**EVOMIN transport protocol**

**Procedure**

1. Enable transport layer (low-level), i.e. I2C, SPI

2. Use evoMin\_Init() to initialize a passed instance of evoMin\_Interface

3. evoMin\_RXHandler() needs to be called from within the low-level transport routine (i.e IRQHandler for SPI/I2C/UART) with the received byte (if you want to receive data, otherwise just skip this method)

4. Whenever a new, valid frame is received, the evoMin\_Handler\_FrameRecvd() callback gets called from evoMIN

5. To send data, one must implement a TX callback method (low-level implementation for the send transport layer, i.e. I2C, SPI) and assign the implemented callback to the interface by using evoMin\_SetTXHandler()

If the device is receive-only, you can disable sending by compiling with the EVOMIN\_TX\_DISABLE statement.

To eventually send a frame, use the evoMin\_sendFrame() method

**Buffer status management**

The buffer of each individual frame can be checked against overflows etc. Therefor, use the EVOMIN\_BUF\_STATUS\_MASK\_xxx masks.

**CRC8**

The checksum CRC8 is only calculated over the payload bytes EXCLUDING the stuff bytes and also EXCLUDING the frame header bytes, but INCLUDING the command and the payload length byte!

|  |  |
| --- | --- |
| 0 | command byte |
| 1 | payload length byte |
| 2..n | payload bytes |

**Callbacks / Application-dependent handlers**

**(see evomin.h)**

*/\* -- Custom handlers, must be implemented on the application side -- \*/  
  
/\* evoMin\_RXHandler must be called through the low-level byte receive method, i.e. the SPI RX IRQHandler  
 whenever a byte is received on the line \*/***void** evoMin\_RXHandler(**struct** evoMin\_Interface\* interface, uint8\_t cByte);  
  
*/\* Application dependent CRC8 calculation, must be implemented! \*/*uint8\_t evoMin\_CRC8(uint8\_t\* bytes, uint32\_t bLen);  
  
*/\* evoMin\_Handler\_FrameRecvd callback gets called by evoMIN whenever a new, valid frame has been received  
 It contains a pointer to the received frame, including the command, it's payload length and the payload itself  
 in a buffer \*/***void** evoMin\_Handler\_FrameRecvd(**struct** evoMin\_Frame\* frame);

**You must implement above listed handlers / callcacks.**

**Use case**

Senden von Daten:

*/\* Test sending of a frame \*/*

**const** uint32\_t sendBufferLen = 9;

uint8\_t sendBuffer[] = {

0xAA,

0xAA,

0xAA,

0xBB,

0xAA,

0xAA,

0xAA,

0xCC,

0xDD

};

evoMin\_sendFrame(&comInterface, ***EVOMIN\_CMD\_CHIP***, sendBuffer, sendBufferLen);

Resultiert in:

Send byte: 170 (AA)  
Send byte: 170 (AA)  
Send byte: 170 (AA)  
Send byte: 15 (F)  
Send byte: 9 (9)  
Send byte: 170 (AA)  
Send byte: 170 (AA)  
Send byte: 85 (55)  
Send byte: 170 (AA)  
Send byte: 187 (BB)  
Send byte: 170 (AA)  
Send byte: 170 (AA)  
Send byte: 85 (55)  
Send byte: 170 (AA)  
Send byte: 204 (CC)  
Send byte: 221 (DD)  
Send byte: 141 (8D)  
Send byte: 85 (55)

Jedes Byte verursacht einen Call des

evoMin\_comTXImplementation

Callbacks.

Was man hier schön sieht, ist das automatische Einfügen eines Stopfbytes (0x55 = 85) nach zwei aufeinanderfolgenden 0xAA Bytes innerhalb der Payload.

**Test software (Python)**

You can use the following script to generate test data to be send or received via evoMIN.

from random import randint

n = 20

j\_data = ''

for i in range(0, n):

payload\_length = 4

payload = ''

for r in range(0, payload\_length):

payload += '{0},'.format(randint(0x00, 0xFE))

payload = payload[:-1]

msg = '''uint8\_t testData{0}[] = {1}0xAA,0xAA,0xAA,{5},{2},{3},0xCC,0x55{4};'''.format(i,'{', payload\_length, payload, '}', i)

j\_data += 'testData{0},'.format(i)

# Print the single rows

print(msg)

j\_data = j\_data[:-1]

jagged\_array = 'uint8\_t\* testData[] = {0} {1} {2};'.format('{', j\_data, '}')

# Print the jagged array

print(jagged\_array)