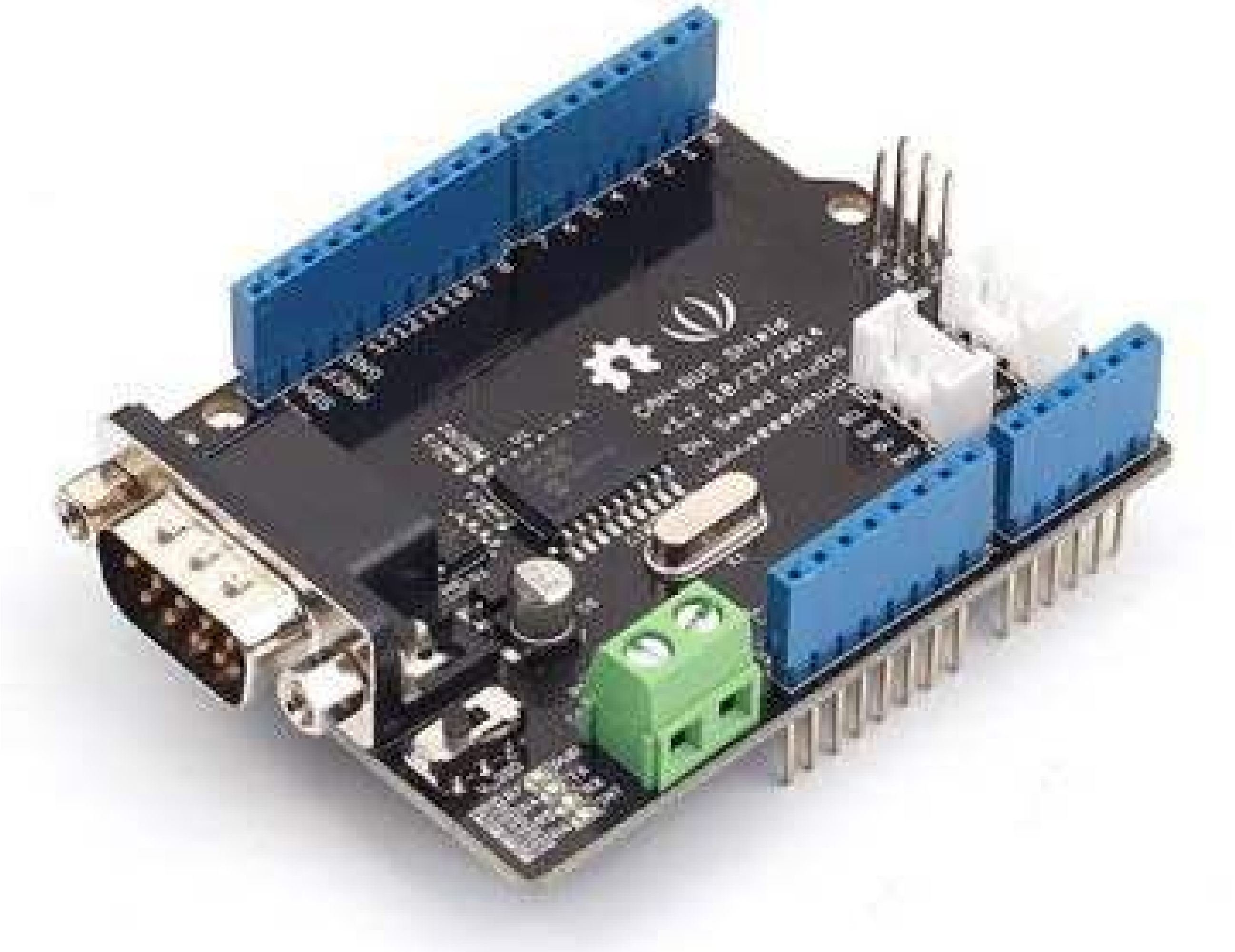


CAN-BUS Shield V1.2

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:

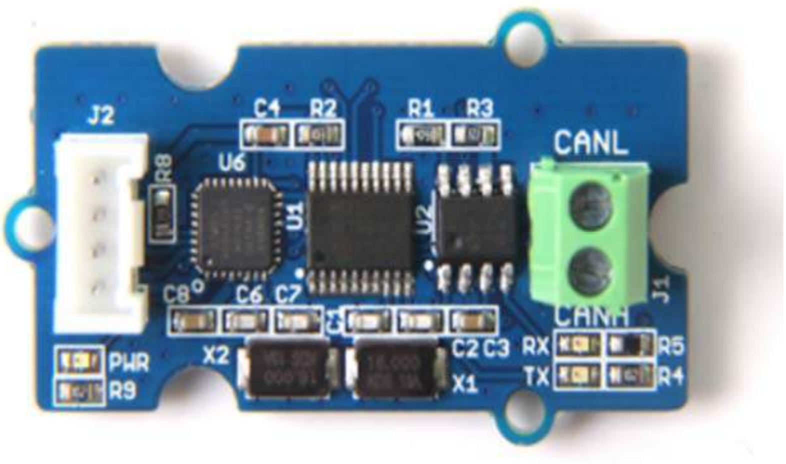
Version	Released Date	How to Buy
CAN BUS Shield V1.0	Oct 14, 2012	E.O.L
CAN BUS Shield V1.1	Aug 10, 2013	E.O.L
CAN BUS Shield V1.2	Jan 5, 2015	Get One Now
CAN BUS Shield V2.0	Aug 01,2017	Get One Now

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 cablemake 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

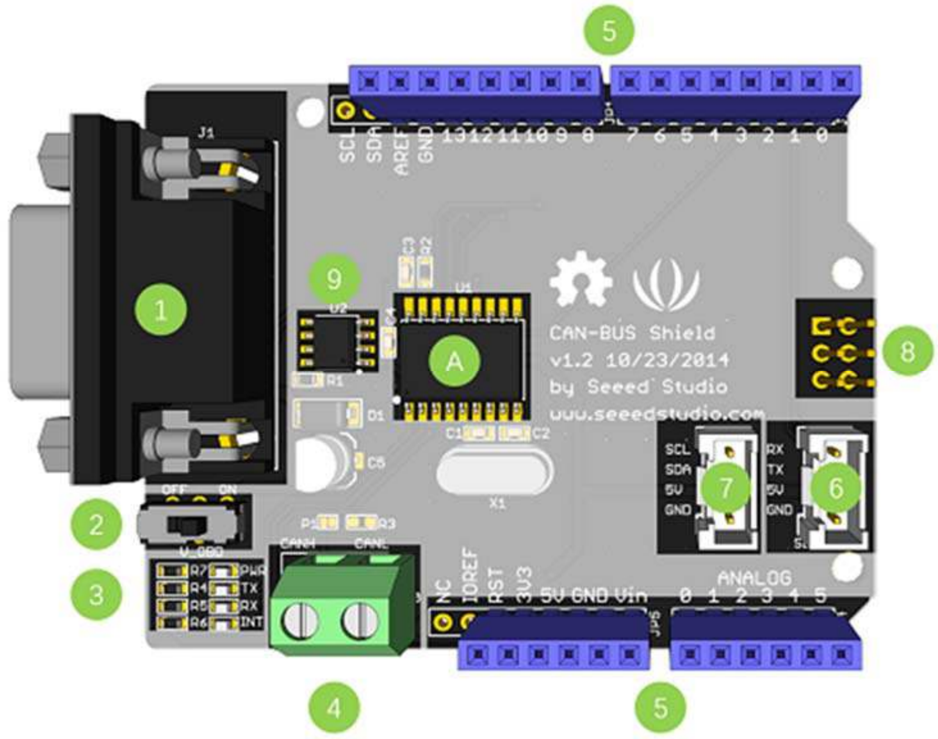
If you want a CAN Bus Analyzer to debug your CAN Bus, this USB-CAN Analyzer is recommended.

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

Note
CAN BUS Shield Work well with Arduino UNO (ATmega328), Arduino Mega (ATmega1280/2560) as well as Arduino Leonardo (ATmega32U4).

Hardware Overview

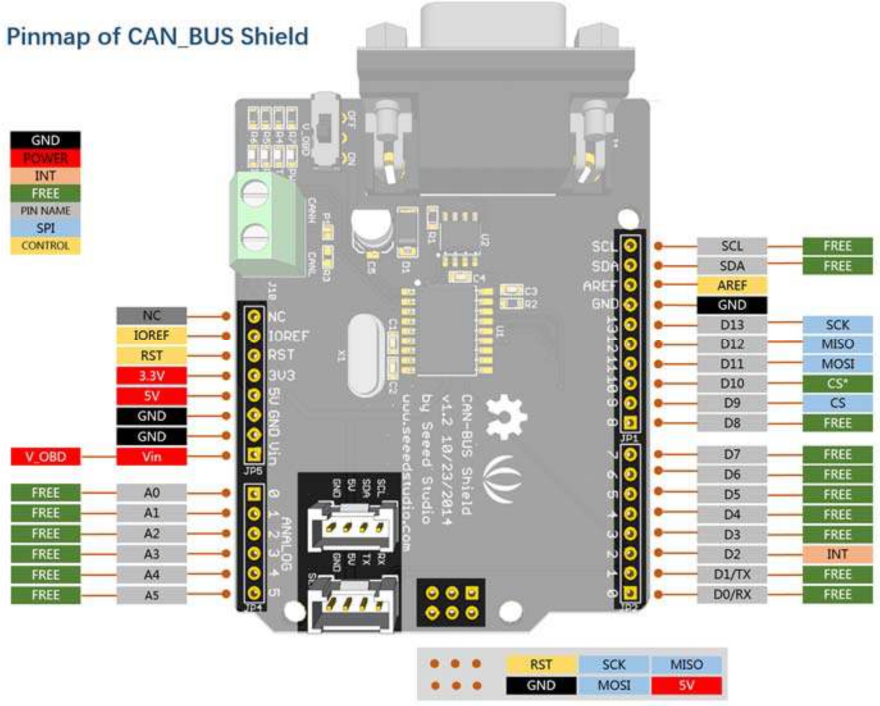


1. **DB9 Interface** - to connect to OBDII Interface via a DBG-OBD Cable.
2. **V_OBD** - It gets power from OBDII Interface (from DB9)

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)

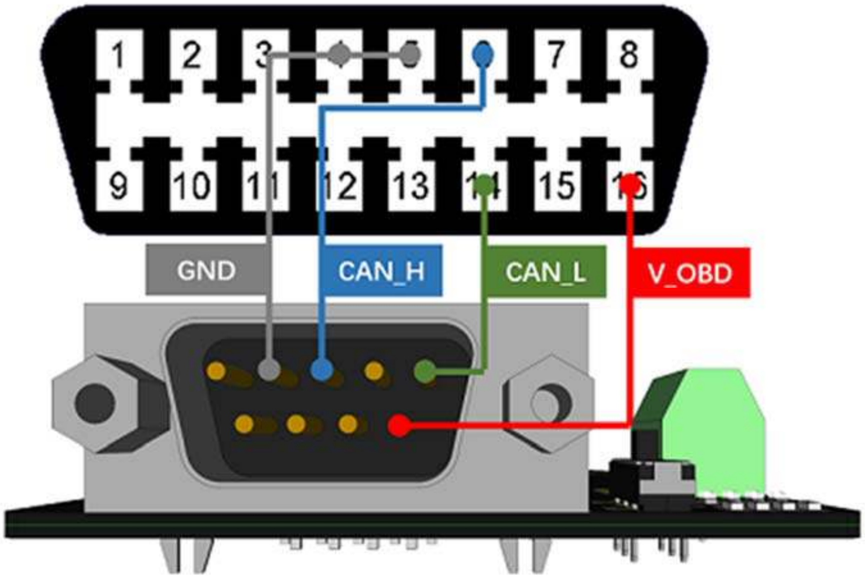
Warning
When you use more than two CAN Bus Shield in one net, you should take the impedance into consideration. You should either cut P1 in the PCB with a knife, or just remove R3 on the PCB.

Pin map



Note
The FREE pin is available for the other usages.

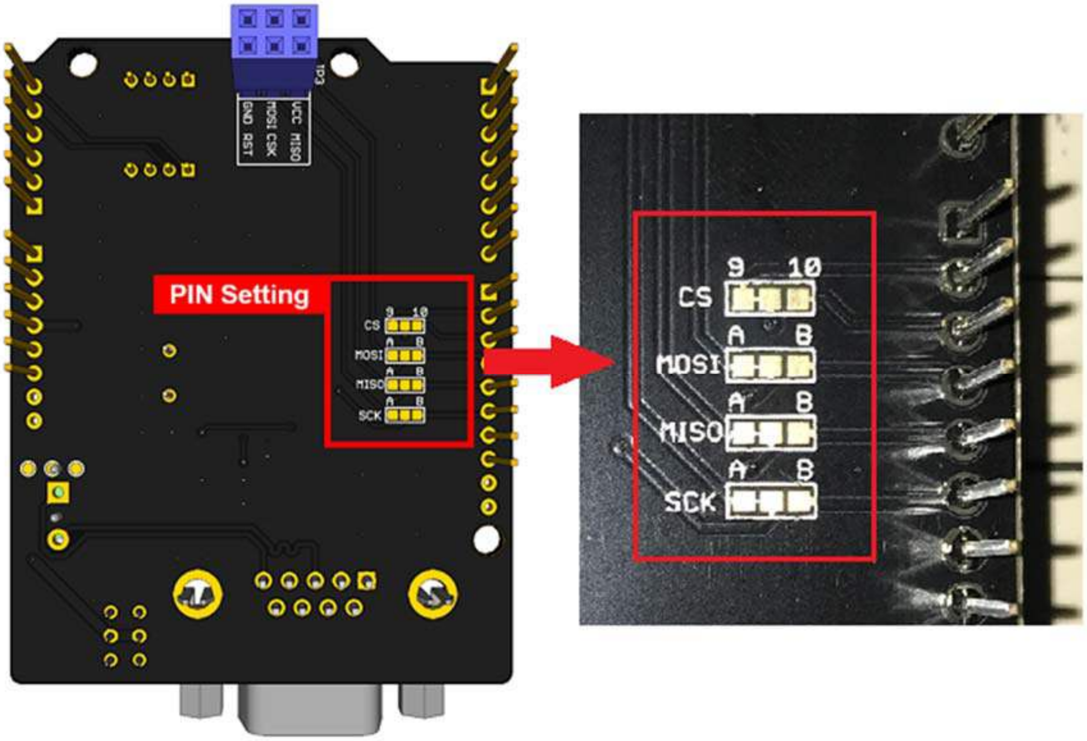
DB9&OBDii Interface



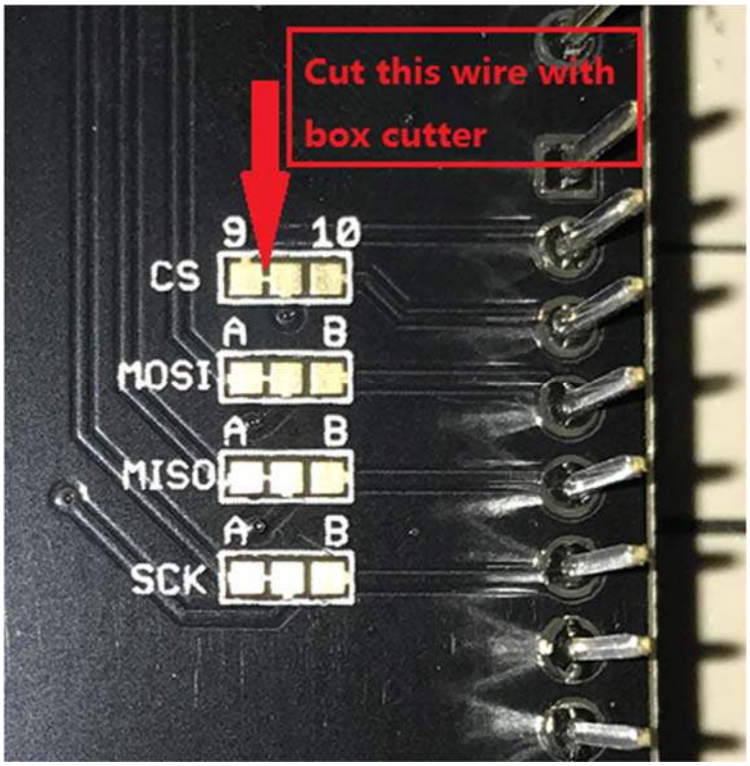
CS pin

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

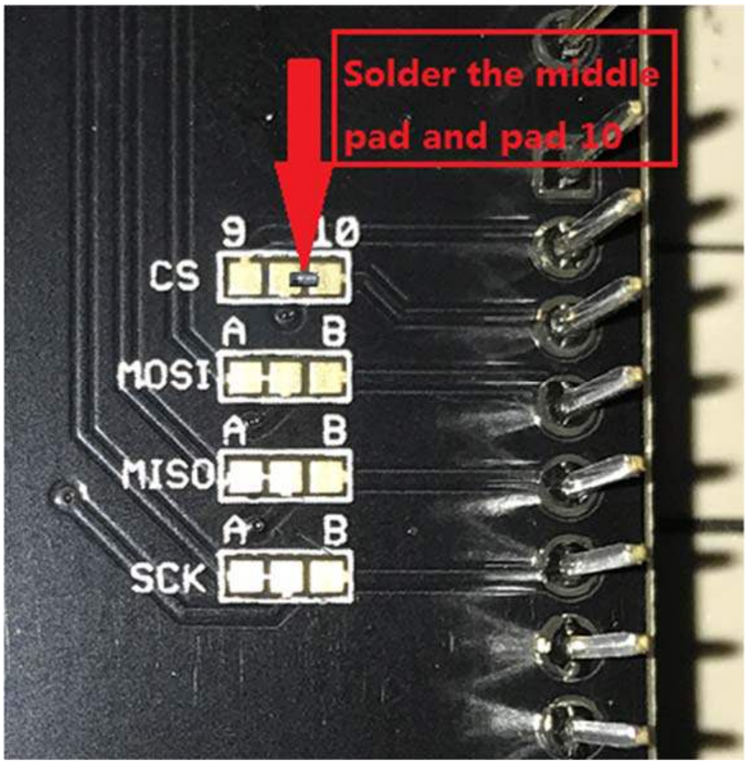
- Step1: Take a look at the backside of the PCBA, you will find a pad named CS.



- Step2: Cut the wire between pad9 and the middle pad.



- Step3: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.

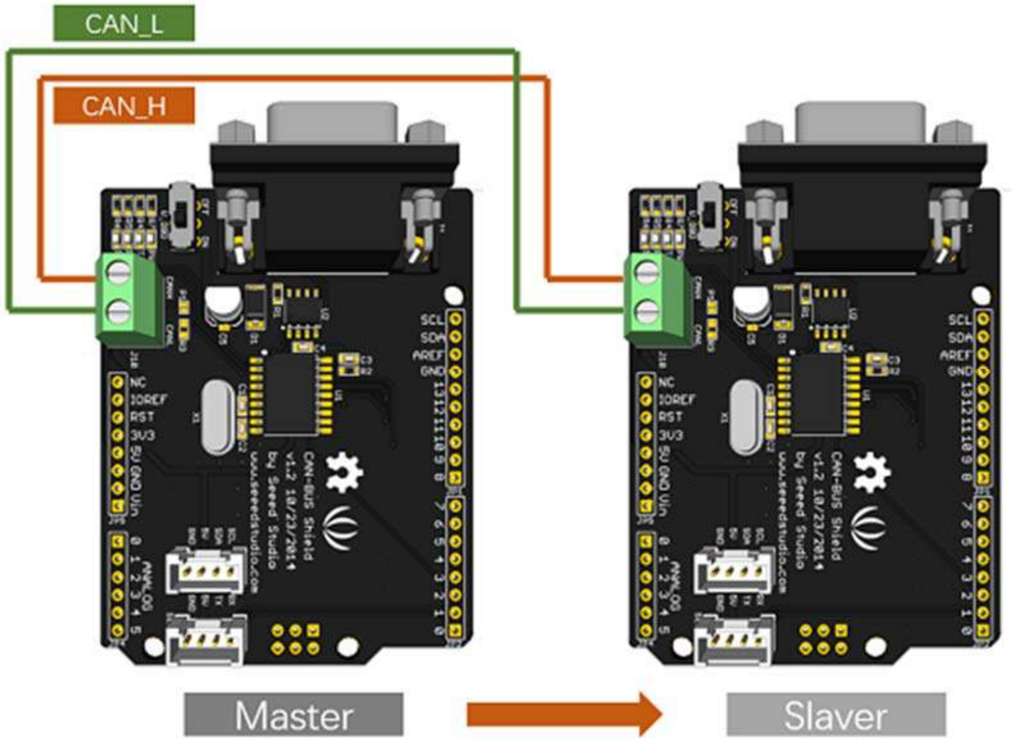
Note
This example is built under Arduino IDE version 1.6.9.

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.

Click on below button to download the library.

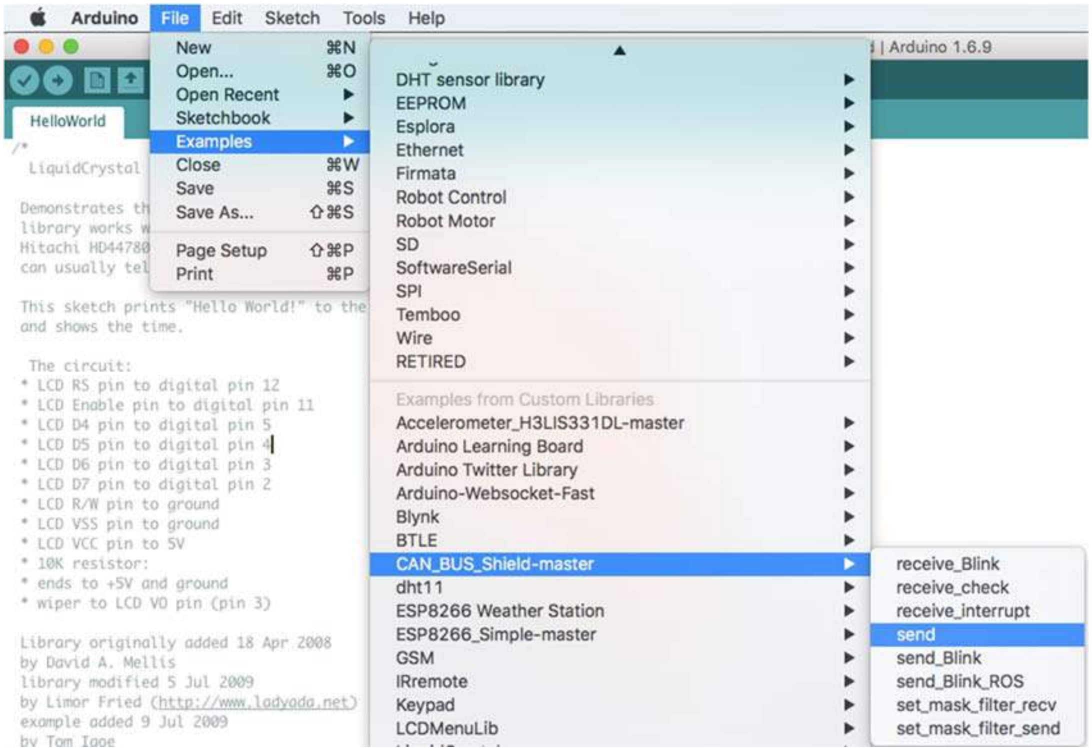
Download CAN BUS Shield Library

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.

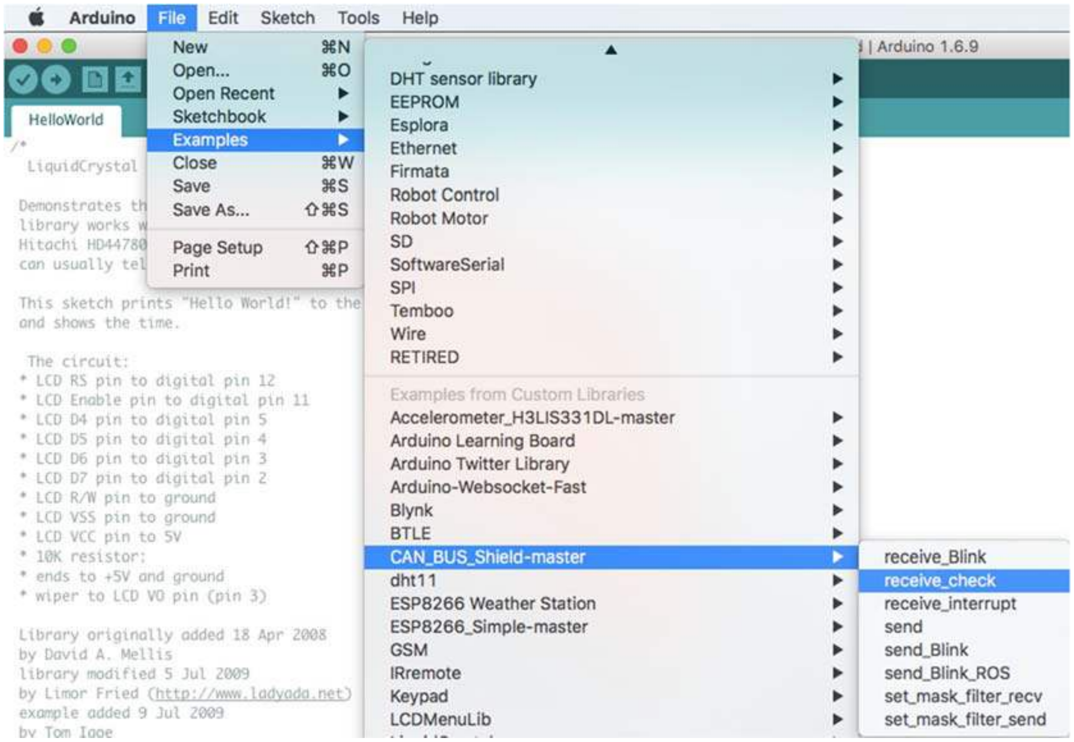
Note
Each node can act as master before the code being uploaded.

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



```
/*
  LiquidCrystal

  Demonstrates th
  library works w
  Hitachi HD44780
  can usually tel

  This sketch prints "Hello World!" to the
  and shows the time.

  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
```

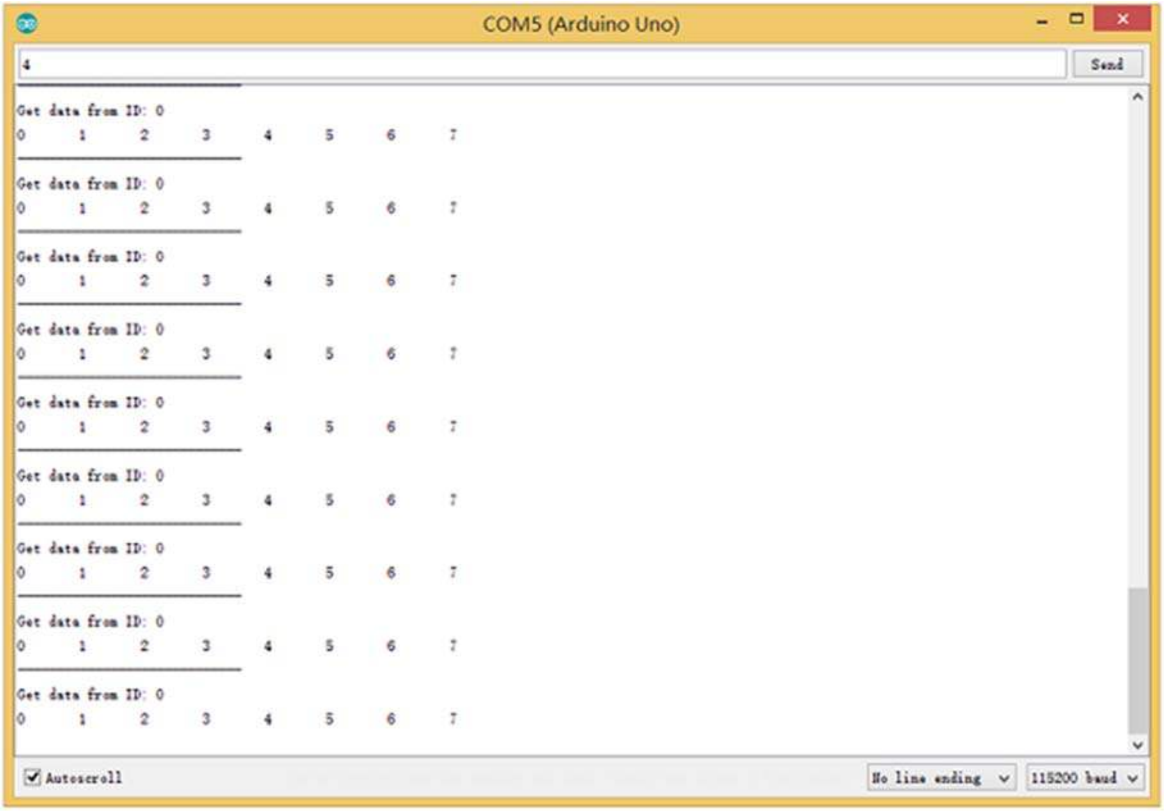
Open the **receive_check** example (**File > Examples > CAN_BUS_Shield-master > receive_check**) and upload to the **slaver**



```
/*
  LiquidCrystal

  Demonstrates th
  library works w
  Hitachi HD44780
  can usually tel

  This sketch prints "Hello World!" to the
  and shows the time.

  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
```

STEP4: View Result

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



```
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
-----------------------------
Get data from ID: 0
0       1       2       3       4       5       6       7
```

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:

```
#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
```

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:

```
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.
```

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.

Generate a New BaudRate

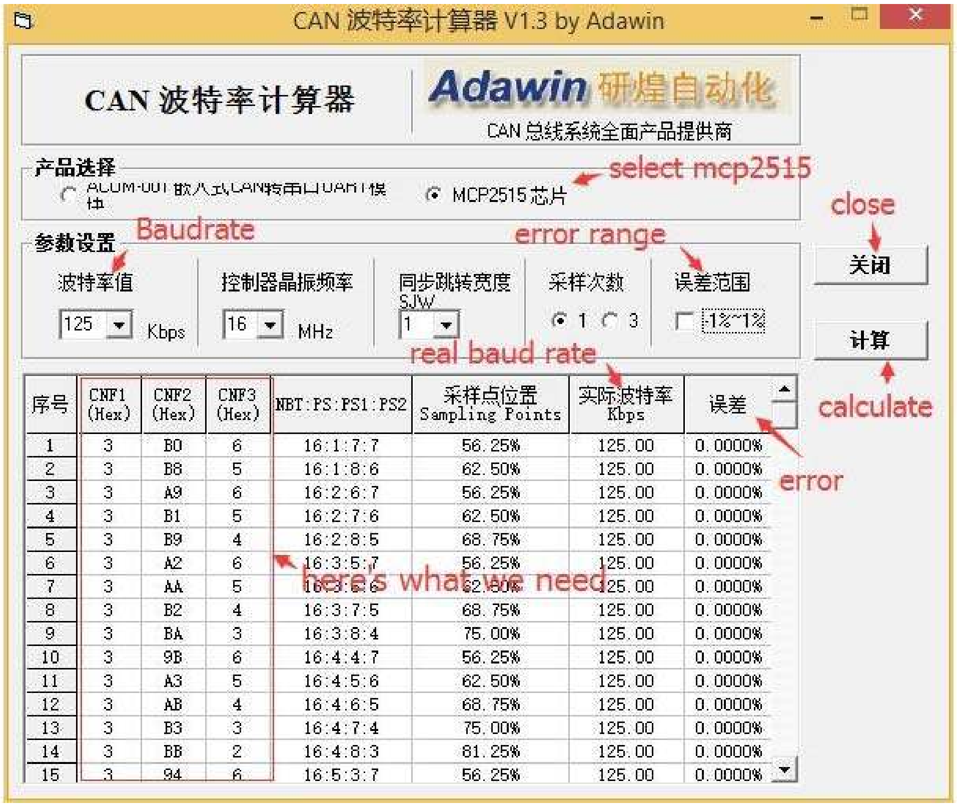
We had provided many frequently-used baud rates, as below:

```
#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 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.



CAN 波特率计算器 V1.3 by Adawin

CAN 波特率计算器

Adawin 研煌自动化

CAN 总线系统全面产品提供商

产品选择

○ ACUM-001 嵌入式CAN转串口UART模块 ● MCP2515芯片

参数设置

波特率值: 125 Kbps

控制器晶振频率: 16 MHz

同步跳转宽度 SJW: 1

采样次数: ● 1 ○ 3

误差范围: □ -1%~1%

序号	CNF1 (Hex)	CNF2 (Hex)	CNF3 (Hex)	NBT:PS:PS1:PS2	采样点位置 Sampling Points	实际波特率 Kbps	误差
1	3	B0	6	16:1:7:7	56.25%	125.00	0.0000%
2	3	B8	5	16:1:8:6	62.50%	125.00	0.0000%
3	3	A9	6	16:2:6:7	56.25%	125.00	0.0000%
4	3	B1	5	16:2:7:6	62.50%	125.00	0.0000%
5	3	B9	4	16:2:8:5	68.75%	125.00	0.0000%
6	3	A2	6	16:3:5:7	56.25%	125.00	0.0000%
7	3	AA	5	16:3:6:6	62.50%	125.00	0.0000%
8	3	B2	4	16:3:7:5	68.75%	125.00	0.0000%
9	3	BA	3	16:3:8:4	75.00%	125.00	0.0000%
10	3	9B	6	16:4:4:7	56.25%	125.00	0.0000%
11	3	A3	5	16:4:5:6	62.50%	125.00	0.0000%
12	3	AB	4	16:4:6:5	68.75%	125.00	0.0000%
13	3	B3	3	16:4:7:4	75.00%	125.00	0.0000%
14	3	BB	2	16:4:8:3	81.25%	125.00	0.0000%
15	3	94	6	16:5:3:7	56.25%	125.00	0.0000%

Note
This software supports Windows system only. If you can't open it, please feel free to contact loovee@seeed.ccfor support.

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;
```

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.

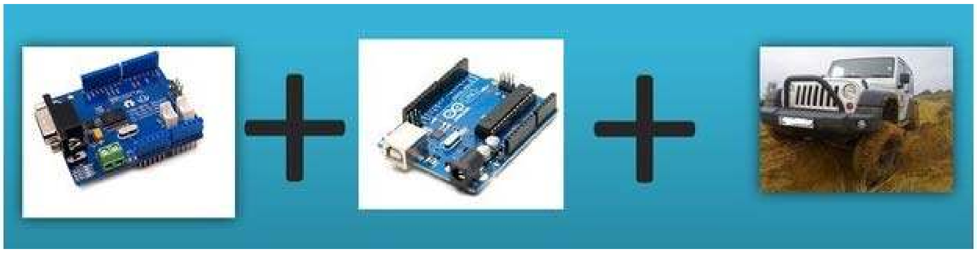
Projects

If you want to make some awesome projects with CAN-BUS shield, here are some projects for reference.**Volkswagen CAN BUS Gaming**

Ever wanted to play a car/truck simulator with a real dashboard on your PC? Me too! I'm trying to control a VW Polo 6R dashboard via CAN Bus with an Arduino Uno and a Seeed CAN Bus Shield. Inspired by Silas Parker. Thanks Sepp and Is0-Mick for their great support!

Make it NOW

Hack your vehicle CAN-BUS



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.

All electronic functions are connected to the TIPM, (Totally integrated Power Module), such as solenoids/relays to lock the doors or mini motors to wind the windows etc.

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.

Make it NOW

- You can post a question to Seeed Forum or send an email to **techsupport@seeed.cc**.

Resources

- **【PDF】** CAN-BUS Shield V1.2 Schmatics
- **【Eagle】** Schematic of CAN-BUS Shield V1.2
- **【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

Tech Support

Please submit any technical issue into our forum