CAN-BUS Shield V2.0

CAN-BUS is a common industrial bus because of its long travel distance, medium communication speed and high reliability. It is commonly found on modern machine tools, such as an automotive diagnostic bus.

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.

Version

This document applies to the following version of products:

Version	Released Date	How to Buy
CAN BUS Shield V1.0	Oct 14, 2012	E.O.L
CAN BUS Shield V1.1	Aug 10, 2013	E.O.L
CAN BUS Shield V1.2	Jan 5, 2015	E.O.L
CAN BUS Shield V2.0	Aug 01,2017	

What's new in CAN BUS Shield V1.2

- Pads on the backside of PCBA
- Change terminal resistor to 120 Ohm

Alternative Choice

If your project is space limited and also don't need other fuctions except CAN-BUS, here is a Grove CAN-BUS module which is Arduino compatible, more compact and cost effective, please click here to visit its page.
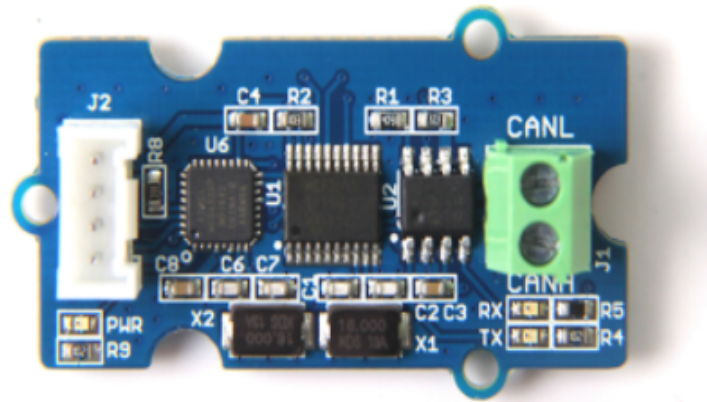
What if I want to connect this shield to my car

If you want to read data or control your car, there's an OBD>DB9 cable available for you, this cable make easier to connect to OBD-connector and DB9-connector. This cable will also work with anything that has a OBD-connector. Add a power switch makes such a satisfying click.

USB-CAN Analyzer

If you want a CAN Bus Analyzer to debug your CAN Bus, this USB-CAN Analyzer is recommended.

Features

- Implements CAN V2.0B speed up to **1 Mb/s**

- SPI Interface speed up to **10 MHz**

- Standard (**11 bit**) and extended (**29 bit**) data and remote frames

- Two receive buffers with prioritized message storage

- Industrial standard DB-9 connector

- LED indicators

Hardware Overview
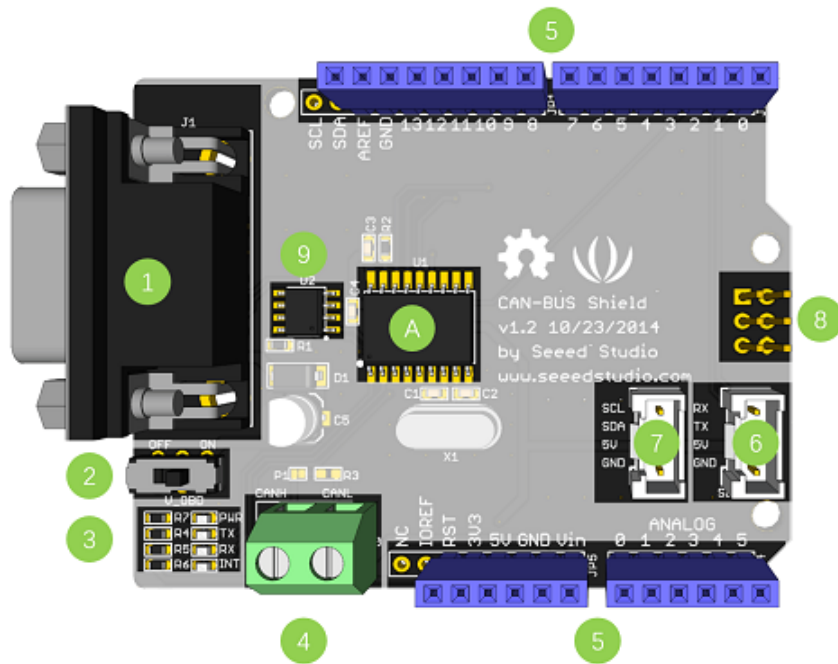
1. **DB9 Interface** - to connect to OBDII Interface via a DBG-OBD Cable.

2. **V_OBD** - It gets power from OBDII Interface (from DB9)

3. **Led Indicator**:

- **PWR**: power

- **TX**: blink when the data is sending

- **RX**: blink when there's data receiving

- **INT**: data interrupt

4. **Terminal** - CAN_H and CAN_L

5. **Arduino UNO pin out**

6. **Serial Grove connector**

7. **I2C Grove connector**

8. **ICSP pins**

9. **IC** - MCP2551, a high-speed CAN transceiver (datasheet)

10. **IC** - MCP2515, stand-alone CAN controller with SPI interface (datasheet)

> ⚠️ **Warning**
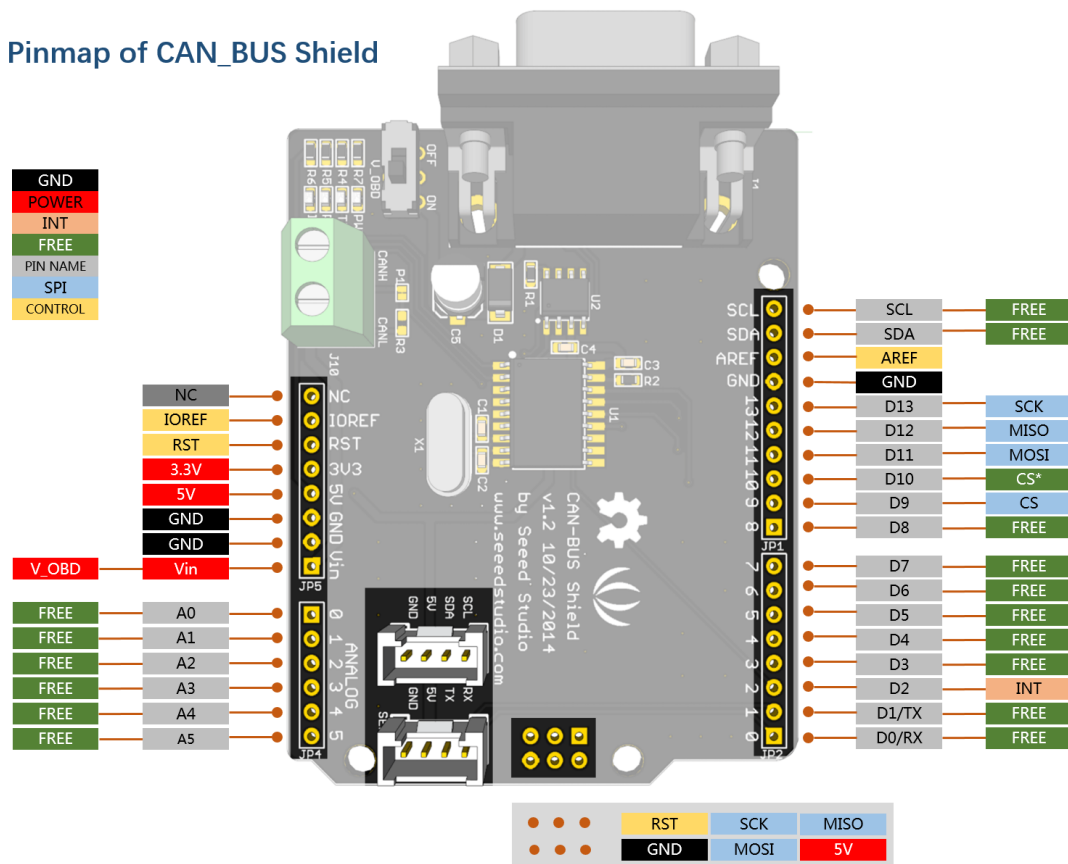> When you use more than two CAN Bus Shield in one net, you should take the impedance into consideration. You should either cut P1 in the PCB with a knife, or just remove R3 on the PCB.

Pin map

Pinmap of CAN_BUS Shield

Legend	
GND	
POWER	
INT	
FREE	
PIN NAME	
SPI	
CONTROL	

> **Note**
> The FREE pin is available for the other usages.

DB9&OBDii Interface
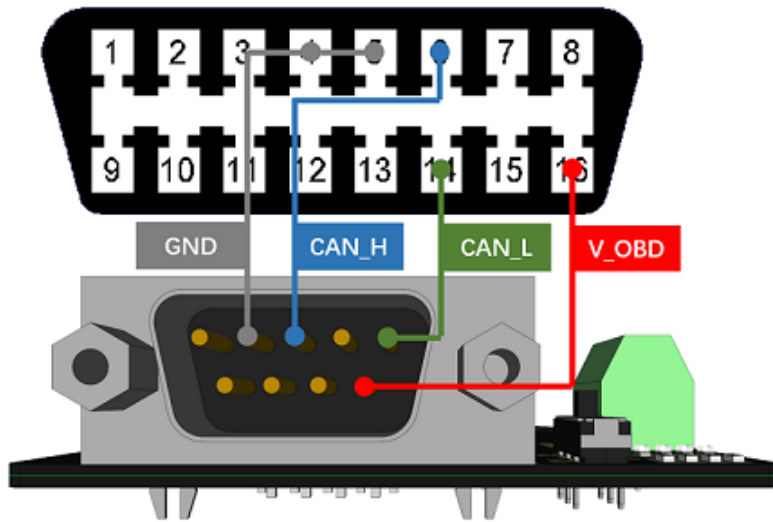
CS pin

⚠️ **Attention**
When we produced the new batch of CAN-BUS Shield V2, the wire of the back pads was embedded inside the PCB, although the wire between the pads is now not visible on the outside, the inside is still connected, if you want to change the wiring of the pads, you still need to cut the wiring in the PCB first.

SPI_CS pin of V1.2 is connected to **D9** by default. If you want to change to **D10**, please follow below instructions.

- Step1: Take a look at the backside of the PCBA, you will find a pad named CS.
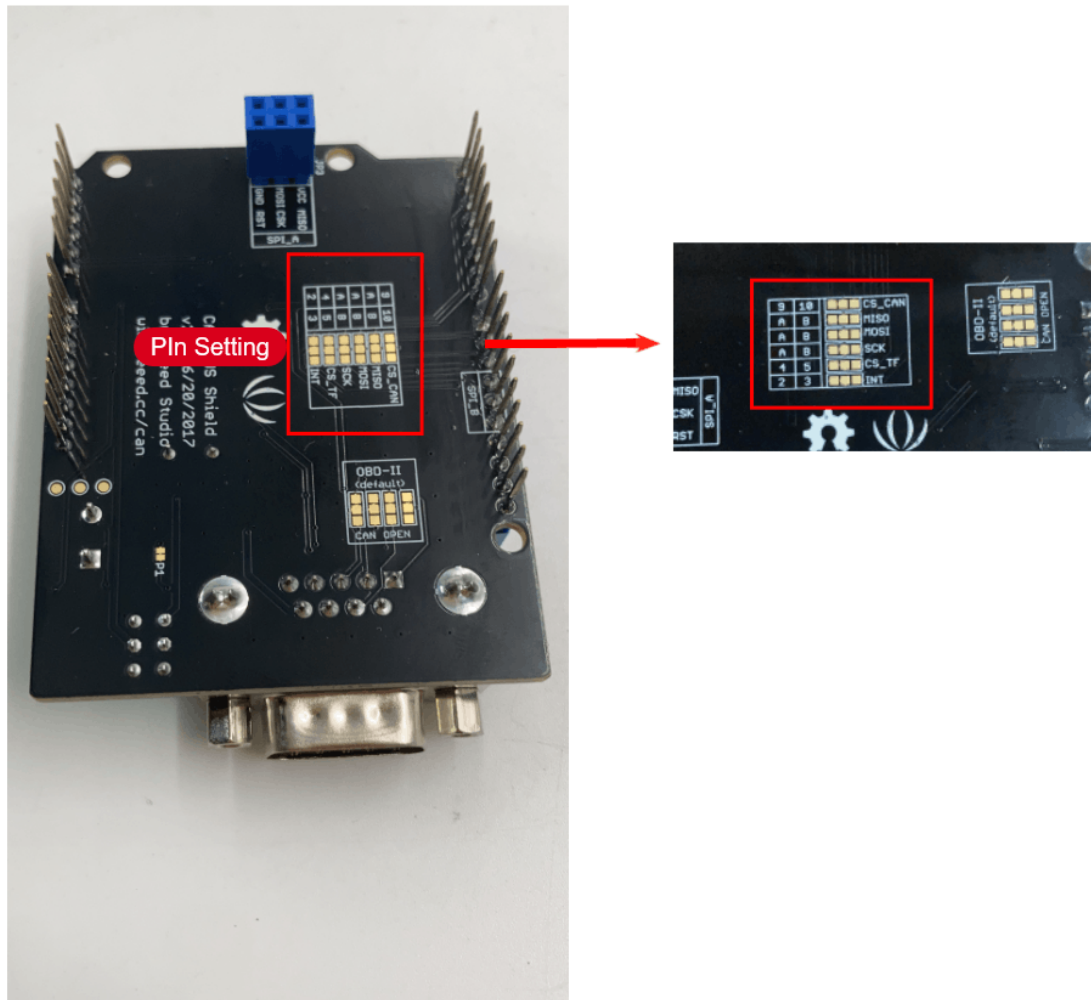
- Step2: Cut the wire between pad9 and the middle pad.

Cut this wire with box cutter

9	10		CS_CAN
A	B		MISO
A	B		MOSI
A	B		SCK
4	5		CS_TF
2	3		INT

- Step3:Solder the middle pad and pad 10.
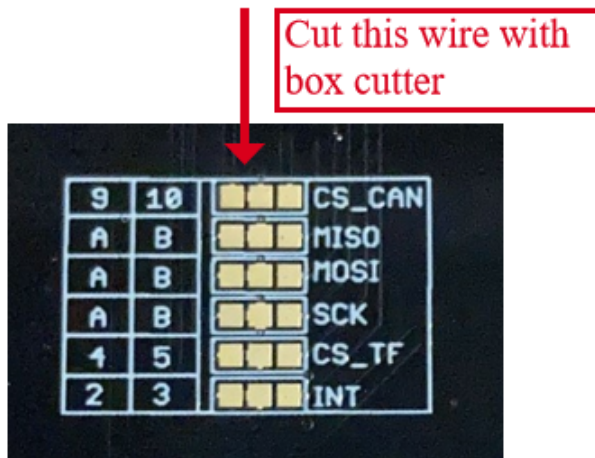
solder the middle pad and pad 10
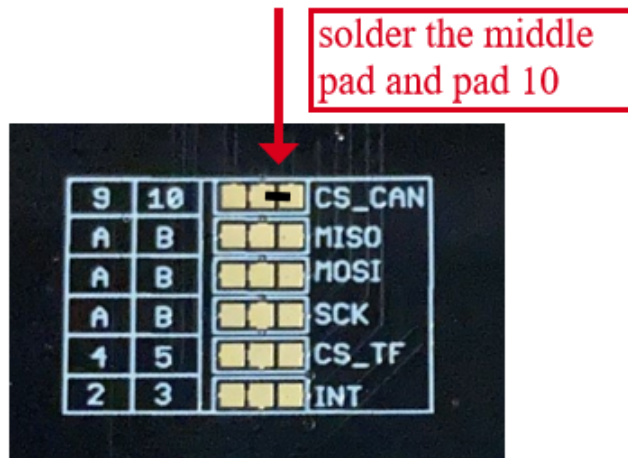
> ⚠ **Warning**
> Be careful with the box cutter, it's easy to hurt yourself or the PCBA.

SPI pins

The SPI pins (SCK, MISO, MOSI) are routed to the ICSP pins by default. But for some boards, the SPI pins are located at D11~D13. if this happens, you need make some change to the PCBA. Take a look at the backside of the PCBA, there're three pads, MOSI, MISO and SCK, they are connected to A by default. You can change them to B if needed.

> ✏️ **Note**
> For Arduino UNO, Arduino Mega, Arduino Leonardo and any others AVR based Arduino boards, it works well by default setting.

> ⚠️ **Warning**
> Be careful when you are going to change SPI pins, it's easy to hurt yourself or the PCBA.

Getting Started

Here's a simple example to show you how CAN-BUS Shield works. In this example we need 2 pieces of CAN-BUS Shields as well as Arduino or Seeeduino.

> ✏️ **Note**
> This example is built under Arduino IDE version 1.6.9.

STEP1: What do we need

Name	Function	Qty	Link
CAN-BUS Shield	CAN Bus communication	2	link
Seeeduino V4.2	Controller	2	link
Jumper Wire	connection	2	link

STEP2: Hardware Connection

Insert each CAN-BUS Shield into Seeeduino V4.2, and connect the 2 CAN-BUS Shield together via 2 jumper wires. Shown as below images.
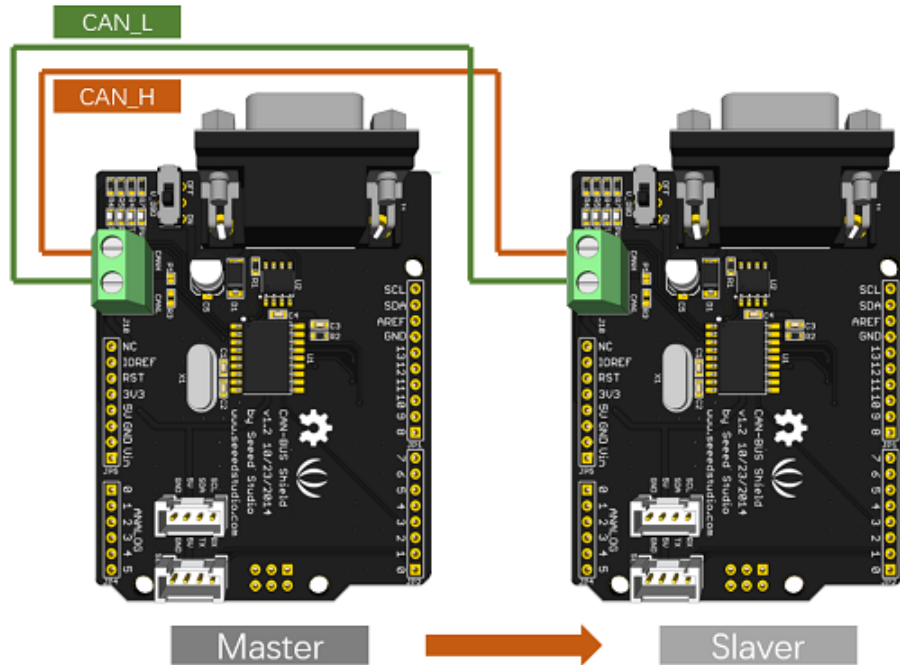
> **Note**
> CAN_H to CAN_H, CAN_L to CAN_L

STEP3: Software

Please follow how to install an arduino library procedures to install CAN BUS shield library.

- Download the **Seeed_Arduino_CAN** Arduino library here.

Install the library to your Arduino IDE when it is downloaded.

One of the node (a node means Seeeduino + CAN_BUS Shield) acts as master, the other acts as slaver. The master will send data to slaver constantly.

> **Note**
> Each node can act as master before the code being uploaded.

Open the **send** example (**File > Examples > Seeed_Arduino_CAN > send**) and upload to the **master**.

Arduino   File   Edit   Sketch   Tools   Help

New   ⌘N
Open...   ⌘O
Open Recent   ▶
Sketchbook   ▶
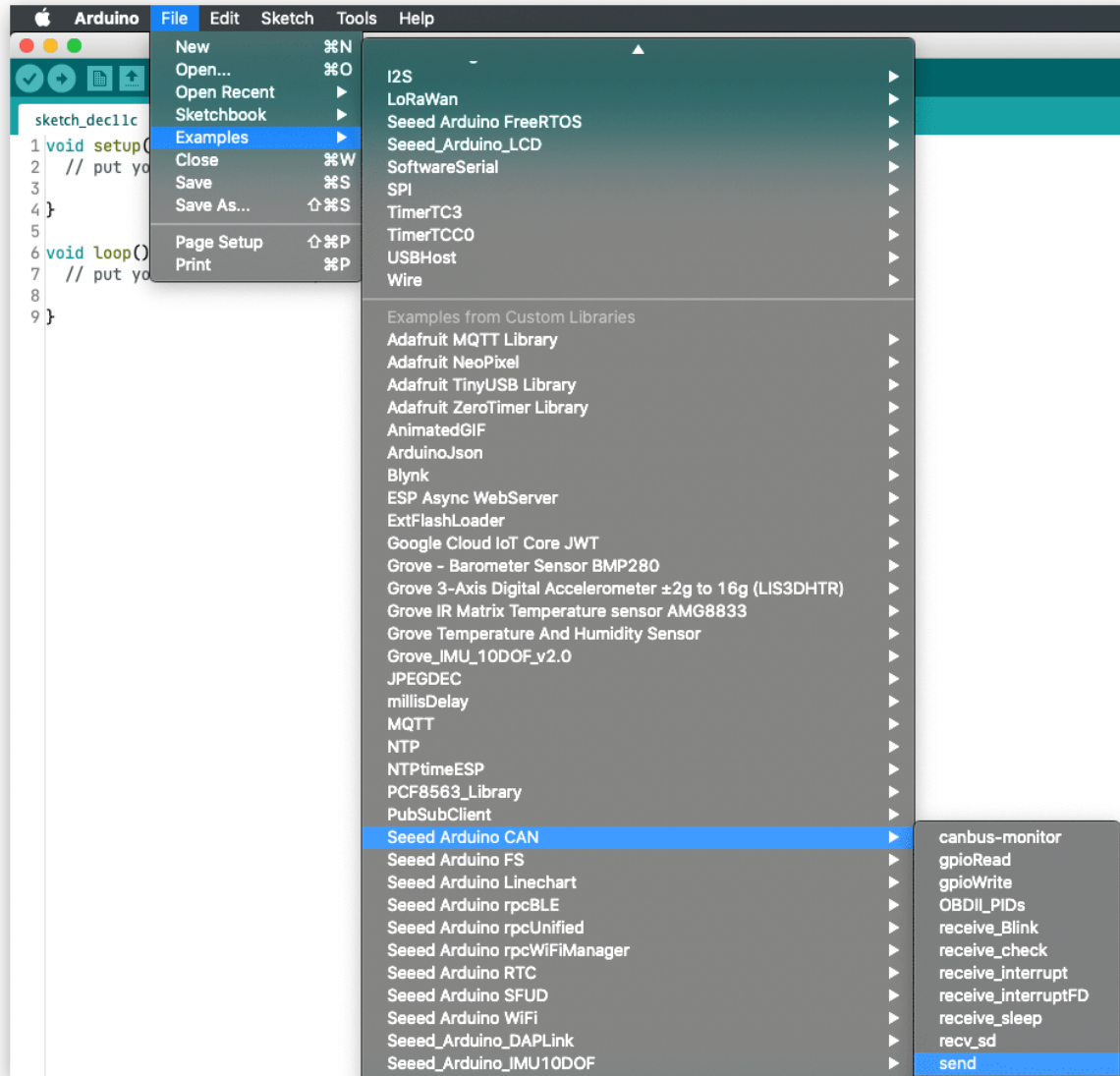Examples   ▶
Close   ⌘W
Save   ⌘S
Save As...   ⇧⌘S

Page Setup   ⇧⌘P
Print   ⌘P

sketch_dec11c
1 void setup(
2   // put yo
3
4 }
5
6 void loop()
7   // put yo
8
9 }

I2S   ▶
LoRaWan   ▶
Seeed Arduino FreeRTOS   ▶
Seeed_Arduino_LCD   ▶
SoftwareSerial   ▶
SPI   ▶
TimerTC3   ▶
TimerTCC0   ▶
USBHost   ▶
Wire   ▶

Examples from Custom Libraries
Adafruit MQTT Library   ▶
Adafruit NeoPixel   ▶
Adafruit TinyUSB Library   ▶
Adafruit ZeroTimer Library   ▶
AnimatedGIF   ▶
ArduinoJson   ▶
Blynk   ▶
ESP Async WebServer   ▶
ExtFlashLoader   ▶
Google Cloud IoT Core JWT   ▶
Grove - Barometer Sensor BMP280 ▶
Grove 3-Axis Digital Accelerometer ±2g to 16g (LIS3DHTR) ▶
Grove IR Matrix Temperature sensor AMG8833 ▶
Grove Temperature And Humidity Sensor ▶
Grove_IMU_10DOF_v2.0   ▶
JPEGDEC   ▶
millisDelay   ▶
MQTT   ▶
NTP   ▶
NTPtimeESP   ▶
PCF8563_Library   ▶
PubSubClient   ▶
Seeed Arduino CAN   ▶
Seeed Arduino FS   ▶
Seeed Arduino Linechart   ▶
Seeed Arduino rpcBLE   ▶
Seeed Arduino rpcUnified   ▶
Seeed Arduino rpcWiFiManager   ▶
Seeed Arduino RTC   ▶
Seeed Arduino SFUD   ▶
Seeed Arduino WiFi   ▶
Seeed_Arduino_DAPLink   ▶
Seeed_Arduino_IMU10DOF   ▶

canbus-monitor
gpioRead
gpioWrite
OBDII_PIDs
receive_Blink
receive_check
receive_interrupt
receive_interruptFD
receive_sleep
recv_sd
send

Or copy the following to the Arduino IDE and upload:

```cpp
#include <SPI.h>
#include "mcp2515_can.h"

/*SAMD core*/
#ifdef ARDUINO_SAMD_VARIANT_COMPLIANCE
    #define SERIAL SerialUSB
#else
    #define SERIAL Serial
#endif

const int SPI_CS_PIN = 9;
mcp2515_can CAN(SPI_CS_PIN); // Set CS pin

void setup() {
    SERIAL.begin(115200);
    while(!Serial){};

    while (CAN_OK != CAN.begin(CAN_500KBPS)) {              // init can b
        SERIAL.println("CAN BUS Shield init fail");
        SERIAL.println(" Init CAN BUS Shield again");
        delay(100);
    }
    SERIAL.println("CAN BUS Shield init ok!");
}

unsigned char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};
void loop() {
    // send data:  id = 0x00, standrad frame, data len = 8, stmp: data bu:
    stmp[7] = stmp[7] + 1;
    if (stmp[7] == 100) {
        stmp[7] = 0;
        stmp[6] = stmp[6] + 1;
```

```
33
34            if (stmp[6] == 100) {
35                stmp[6] = 0;
36                stmp[5] = stmp[6] + 1;
37            }
38        }
39
40      CAN.sendMsgBuf(0x00, 0, 8, stmp);
41      delay(100);                              // send data per 100ms
42      SERIAL.println("CAN BUS sendMsgBuf ok!");
43  }
```

STEP4: View Result

Open the Serial Monitor of Arduino IDE(**slaver**), you will get the data sent from the master.
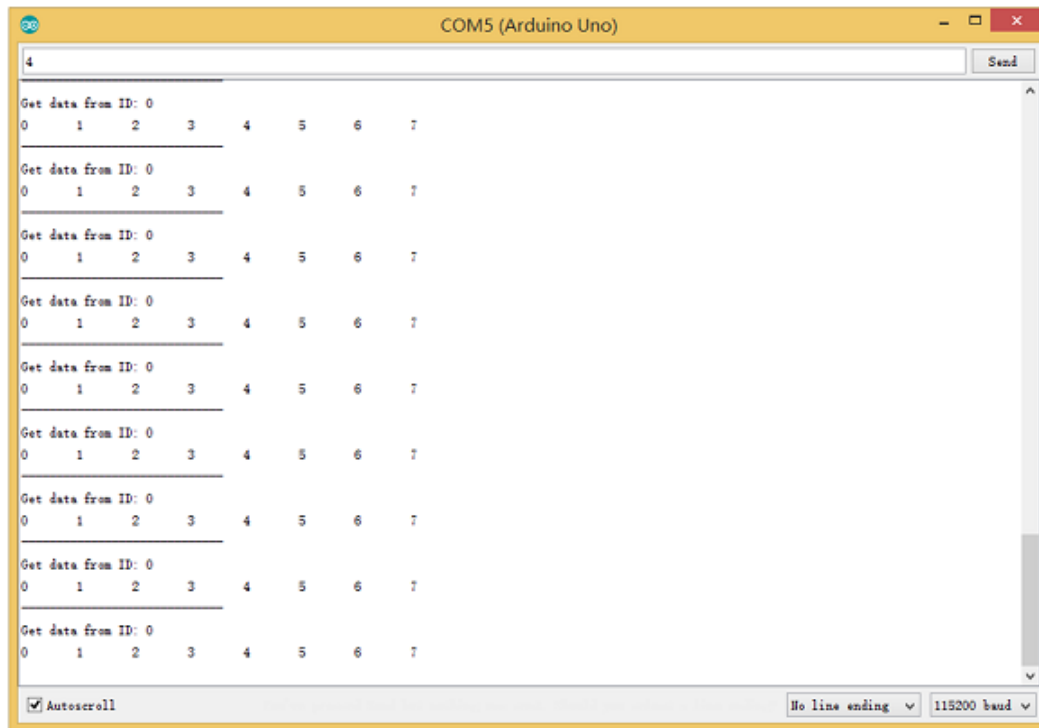
APIs

1.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:

```
1   #define CAN_5KBPS    1
```

```
 2  #define CAN_10KBPS    2
 3  #define CAN_20KBPS    3
 4  #define CAN_25KBPS    4
 5  #define CAN_31K25BPS 5
 6  #define CAN_33KBPS    6
 7  #define CAN_40KBPS    7
 8  #define CAN_50KBPS    8
 9  #define CAN_80KBPS    9
10  #define CAN_83K3BPS  10
11  #define CAN_95KBPS   11
12  #define CAN_100KBPS  12
13  #define CAN_125KBPS  13
14  #define CAN_200KBPS  14
15  #define CAN_250KBPS  15
16  #define CAN_500KBPS  16
17  #define CAN_666kbps  17
18  #define CAN_1000KBPS 18
```

2.Set Receive Mask and Filter

There are **2** receive mask registers and **5** filter registers on the controller chip that guarantee 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 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 a extended frame.

- **ulData** represents the content of the mask of filter.

3.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.

4.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)
```

5.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 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:

```
1  unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
2  CAN.sendMsgBuf(0x00, 0, 8, stmp); //send out the message 'stmp' to the bu
```

6.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.

7.init_CS

it is very usefule for you to use two CAN-BUS shields with one Arduino.we provide CS_CAN pin to select for you.if you cannot kown how to use CS_CAN pin , you can go to CS_CAN pin to learn.

```
void MCP_CAN::init_CS(byte _CS)
```

- **_CS** represents what you select pin.(9 or 10)

Generate a New BaudRate

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

```
1  #define CAN_5KBPS    1
2  #define CAN_10KBPS   2
3  #define CAN_20KBPS   3
4  #define CAN_25KBPS   4
5  #define CAN_31KBPS   5
```

```
 6   #define CAN_33KBPS    6
 7   #define CAN_40KBPS    7
 8   #define CAN_50KBPS    8
 9   #define CAN_80KBPS    9
10   #define CAN_83KBPS    10
11   #define CAN_95KBPS    11
12   #define CAN_100KBPS   12
13   #define CAN_125KBPS   13
14   #define CAN_200KBPS   14
15   #define CAN_250KBPS   15
16   #define CAN_500KBPS   16
17   #define CAN_666KBPS   17
18   #define CAN_1000KBPS  18
```

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

Click here to download the software, it's in Chinese, but never mind, it's easy to use.

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:

```
1   #define MCP_16MHz_xxxkBPS_CFG1 (cfg1)      // xxx is the baud rate you need
2   #define MCP_16MHz_xxxkBPS_CFG2 (cfg2)
3   #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 i.
```

Open **mcp_can.cpp**, goto the function **mcp2515_configRate**(at about line 190), then add below code:

```
1   case (CAN_xxxKBPS):
2       cfg1 = MCP_16MHz_xxxkBPS_CFG1;
3       cfg2 = MCP_16MHz_xxxkBPS_CFG2;
4       cfg3 = MCP_16MHz_xxxkBPS_CFG3;
5       break;
```

Then you can use the baud rate you need. And please give me a pull request at github when you use a new rate, so I can add it to the library to help other guys.

FAQ

Q1: I can't get data from other CAN device.

- Check if the connection is right

- Check if the baud rate setting is right

Q2: The serial monitor print Init Fail.

- Check if the CS pin setting is matched with the code. For CAN Bus Shield V1.1/1.2, CS pin is connected to D9, others are to D10.

Schematic Online Viewer

Resources

- **[PDF]**CAN-BUS Shield V2.0 Schmatics

- **[Eagle]**Schematic & PCB of CAN-BUS Shield V2.0

- **[Library]**Arduino Library for CAN-BUS Shield

- **[Datasheet]**MCP2515 datasheet

- **[Datasheet]**MCP2551 datasheet

- **[Demo]**An OBD Demo

- **[Tool]**MCP2515 Baud Rate Tool

- **[Tool]**USB-CAN Analyzer

- **[Cable]**DB9 to OBD2 Cable

- **[Drawing]**CAN BUS Shield v2.0 Drawing in PDF

- **[Drawing]**CAN BUS Shield v2.0 Drawing in DXF

Tech Support

Please submit any technical issue into our forum.