June 2013

CPE Senior Project

Andrew Fong

[OBD2 PID READER]

Table of Contents

Table of Contents 2

List of Tables and Figures 3

Acknowledgements 4

Abstract 5

I. Introduction 6

II. Background 7

III. Requirements 8

IV. Design 9

VI. Development 10

V. Testing and Results 12

VI. Conclusion and Recommendations 16

VII. Bibliography 18

VIV: Appendixes 19

Appendix A: Analysis of Senior Project Design 19

Appendix B: Parts List 21

Appendix C: Hardware and Connections 21

Appendix D: How to Manual 21

Appendix D: OBD Visual Basic Code 23

List of Tables and Figures

Figure 1: OBD2 Connector 7

Figure 2: Software Flow Diagram 9

Figure 3: HyperTerminal ELM327 Read 122

Figure 4: PID List(Tab2) Select PIDs 133

Figure 5: Show PID Data for Selected PIDs 144

Figure 6: More extensive PID Read 144

Figure 7: Custom PIDs on Volt 155

Figure 8a: Car's OBD Connector 21

Figure 8b: ELM327 Connected 21

Figure 9: Mode Select 22

Acknowledgements

I would like to thank Dr. Dolan for his insight and guidance throughout the project. In addition, he provided the Bluetooth ELM327 device as well as a Chevy Volt to test the project on.

Sincerely,

Andrew Fong

Abstract

The most current standard on automobiles for monitoring data is OBD-II. Using On board diagnostic Parameter ID (OBD PID) codes hobbyists and mechanics have the ability to monitor, chart, and obtain data on cars by communicating to the ECU. The OBD-II connector is often located under the driver's side inside of the car. The hardware will consist of a generic Bluetooth OBD2 connector and a laptop running windows. The software will be an program exe created by visual basic. The software created in visual basic will allow the user to examine PID codes in order to record and discover new data on the OBD bus.

I. Introduction

General Knowledge

An OBD-II standard was mandated for all automobiles sold in the United States since 1996 [2]. The port is often located on the driver's side of the car underneath the steering wheel on the left or right side. This data is read-only so it can in no way cause harm to the car as it is merely a diagnostics tool. A connector is hooked up to this port and is usually interfaced with by Bluetooth, usb, or a connected display. Reading this port gives the user access to data to monitor the chasis, body, and accessories of the car as well as other diagnostic data [2].

The device that this project uses in particular is the ELM327 chip. Its peripheral interface is either Bluetooth or usb, in this project a Bluetooth connection was used. This chip is able to read five different signaling interfaces: SAE J1850 PWM, SAE J1850 VPW, ISO 9141-2, ISO 14230 KWP2000, ISO 15765 CAN [5]. However, the CAN interface has become a requirement on vehicles in the US starting from the year 2008 [5]. Using this interface, allows for a higher possible range of data as it is a 11-bit BUS. The ELM327 Bluetooth device handles all the interfaces by using its onboard commands to select the protocol. The device pre-handles the headers and format. For instance, if one wants to send the a Parameter-ID, the user just has to enter the mode and then a supported OBD2 PID [5]. The result is then responded in a hex message string which one needs to use a program to decode it into human readable data.

The goal of the project is to read custom and standard (Mode 01) On-Board Parameter-IDs for OBD2 vehicles. Using the ELM327, the user will be able to use the program to monitor and poll for new codes later being able to chart the hex strings to determine what the code represents exactly. In addition, the operator can simply use draw from the standard PID list that is supported on all vehicles to access known data.

II. Background

Hardware

A 16 pin J1962 connector is standard for all cars to interface with. There are five communication types using this layout: J1850 VPW, ISO-9141-2, KWP2000, J1850 PWM, and CAN are set by the metal contacts that the connector touches [5]. At the start of 2008 cars in the United States were required to have a new standard on top of this which was using a CAN bus [3].

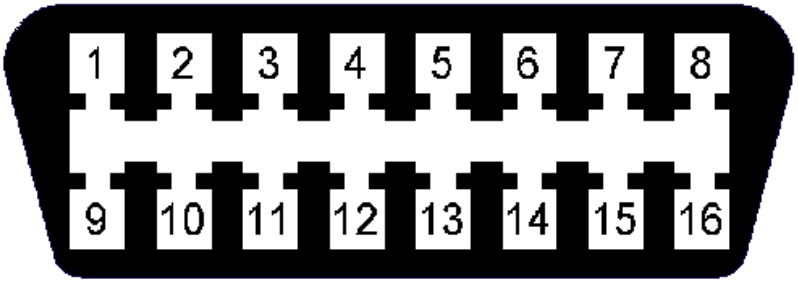

Figure 1: OBD2 Connector

In all cases pins 4 and 5 are both tied to ground and 16 is to the 12V battery. Based on what interface is used, two other pins are used for the data lines. In the CAN implementation, 6 and 14 are used. The advantages of CAN are that it supports a higher bit transfer rate and 2 buses support by a CANH (Pin 6) and CANL(Pin 14) [5].

The device used in this project is a generic Bluetooth ELM327. The Bluetooth connectivity allows for freedom to place the device accessing the ELM327, but at the cost of some speed. This connector allows for the user to monitor data through a computer or phone and potentially could be added to onboard GPS navigation systems. The ELM327 software addresses the CAN implementation by preloading the message header with 0x7DF and formats the hex string appropriately to a 7 byte CAN message [6]. Elm Electronic Inc's ELM327 microchip handles the detection of the proper protocol used by the car as well as sets up the header for the protocol to be used this is hidden by default and can be shown with the command AT H1 [1].

III. Requirements

Marketing Requirements	Engineering Specifications	Justification
1	This system costs less than $30.	Just need to purchase a Bluetooth ELM327 chip to use to connect to a PC.
3	Support all OBD2 Protocols	To use the software on any vehicle.
2, 3	Monitor OBD2 Data	Monitor list of standard PIDs [5].
2, 3, 4	Send Custom OBD PIDs	Send non-standard/special PIDs to ELM327 in an attempt and to read new data.
5	Bluetooth ELM327	Allows user to connect a laptop without requiring a cable to reach across the user.
1,2,3, 4	Create software to run on Windows	Read and send codes to the ELM327 device.

Marketing Requirements

1. Low cost.
2. Easy to use
3. Support all protocols
4. User input
5. Portability

IV. Design

Hardware

See Hardware connection in Appendix C.

Software

After connecting the ELM327, the device is to be connected using Bluetooth from a computer using windows. A program will be devised in visual basic to allow users to check for standard PIDs as well as monitor non-standard PIDs such as manufacturer specific codes that are unreleased to the public. A visual basic program was constructed to allow the user to examine data. The device is to be interfaced using serial and the com connection created by the Bluetooth device. After connecting to the COM port the code will the initialize an array that corresponds to the standard PID table [6]. When initialization is finished, the user will be able to select to look at the standard PIDs or enter their own (Figure 2). The standard PID populates the list in Figure 4 and then pushes the checked list to the readPID tab shown in Figure 5/6. The other two modes allow the user to input their own custom PID and receive results.

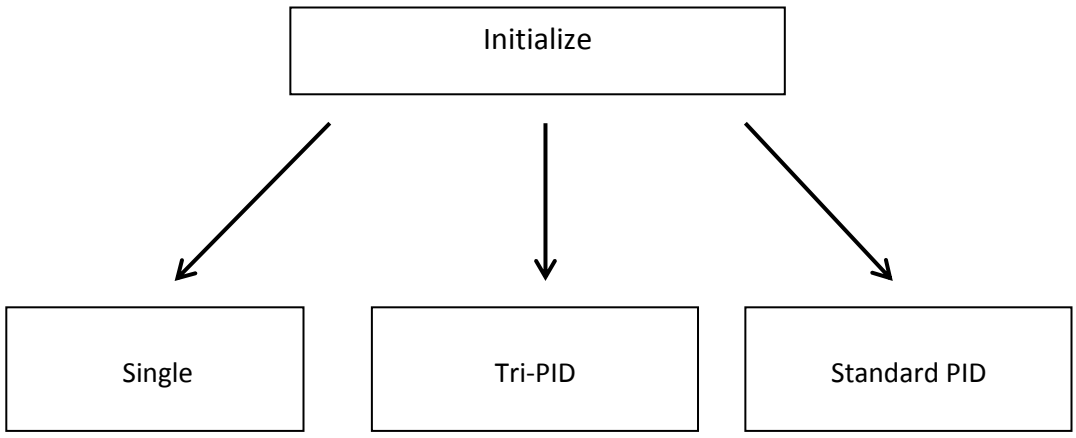

Figure 2: Software Flow Diagram

VI. Development

The development was done using Microsoft Visual Studio to create visual basic code to connect to the hardware. Using the ELM327 device and a laptop with both Bluetooth and a windows operating system, initial tests were done using a terminal client to read data from the ELM327. After sending this data, bytes of data were returned at monitored to understand the data collection process. Tests were done using a terminal application and reading data from the program.

To initialize the ELM327 chip for use, the program first connected to the serial COM port created by the Bluetooth link. First, an AT Z command is sent to reset the settings on the ELM327 [1]. Next, the command AT L1 sets up the output for terminal style feeds [1]. The code also toggles off programmable parameters, turns echo on, and turns on headers [1]. While testing on the Chevy Volt, an issue occurred as Flow Control was also enabled by default. As opposed to the previous CAN vehicle I tested on it ran a loop reading multiline input that would not stop, this hindered the results of my program. As a fix, I turned flow control off allowing for single line input feeds using the command AT CFC0 [1].

After making comparisons, a standard PID calculation list was constructed to support the 101 PIDs for Mode 1 for PIDs 0x00 to 0x64 [6]. A successful response to a mode 01 PID would be for example 41 00 to a 01 00 query would be followed by 4 bytes of data [6]. In Figure 9, the section above the single line PID entry (1) generates a header that shows the connected status of the device. Looking at Figure 7, information on the ELM327 device is shown in the top section. Next, the OBD protocol the ELM327 uses to connect to the vehicle is specified. Finally, the manufacturer of the device is listed at the end.

The other functionality is to allow users to attempt to read custom PID codes. As shown in Figure 9, the design to add the option to input up to 3 custom PIDs was added. For the tests on the Chevy Volt, it was noted that GM vehicles use mode 22 [6]. This section sends the PID entered into the

textbox from left to right, first sending the hex string to the ELM327 and waiting on the response then sending the next PID. After completing the loop, the thread for then loops on itself, and will continue to print until it is cancelled by the button. The cancel button for the tri-line PID also stores the data into the file “C:\EngineMonitoring\CPID.txt”. The single line PID sends the PID only once but still requires the user to cancel the string in order to be run again. The standard PID is used in the same method where the ReadPID in Figure 5 starts the thread. After this, the overhead for the program generates the supported PID list for the vehicle and updates it in Figure 4. The user then selects the PIDs they wish to monitor using the check boxes and presses refresh to push the ids to the screen in Figure 5. The data in Figure 5 automatically updates after a loop of reading values is completed.

V. Testing and Results

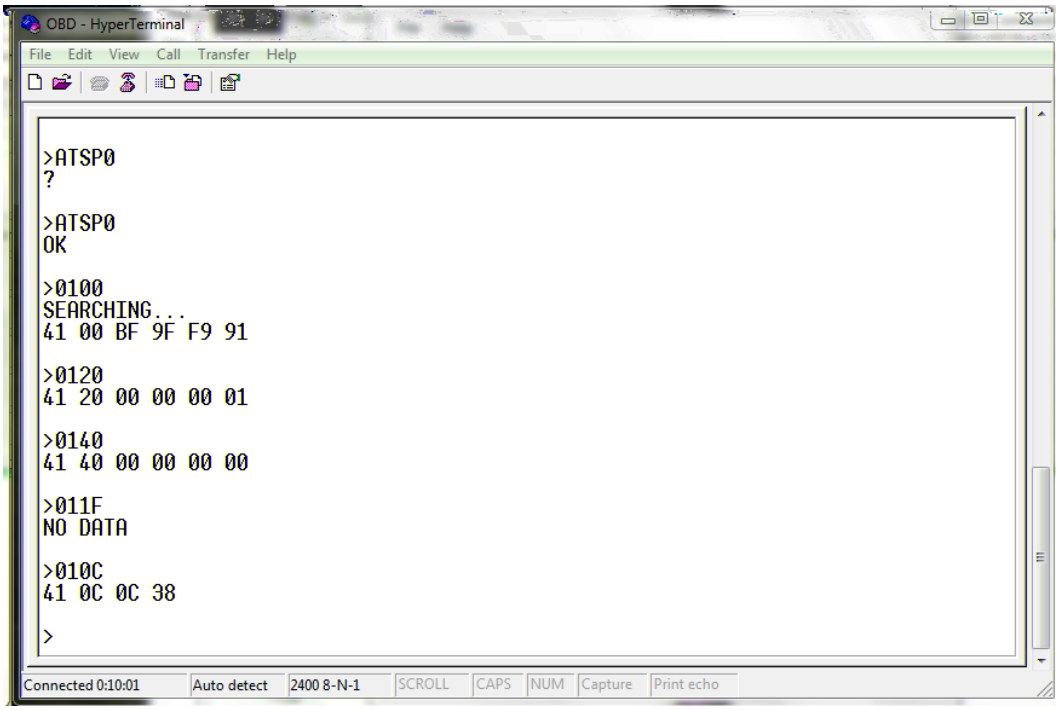

Figure 3: HyperTerminal ELM327 Read

Testing was initially done using HyperTerminal as shown in Figure l. After connecting to the Bluetooth device in windows, the Bluetooth manager was opened to detect that the device used COM ports 3 and 4. PIDs 00, 20, and 40 were read in mode 1 to determine which standard PIDs were supported by the vehicle. In figure, the responses after the requests for the available pids returns 41 to confirm a valid response to mode 1 followed by the pid entered [6]. After that the following, data is a bitmap to which PIDs in mode 1 are supported. In the 4 byte string response for the code 01 20, the last bit of the hex string means that PID 0x40 is available. Bit masking is done to check the status of supported bits for these PIDs.

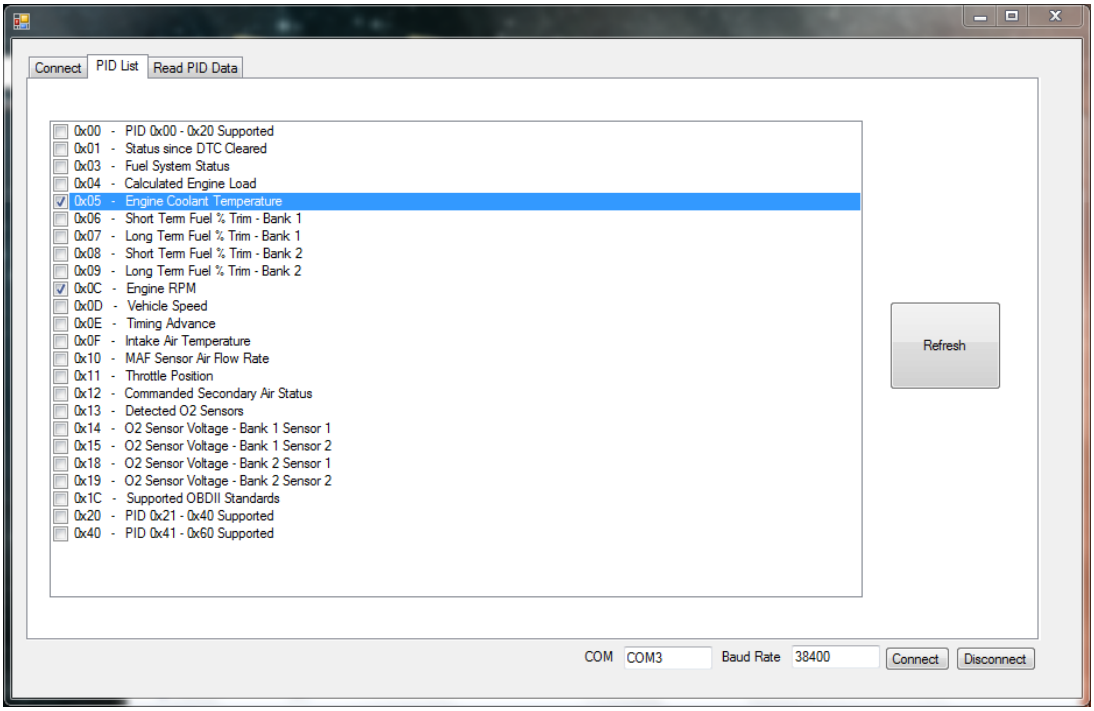

Figure 4: PID List(Tab2) Select PIDs

Next, I began to construct my program in which I constructed a table to convert the hex string responses into human readable data. After using others' code to connect to the Bluetooth using the serial COM port, the code read in the strings into a function. At first, work was done to determine the bitmap and determine which PIDs were supported by the vehicle, this turned on or off elements in my array. In Figure 4, shows the program determining which PIDs were supported and creating a list. After generating this list, the user is able to select what PIDs to monitor.

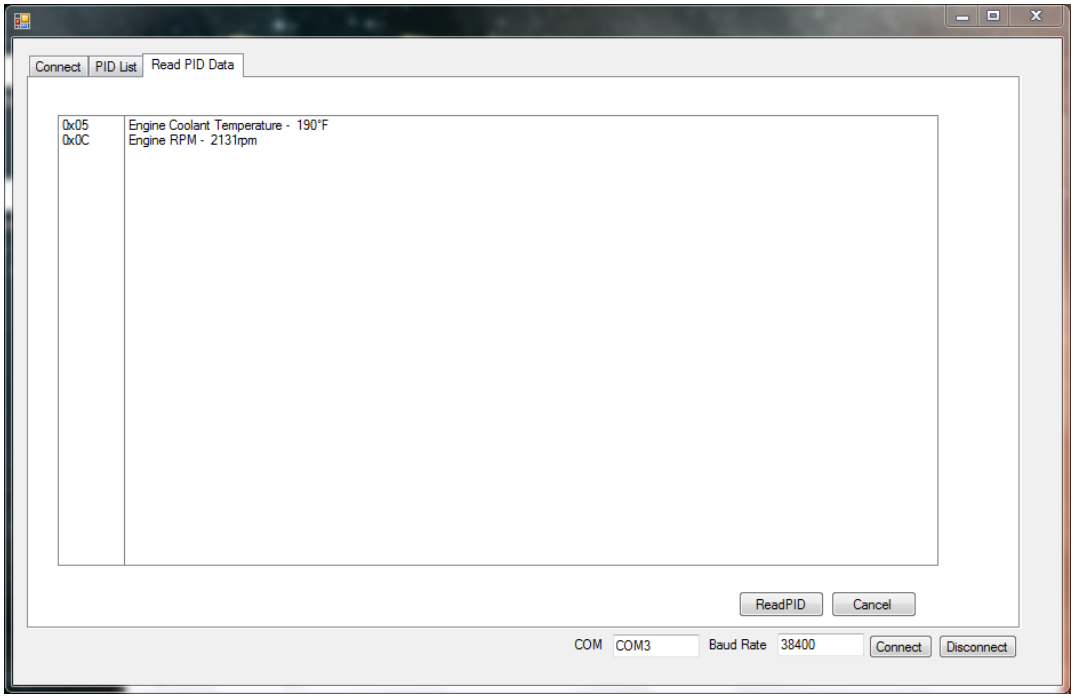

Figure 5: Show PID Data for Selected PIDs

In tab 3 shown in Figure 5/6, the user is reading the data select by the boxes in the previous tab (PID List). Testing was done on the custom program created in visual basic and a reference program. After verifying consistent data, tests were done on the all supported PIDs for the vehicle [6].

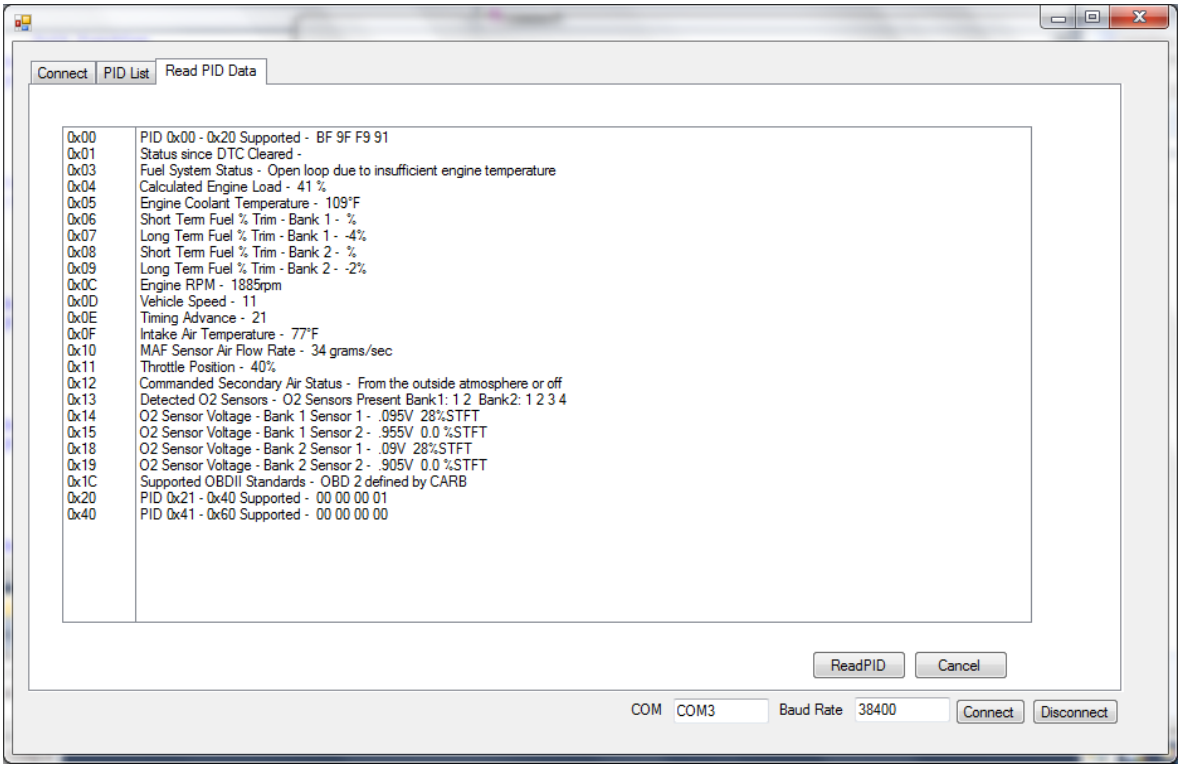

Figure 6: More extensive PID Read

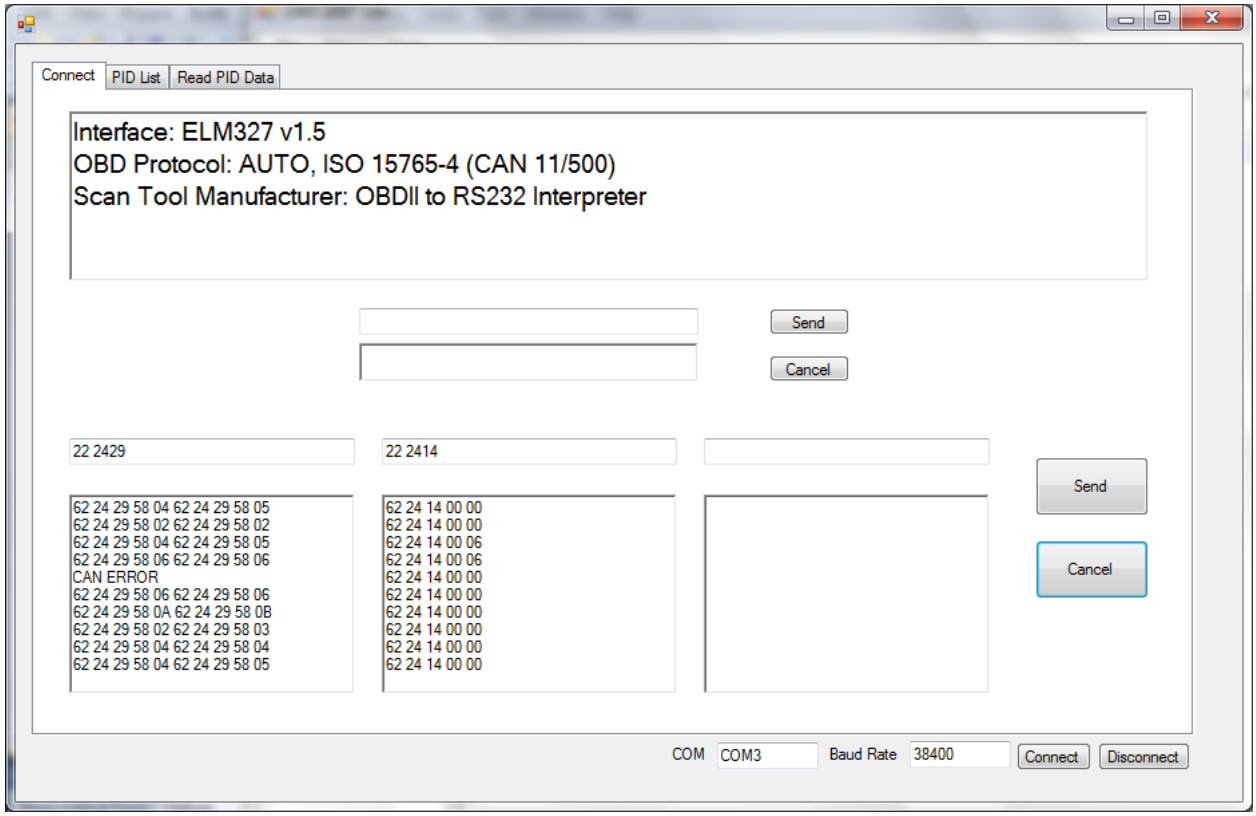

Figure 7: Custom PIDs on Volt

Finally, the next step involved testing custom PIDs (Figure 7). The final tests were done a Chevy Volt. On GM cars, there is a special pid mode 22 the valid response for this query would have 62 then the PID[6]. Using discovered data users would try to poll through the mode 22 and a 2 byte PID to discover new PIDs that are not required by the government via the standard PID list [6]. These manufacturer specific codes have much more data and information. In this case, the program wants to monitor 22 2414 and 22 2429, which are the Hybrid Voltage discharge amps and voltage respectively. In this mode, valid responses show 62 to confirm the mode 22 followed by the custom pid in this case which is 2 bytes. The string responded with 62 24 29 58 04. To convert this, the formula was described to be (Signed(A)*256 + B)/64. The result for the string would be approximately 352V. A is multiplied by 256 since theres multiple bytes in the calculation, the string is merely converted into its decimal value. After polling the data, users have to then convert the hex string into decimal and understandable data.

VI. Conclusion and Recommendations

The system completed the requirements of allowing the user to input custom codes. A response was given if the code was invalid and the triple PID input did its job of allowing the user freedom to monitor multiple codes at once. One issue that was encountered during the tests on the Chevy Volt, was that for mode 01 PID 00 and all of the bit masking queries there would be multiple lines of data. This is due to the fact there are multiple ECUs on the vehicle and it is sending the responses from each one that holds a different amount of data to query the bus for. This issue needs to be solved or else the automation on the read of standard PIDs. One method would be to break up the string returned if it is greater than the size of a response that is entered into this field which is 3 bytes for the validity of the PID and 4 data bytes. After this, the user would compare how many bits are enabled for each of the strings and decide to use the longest string. Other than that the code accomplished its task and allowed the user to monitor the hex strings and is a start for allowing hobbyists to discover new PIDs.

To improve the project there are a few tasks that need to be accomplished. The first would be auto-detecting a baud rate. Currently, the program requires the user to input the baud rate that the Bluetooth ELM327 uses. To solve this issue, the program would most likely send a reset command (AT Z) to the device at baud rates of 4800, 9600, 19200, 38400, and so on [1]. If a response of OK is received, then the ELM327 has successfully responded to the user and reset the ELM327. Next, the task would be to improve the data logging functionality of the program, this would enable a button to save the data to text file. Some of this functionality is enabled in the cancel button of the Tri-Line PID but one may want to record timing to monitor changes in data. Prior to storing, some checks on valid PIDs such that the string needs to be at least 4 characters and if it is an ELM327 message it starts with AT could improve the program for the user. In addition, to logging the data of custom PIDs, graphing the data using a line graph could assist the user in isolating what data/component the PID corresponds to or even monitor

standard PIDs. Furthermore, smartphone could be implemented to minimize on spatial and increase portability.

VII. Bibliography

[1] "ELM327 Datasheet." ELM Electronics

URL: http://elmelectronics.com/DSheets/ELM327DS.pdf

[2] "OBD-II Background Information." *OBD-II Background Information*. B&B Electronics, 2011.

URL: http://www.obdii.com/background.html

[3] "Control of Air Pollution From New Motor Vehicles and New Motor Vehicle Engines; Modification of Federal On-Board Diagnostic Regulations For: Light-Duty Vehicles, Light-Duty Trucks, Medium Duty Passenger Vehicles, Complete Heavy Duty Vehicles and Engines Intended for Use in Heavy Duty Vehicles Weighing 14,000 Pounds GVWR or Less." EPA. Environmental Protection Agency, 19 Dec. 2005.

URL: http://www.epa.gov/fedrgstr/EPA-AIR/2005/December/Day-20/a23669.htm

[4] Chevy Volt Mode 22 PIDs "VPIDs-labeled Spreadsheet." Professor at UCCS via GM-Volt Forums.

URL:
https://docs.google.com/spreadsheet/ccc?key=0AvK9F6VeA7dvdFUwSjBzR2FQM1BJTWZoLW43ZFdoTFE#gid=0

[5] "OBD-II PIDs." *Wikipedia*. Wikimedia Foundation, 28 May 2013.

URL: http://en.wikipedia.org/wiki/OBD-II_PIDs

[6] "On-board Diagnostics." *Wikipedia*. Wikimedia Foundation, 06 Dec. 2013.

URL: http://en.wikipedia.org/wiki/On-board_diagnostics#OBD-II

VIV: Appendixes

Appendix A: Analysis of Senior Project Design

Project Title: OBD2 PID Reader

Student's Name:	Andrew Fong	Student's Signature:	
Advisor's Name:	Dale Dolan	Advisor's Initials:	Date:

• Summary of Functional Requirements

The software is designed to work with an ELM327 device and allow the user to obtain data for standard PIDs. The user can also enter in a custom OBD PID to scan for non-standard PIDs allowing them to acquire additional data on the vehicle. Custom PIDs work in two conditions single one line entry or multiple line reads.

• Primary Constraints

Significant challenges occurred with testing on different vehicles. One CAN vehicle I tested worked perfectly, however another had preset modes on the ELM327 enabled that were not supported after a reset on the first device. Time constraints as well as lacking a CAN device to consistently test on limited my approach. Issues such as the automatic enabling of Flow control mode (multiline output) which affected my design initially as the ELM327 would not stop outputting until a break message was sent. This directed the program to enable more commands at initialization to provide consistency across different vehicles.

• Economic

The components for the system cost roughly between $20-30 USD.
The project used a donated ELM327.
Materials needed for the project are: ELM327, 1996 or newer automobile, Laptop running windows with Bluetooth capabilities
This project is free and allows hobbyists to share information and reverse engineer data from their car's ECU.

• Environmental

The ELM327 device draws power from the car's battery this requires the key to be in the ignition. The device contains a sleep mode as well as a low power mode. Bluetooth may cause interference with wifi and other Bluetooth devices this may affect the reliability of the data sent to the device or other devices in the area. Manufacturing of the ELM327 device requires the resources of plastic, silicon, and copper. The manufacturing plants also emit greenhouse gases as well as use the resource of power in some way, which in turn can affect other species.

• Manufacturability

This project wasn't developed to be manufactured however; other companies along the same line have created software similar to read standard PIDs. The interface would need to be more user friendly to allow less technical users to use the device appropriately. Besides this, manufacturing can also be done to re-implement the ELM327 chips using an alternative microcontroller for control such as an Arduino uno.

• **Sustainability**

The project is sustainable since it uses existing technology for the ELM327. This device can also features a low power mode (AT LP) and sleep functionality. Other than this hardware device, the software is stored in memory and has little issue with real world resources besides a small amount of memory on a computer. Since the OBDII standard has existed for over a decade it's a reliable and consistent format that may not be upgraded for a while.

The requirement of a laptop with windows is not exactly the most sustainable device to always lug around for monitoring a vehicle, thus upgrades can be made to convert the project over to a smartphone, tablet, or possibly integrated into a navigation system changing the portability of the system. Issues with this design may arise depending on how the Bluetooth stack is integrated.

• **Ethical**

Since the device acts as a read-only connection to the ECU (Central computer for cars) the code cannot harm the car's digital circuits or reroute information. However, users can still clear diagnostic troubleshooting codes on the car or access the car's VIN number. These are minor changes that in reality won't harm the vehicle.

• **Health and Safety**

There are no health and safety issues with the project.

• **Social and Political**

There are no major social issues associated with this project. Political issues associated with reverse engineering non-standard PID codes and selling them may arise with conflict to the Equipment and Tool Institute and manufacturers. Not much data if any, has been leaked from these sources, however an online community online seeks to discover the information so far not making any changes to affect stakeholders' stance. Stakeholders of the ELM Electronics and other stakeholders may have issues regarding the creation of fake ELM327s created for profit since they created both the circuit and software for the device.

• **Development**

New tools for development were learned from this project in the coding language of Microsoft Visual Basic. For example, it was learned how to create forms to display and buttons for an executable in which the data for the project was shown.

Appendix B: Parts List

Generic ELM327 Bluetooth Adapter $20-40
OBD2 Supported Vehicle 1996 or Newer, 2008+ if CAN interface desired
PC running Windows with Bluetooth adapter

Appendix C: Hardware and Connections

Figure 8a: Car's OBD Connector

Figure 8b: ELM327 Connected

Appendix D: How to Manual

Step 1) Connect Bluetooth ELM327 to car. See Appendix C.

Step 2) Connect to Bluetooth from PC.

Step 3) Run devmgmt.msc or open up bluetooth devices to find the COM port that the device is using

Step 4) Open the program OBDXR.exe

Step 5) Set the COM port to the one in Step3 and the baud rate at 38400. See bottom of Figure 8.

Step 6) Press connect and if the top box in the connect tab show the interface, protocol, and scan tool manufacturer the user has successfully connected. See bottom of Figure 8.

Step 7) Choose to use one of the three modes:

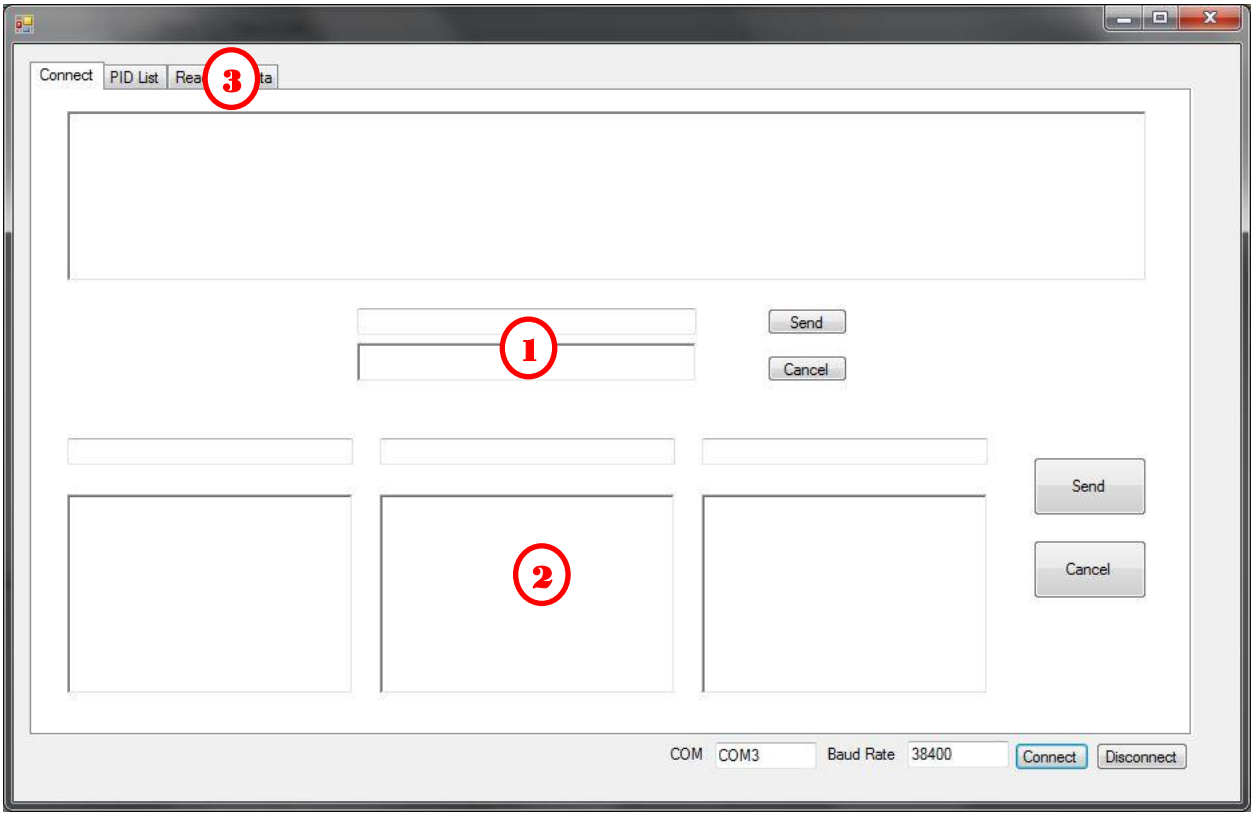

Figure 9: Mode Select

Single PID Entry (1)

Use the middle box to send a mode (1 byte) then pid (1 byte up to 7 bytes) message.

Tri PID Entry (2)

Enter in as many PID entries into the three boxes. Press send to begin the thread until the user wants to stop querying the device by pressing cancel then stores the data into

Standard PID Reader (3)

To begin, go to the Read PID Data tab from this tab press readPID to begin the thread, this initializes the code to find the supported PIDs for the vehicle. To select PIDs to view, the user navigates to the PID List tab, and then checks the desired PIDs, push refresh to push the choices to the readPIDdata tab, it updates automatically after the complete checked list is read.

Appendix D: OBD Visual Basic Code

In MainForm Class

```
Starts a thread which runs the single line PID once.
Public Sub sLineThreadLoop()

Starts a thread which runs a loop on up to three PIDs.
Public Sub cPIDThreadLoop()

Starts a thread to generate list of standard PID list and loops through checked PIDs to
display
Public Sub sPIDThreadLoop()

Initializes arrays PIDCode (PID ID for Mode 01), PIDDesc(Description for corresponding
PID), PIDData to 0.
Function initPIDLib()

Takes the string and does math on it to convert the PID hex string into a human
understandable data piece such as Vehicle Speed – mph. Handles Mode 1 PIDs from 0x00 to
0x64.
Private Sub ObdResponse(ByVal result As String) Handles obd.obdResponse

Sets the PID list in Tab2.
Private Sub getPIDList()

Determines if character is a hex character 0 to F.
Private Function isHex(ByVal myChar As Char) As Boolean

Checked the response from a PID request to 0x00, 0x20, 0x40 by determining the active
bits and masking it to the array.
Function getCharValidBits(ByVal setPID As String, ByVal myOffset As Integer)

Updates the data in the ReadPIDTab
Function updatePIDData() Handles PIDRefresh.Click
Sets up header that displays on Figure9 above the single PID entry
Public Sub setHeader(ByVal headerStr As String)


Starts Standard PID list Thread
Private Sub ReadPID_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles ReadPID.Click

Aborts Standard PID list Thread
Private Sub CancelRead_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles CancelRead.Click

Starts TriLine PID list Thread
Private Sub Tri_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
StartTri.Click
```

```
Aborts TriLine PID list Thread
Private Sub CTri_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles CancelTri.Click

Starts Single Line PID Thread
Private Sub Single_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles SLineStart.Click

Aborts Single Line PID Thread
Private Sub CSingle_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles SLineCancel.Click
```

In OBD Class

```
This Function turns off FlowControl by sending the command AT CFC0
Public Function FlowControlOff() As Boolean
```