https://www.seeedstudio.com/blog/2019/11/27/introduction-to-can-bus-and-how-to-use-it-with-arduino/

**Introduction to CAN-BUS and How to use it with Arduino**

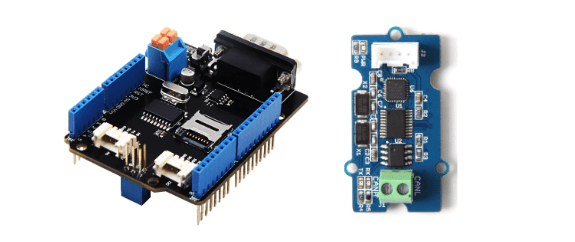
By [yida](https://www.seeedstudio.com/blog/author/yida/) 2 years ago



Ever heard of CAN-BUS but don’t exactly know what it does? Fret not! We will be discussing everything about CAN-BUS today. I hope that you’ll get a better understanding of it through this blog:

* Introduction to CAN BUS
* Why use CAN BUS protocol
* CAN BUS vs OBD2
* Getting started with CAN BUS with Arduino
* CAN BUS Projects

**Introduction to CAN BUS**

[CAN-BUS Shield V2](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html?utm_source=blog&utm_medium=blog), [Serial CAN-BUS Module,](https://www.seeedstudio.com/Serial-CAN-BUS-Module-based-on-MCP2551-and-MCP2515.html?utm_source=blog&utm_medium=blog)

**What is CAN BUS?**

CAN stands for *Controller Area Network*, it is used to **allow microcontrollers and devices to communicate with each other within a vehicle** without a host computer which allows for control and data acquisition. These devices are also called Electronic Control Units (ECU) and they enable communication between all parts of a vehicle.

Today, you can find up to 70 ECUs in a modern car. CAN is a serial communication bus designed for industrial and automotive applications. For example, they are found in vehicles, farming equipment, industrial environments, etc.

**How does CAN BUS works?**

The fuel level, door sensors, odometer, and many more parts of a car have to communicate with each other some how, and CAN BUS is what they used to do that. These CAN compatible components, which are called “nodes” are connected with a 3-string copper wires, with no central router to govern the flow of data. Every node can hear the messages of every other node.

Every node has an ID, where the ones with the higher priority ID can have the priority to “talk” first while the others “listen”. This is to ensure that there are never two nodes talking at the same time. The biggest benefit of CAN BUS is to be able to just connect components without having to worry about signal routing.

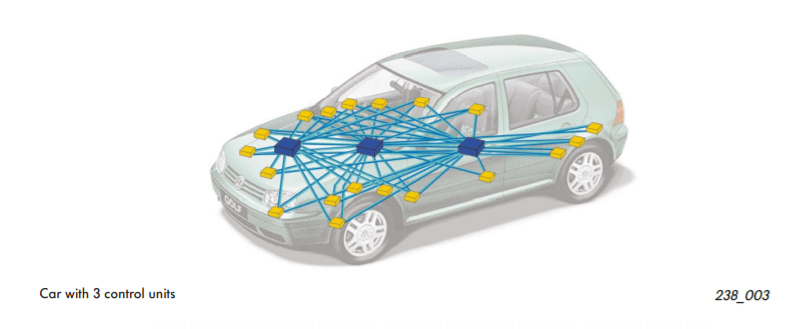
**So, why not use other communication protocols like UART, SPI and I2C?**

Compared to other communication protocols like UART, SPI and I2C, using CAN BUS communication protocol are *much more reliable* as they are standard automotive communication protocols that are used to transmit vital data like a throttle position in a vehicle. If miscommunication or loss of data occurs, it could lead to critical failures.

In a vehicle, safety and reliability are the two most valued features. CAN BUS are therefore the ideal protocols for vehicular usage.

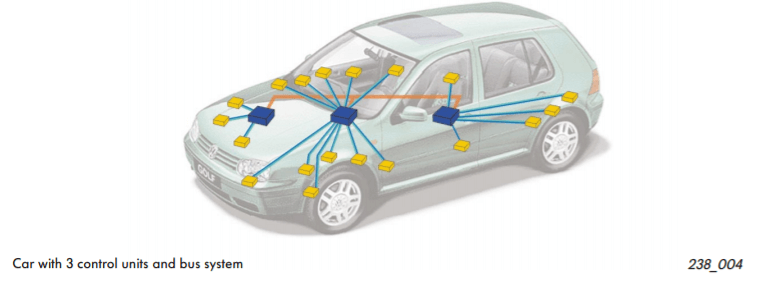
**The role of CAN**

**What will happen if there is no CAN in the car?**

Ref: [Volkspage](http://www.volkspage.net/technik/ssp/ssp/SSP_238.pdf)

Without CAN BUS protocol, electronic modules in vehicles will have to communicate with each other using direct, point-to-point analog signal lines. With each module requiring a direct line connected for communication, not only is it time-consuming, it will be messy with all the excessive amount of wiring as seen on the picture above. And there may be unreliable communication between devices. Excessive wires may require additional equipment, which also creates cost issues.

**What about a car with CAN?**

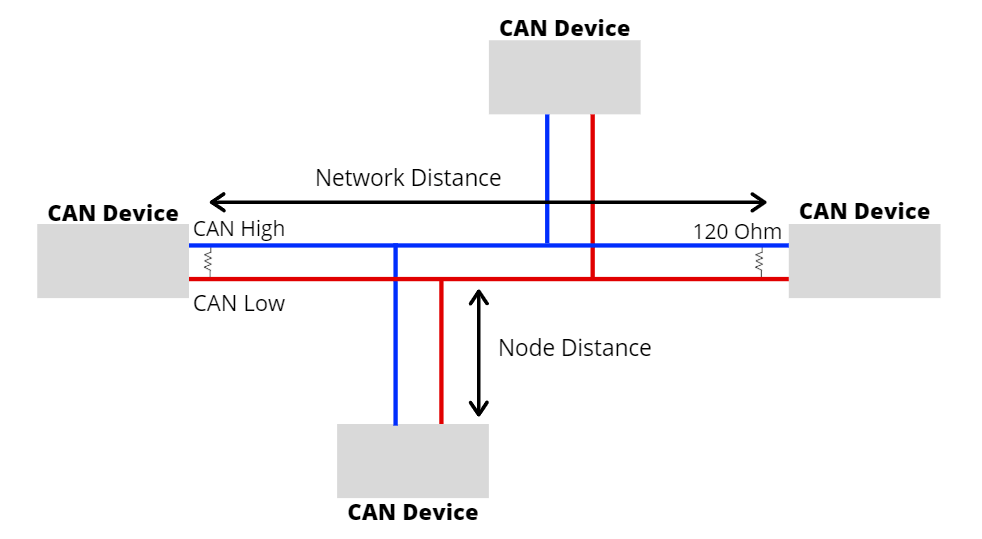
Ref: [Volkspage](http://www.volkspage.net/technik/ssp/ssp/SSP_238.pdf)

With CAN BUS protocol, it eliminates the need of all these wirings by enabling electronic devices to communicate with each other with a single multiplex wire that connects each node in the network to the main dashboard as seen on the picture above.

The multiplex wire and architecture enable signals to be combined and transmitted over the entire network with just a single wire while ensuring each electronic module in vehicles receives data from sensors and actuators. This allows the user to be able to connect any number of ECUs in your vehicle through the two-wire bus.

It also allows for several features to be added via just software. Furthermore, an ECU is able to use data from another ECU which eliminates the need to install the same sensors in multiple devices.

**CAN BUS Wiring Sequence**



* The CAN BUS protocol consists of two wires for bi-directional data transmission as shown above which are
  + CAN\_H (CAN High)
  + CAN\_L (CAN Low)
* The wires act as a differential line which means the CAN signal either 0 or 1, can be represented by the potential difference between the two wires
  + For example, if the difference is positive and large than a certain minimum voltage = 1. If the difference is negative = 0
* For CAN termination, as you can see from the picture above, a single 120 Ohm is generally used at the 2 ends of the CAN network.

**CAN BUS Protocol Speed and Range**

* Communication speeds of the CAN BUS protocol ranges from 10kpbs to 1Mbps.
  + The speed also depends on the length of wire used. The shorter it is, the faster the communication speed and the longer it is, the slower the communication speed.
  + For example, at 40 meters, the speed will be at 1Mbps. At 1000 meters, the speed can be at 50kpbs.
* The node distance is generally advised to be no more than 0.3 meters / 1 foot.

**CAN Message**

* To fully understand how the CAN BUS protocol works, let us look at the frames sent over the network.
* The CAN message contains many segments. The 2 main segments, identifier and data, will be the ones transmitting the data.
* The identifier is used to identify CAN devices in a CAN network while data will be the sensor or control data that have to be sent from one device to another.
* The identifier or CAN ID is either 11 or 29 bits in length depending on the type of CAN protocol used.
  + Standard CAN = 11 bit
  + Extended CAN = 29 CAN
* While the data can be anywhere from 0 to 8 bytes.

**Why use CAN BUS protocol?**

CAN BUS protocol is widely used thanks to the following advantages：

**Low Cost**

* ECUs communicate via a single CAN interface compared to, for example, direct analog signal lines that reduce errors, weight, and costs.
* With its multiplex wiring that combines analog and digital signals and their transmission over a shared medium, it reduces the amount of wiring needed.
* When adding or reducing some equipment, it can be easily operated, and there is no need to carry out large-scale transformation of the system, which saves a lot of manpower.

**Centralized**

* As CAN BUS supports centralized control over electronic devices that are connected to the network, it allows for central error diagnosis and configuration across all ECUs
* Error handling is also built into the CAN protocol where nodes can check for errors in transmission while maintaining its own error counter. For example, the protocol supports different error detection capabilities such as bit error, ack error, form error, CRC error, etc.

**Flexible**

* As each CAN connected ECU can receive all transmitted messages, it can also decide whether it is relevant and act accordingly.
* The CAN BUS protocol is also a message-based communication protocol where nodes on the bus have no identifying information.
* With the above features, nodes can easily be added or remove and modified.
* For beginners, it will be easy to integrate new electronic devices into the CAN BUS network without any significant programming overhead.

**Robust**

* When choosing a communication protocol, durability and reliability are very important. You would want your communication protocol to be self-sustaining and durable for a long period of time without maintenance.
* With the CAN BUS, the system is robust towards electric disturbances and electromagnetic interference which makes it ideal for vehicles.

**Efficient**

* CAN messages frames are prioritized by ID where the top priority will get bus access and yet frames would not be interrupted.
* Due to flash programming, it also saves time together with less and simple wiring.

**CAN BUS vs OBD2**

I am pretty sure you will come across other “higher-layer protocols” that are related to CAN BUS like OBD2 **BUT** they are not the same!

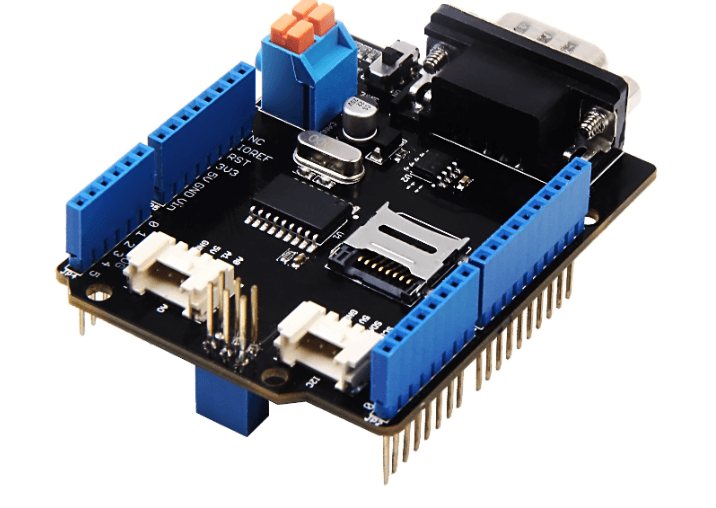
[OBD-II CAN Bus GPS Development Kit](https://www.seeedstudio.com/OBD-II-CAN-Bus-GPS-Development-Kit-p-2868.html?utm_source=blog&utm_medium=blog)

* The CAN standard does not specify how to handle messages larger than 8 bytes or how to decode the raw data. Therefore, a set of standardized protocols are developed to further specify how data is communicated between ECUs of a given network and OBD2 is one of them.
* OBD which stands for on-board diagnostics is your vehicle built-in self-diagnostic system. OBD2 can use one of (many) different bus systems to transfer diagnostic data from and to your car.
* One simple analogy I would use to differentiate between two of them will be OBD2 is like a language that we speak where we use CAN as a communication device like a phone to talk to someone which in this case a vehicle and its state of health.
* To read more about OBD2, you can check out the [Wiki Page on OBD.](https://en.wikipedia.org/wiki/On-board_diagnostics)

**Explore Seeed’s CAN BUS products!**

Here at Seeed, we offer a variety of CAN BUS products for your project needs! As shown below:

[**CAN-BUS Shield V2 ($24.50)**](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html?utm_source=blog&utm_medium=blog)

[](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html)

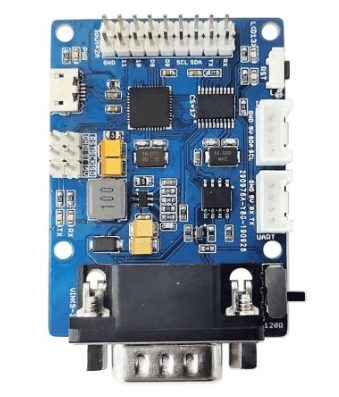
This CAN-BUS Shield adopts **MCP2515** CAN Bus controller with SPI interface and **MCP2551** CAN transceiver to give your Arduino/Seeeduino CAN-BUS capability. With an **OBD-II** converter cable added on and the OBD-II library imported, you are ready to build an onboard diagnostic device or data logger.

To make this CAN BUS shield convenient to interact with cars, it has an MCP2515 CAN-BUS controller with SPI interface and MCP2551 CAN transceiver to give you Arduino/Seeeduino CAN-BUS capability.

Features of this CAN BUS shield includes:

* Implements CAN V2.0B at up to 1 Mb/s
* Industrial standard 9 pin sub-D connector
* OBD-II and CAN standard pinout selectable.
* Changeable chip select pin
* Changeable CS pin for TF card slot
* Changeable INT pin
* Screw terminal that easily to connect CAN\_H and CAN\_L
* Arduino Uno pin headers
* 2 Grove connectors (I2C and UART)
* SPI Interface up to 10 MHz
* Standard (11 bit) and extended (29 bit) data and remote frames
* Two receive buffers with prioritized message storage

[**CANBed – Arduino CAN-BUS Development Kit (Atmega32U4 with MCP2515 and MCP2551) ($14.90)**](https://www.seeedstudio.com/CANBed-Arduino-CAN-BUS-Development-Kit-Atmega32U4-with-MCP2515-and-MCP2551-p-4365.html?utm_source=blog&utm_medium=blog)

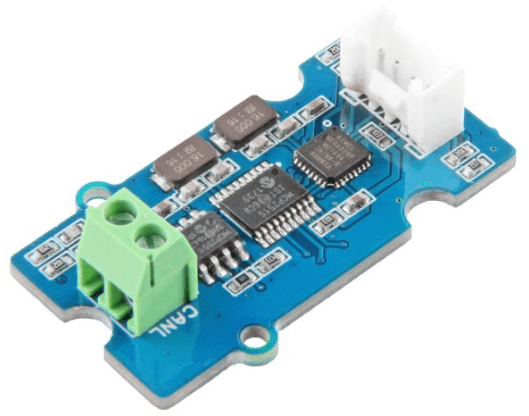
[](https://www.seeedstudio.com/CANBed-Arduino-CAN-BUS-Development-Kit-Atmega32U4-with-MCP2515-and-MCP2551-p-4365.html)

CANBed – Arduino CAN-BUS Development Kit carries an Atmega32U4 chip and MCP2515, MCP2551 CAN-BUS controller and transceiver to realize the CAN-BUS communication protocol on a single board without other MCU to control, it is a CAN-BUS Development Board itself! Compared to the above CAN BUS shield, this CAN BED carries an Arduino chip – Atmega32U4 which means it is a combination of the CAN-BUS Shield and [Arduino Development Board](https://www.seeedstudio.com/catalogsearch/result/?q=Arduino+Development+Board) together on one single board.

Features of the CANBED includes:

* Atmega32U4 with Arduino Leonardo bootloader on the board
* MCP2515 CAN Bus controller and MCP2551 CAN Bus transceiver
* BD-II and CAN standard pinout selectable at the sub-D connector

[**Serial CAN-BUS Module based on MCP2551 and MCP2515 ($14.90)**](https://www.seeedstudio.com/Serial-CAN-BUS-Module-based-on-MCP2551-and-MCP2515.html?utm_source=blog&utm_medium=blog)

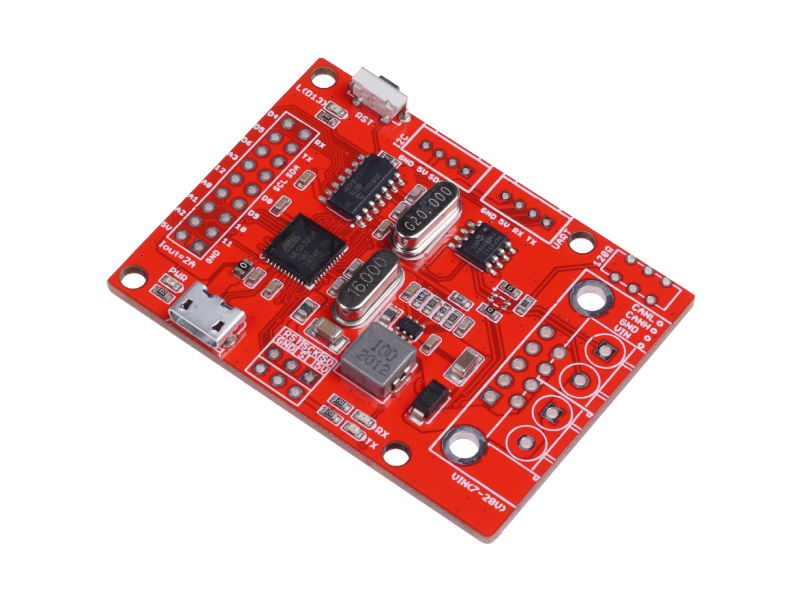
[](https://www.seeedstudio.com/Serial-CAN-BUS-Module-based-on-MCP2551-and-MCP2515.html)

If your project is space limited, here is a Serial CAN-BUS module which has the full features of CAN Bus. The Serial CAN-BUS provides your Arduino or other MCU with the capability of communication to CAN Bus, such as hacking your vehicle. This Grove CAN-BUS module is controlled by UART, which means if your MCU has a UART interface, this serial CAN BUS is available.

Features of this Serial CAN BUS module includes:

* UART to CAN-BUS communication
* AT command
* Up to 115200 UART baud rate (default 9600)
* Up to 1Mb/s CAN-BUS baud rate
* TX and RX led indicator
* 4pin Grove connector
* 3.3V working voltage
* Easy-to-use [Arduino library](https://github.com/Longan-Labs/Serial_CAN_Arduino)
* Small size

[**CANBed FD – Arduino CAN-FD Development Kit ($17.90)**](https://www.seeedstudio.com/CANBed-FD-Arduino-CAN-FD-Development-Kit-p-4724.html)



The newest addition to the CAN family, this CAN Bed-FD carries MCP2517FD CAN Bus controller with SPI interface and MCP2542FD CAN transceiver to achieve the CAN-BUS capability. There is also an ATmega32U4 with Arduino Leonardo bootloader on the board, which allows for easy programming using Arduino IDE. Moreover, there is an OBD-II converter cable added on along with OBD-II library imported, you’re able to build an onboard diagnostic device whenever you feel like it! Not only that, but CAN Bed-FD comes in a compact size of 56x41mm which for easy storage and suitable for small projects

Features of the CANBed-FD includes:

* Wide power input range from 7-28V
* 2 x 4-Pin Grove connectors compatible with the Grove ecosystem
* OBD-II and CAN standard pinout selectable at the sub-D connector
* Industrial standard 9 pin sub-D connector or 4-pin terminal
* Work at CAN-FD and CAN 2.0
* SPI Interface up to 10 MHz

[**OBD-II CAN Bus GPS Development Kit ($29.90)**](https://www.seeedstudio.com/OBD-II-CAN-Bus-GPS-Development-Kit-p-2868.html?utm_source=blog&utm_medium=blog)

[](https://www.seeedstudio.com/OBD-II-CAN-Bus-GPS-Development-Kit-p-2868.html)

If you want to record the GPS log at the same time, this OBD-II CAN Bus GPS Development Kit is highly recommended! The board can be plugged directly into the car via the ODB-II interface and supports Arduino programming. The baseboard of the development kit is integrated with an Atmega32U4 microprocessor. The CAN-Bus library is available to write sketches using Arduino IDE to send and receive messages from the CAN bus network and also allows you to fetch useful data from the messages.

**Getting started with CAN BUS with Arduino**

Getting Started with CAN BUS can be very fulfilling. It can be daunting as well, but this guide will provide you a step-by-step guide to get started!

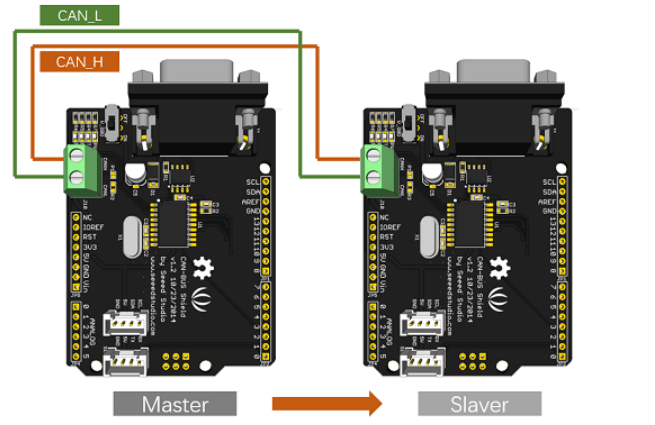
**What do you need?**

* 2 x [Seeeduino V4.2](https://www.seeedstudio.com/Seeeduino-V4.2-p-2517.html?utm_source=blog&utm_medium=blog)
* 2 x [CAN-BUS Shield V2](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html?utm_source=blog&utm_medium=blog)
* 2 x [Jumper Wire](https://www.seeedstudio.com/Breadboard-Jumper-Wire-Pack-241mm-200mm-160mm-117m-p-234.html?utm_source=blog&utm_medium=blog)\

**Instructions**

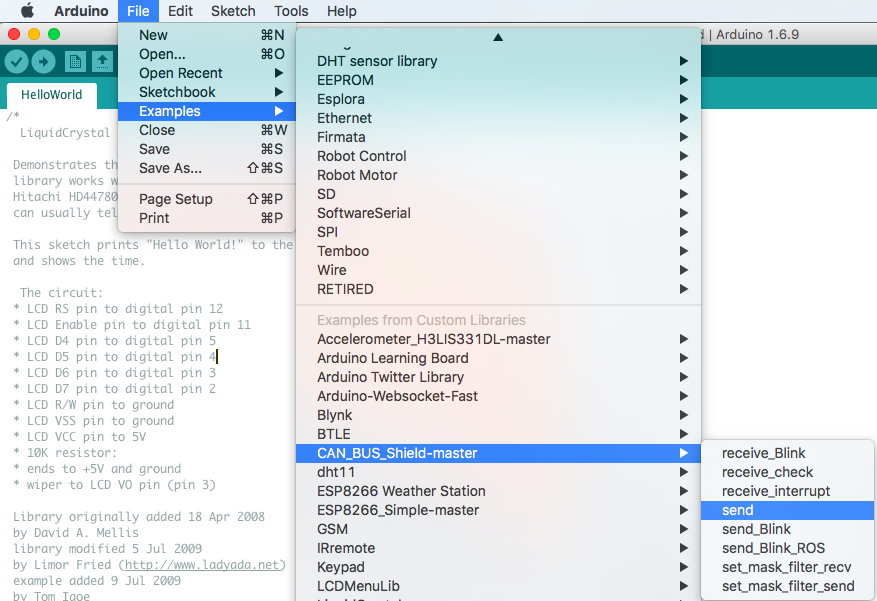
**Step 1: Hardware Connection**

* Firstly, connect each CAN-BUS shield into Seeeduino V4.2 and connect the 2 CAN-BUS shield together via 2 jumper wires as shown below. (CAN\_H to CAN\_H and CAN\_L to CAN\_L)

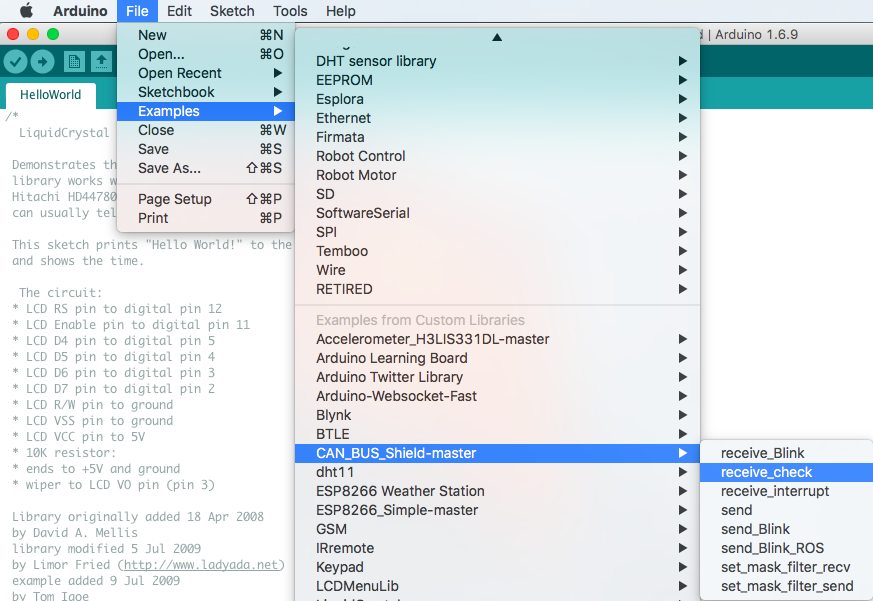


**Step 2: Setting up Software**

* Download the library for CAN BUS [here](https://github.com/Seeed-Studio/CAN_BUS_Shield) and install the library to Arduino IDE after the download has completed.
* If you do not know how to install an Arduino library, you can follow the instructions here.
* One of the nodes (a node means Seeeduino + CAN\_BUS Shield) acts as master, the other acts as a slaver. The master will send data to slaver constantly.
* Open the **send** example (**File > Examples > CAN\_BUS\_Shield-master >send**) and upload to the **master** as shown below:

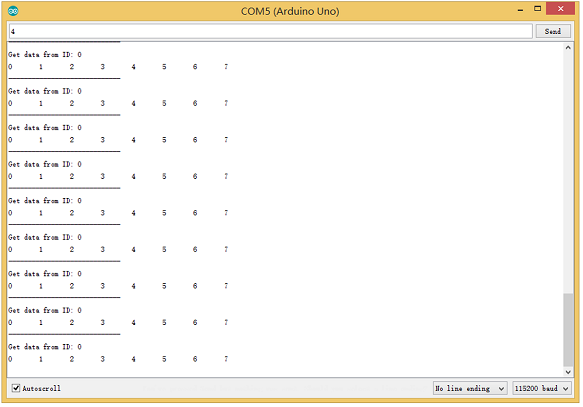


* After that, Open the **receive\_check** example (**File > Examples > CAN\_BUS\_Shield-master > receive\_check**) and upload to the **slaver** as shown below



**Step 3: View Results**

* Open the Serial Monitor of Arduino IDE (**slaver**), you will get the data sent from the master. Below is a photo of what it will look like:



**Step 4: APIs (Application Program Interface)**

For the APIs of this software, you can:

**Set the Baud Rate**

* This function is used to initialize the baud rate of the CAN Bus system. The available baud rates are listed as follows:

#define CAN\_5KBPS 1

#define CAN\_10KBPS 2

#define CAN\_20KBPS 3

#define CAN\_25KBPS 4

#define CAN\_31K25BPS 5

#define CAN\_33KBPS 6

#define CAN\_40KBPS 7

#define CAN\_50KBPS 8

#define CAN\_80KBPS 9

#define CAN\_83K3BPS 10

#define CAN\_95KBPS 11

#define CAN\_100KBPS 12

#define CAN\_125KBPS 13

#define CAN\_200KBPS 14

#define CAN\_250KBPS 15

#define CAN\_500KBPS 16

#define CAN\_666kbps 17

#define CAN\_1000KBPS 18

**Set Receive Mask and Filter**

* There are 2 receive mask registers and **5** filter registers on the controller chip that guarantees you getting data from the target device. They are useful especially in a large network consisting of numerous nodes.
* We provide two functions for you to utilize these mask and filter registers. They are:

Mask

init\_Mask(unsigned char num, unsigned char ext, unsigned char ulData);

Filter

init\_Filt(unsigned char num, unsigned char ext, unsigned char ulData);

* num represents which register to use. You can fill 0 or 1 for mask and 0 to 5 for the filter.
* ext represents the status of the frame. 0 means it’s a mask or filter for a standard frame. 1 means it’s for an extended frame.
* ulData represents the content of the mask of the filter.

**Check Receive**

* The MCP2515 can operate in either a polled mode, where the software checks for a received frame, or using additional pins to signal that a frame has been received or transmit completed.
* Use the following function to poll for received frames:

INT8U MCP\_CAN::checkReceive(void);

The function will return 1 if a frame arrives, and 0 if nothing arrives.

**Get CAN ID**

* When some data arrive, you can use the following function to get the CAN ID of the “send” node.

INT32U MCP\_CAN::getCanId(void)

**Send Data**

CAN.sendMsgBuf(INT8U id, INT8U ext, INT8U len, data\_buf);

It is a function to send data onto the bus. In which:

* id represents where the data comes from.
* ext represents the status of the frame. ‘0’ means standard frame. ‘1’ means an extended frame.
* len represents the length of this frame.
* data\_buf is the content of this message.

For example, In the ‘send’ example, we have:

unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};

CAN.sendMsgBuf(0x00, 0, 8, stmp); //send out the message 'stmp' to the bus and tell other devices this is a standard frame from 0x00.

**Receive Data**

* The following function is used to receive data on the ‘receive’ node:

CAN.readMsgBuf(unsigned char len, unsigned char buf);

In conditions that masks and filters have been set. This function can only get frames that meet the requirements of masks and filters.

* len represents the data length.
* buf is where you store the data.

**Generate a New BaudRate**

We had provided many frequently-used baud rates, as shown below:

#define CAN\_5KBPS 1

#define CAN\_10KBPS 2

#define CAN\_20KBPS 3

#define CAN\_25KBPS 4

#define CAN\_31KBPS 5

#define CAN\_33KBPS 6

#define CAN\_40KBPS 7

#define CAN\_50KBPS 8

#define CAN\_80KBPS 9

#define CAN\_83KBPS 10

#define CAN\_95KBPS 11

#define CAN\_100KBPS 12

#define CAN\_125KBPS 13

#define CAN\_200KBPS 14

#define CAN\_250KBPS 15

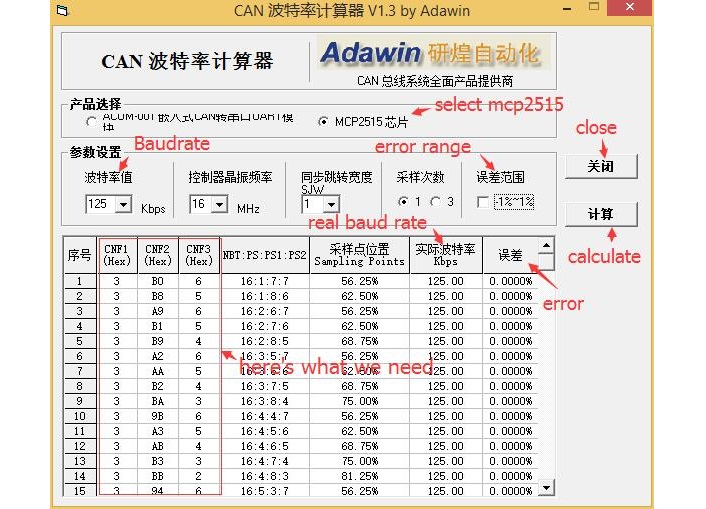
#define CAN\_500KBPS 16

#define CAN\_666KBPS 17

#define CAN\_1000KBPS 18

Yet you may still can’t find the rate you want. Here we provide software to help you to calculate the baud rate you need.

Click [here](https://github.com/SeeedDocument/CAN_BUS_Shield/raw/master/resource/CAN_Baudrate_CalcV1.3.zip) to download the software, it’s in Chinese, but it’s easy to use. Here is the interface translated for easier usage:



* Open the software, what you need to do is to set the baud rate you want, and then do some simple setting, then click calculate.
* Then you will get some data, cfg1, cfg2 and cfg3.
* You need to add some code to the library.
* Open mcp\_can\_dfs.h, you need to add below code at about line 272

#define MCP\_16MHz\_xxxkBPS\_CFG1 (cfg1) // xxx is the baud rate you need

#define MCP\_16MHz\_xxxkBPS\_CFG2 (cfg2)

#define MCP\_16MHz\_xxxkBPS\_CFG3 (cfg2)

* Then let’s go to about line 390, add below code:

#define CAN\_xxxKBPS NUM // xxx is the baudrate you need, and NUM is a number, you need to get a different from the other rates.

* Open mcp\_can.cpp, goto the function mcp2515\_configRate(at about line 190), then add below code:

case (CAN\_xxxKBPS):

cfg1 = MCP\_16MHz\_xxxkBPS\_CFG1;

cfg2 = MCP\_16MHz\_xxxkBPS\_CFG2;

cfg3 = MCP\_16MHz\_xxxkBPS\_CFG3;

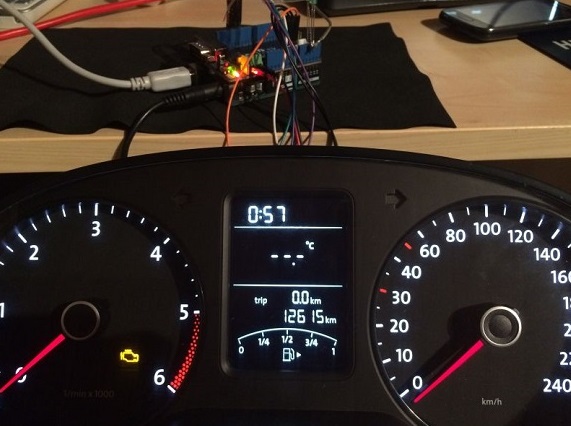
break;

That’s all! Now, you can use the baud rate you need!

Now that you’ve learnt how to interface CAN BUS with the Arduino using the CAN BUS shield, here are some CAN BUS projects to get you started!

**CAN BUS Projects**

[**Volkswagen CAN BUS Gaming**](https://community.seeedstudio.com/project_detail.html?id=291)

Inspired by Silas Parker

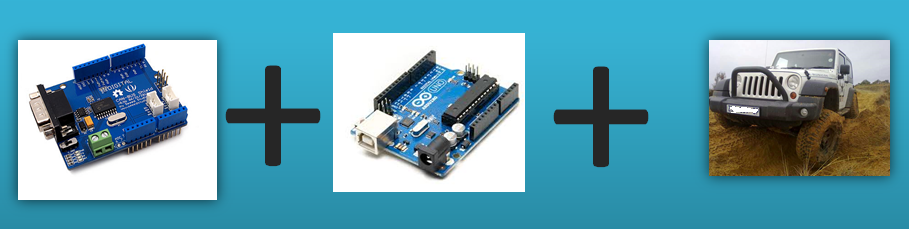
Ever wanted to play a car/truck simulator with a real dashboard on your PC? With the CAN BUS shield, you can! In this project, I’m trying to control a VW Polo 6R dashboard via CAN Bus with an Arduino Uno and a Seeed CAN Bus Shield.

**What you’ll need:**

* [Seeeduino V4.2](https://www.seeedstudio.com/Seeeduino-V4-2-p-2517.html?utm_source=blog&utm_medium=blog)/ [Arduino UNO Rev3](https://www.seeedstudio.com/Arduino-Uno-Rev3-p-2995.html?utm_source=blog&utm_medium=blog)
* [CAN-BUS Shield V2](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html?utm_source=blog&utm_medium=blog)
* [Jumper Wires](https://www.seeedstudio.com/Breadboard-Jumper-Wire-Pack-241mm-200mm-160mm-117m-p-234.html?utm_source=blog&utm_medium=blog)
* [Type-B USB cable for Arduino Diecimila and Freeduino](https://www.seeedstudio.com/Type-B-USB-cable-for-Arduino-Diecimila-and-Freeduino-p-130.html?utm_source=blog&utm_medium=blog)

Interested? You can check out the full tutorial at [SeeedStudio Community!](https://community.seeedstudio.com/project_detail.html?id=291)

[**Hacking your Vehicle**](https://www.instructables.com/id/Hack-your-vehicle-CAN-BUS-with-Arduino-and-Seeed-C/)

Photo by [mviljoen2](https://www.instructables.com/member/mviljoen2/) on Instructables

Modern Vehicles all come equipped with a CAN-BUS Controller Area Network, Instead of having a million wires running back and forth from various devices in your car to the battery, its making use of a more clever system. From each node (IE Switch pod that controls your windows or electric door locks) it broadcasts a message across the CAN. When the TIPM detects a valid message it will react accordingly like, lock the doors, switch on lights and so on. With the CAN BUS shield, you can now hack your vehicle!

**What do you need?**

* [Seeeduino V4.2](https://www.seeedstudio.com/Seeeduino-V4-2-p-2517.html?utm_source=blog&utm_medium=blog)/ [Arduino UNO Rev3](https://www.seeedstudio.com/Arduino-Uno-Rev3-p-2995.html?utm_source=blog&utm_medium=blog)
* [CAN-BUS Shield V2](https://www.seeedstudio.com/CAN-BUS-Shield-V2.html?utm_source=blog&utm_medium=blog)
* [Jumper Wires](https://www.seeedstudio.com/Breadboard-Jumper-Wire-Pack-241mm-200mm-160mm-117m-p-234.html?utm_source=blog&utm_medium=blog)

Interested? You can find the full tutorial by [mvilijoen2 on Instructables!](https://www.instructables.com/id/Hack-your-vehicle-CAN-BUS-with-Arduino-and-Seeed-C/)

**Summary**

With the basic knowledge of CAN BUS protocol in your hands, one way that you can use it is to analyze the data/messages passing through your vehicle and also hack your vehicle!

I hope you learn more about the CAN-BUS protocol in this guide! If you have any questions regarding CAN BUS, feel free to drop a comment down in the comment section down below.

Interested in more CAN BUS products? You can check out all of our CAN BUS products [here](https://www.seeedstudio.com/catalogsearch/result/?q=CAN+BUS)!

[](https://www.seeedstudio.com/blog/author/yida/)

[**yida**](https://www.seeedstudio.com/blog/author/yida/)

[See author's posts](https://www.seeedstudio.com/blog/author/yida/)

Tags: [Advantage of using CAN BUS](https://www.seeedstudio.com/blog/tag/advantage-of-using-can-bus/), [Arduino CAN BUS](https://www.seeedstudio.com/blog/tag/arduino-can-bus/), [CAN BUS](https://www.seeedstudio.com/blog/tag/can-bus/), [CAN BUS Message](https://www.seeedstudio.com/blog/tag/can-bus-message/), [CAN BUS Projects](https://www.seeedstudio.com/blog/tag/can-bus-projects/), [CAN BUS Protocol](https://www.seeedstudio.com/blog/tag/can-bus-protocol/), [CAN BUS Protocol Range](https://www.seeedstudio.com/blog/tag/can-bus-protocol-range/), [CAN BUS Protocol Speed](https://www.seeedstudio.com/blog/tag/can-bus-protocol-speed/), [CAN BUS System](https://www.seeedstudio.com/blog/tag/can-bus-system/), [CAN BUS vs OBD2](https://www.seeedstudio.com/blog/tag/can-bus-vs-obd2/), [CAN BUS Wires](https://www.seeedstudio.com/blog/tag/can-bus-wires/), [CAN Protocol](https://www.seeedstudio.com/blog/tag/can-protocol/), [CAN-BUS Shield](https://www.seeedstudio.com/blog/tag/can-bus-shield/), [Introduction to CAN BUS](https://www.seeedstudio.com/blog/tag/introduction-to-can-bus/), [OBD2](https://www.seeedstudio.com/blog/tag/obd2/), [Tutorial](https://www.seeedstudio.com/blog/tag/tutorial/)