MULTIPROTOCOL OBD TO UART INTERPRETER

OBD to UART Interpreter User Manual

Purpose of the Document

The purpose of this document is to explain the OBD to UART interpreter. This document contains the features of the multiprotocol OBD to UART interpreter.

Document History

Version	Author	Date	Description
A	5G HUB	10.12.2020	Initial Document
B	5G HUB	03.25.2021	Add Tera Term

Table of Contents

1 Introduction

The hardware board is an OBD to UART interpreter hardware board that provides bi-directional, half-duplex communication with the vehicle's On-Board Diagnostic system (OBD-II). It supports all legislated OBD-II protocols.

A wealth of information can be obtained by tapping into the OBD bus, including the status of the malfunction indicator light (MIL), diagnostic trouble codes (DTCs), inspection and maintenance (I/M) information, freeze frames, VIN, hundreds of real-time parameters, and more.

The hardware board features the STN2100. It is fully compatible with the *de facto* industry standard ELM327 command set. Based on a 16-bit processor core, the STN2100 offers more features and better performance than any other ELM327 compatible IC.

> On-Board Diagnostics, or OBD, is a computer-based system built into all 1996 and later light-duty vehicles and trucks, as required by the Clean Air Act Amendments of 1990. OBD systems are designed to monitor the performance of some of an engine's major components including those responsible for controlling emissions.

OBD is the language of the **Engine Control Unit (ECU)**, and it was designed to help fight emissions and engine failures. You can determine what the **Malfuction Indicator Light (MIL)** (aka the Check Engine Light) on your dash is referring to when it tells you there is an engine problem. If you or your mechanic has ever read the **DTCs (Diagnostic Trouble Codes)** on your vehicle, they are using OBD-II.
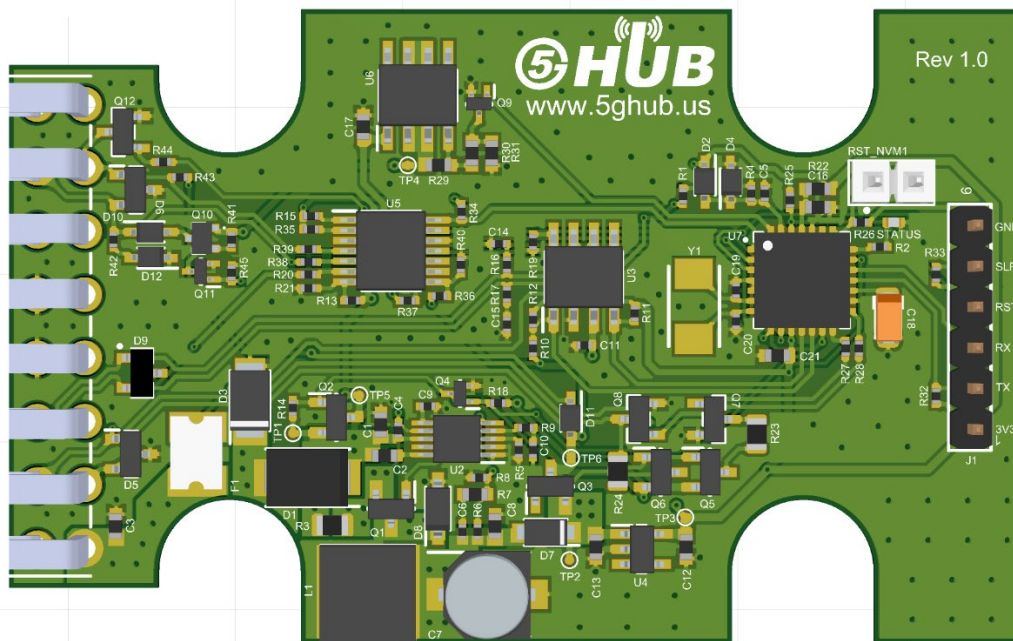
2 Feature Highlights

- Stable, field-tested firmware
- Fully **compatible with the ELM327** AT command set
- Fully **backwards compatible with the STN1110 and STN1170** command set
- Extended ST command set
- **UART interface** (baud rate is 9600 bps)
- Secure **bootloader** for easy firmware updates
- Support for **all legislated OBD-II protocols:**
    - SAE J1850 VPW (GM vehicles)
    - SAE J1850 PWM (Ford vehicles)
    - ISO 9141-2 (Asian, European, Chrysler vehicles)
    - ISO 14230-4 (Keyword Protocol 2000)
    - ISO 15765-4 (CAN)
- Support for **non-legislated OBD protocols:**
    - ISO 15765
    - ISO 11898 (raw CAN)
- Support for the heavy-duty **SAE J1939 OBD protocol**
- Superior **automatic protocol detection** algorithm
- Large message buffer
- Sophisticated **PowerSave Sleep/Wakeup Triggers**
- **RoHS** compliant

3   Typical Applications

- Vehicle telematics
- Fleet management and tracking applications
- Usage-based insurance (UBI)
- OBD data loggers
- Automotive diagnostic scan tools and code readers
- Digital dashboards

4   Hardware Board and Case Diagram

Pin #	Feature	Description
1	3.3V	Output 3.3V from the board
2	UART_TX	UART transmit output. Open drain – requires a pull-up to Vdd or 5V. Pull-up value depends on UART baud rate and the trace length (higher resistor values can be used with lower baud rates and shorter traces); typical value is 1 kΩ (1.5 kΩ, if pulled up to 5V).
3	UART_RX	UART receive input. Compatible with 3.3V and 5V logic.
4	RESET	Device reset input. A logic low pulse (min 2 μs) on this pin will reset the device. Apply a continuous logic low to hold the device in reset.
5	SLEEP	External sleep control input. It puts the device into low-power sleep mode. It is active low.
6	GND	

5 Using USB to UART Cable

You can use the OBD with any device that has UART (Tx/Rx) interface such as an Arduino board or any other hardware board available.

You can also use and connect the OBD to a USB port in the computer through a USB-to-UART cable. You can use cable such as this one: [USB TO UART TTL (Wires) Serial Cable (PL2303HX) MCP00102W Programmer Arduino Compatible in Elecrow bazaar!](#)
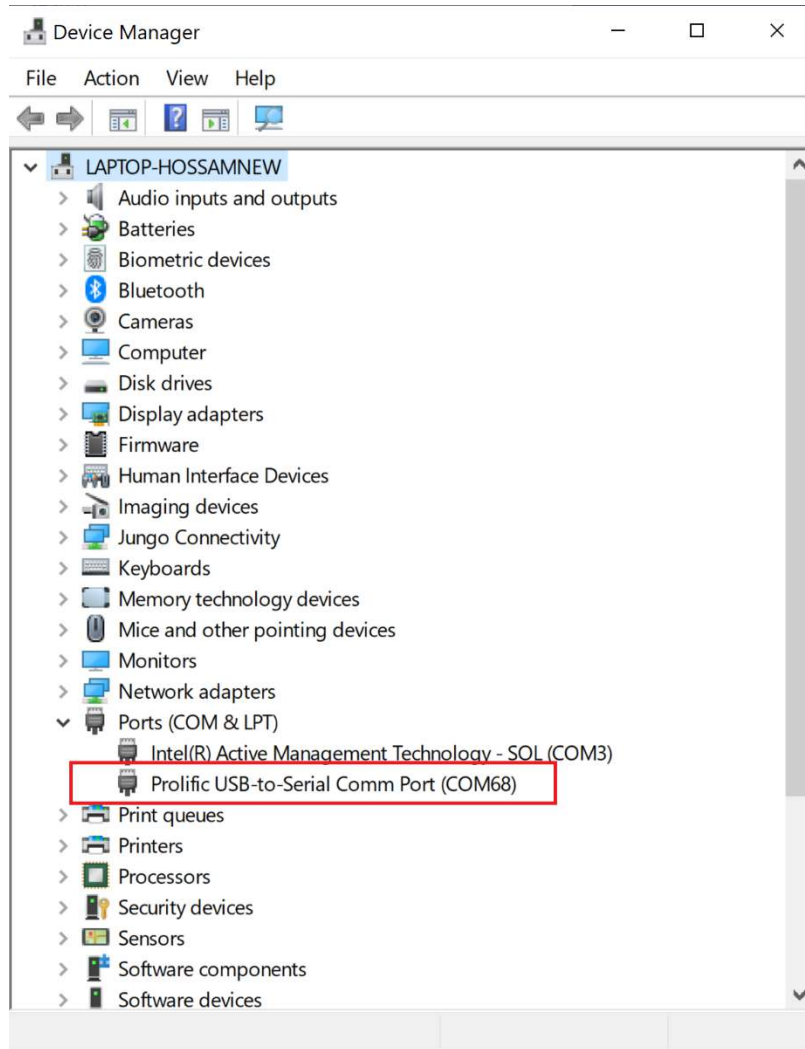
Such a cable has four wires colored as follow:

Red: +5V

White: Tx

Green: Rx

Black: GND

Connect the **White** cable to **UART_TX** (**PIN2**) and **Green** cable to **UART_RX** (**PIN3**). Also connect the **Black** cable to the **GND** (**PIN6**).

Make sure when you connect the USB-to-UART cable, it shows correctly in Windows device manager and all its driver is installed as in this screenshot.

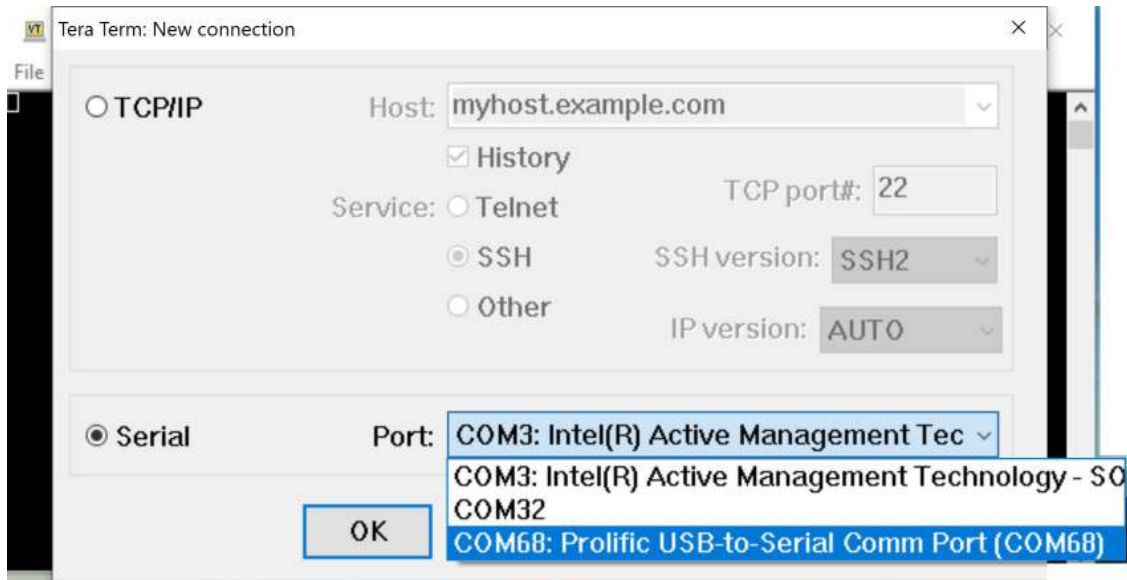# 6 Using Serial Terminal

It is recommended to use Tera Term tool as the serial terminal. You can download it from here:

https://osdn.net/projects/ttssh2/downloads/54081/teraterm-4.72.exe/

Launch Tera Terminal and select the **Serial** option and select USB-to-Serial port.

In Tera Term, choose **Setup->Serial port** and configure serial ports according to the following:
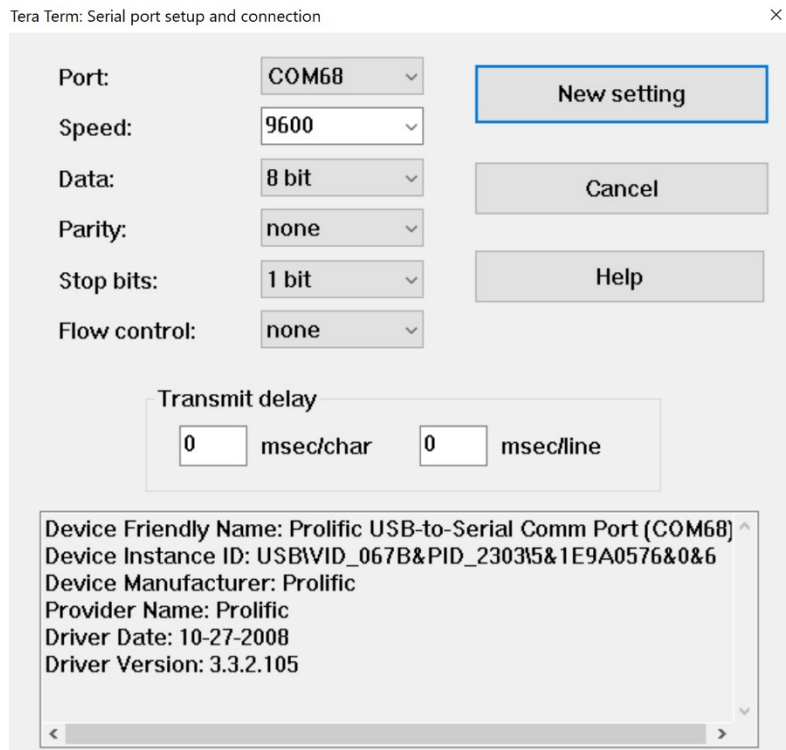
In Tera Term, you can select **Setup->Terminal** and configure the serial according to the following parameters:

7 OBD

Any vehicle manufacture from 1996 or later is required by law to have the OBD-II computer system. You can access this system through the **Data Link Connector (DLC)**. It is a 16 pin connector that can tell you which protocol your car communicates with, depending on which pins are populated in it.
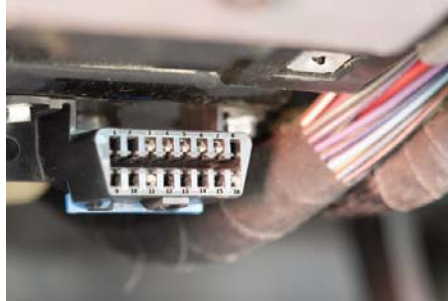
8 Terminology

Before we get too much farther, let us make sure we understand all the keywords used in these protocols.

8.1 Engine/Electronic Control Unit (ECU)

The ECU can refer to a single module or a collection of modules. These are the brains of the vehicle. They monitor and control many functions of the car. These can be standard from the manufacturer, reprogrammable, or have the capability of being daisy-chained for multiple features. Tuning features on the ECU can allow the user to make the engine function at various performance levels and various economy levels. On new cars, these are all typically microcontrollers. Some of the more common ECU types include:

- **Engine Control Module (ECM)** - This contsrols the actuators of the engine, affecting things like ignition timing, air to fuel ratios, and idle speeds.
- **Vehicle Control Module (VCM)** - Another module name that controls the engine and vehicle performance.
- **Transmission Control Module (TCM)** - This handles the transmission, including items like transmission fluid temperature, throttle position, and wheel speed.
- **Powertrain Control Module (PCM)** - Typically, a combination of an ECM and a TCM. This controls your powertrain.
- **Electronic Brake Control Module (EBCM)** - This controls and reads data from the anti-lock braking system (ABS).
- **Body Control Module (BCM)** - The module that controls vehicle body features, such as power windows, power seats, etc.

8.2   Diagnostic Trouble Code (DTC)

These codes are used to describe where an issue is occurring on the vehicle and are defined by SAE (you can find the whole spec here for a cost). These codes can either be generic or unique to the vehicle manufacturer. These codes take the following format:

XXXXX

- First unit identifies the type of error code:
    - **P**xxxx for powertrain
    - **B**xxxx for body
    - **C**xxxx for chassis
    - **U**xxxx for class 2 network
- Second digit shows whether the code is manufacturer unique or not:
    - x**0**xxx for government-required code
    - x**1**xxx for manufacturer-specific code
- Third digit shows us what system the trouble code references:
    - xx**1**xx/xx**2**xx show air and fuel measurements
    - xx**3**xx shows ignition system
    - xx**4**xx shows emissions systems
    - xx**5**xx references speed/idle control
    - xx**6**xx deals with computer systems
    - xx**7**xx/xx**8**xx involve the transmission
    - xx**9**xx notates input/output signals and controls
- Digits four and five show the specific failure code.
    - xxx**00** to xxx**99** - these are based on the systems defined in the third digit.

You can find some incomplete lists of DTCs here and here.

8.3   Parameter Identification (PID)

These are the actual meat and potatoes of the information you can pull off of an OBD-II system. The PIDs are the definitions of the different parameters you could be interested in checking out. These are similar to the third digit in the DTCs.

Not all PIDs are supported on all protocols, and there can be several unique, custom PIDs for each manufacturer. Unfortunately, these also are not generally published, so you may need to do a lot of hunting and/or reverse engineering to determine to which system each PID relates.

There are different modes available, and each mode has several options of PIDs available in that mode. For more general information on that, please check out the PID wiki page.

8.4   Malfunction Indicator Lamp (MIL)

The MIL is that terrible little light in the dash that indicates a problem with the car. There are a few variations, but they all indicate an error found by the OBD-II protocol.

Another possibility you might find on your dash includes this option:

No matter which one it is, these usually are not great lights to see.

9 OBD-II Protocols

There are five different communication protocols available under the OBD-II spec. Like so many things, manufacturers tend to have their preferences and think their protocol is best, hence the variation. Here is a quick overview of each and a description of the pins used on the DLC for each.
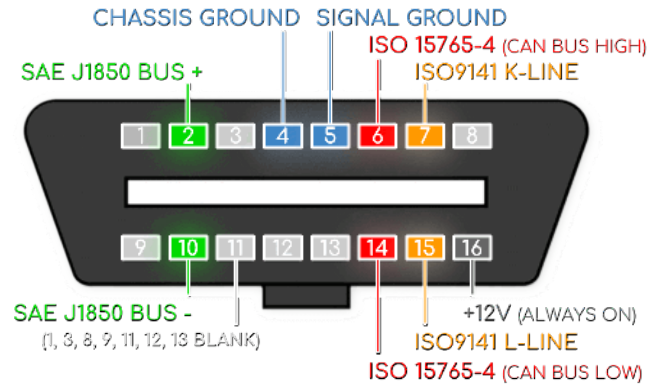

Figure 1: OBD-II Interface.

9.1 SAE J1850 PWM

This signal is Pulse Width Modulation, which runs at 41.6 kbps. This protocol is generally used on Ford vehicles.

Table 1: SAE J1850 PWM.

Pin #	Description
2	BUS+
10	BUS-
16	12V
4	GND
5	GND
Bus State	Active when BUS + is pulled HIGH, BUS – is pulled LOW
Maximum Signal Voltage	5V
Minimum Signal Voltage	0V
Number of Bytes	12
Bit Timing	'1' bit - 8uS, '0' bit - 16uS, Start of Frame - 48uS

9.2 SAE J1850 VPW

This protocol is Variable Pulse Width, which runs at 10.4 kbps. GM vehicles typically use this version.

Table 2: SAE J1850 VPW.

Pin #	Description
2	BUS+
16	12V
4	GND
5	GND
Bus State	Bus idles low
Maximum Signal Voltage	7V
Decision Signal Voltage	3.5V
Minimum Signal Voltage	0V
Number of Bytes	12
Bit Timing	'1' bit -HIGH 64uS, '0' bit -HIGH 128uS, Start of Frame - HIGH 200uS

9.3 ISO 9141-2

If you have a Chrysler, European, or Asian vehicle, this is your protocol. It runs at 10.4 kbps and is asynchronous serial communication.

Table 3: ISO 9141-2.

Pin #	Description
7	K Line (bidirectional)
15	L Line (unidirectional, optional)
16	12V
4	GND
5	GND
Bus State	K Line idles HIGH. Bus is active when driven LOW
Maximum Signal Voltage	12V
Minimum Signal Voltage	0V
Number of Bytes	Message: 260, Data: 255
Bit Timing	UART: 10400bps, 8-N-1

9.4 ISO 14230-4 (KWP2000)

This is the Keyword Protocol 2000, another asynchronous serial communication method that also runs at up to 10.4 kbps. This also is used on Chrsyler, European, or Asian vehicles.

Table 4: 14230 KWP2000.

Pin #	Description
7	K Line (bidirectional)
15	L Line (unidirectional, optional)
16	12V
4	GND
5	GND
Bus State	Active when driven LOW
Maximum Signal Voltage	12V

Minimum Signal Voltage	0V
Number of Bytes	Data: 255
Bit Timing	UART: 10400bps, 8-N-1

9.5   ISO 15765 CAN

This protocol has been mandated in all vehicles sold in the US from 2008 and later. However, if you have a European car from 2003 or later, the vehicle may have CAN. It's a two-wire communication method and can run at up to 1Mbps.

Table 5: ISO 15765 CAN.

Pin #	Description
6	CAN HIGH (CANH)
14	CAN LOW (CANL)
16	12V
4	GND
5	GND
Bus State	Active when CANH pulled HIGH, CANL pulled LOW. Idle when signals are floating.
CANH Signal Voltage	3.5v
CANL Signal Voltage	1.5V
Maximum Signal Voltage	CANH = +4.5V, CANL = +2.25V
Minimum Signal Voltage	CANH = +2.75V, CANL = +0.5V
Number of Bytes	L
Bit Timing	250kbit/sec or 500kbit/sec

10 Extended AT Command Set

ELM327 has extended AT commands set for following purposes:

- Reading built-in MEMS motion sensor data
- Reading orientation result from sensor fusion computation
- Data bus specific controls
- CAN bus sniffing

10.1 Motion Sensor Access

ATACL

- Function: reading accelerometer data
- Response format: X,Y,Z (in G)

ATGYRO

- Function: reading gyroscope data
- Response: X,Y,Z (in degree)

ATMAG

- Function: reading magnetometer data
- Response: X,Y,Z (in milli-Gauss)

ATTEMP

- Function: reading temperature data
- Response: temperature (raw) data

10.2 Quaternion Control and Orientation

ATQU0/ATQU1

- Function: disable/enable 9-DOF sensor fusion (disabled by default)
- Response: OK

ATORI

- Function: retrieving orientation parameters from 9-DOF sensor fusion
- Response: <yaw>,<pitch>,<roll> (in degree)

10.3 K-Line and ISO9141-2 Specific

ATSH <AA> <BB> <CC>

- Function: set header bytes
- Example: ATSH C1 33 F1

ATPTH <AA>

- Function: set initializing pulse time for (in hex, range of 0~255ms)
- Example: ATPTH 19 (set to 25ms)

ATPTA <BB>

- Function: set negative pulse duration before first data (in hex, range of 0~255ms)
- Example: ATPTA 32 (set to 50ms)

10.4 CAN Bus Specific

ATSH <ABC>

- Function: set CAN message header (on 11-bit CAN)
- Example: ATSH 7DF

ATSH <AA> <BB> <CC>

- Function: set lower 24 bits of CAN message header (on 29-bit CAN), higher 5 bits are set by ATCP command
- Example: ATSH DB 33 F1

ATCP <HH>

- Function: set CAN priority/higher 5 bits of header (on 29-bit CAN)
- Example: ATSH 18

ATCF <ABC> or <AA> <BB> <CC>

- Function: set CAN message header filter for CAN sniffing, <ABC> for CAN 11-bit, <AA> <BB> <CC> for CAN 29-bit
- Example: ATCF 7E8

ATCM <AA> <BB> <CC> <DD>

- Function: set CAN message filtering bit mask (32-bit)
- Example: ATCM FFFFFFFE (ignore the lowest bit when comparing with header filter number)

ATM1

- Start sniffing mode

ATM0

- Stop sniffing mode

10.5 Examples

Typical sniffing on 11-bit 500kbps CAN bus

1. ATSP6
2. ATCF 700
3. ATCM FFFFFF00
4. ATM1

Typical sniffing on 29-bit 500kbps CAN bus

1. ATSP7
2. ATCF 18DBF133
3. ATCM FF000000
4. ATM1

11 Using a Simulator

While these protocols are great for collecting data from your vehicle, it can be a real pain when prototyping to have to sit with a computer, various electronics, and cables running all over the place in the front of your car. Luckily, there are many simulators out there that allow basic prototyping and testing of OBD-II systems.

We have a few different simulators laying around here that are useful for working with these protocols. The simulator used here is ECUSim 2000 and can be found at:

https://www.obdsol.com/solutions/development-tools/obd-simulators/ecusim-2000/

and its full command manual is available here:

https://www.scantool.net/static/documentation/ecusim/ecusim-pm.pdf

Figure 2: ECUsim 2000 simulator.

To get started using this simulator, you must make the following connections:

1. Install Tera Term as described in a previous Section.
2. Plug a USB-to-Serial cable to the simulator and the computer. Install the necessary drivers.
3. Plug in the OBD-II cable to the simulator.
4. Power your simulator off of the supplied 12V power supply.
5. Open up a Tera Term tool at **9600 bps, 8, N,1** connecting to the serial port of the USB-to-serial cable.
6. Configure the simulator to the protocol you desire to test.
7. Connect to your ECU device (OBD-II board, CAN-Bus Shield, Raspberry Pi, etc.)

Now, you can leverage the power of the simulator by verifying that the data being transmitted over the bus is what your ECU reader is receiving and vice-verse.

For example: In the following screen-shot you request data from the OBD which s specified by **Mode** and **PID**:

Mode can be:

0x01: show current data

0x02: show freeze frame data
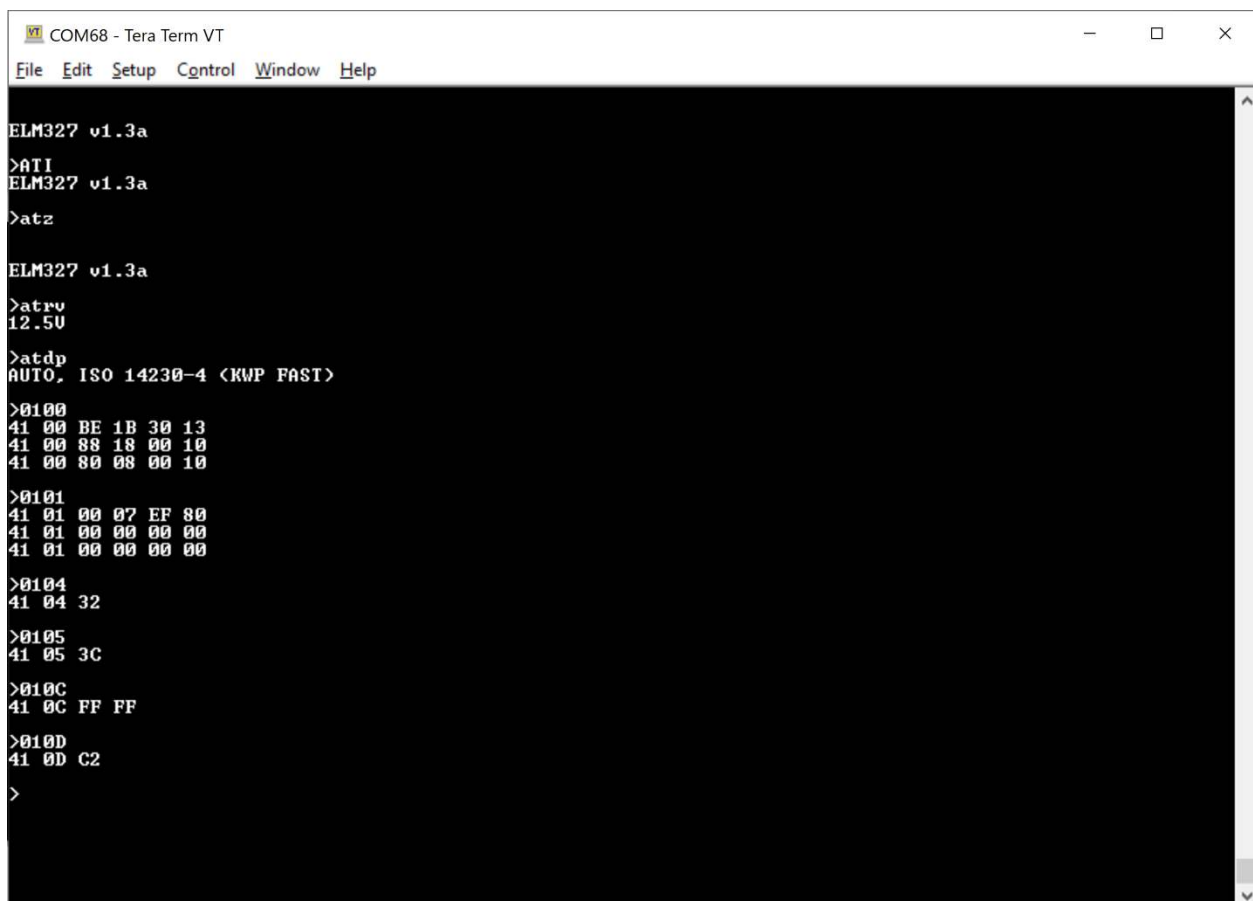
0x03: show stored diagnostic trouble code.

And commonly used **PIDs** are

0x04: engine load

0x05: engine coolant temperature

0x0C: engine rpm

0x0D: vehicle speed