SparkFun Electronics

Car Diagnostics Kit Landing Page

by zagGrad | June 22, 2011 | *5 comments* Skill Level: ★★★ Advanced

Overview

The car diagnostic kit is designed to allow you to interface with your vehicle's On Board Diagnostic computer (OBD). The kit includes an OBD to Serial cable, which plugs in to the vehicles OBD port and connects to the OBD-II-UART board. Once connected to the vehicle, diagnostic information can be retrieved from the OBD-II-UART by interfacing with the board’s serial port.

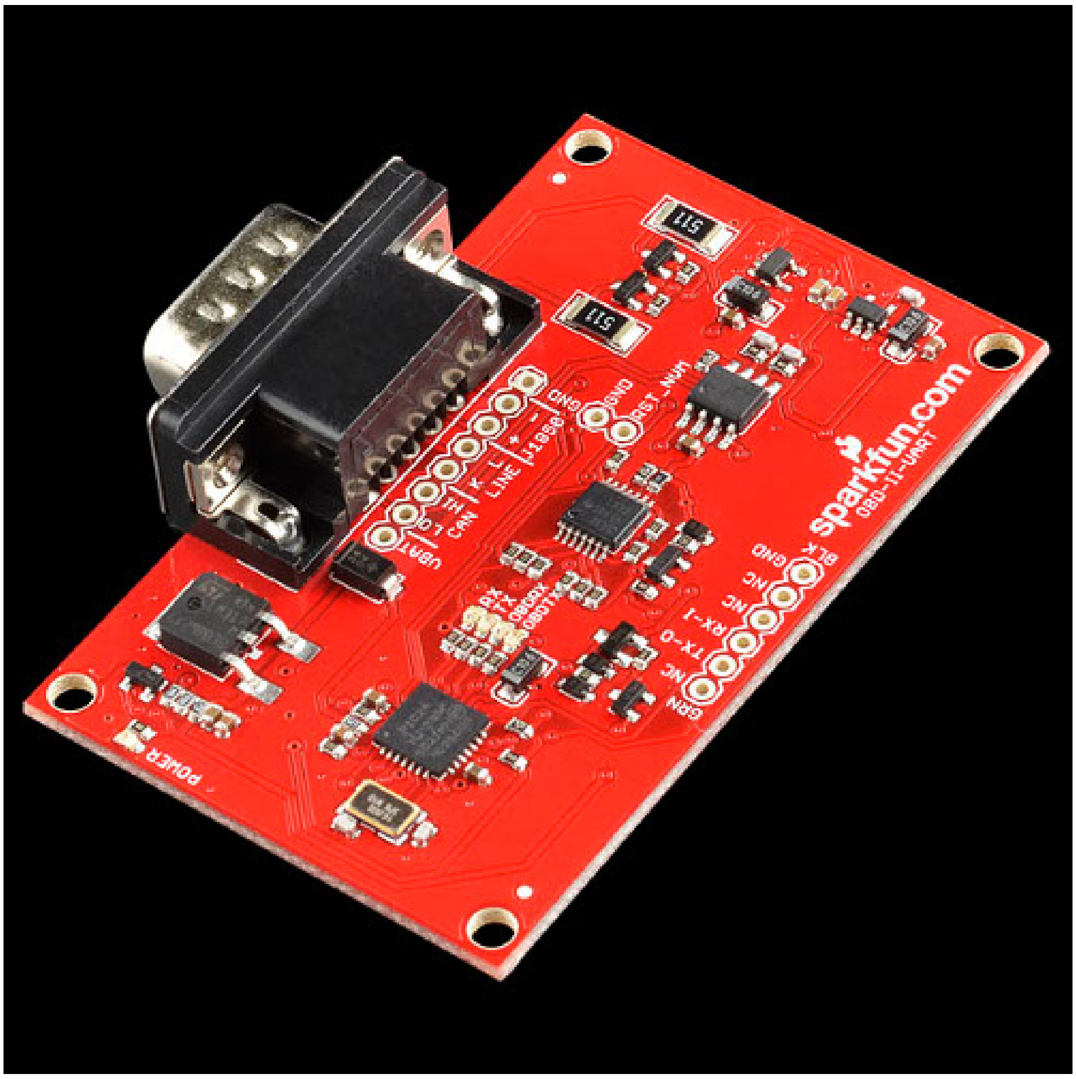
This board allows you to interface with your car's OBD-II bus. It provides you a serial interface using the ELM327 command set and supports all major OBD-II standards such as CAN and JBUS. The board also provides a footprint which mates directly to our FTDI Basic or a Bluetooth Mate. The DB9 connector mates with our DB9 to OBD-II cable listed below.

On-Board Diagnostics, Second Generation (OBD-II) is a set of standards for implementing a computer based system to control emissions from vehicles. It was first introduced in the United States in 1994, and became a requirement on all 1996 and newer US vehicles. Other countries, including Canada, parts of the European Union, Japan, Australia, and Brazil adopted similar legislation. A large portion of the modern vehicle fleet supports OBD-II or one of its regional flavors.

Among other things, OBD-II requires that each compliant vehicle be equipped with a standard diagnostic connector (DLC) and describes a standard way of communicating with the vehicle’s computer, also known as the ECU (Electronic Control Unit). 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.

STN1110 is an OBD to UART interpreter that can be used to convert messages between any of the OBD-II protocols currently in use, and UART. It is fully compatible with the de facto industry standard ELM327 command set. Based on a 16-bit processor core, the STN1110 offers more features and better performance than any other ELM327 compatible IC.

This quick-start guide will lead you through a couple example projects. In the first example we’ll discover how to use a computer to retrieve some basic information from the OBD-II-UART board; in the final example we’ll display real time engine data on an LCD using an arduino.



Software Required

- TeraTerm (or any other terminal program)
- Arduino 0022 (or later)

Hardware Required

- OBD-II-UART (Included in Car Diagnostic Kit)
- OBD-II-Serial Cable (Included in Car Diagnostic Kit)
- Male Headers
- FTDI Basic (Example One)
- Laptop (Example One)
- Mini USB Cable (Example One)
- Arduino Uno or similar (Example Two)
- Serial LCD (Example Two)
- M/F Jumper Wires

Firmware Required

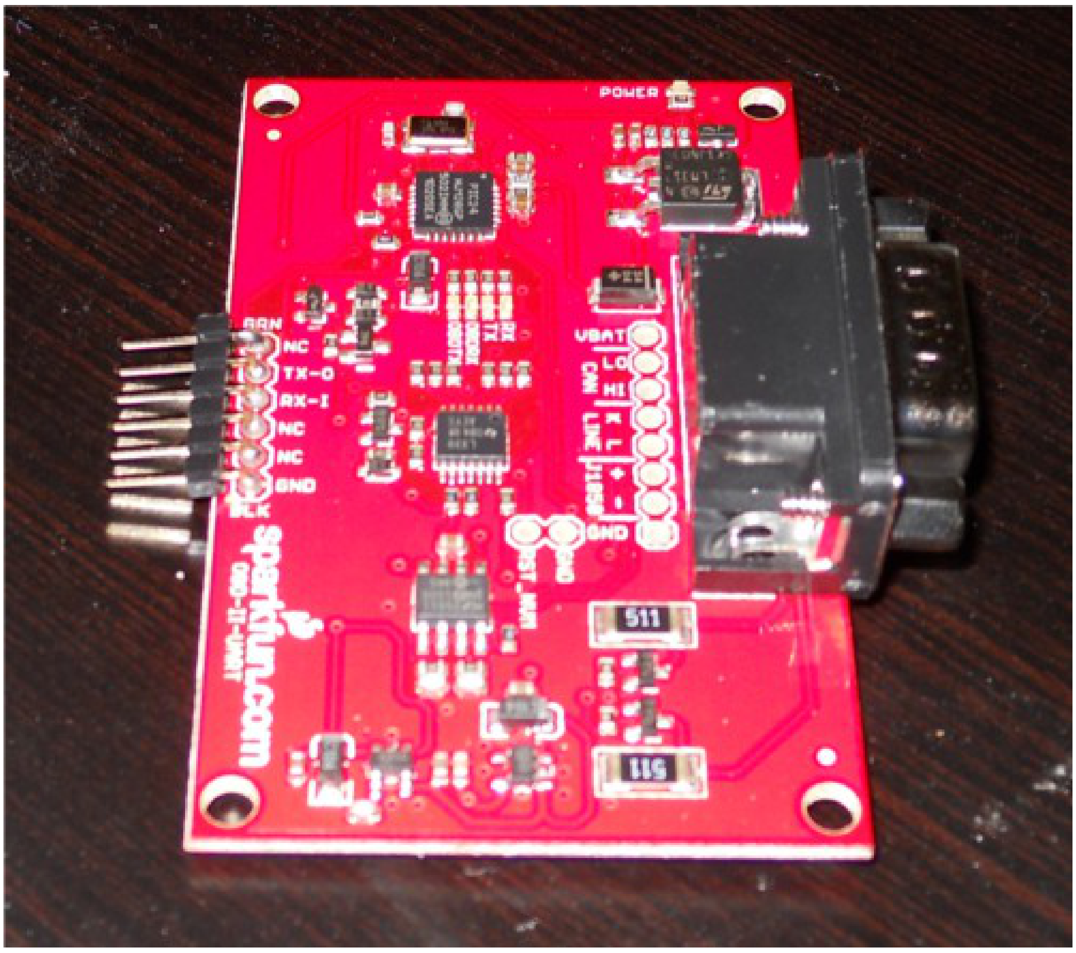
- OBD II Uart Example Sketch
- NewSoftSerial Library (Unzip this in your Arduino Libraries folder)

Relevant Documentation

- ELM327 Datasheet
- ELM327 Command Set Overview
- Wikipedia Article on OBD
- Visit the main product page for much more information regarding the OBD-II-UART module

Assembly Guide

Putting the OBD-II-UART board together is very simple. In order to interface with the OBD-II-UART, we'll need to interact with the pins on the board. All of the pins that we need are accessible from the header holes located along the bottom of the OBD-II-UART board. I'd recommend soldering some male headers to the header pins. Though we don't use all of the pins that are exposed, it's best to just solder headers into the whole row of holes. Here's a picture of how my board turned out after soldering the pins on.

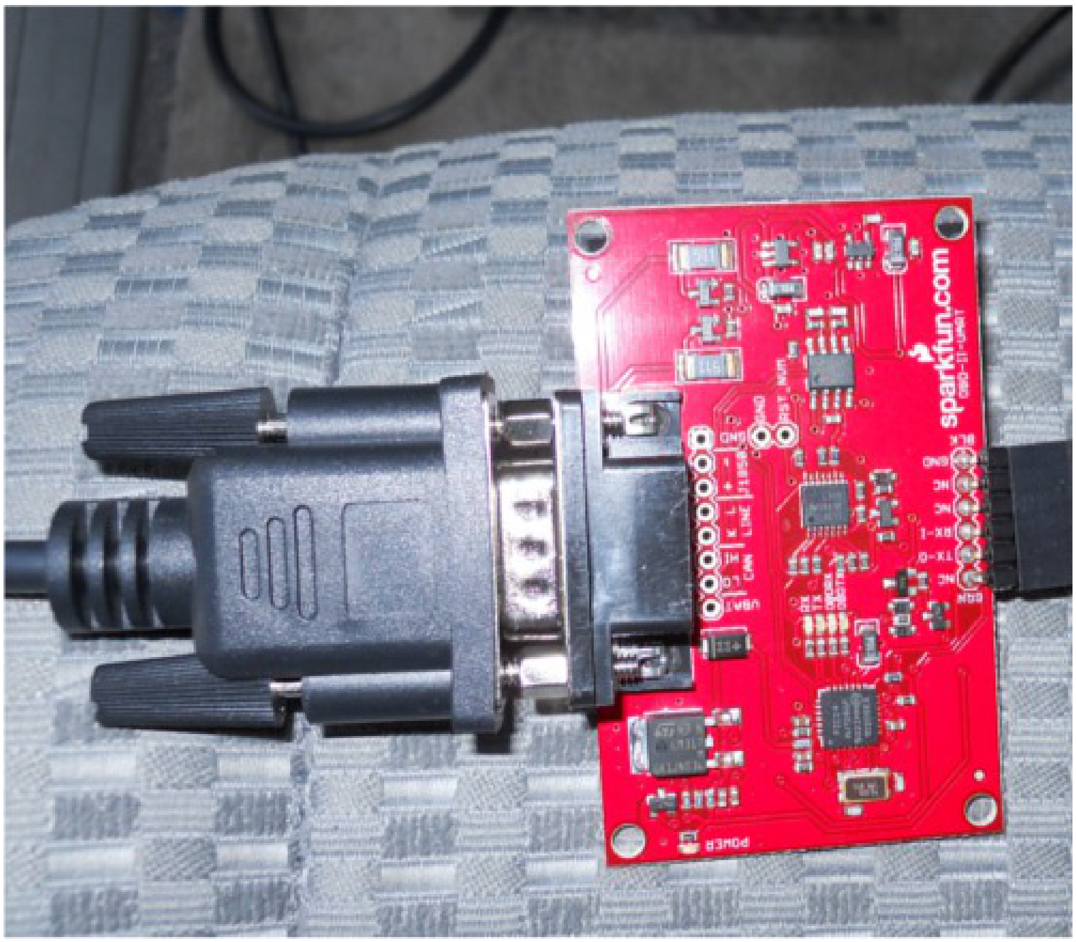


You could also solder female headers into the holes, or even solder wires directly to them if you want. I used male headers because we're going to be interfacing the OBD-II-UART board with an FTDI basic, and these come with a female connector that connects directly to the OBD-II-UART if it has male header pins installed.

Connecting the OBD-II-UART to a Vehicle

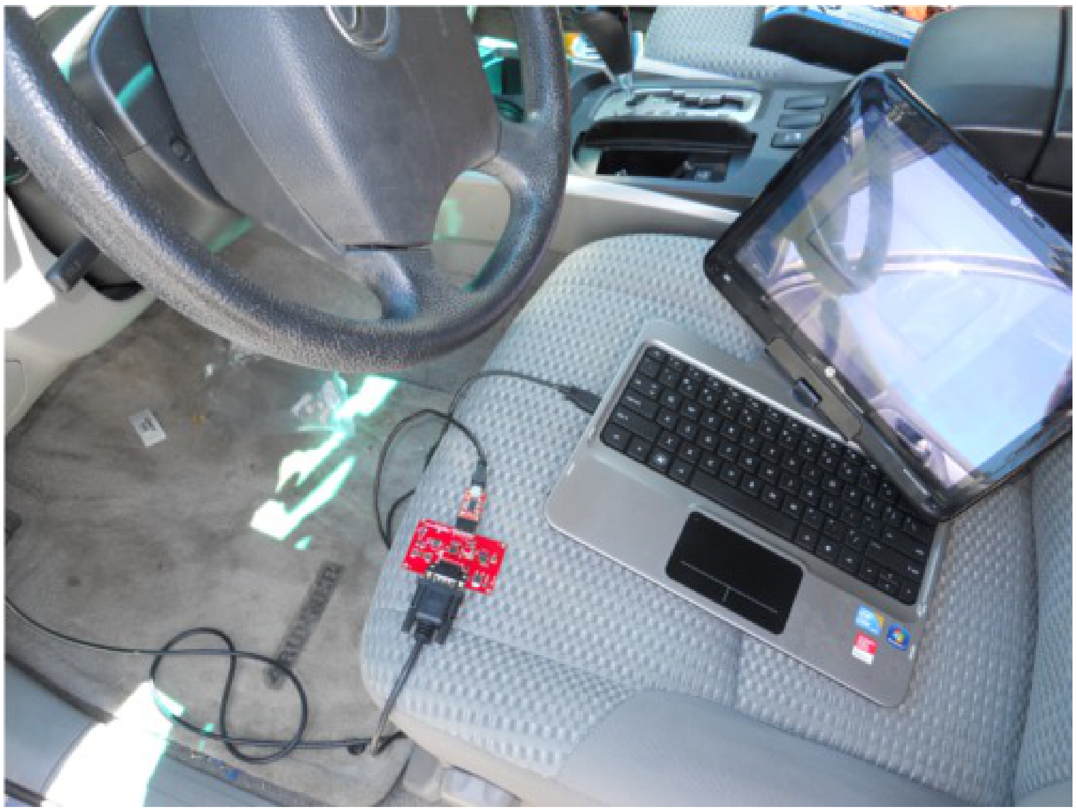
In order to start retrieving data from the vehicle we must connect the OBD-II-UART to the vehicles OBD connector using the interface cable. The OBD connector can be tricky to find; different vehicle manufacturers put the connector in different places. If you can't find the connector in your vehicle, consult the vehicle's owner's manual. You can also check out this link from nology.com for a quick reference on the most common places to find the OBD connector.

Once you find the OBDII connector in your vehicle you can connect the mating end of the cable. The OBD-II-UART board is powered from the vehicle. I recommend leaving the vehicle off while plugging the cable into the OBD connector. It would probably be fine to connect the board with the engine running, I just feel a bit more comfortable if the power is off while I'm trying to get things connected; especially since the area in which you'll be working can be a bit cramped. The connector has a very tight fit; you'll likely have to give it a decent amount of force to get the connector to mate snugly with the cable. Once you've connected the cable to the OBD connector on the vehicle go ahead and connect the other end of the cable to the OBD-II-UART board.



Example One: Reading from the OBD-II-Uart with a Laptop

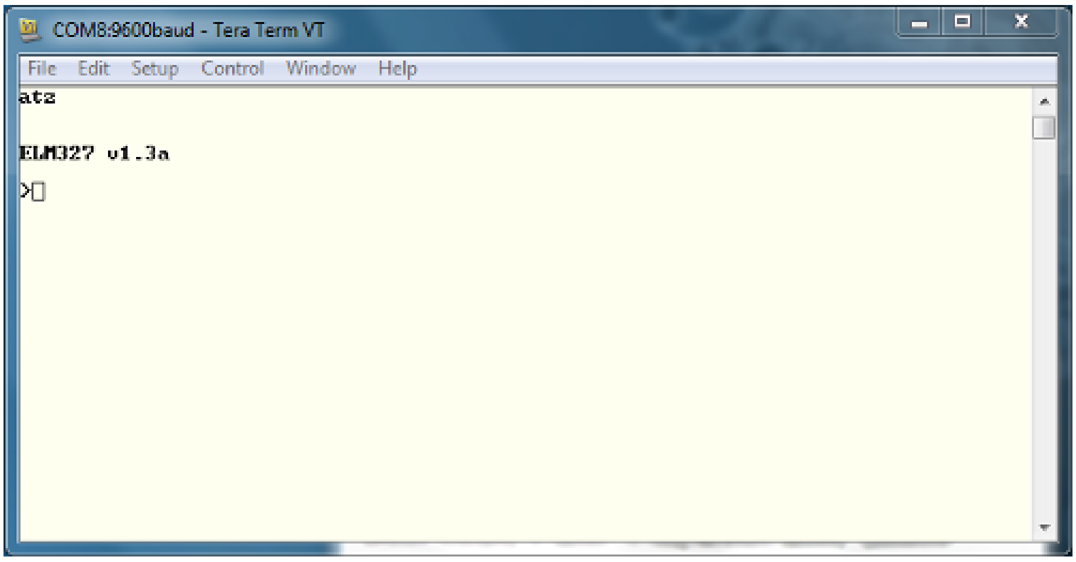
In the first example, we'll connect the OBD-II-UART to a vehicle and read some data from the device using a laptop. To complete this example you'll need a laptop, a miniUSB cable and an FTDI Basic (or just an FTDI cable). Assuming you've connected the OBD-II-UART board to the vehicle already, go ahead and connect the FTDI Basic to the OBD-II-UART header, and then use the USB cable to connect to the computer. Before we start sending commands to the module, let's briefly review what's going on with the OBD-II-UART board.



Commands and responses from the OBD-II-UART are transmitted using simple serial strings. This means that to send and receive data to and from the OBD-II-UART board using a computer all we need to do is use a terminal program (like hyper-terminal, teraterm, etc). There are essentially two different types of commands that can be sent to the OBD-II-UART: configuration commands and OBD commands. The configuration commands are used to configure the OBD-II-UART while the OBD commands are used to actually communicate with the vehicle. The ELM327 Datasheet does an excellent job outlining these commands, and I'd highly recommend that you read it (or at least the parts pertaining to your application).

We'll start by sending some AT commands to the OBD-II-UART board. AT commands always start with the letters 'AT', though they do not have to be capitalized because all of the commands received by the OBD-II-UART are case insensitive. After the 'AT' comes the letters indicating the command that should be executed. The ELM327 Command Set Overview covers all of the available AT commands.

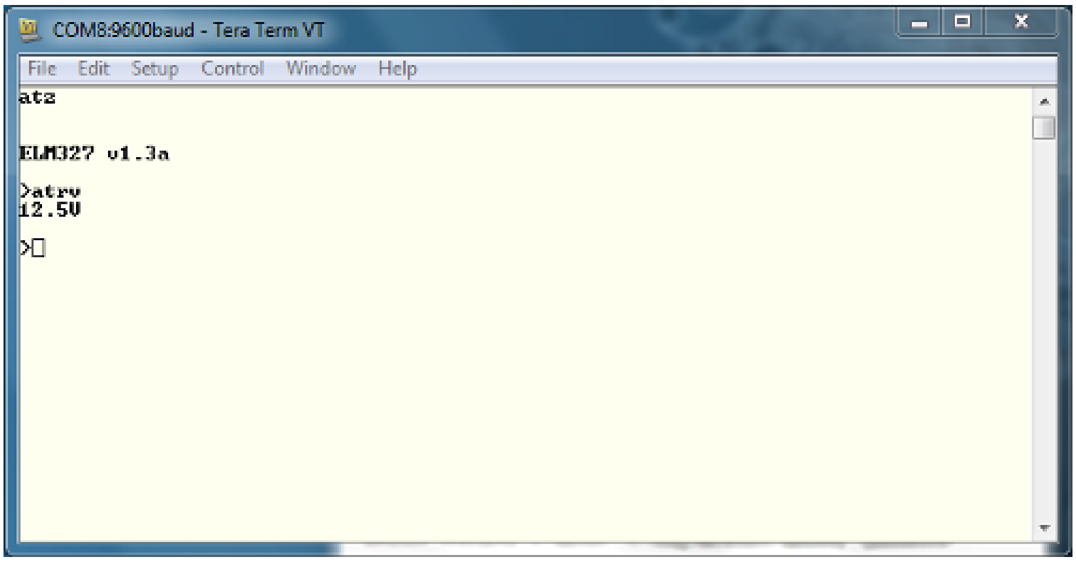
Once you've connected a USB cable and FTDI basic (or FTDI cable) to the OBD-II-UART, open a serial terminal on the laptop. Configure the settings for the terminal to use the serial port that you've connected the USB cable to, the baud settings should be configured for 9600 bps, 8 data bits, 1 stop bit, no parity. Make sure that the OBD-II-UART is plugged into the vehicle with the OBDII cable. Then, to reset the OBD-II-UART board, type the characters 'ATZ' into the terminal and press enter.


COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
atz

ELM327 v1.3a

>
```

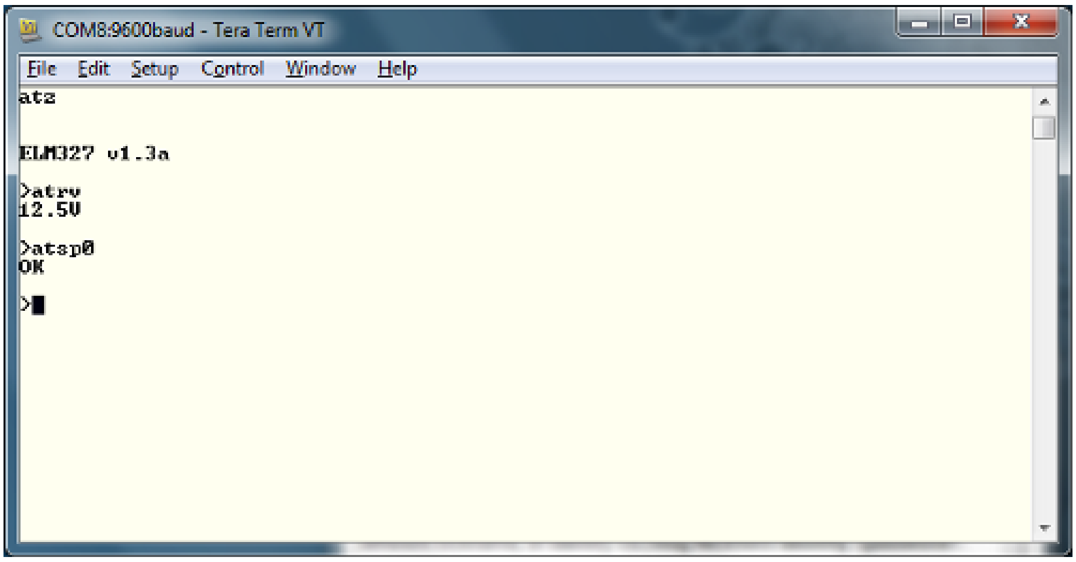
This command will cause the OBD-II-UART board to reset. On the board you'll see some LEDs flash, and then you'll receive the start-up prompt in the terminal window. If you see strange characters instead of the welcome string that's pictured above, you likely haven't configured the serial port properly. Double check all of the settings and try again.

Now that we've verified communication with the board, let's try reading the OBD-II-UART system voltage. The command to read the voltage is 'ATRV.' Enter those four characters into the terminal window and press enter and the OBD-II-UART will return the voltage.


COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
atz

ELM327 v1.3a

>atrv
12.5V

>
```

The voltage should represent the battery voltage of the vehicle. If it doesn't, well...check your battery. We're about ready to read some parameters from the OBD of the vehicle, but first we need to configure the OBD-II-UART to use the correct OBD protocol. This can be confusing because there are several different protocols that can be used for OBD; fortunately though, the OBD-II-UART has an auto-detect feature that can determine which protocol should be used. To take advantage of the auto-detect feature send the at command 'ATSP0' (that's a zero, not an Oh). Once the OBD-II-UART has detected the proper protocol, an 'OK' response will be issued and seen in the terminal window. This command will only work if the vehicle is On; if you receive an error message try again after turning the ignition to the ON position. It doesn't matter if the vehicle is running or not, the key's just need to be turned to the ON position.


COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

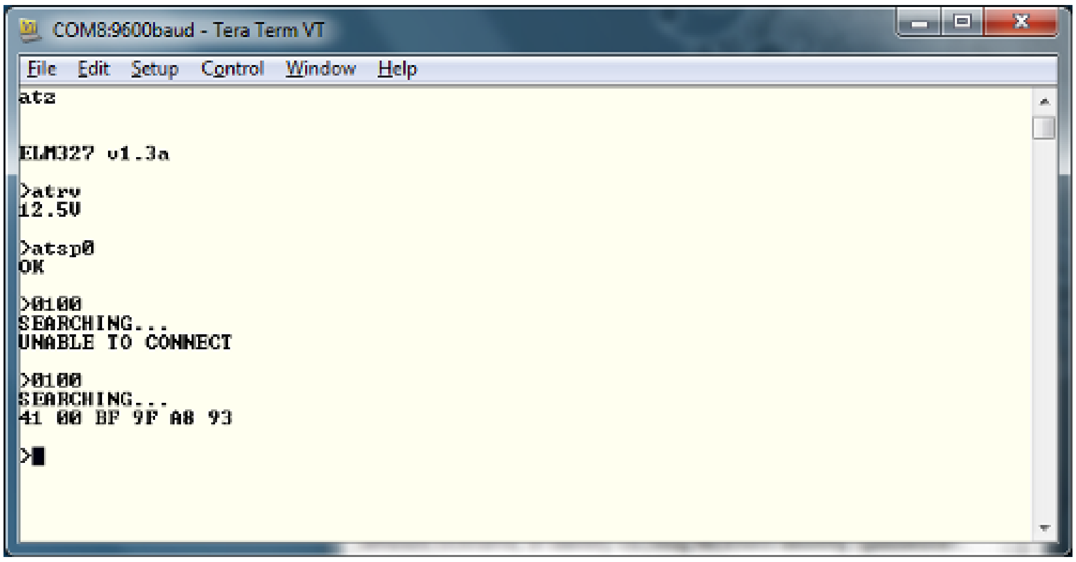
```
atz

ELM327 v1.3a

>atrv
12.5V

>atsp0
OK

>
```

Once the OBD-II-UART has been configured for the proper OBD protocol we can start sending OBD commands to the board. Once again I'd suggest you check out the ELM327 datasheet on the OBD protocol. The OBD commands consist of hexadecimal codes written in Ascii. Typically commands consist of 2 or more pairs of hexadecimal numbers, though there are some commands that only require 1 hex pair (remember, hexadecimal numbers in ascii format consist of the characters 0-9 and A-F). The first pair of hexadecimal numbers represents the OBD mode intended to be used, while the second and following pairs refer to the Parameter ID (PID) to be read from the mode. There are 10 modes in OBD, though not all modes are used in all vehicles (and not all parameters are used in the modes that are supported).

Mode Number	Mode Description
01	Current Data
02	Freeze Frame Data
03	Diagnostic Trouble Codes
04	Clear Trouble Code
05	Test Results/Oxygen Sensors
06	Test Results/Non-Continuous Testing
07	Show Pending Trouble Codes
08	Special Control Mode
09	Request Vehicle Information
0A	Request Permanent Trouble Codes

This Wikipedia page does a good job illustrating many of the Parameter IDs and what their functions are. There are also some proprietary OBD parameters that vehicle manufacturers use for their own purposes. Many PIDs are standard though, so let's explore what some of these do.

One special PID is 00, this parameters ID works on any vehicle with OBD and tells us the other PIDs that the vehicle supports. In the serial terminal, issue the command '0100'. This command roughly translates as 'OBD-II-UART, tell me what PIDs are supported in mode 01.'

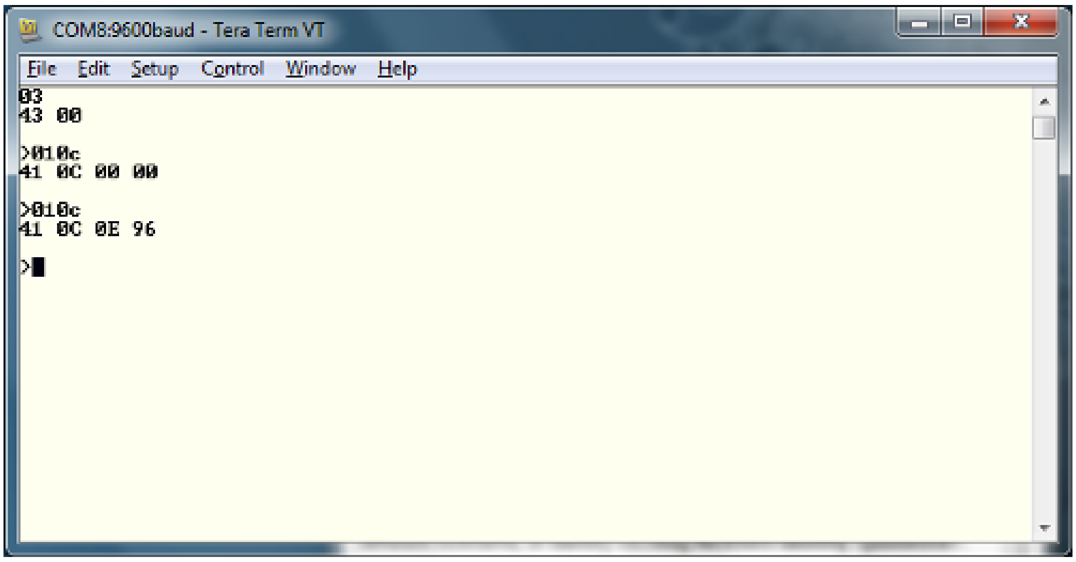

COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
atz


ELM327 v1.3a

>atrv
12.5V

>atsp0
OK

>0100
SEARCHING...
UNABLE TO CONNECT

>0100
SEARCHING...
41 00 BF 9F A8 93

>
```

There are a few things that all responses to OBD commands have in common. The first byte, the response byte, tells us which mode request is being responded to. Since our command used the mode 01, the response byte issues 0x41, or 0x40 + 0x01. After that the response lists the actual PID that was requested in the command, in our case 00. Following the first two bytes are the response to the command. The response to our first command tells us what PIDs are supported by the vehicle.

The response to our command has 4 pairs of hexadecimal numbers, which is 32 bits long. The highest bit corresponds to PID 01(0x01) while the lowest bit corresponds to PID 32(0x20). If the bit is a '1', the PID is supported; but if the bit is a '0' the PID is not supported.

Before we head over to the next example let's issue one more OBD parameter. The 'Read Engine RPM' parameter is another that is commonly supported. The PID for this parameter is 0C. To find the current engine RPM, issue the command '010C' and press enter.


COM8:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
03
43 00

>010c
41 0C 00 00

>010c
41 0C 0E 96

>
```

Following the first two response bytes we see the Engine RPM value. This value is listed in hex, so to get anything useful we must convert the number to decimal. Still though, the decimal value of the RPM is 3734. I was just idling when I issued this command, so obviously this is way too high. It turns out the RPM is listed in quarters of RPM, to get the acual RPM we must divide our result by 4. Now the RPM for our response is

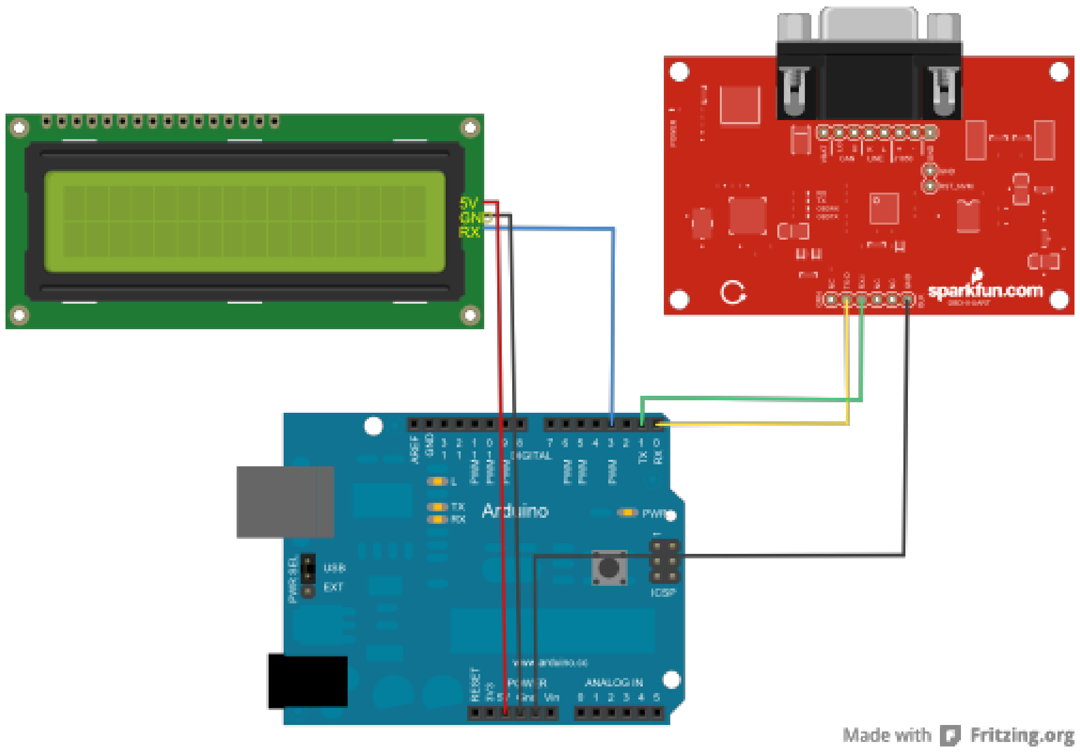
933, much more reasonable!

To find out more about how to read PIDs using the OBD-II-UART, read the ELM327 Datasheet (especially the 'OBD Commands' and 'Talking to the Vehicle' sections) and refer to this Wikipedia page. The Wikipedia page does a good job of listing the PIDs and how the PID responses are formatted.

Example Two: Printing OBD Parameters on a Serial LCD using an Arduino

In this example, we'll read parameters from the OBD-II-UART using an Arduino and print them on an LCD. To complete this example you'll need an Arduino Uno (or other 5V Arduino system), some M/F jumper wires and a serial LCD. If you didn't solder on male headers to the OBD-II-UART for the last example you can get by without the M/F jumper wires and just use wires to make the connections. You'll also need a way to power the Arduino while it's in your vehicle since power can't be taken from the OBD-II-UART board. I had my laptop in my car while I was debugging code, so I just powered the Arduino over USB. You could also use a 9V battery and 9V battery clip.

Before we get started we need to connect the Arduino to the OBD-II-UART and to the serial lcd. Since both of these are serial devices it's pretty simple, we'll only need to create 6 connections. The image below, along with the table, will show you how to connect all the wires to get the system ready for reading data from the OBD bus in your vehicle and displaying the information on the LCD screen. You will also need to connect the OBDII cable from the vehicle to the OBD-II-UART board, but that should be obvious by now.



Arduino Pin	Serial LCD Pin	OBD-II-UART Pin
GND	GND	GND
5V	5V	None
D3	Rx	None
D0(Rx)	None	Tx-O

D1(Tx)	None	Rx-I

Now that the hardware has been connected let's look at the sketch. If you haven't downloaded the obdIIUartQuickstart sketch, download it and open it up in Arduino. You'll also need to have the NewSoftSerial library installed in the libraries folder. If you're too excited to see what happens to read the description of the sketch go ahead and upload the code to the Arduino. Make sure the OBD-II-UART is plugged into the vehicle and the Arduino is powered, then turn on the vehicle. You'll see the speed and RPM displayed on the LCD! Do not, I repeat, do not debug/stare at the LCD while you're driving! There are safer ways to do things...

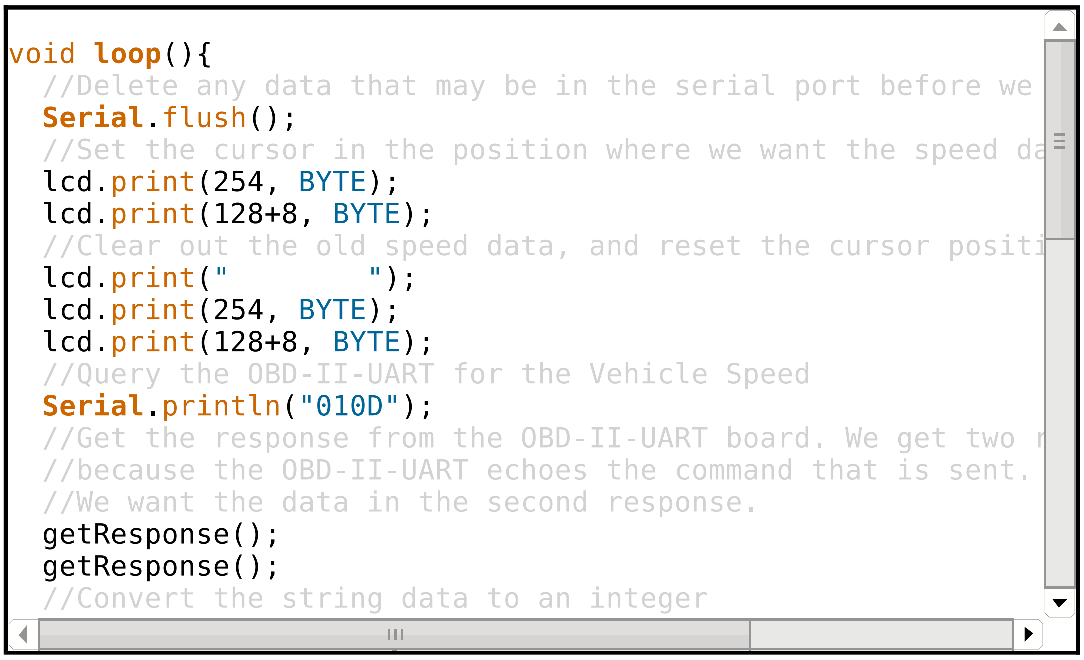
The sketch is pretty simple actually, there are only two things we're working with: communicating with the OBD-II-UART and sending data to the serial LCD. Here's the initialization and setup section of the sketch:

```
#include <NewSoftSerial.h>

//Create an instance of the new soft serial library to contro
//Note, digital pin 3 of the Arduino should be connected to R
NewSoftSerial lcd(2,3);

//This is a character buffer that will store the data from th
char rxData[20];
char rxIndex=0;

//Variables to hold the speed and RPM data.
int vehicleSpeed=0;
int vehicleRPM=0;

void setup(){
  //Both the Serial LCD and the OBD-II-UART use 9600 bps.
  lcd.begin(9600);
  Serial.begin(9600);
```

In the initialization section at the top you can see that there are only a couple variables. We created a NewSoftSerial port to talk to the LCD. We also created a character array which will be used to collect the data coming in on the serial port, along with an index to keep track of our position in the array. Finally there are two variables that will hold the speed and rpm data retrieved from the vehicle.

The setup section is also pretty straightforward. The lcd serial port is initialized to 9600 bps to talk to the serial lcd, and the serial port (which is connected to the OBD-II-UART) is also configured for 9600 bps just like in the previous example. I decided I wanted the speed data to be displayed on the first row, and the RPM data on the second row; so I printed the data headers in the appropriate places. Keep in mind that the serial lcd has some special commands that allow you to clear the display and place the cursor wherever you'd like.

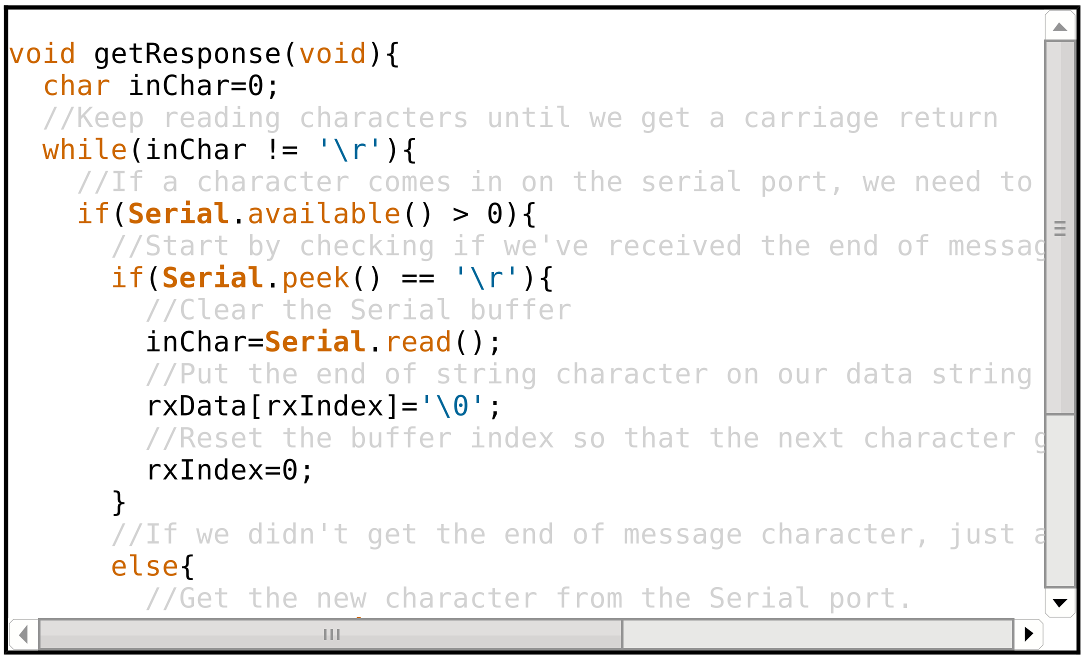
After configuring the LCD, the OBD-II-UART module is reset. Finally, we flush any miscellaneous data out of the serial port before moving onto the main loop. This is important because the OBD-II-UART will be spitting out some data while it's powered on and reset, and we don't want to copy this into our character buffer.



```
void loop(){
  //Delete any data that may be in the serial port before we
  Serial.flush();
  //Set the cursor in the position where we want the speed da
  lcd.print(254, BYTE);
  lcd.print(128+8, BYTE);
  //Clear out the old speed data, and reset the cursor positi
  lcd.print("        ");
  lcd.print(254, BYTE);
  lcd.print(128+8, BYTE);
  //Query the OBD-II-UART for the Vehicle Speed
  Serial.println("010D");
  //Get the response from the OBD-II-UART board. We get two r
  //because the OBD-II-UART echoes the command that is sent.
  //We want the data in the second response.
  getResponse();
  getResponse();
  //Convert the string data to an integer
```

The main loop of our sketch has a moderate amount of lines in it, but it's actually not doing much. Basically we set the cursor position where we want to display the speed data, then we send a request for the vehicle speed data from the OBD-II-UART to the Serial terminal. The command '010D' was found in the wikipedia article mentioned earlier, you'll notice that the format is identical to the way we requested information from the OBD-II-UART in the previous example. After the command is sent we wait for the appropriate response. When the response is received, we convert the data to an integer and display it on the LCD.

The second half of the loop is exactly the same as the first section, but we're retrieving the RPM data rather than the vehicle speed data. There's kind of a funky line that converts the RPM response to an integer. Keep in mind that the RPM response from the OBD-II-UART is 2 bytes long, and it's reported in quarter RPMs, not full RPMs. This line of the sketch gets the first byte of data and multiplies it by 256 (since it's the high byte), and then adds the low byte to this value. Now we've got the RPM value in a variable as an integer instead of a string, all we have to do to get the actual RPM is divide by 4.



```
void getResponse(void){
  char inChar=0;
  //Keep reading characters until we get a carriage return
  while(inChar != '\r'){
    //If a character comes in on the serial port, we need to
    if(Serial.available() > 0){
      //Start by checking if we've received the end of messag
      if(Serial.peek() == '\r'){
        //Clear the Serial buffer
        inChar=Serial.read();
        //Put the end of string character on our data string
        rxData[rxIndex]='\0';
        //Reset the buffer index so that the next character g
        rxIndex=0;
      }
      //If we didn't get the end of message character, just a
      else{
        //Get the new character from the Serial port.
```

The last part of the sketch is just a function that retrieves a line of characters from the serial port and saves the data to the rxBuffer. Basically it keeps copying all of the characters that come into the serial port into the rxBuffer array until it sees a carriage return come in. If a carriage return is detected the function puts a null terminator at the end of the buffer and resets the index to 0 to prepare for the next string.

Conclusion

That's all there is to it! Reading data from a vehicles OBD port is surprising simple if you're using the OBD-II-UART module. Read through the other linked documentation for more information on the OBD standards and see what you can make! If you have any problems, feel free to contact SparkFun Technical Support at techsupport@sparkfun.com