[**https://github.com/sandeepmistry/arduino-CAN/blob/master/API.md**](https://github.com/sandeepmistry/arduino-CAN/blob/master/API.md)

**CAN API**

**Include Library**

#include <CAN.h>

**Setup**

**Begin**

Initialize the library with the specified bit rate.

CAN.begin(bitrate);

* bitrate - bit rate in bits per seconds (bps) (1000E3, 500E3, 250E3, 200E3, 125E3, 100E3, 80E3, 50E3, 40E3, 20E3, 10E3, 5E3)

Returns 1 on success, 0 on failure.

**Set pins**

**MCP2515**

Override the default CS and INT pins used by the library. **Must** be called before CAN.begin(...).

**CAN.setPins(cs, irq); nálam CS=5, INT=4**

* cs - new chip select pin to use, defaults to 10
* irq - new INT pin to use, defaults to 2. **Must** be interrupt capable via [attachInterrupt(...)](https://www.arduino.cc/en/Reference/AttachInterrupt).

This call is optional and only needs to be used if you need to change the default pins used.

**ESP32 nálam RX 16 TX 17**

Override the default CTX and CRX pins used by the library. **Must** be called before CAN.begin(...).

CAN.setPins(rx, tx);

* rx - new CRX pin to use, defaults to 4
* tx - new CTX pin to use, defaults to 5.

This call is optional and only needs to be used if you need to change the default pins used.

**Set SPI Frequency**

**MCP2515 only**

Override the default SPI frequency of 10 MHz used by the library. **Must** be called before CAN.begin(...).

CAN.setSPIFrequency(frequency);

* frequency - new SPI frequency to use, defaults to 10E6

This call is optional and only needs to be used if you need to change the default SPI frequency used. Some logic level converters cannot support high speeds such as 10 MHz, so a lower SPI frequency can be selected with CAN.setSPIFrequency(frequency).

**Set Clock Frequency**

**MCP2515 only**

Override the default clock source frequency that is connected to the MCP2515. **Must** be called before CAN.begin(...).

CAN.setClockFrequency(clockFrequency);

* clockFrequency - new clock frequency to use (8E6, 16E6) connected to MCP2515, defaults to 16 Mhz

This call is optional and only needs to be used if you need to change the clock source frequency connected to the MCP2515. Most shields have a 16 MHz clock source on board, some breakout boards have a 8 MHz source.

**End**

Stop the library

CAN.end()

**Sending data**

**Begin packet**

Start the sequence of sending a packet.

CAN.beginPacket(id);

**CAN.beginPacket(id, dlc);**

CAN.beginPacket(id, dlc, rtr);

CAN.beginExtendedPacket(id);

CAN.beginExtendedPacket(id, dlc);

CAN.beginExtendedPacket(id, dlc, rtr);

* id - 11-bit id (standard packet) or 29-bit packet id (extended packet) **itt még gond van**
* dlc - (optional) value of Data Length Code (DLC) field of packet, default is size of data written in packet **legyen 8**
* rtr - (optional) value of Remote Transmission Request (RTR) field of packet (false or true), defaults to false. RTR packets contain no data, the DLC field of the packet represents the requested length.

Returns 1 on success, 0 on failure.

**Writing**

Write data to the packet. **Each packet can contain up to 8 bytes.**

CAN.write(byte);

**CAN.write(buffer, length);**

* byte - single byte to write to packet

or

* **buffer - data to write to packet**
* length - size of data to write

Returns the number of bytes written.

**Note:** Other Arduino Print API's can also be used to write data into the packet

**End packet**

End the sequence of sending a packet.

**CAN.endPacket()**

Returns 1 on success, 0 on failure.

**Receiving data**

**Parsing packet**

Check if a packet has been received.

int packetSize = **CAN.parsePacket();**

Returns the packet size in bytes or 0 if no packet was received. For RTR packets the size reflects the DLC field of the packet.

**Register callback**

Register a callback function for when a packet is received.

**CAN.onReceive(onReceive);**

void onReceive(int packetSize) {

// ...

}

* onReceive - function to call when a packet is received.

**Packet ID**

long id = **CAN.packetId();**

Returns the id (11-bit or 29 bit) of the received packet. Standard packets have an 11-bit id, extended packets have an 29-bit id.

**Packet Extended**

bool extended = CAN.packetExtended();

Returns true if the received packet is extended, false otherwise.

**Packet RTR**

bool rtr = CAN.packetRtr();

Returns the value of the Remote Transmission Request (RTR) field of the packet true/false. RTR packets contain no data, the DLC field is the requested data length.

**Packet DLC**

int DLC = **CAN.packetDlc();**

Returns the value of the Data Length Code (DLC) field of the packet.

**Available**

int availableBytes = **CAN.available()**

Returns number of bytes available for reading.

**Peeking**

Peek at the next byte in the packet.

int b = CAN.peek();

Returns the next byte in the packet or -1 if no bytes are available.

**Reading**

Read the next byte from the packet.

int b = **CAN.read();**

Returns the next byte in the packet or -1 if no bytes are available.

**Note:** Other Arduino [Stream API's](https://www.arduino.cc/en/Reference/Stream) can also be used to read data from the packet

**Filtering**

Filter packets that meet the desired criteria.

**CAN.filter(id);**

CAN.filter(id, mask);

CAN.filterExtended(id);

CAN.filterExtended(id, mask);

* id - 11-bit id (standard packet) or 29-bit packet id (extended packet)
* mask - (optional) 11-bit mask (standard packet) or 29-bit mask (extended packet), defaults to 0x7ff or 0x1fffffff (extended)

Only packets that meet the following criteria are acknowleged and received, other packets are ignored:

if ((packetId & mask) == id) {

// acknowleged and received

} else {

// ignored

}

Returns 1 on success, 0 on failure.

**Other modes**

**Loopback mode**

Put the CAN controller in loopback mode, any outgoing packets will also be received.

CAN.loopback();

**Sleep mode**

Put the CAN contoller in sleep mode.

**CAN.sleep();**

Wake up the CAN contoller if it was previously in sleep mode.

**CAN.wakeup();**