# diyBMSv4ESP32 - Developer Guide (English & German)

## Open-Source Development & Developer Information (English)

This implementation was developed as part of an open-source project.

Developer: Trajilovic Goran

This document describes the new functions integrated into the original diyBMSv4ESP32 project. The goal is to provide developers with a complete guide for implementing these improvements.

## Open-Source Development & Developer Information (Deutsch)

Diese Implementierung wurde im Rahmen eines Open-Source-Projekts entwickelt.

Entwickler: Trajilovic Goran

Dieses Dokument beschreibt die neuen Funktionen, die in das ursprüngliche diyBMSv4ESP32-Projekt integriert wurden. Ziel ist es, Entwicklern eine vollständige Anleitung zur Implementierung dieser Verbesserungen zu bieten.

## RS485 Communication Overview (English)

RS485 communication enables robust data exchange between the BMS controller and external devices, such as a current monitor. This implementation ensures that only the master device sends data while all slave devices listen.

## RS485 Communication Overview (Deutsch)

Die RS485-Kommunikation ermöglicht einen zuverlässigen Datenaustausch zwischen dem BMS-Controller und externen Geräten wie einem Current Monitor. Diese Implementierung stellt sicher, dass nur das Master-Gerät Daten sendet, während alle Slave-Geräte zuhören.

## CAN-Bus Communication with PylonTech (English)

The CAN-Bus system allows communication between diyBMSv4ESP32 and PylonTech batteries. This implementation enables:

- Transmission of battery data (SoC, voltage, temperature)
- Receiving control commands for the BMS
- Synchronizing charging operations

## CAN-Bus Communication with PylonTech (Deutsch)

Das CAN-Bus-System ermöglicht die Kommunikation zwischen diyBMSv4ESP32 und PylonTech-Batterien. Diese Implementierung umfasst:

- Übertragung von Batteriedaten (SoC, Spannung, Temperatur)
- Empfang von Steuerbefehlen für das BMS
- Synchronisation von Ladevorgängen

## bms_id_manager.h - ID Management (English)

The file `bms_id_manager.h` manages the unique identification numbers (IDs) of the BMS modules. This ensures that each module is correctly recognized within the network and that the master-slave transition works reliably.

## bms_id_manager.h - ID Management (Deutsch)

Die Datei `bms_id_manager.h` verwaltet die eindeutigen Identifikationsnummern (IDs) der BMS-Module. Dies stellt sicher, dass jedes Modul im Netzwerk korrekt erkannt wird und der Master-Slave-Wechsel zuverlässig funktioniert.

## Integration into the Existing Codebase (English)

To integrate these functions into the original diyBMSv4ESP32 project, follow these steps:

1. Add `bms_id_manager.h` and initialize the ID manager.
2. Include `pylon_canbus.h` and `pylon_canbus.cpp` into the CAN-Bus system.
3. Use `HardwareSerialManager` for RS485 communication.
4. Ensure that all functions are correctly called within `setup()` and `loop()`.

## Integration into the Existing Codebase (Deutsch)

Um diese Funktionen in das ursprüngliche diyBMSv4ESP32-Projekt zu integrieren, folgen Sie diesen Schritten:

1. Fügen Sie `bms_id_manager.h` hinzu und initialisieren Sie den ID-Manager.
2. Binden Sie `pylon_canbus.h` und `pylon_canbus.cpp` in das CAN-Bus-System ein.
3. Nutzen Sie `HardwareSerialManager` für die RS485-Kommunikation.
4. Stellen Sie sicher, dass alle Funktionen korrekt innerhalb von `setup()` und `loop()` aufgerufen werden.

## Open-Source Contribution & Further Development (English)

This project is open-source, and all developers are welcome to contribute. Changes and new features can be submitted via GitHub.

License: MIT License (or the original project's license)

## Open-Source Contribution & Further Development (Deutsch)

Dieses Projekt ist Open-Source, und alle Entwickler sind eingeladen, beizutragen. Änderungen und neue Funktionen können über GitHub bereitgestellt werden.

Lizenz: MIT-Lizenz (oder die Lizenz des Originalprojekts)

## Full Source Code: HardwareSerial.h

```
#ifndef HARDWARE_SERIAL_H
#define HARDWARE_SERIAL_H

#include <Arduino.h>

#define RS485_DE_PIN 16
#define RS485_RE_PIN 17

class HardwareSerialManager {
public:
    HardwareSerialManager();
    void begin(uint32_t baudRate = 9600);
    void setRS485Mode(bool master);
    void sendData(const uint8_t* data, size_t length);
    size_t receiveData(uint8_t* buffer, size_t length);
};

#endif // HARDWARE_SERIAL_H
```

## Full Source Code: HardwareSerial.cpp

```
#include "HardwareSerial.h"

HardwareSerial rs485Serial(2);

HardwareSerialManager::HardwareSerialManager() {}

void HardwareSerialManager::begin(uint32_t baudRate) {
    pinMode(RS485_DE_PIN, OUTPUT);
    pinMode(RS485_RE_PIN, OUTPUT);
    setRS485Mode(false);
    rs485Serial.begin(baudRate, SERIAL_8N1, RS485_DE_PIN, RS485_RE_PIN);
}
```

```
void HardwareSerialManager::setRS485Mode(bool master) {
    if (master) {
        digitalWrite(RS485_DE_PIN, HIGH);
        digitalWrite(RS485_RE_PIN, HIGH);
    } else {
        digitalWrite(RS485_DE_PIN, LOW);
        digitalWrite(RS485_RE_PIN, LOW);
    }
}

void HardwareSerialManager::sendData(const uint8_t* data, size_t length) {
    setRS485Mode(true);
    delay(5);

    rs485Serial.write(data, length);
    rs485Serial.flush();

    delay(5);
    setRS485Mode(false);
}

size_t HardwareSerialManager::receiveData(uint8_t* buffer, size_t length) {
    setRS485Mode(false);

    if (rs485Serial.available()) {
        return rs485Serial.readBytes(buffer, length);
    }
    return 0;
}
```

## Full Source Code: pylon_canbus.h

```
#ifndef PYLON_CANBUS_H
#define PYLON_CANBUS_H

#include <Arduino.h>
#include <CAN.h>

void sendDeviceIdentification();
void send_module_status();
void send_corrected_charge_current();
void pylon_message_351();
void pylon_message_355();
void pylon_message_356();
void pylon_message_35e();

#endif // PYLON_CANBUS_H
```

## Full Source Code: pylon_canbus.cpp

```
#include "pylon_canbus.h"
```

```
void sendDeviceIdentification() {
    uint8_t data[8] = {'P', 'Y', 'L', 'O', 'N', 'T', 'E', 'C'};
    send_canbus_message(0x305, data, sizeof(data));
}

void send_module_status() {
    uint8_t data[8] = {0};
    for (uint8_t i = 0; i < totalSlaves; i++) {
        data[0] = i + 1;
        data[1] = (lowest_cell_voltage_per_module[i] >> 8) & 0xFF;
        data[2] = lowest_cell_voltage_per_module[i] & 0xFF;
        data[3] = (highest_cell_voltage_per_module[i] >> 8) & 0xFF;
        data[4] = highest_cell_voltage_per_module[i] & 0xFF;
        data[5] = module_charging_allowed[i] ? 0x01 : 0x00;
        send_canbus_message(0x381 + i, data, sizeof(data));
    }
}

void send_corrected_charge_current() {
    int16_t corrected_charge_current = currentMonitor.modbus.current - 10;
    uint8_t data[8] = {(corrected_charge_current >> 8) & 0xFF, corrected_charge_current & 0xFF};
    send_canbus_message(0x386, data, sizeof(data));
}

void pylon_message_351() {
    uint8_t data[8] = {0};
    send_canbus_message(0x351, data, sizeof(data));
}

void pylon_message_355() {
    uint8_t data[8] = {0};
    send_canbus_message(0x355, data, sizeof(data));
}

void pylon_message_356() {
    uint8_t data[8] = {0};
    send_canbus_message(0x356, data, sizeof(data));
}

void pylon_message_35e() {
    uint8_t data[8] = {0x50, 0x59, 0x4c, 0x4f, 0x4e, 0x20, 0x20, 0x20};
    send_canbus_message(0x35e, data, sizeof(data));
}
```

## Full Source Code: bms_id_manager.h

```
#ifndef BMS_ID_MANAGER_H
#define BMS_ID_MANAGER_H

#include <Arduino.h>

class BMS_ID_Manager {
public:
```

```
    BMS_ID_Manager();
    uint8_t getMasterID();
    void assignID(uint8_t moduleID);
    bool isMaster();
};

#endif // BMS_ID_MANAGER_H
```