does work most of the time, although it can be buggy at times and starts printing garbage. Typically restarting Tera Term will help, if that doesn’t work power cycle the controller as well.

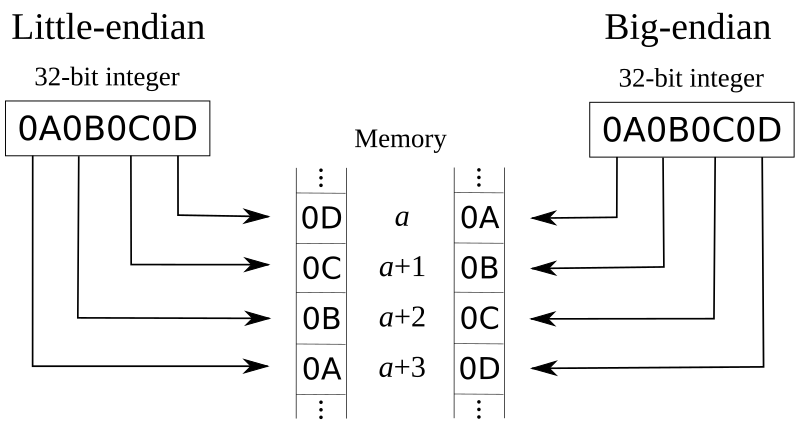
Project

The premise for this project will rely on gathering data sent from the controller on a specific SPN and sending it back to the controller on a different SPN. All the data for SPN(start position/bit length), PGN, Scaling Factor, and Offset will be located in the ICD file.

The states below correlate to a state machine the controller uses to determine what data to send and when to send it. If the correct data is sent, the controller will move on to the next state.

**State Machine Note:** A typical CAN application will cyclically send ***all*** the messages for the application the entire time, unless that specific message is on request. For the sake of simplicity this application will only send the CAN message for the current state the controller is in. This will make it easier to identify what state you are on and what message to reply to.

***CAN Data TransmissionNote:*** All data should be transmitted over the CAN bus in little-endian format. There is a reference image below to visually show how the data is packed for transmission.



***CAN ID***

* The CAN ID used to send out messages from PCAN will be partially made up from the PGN provided to you in each state. The rest is made up of predetermined data that is defined below.
  + Priority = 6
  + EDP = 0
  + DP = 0
  + Source Address = F2

***State Initiate***

1. In the boot up state, *State Initiate*, the controller will be sending out a CAN message on the PGN 0xFF00. The SPN for this state is called ‘State Initiate Rx’ and will be located within the PGN 0xFF00 at bit position 1.1 (*this means the first bit inside the first byte).* 
   1. 0xFF01 [1.1]
2. Using the recorded data, send this data out on the SPN ‘State Initiate Tx’. As defined in the ICD file, this SPN is located on the same PGN 0xFF00, although it will start at bit position 2.1.
   1. 0xFF01 [2.1]
   2. If this is completed correctly, the state machine will move on to State 1.

***State 1: Scaling Factor***

1. Using the ICD file to locate the correct SPN position, record the data from the SPN ‘State 1 Rx’ from the controller.
   1. 0xFF01 [1.1]
2. Once the data is received, apply the scaling factor.
   1. Scaling factor = 0.1
3. Next, add 1 to that converted value.
4. Lastly, you need to send the newly found value back over to the controller, which means you will have to divide that value by the scaling factor to send it.
5. Send out the value on SPN ‘State 1 Tx’
   1. 0xFF01 [2.1]
   2. Scaling factor = 0.1

***State 2: 2 Bit values***

1. Using the ICD file to locate the correct SPN position, record the data from the SPN ‘State 2 Rx’ from the controller.
   1. SPN: 0xFF02 [4.1]
2. Take the recorded data and ***subtract 1 from that value***.
3. Send out the new value on SPN ‘State 2 Tx’
   1. 0xFF02 [4.3]

***State 3: 16 Bit Values***

1. Using the ICD file to locate the correct SPN position, record the data from the SPN ‘State 3 Rx’ from the controller.
   1. ***Note:*** This is the first value that is larger than 8 bits, be sure to send the data in little endian format, otherwise it will be incorrect on the controller side.
   2. 0xFF03 [4.1]
2. Next, ***record this value***. It will be used in a later step.
3. Lastly, send out the recorded data on SPN ‘State 3 Tx’
   1. 0xFF03 [6.1]

***State 4: 32 Bit Values***

1. Using the ICD file to locate the correct SPN position, record the data from the SPN ‘State 4 Rx’ from the controller.
   1. 0xFF04 [1.1]
2. Next, add that value to the value you found in the previous state. This can be done by simply converting both values to decimal and adding them, be sure to convert back to hex before the data is sent out.
3. Send out the new value on SPN ‘State 4 Tx’
   1. 0xFF04 [5.1]