MAN CAN BUS

Getting Started

Getting Started

What is CAN bus?

Controller Area Network (CAN) is a serial communication protocol that lets microcontrollers and devices talk to each other without a central host computer. Developed by Bosch in the 1980s, it is now the standard in automotive and heavy vehicle electronics.

Why buses use it

In a **MAN Lion's City**, dozens of ECUs (Electronic Control Units) share data in real time: the engine tells the instrument cluster the RPM, the EBS reports brake air pressure, the tachograph transmits vehicle speed — all over **two wires** (CAN_H and CAN_L).

The J1939 standard

Heavy vehicles (buses, trucks) use **SAE J1939**, an application layer built on top of CAN. It defines:

Element	Description
29-bit ID	Extended identifier (vs 11-bit basic CAN)
PGN	Parameter Group Number — identifies the data type
SA	Source Address — address of the sending ECU (0x00–0xFE)
SPN	Suspect Parameter Number — each signal within a PGN

Implementation parameters

Parameter	Value
Bus speed	250 kbps
ID type	Extended (29-bit)
Send cycle	20 ms (50 Hz)
Transceiver	MCP2515 + TJA1050
MCP2515 oscillator	8 MHz

Note

`0xFF` bytes in unused positions are the J1939 default for "parameter not available" (SNA — Specific Not Available).

Hardware setup

Compatible boards

Any Arduino with SPI support works. The CS pin and SPI pins vary by board:

Board	SCK	MOSI	MISO	CS (recommended)	Notes
Arduino Mega 2560	52	51	50	53	Hardware SPI, most headroom
Arduino Uno	13	11	12	10	5V, limited RAM
Arduino Nano	13	11	12	10	Same as Uno, compact
Arduino Micro	15	16	14	10	5V, USB HID capable
Arduino Leonardo	15	16	14	10	Same as Micro
ESP32	18	23	19	5	3.3V — use level shifter

For ESP32, the MCP2515 runs at 3.3V — do **not** connect directly to a 5V MCP2515 module without a level shifter.

Required components

Component	Recommended model
Microcontroller	Any board from the table above
CAN module	MCP2515 + TJA1050 breakout board
Bus terminators	2× 120 Ω resistor
Wiring	Twisted pair for CAN_H / CAN_L

Wiring — Arduino Mega

Arduino Mega	MCP2515 module
5V	VCC

Arduino Mega	MCP2515 module
GND	GND
pin 52	SCK
pin 51	MOSI
pin 50	MISO
pin 53	CS
pin 2	INT (optional)

Wiring — Arduino Uno / Nano

Arduino Uno/Nano	MCP2515 module
5V	VCC
GND	GND
pin 13	SCK
pin 11	MOSI
pin 12	MISO
pin 10	CS
pin 2	INT (optional)

Initialization code

Change `SPI_CS_PIN` to match your board:

```
#include <SPI.h>
#include <mcp_can.h>

const int SPI_CS_PIN = 10;  // 53 for Mega, 10 for Uno/Nano
MCP_CAN CAN(SPI_CS_PIN);

void setup() {
    while (CAN_OK != CAN.begin(MCP_ANY, CAN_250KBPS, MCP_8MHZ)) {
        delay(100);
    }
    CAN.setMode(MCP_NORMAL);
}
```

MCP2515 oscillator

Most MCP2515 modules come with an **8 MHz** crystal. Some come with **16 MHz** — check the crystal printed on your module and change the parameter accordingly:

Crystal	Parameter
8 MHz	`MCP_8MHZ`
16 MHz	`MCP_16MHZ`

Installing the library

Install **mcp_can** by Cory Fowler from the Arduino IDE library manager, or clone directly:

```
cd ~/Documents/Arduino/libraries
git clone https://github.com/coryjfowler/MCP_CAN_lib
```

Important

Always place a **120 Ω resistor between CAN_H and CAN_L at each end** of the bus. Without terminators, signal reflections will make the bus unstable.

Message Overview

This implementation sends **6 messages** every 20 ms (50 Hz cycle). All use 29-bit extended IDs and 8 data bytes, conforming to SAE J1939.

Message table

ID (hex)	Name	PGN	Variable	Active bytes
`0x0CF00400`	EEC1 — Engine RPM	61444	`rpm`	3–4
`0x0CFE6CEE`	TCO1 — Vehicle speed	65132	`vel`	6–7
`0x18FEEE00`	ET1 — Engine temperature	65262	`temp`	0
`0x18FEFC21`	ZBR — Throttle	65276	`gas`	1
`0x18FEAE0B`	EBS — Air pressure	65198	`aire1`, `aire2`	2–3
`0x18EEFF21`	Heartbeat	—	—	all `0x00`

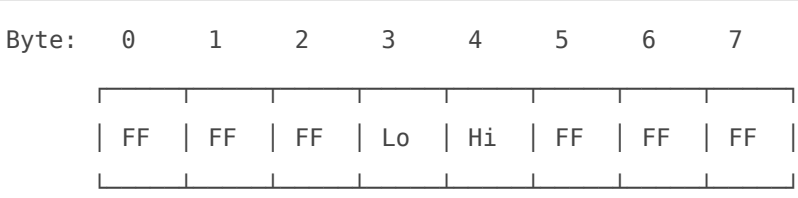
29-bit ID structure

Every J1939 message ID encodes three fields:

Bits	Field	Description
28–26	Priority	0 (highest) to 7 (lowest)
25–8	PGN	Identifies the data type
7–0	SA	Source address — which ECU is sending

Byte convention

Unused bytes are always set to `0xFF`, which in J1939 means **parameter not available** (SNA — Specific Not Available).



```
                          ↑     ↑
                    Actual data (little-endian)
```

Send cycle

All 6 messages are sent sequentially every 20 ms:

```
void loop() {
    // send all messages
    // ...
    delay(20);  // 50 Hz
}
```

Source addresses in use

SA (hex)	Decimal	ECU
`0x00`	0	Engine ECU (RPM, temperature)
`0xEE`	238	Tachograph ECU (speed)
`0x21`	33	Simulator node (throttle, heartbeat)
`0x0B`	11	EBS brake ECU (air pressure)