

Motordaten

V 2.4

Erzeugt von Doxygen 1.13.2

| | |
|---|-----------|
| 1 MotorData NMEA2000 | 1 |
| 1.1 Description | 1 |
| 1.2 Based on the work of | 1 |
| 1.3 Website | 2 |
| 1.4 Plotter | 2 |
| 1.5 Wiring diagram | 2 |
| 1.6 PCB Layout | 2 |
| 1.7 Partlist: | 2 |
| 1.8 Changes | 3 |
| 2 Verzeichnis der Namensbereiche | 5 |
| 2.1 Liste aller Namensbereiche | 5 |
| 3 Klassen-Verzeichnis | 7 |
| 3.1 Auflistung der Klassen | 7 |
| 4 Datei-Verzeichnis | 9 |
| 4.1 Auflistung der Dateien | 9 |
| 5 Dokumentation der Namensbereiche | 11 |
| 5.1 replace_fs-Namensbereichsreferenz | 11 |
| 5.1.1 Variablen-Dokumentation | 11 |
| 5.1.1.1 MKSPIFFSTOOL | 11 |
| 6 Klassen-Dokumentation | 13 |
| 6.1 BoardInfo Klassenreferenz | 13 |
| 6.1.1 Ausführliche Beschreibung | 13 |
| 6.1.2 Beschreibung der Konstruktoren und Destruktoren | 13 |
| 6.1.2.1 BoardInfo() | 13 |
| 6.1.3 Dokumentation der Elementfunktionen | 14 |
| 6.1.3.1 ShowChipID() | 14 |
| 6.1.3.2 ShowChipInfo() | 14 |
| 6.1.3.3 ShowChipTemperature() | 14 |
| 6.1.3.4 ShowChipIDtoString() | 15 |
| 6.1.4 Dokumentation der Datenelemente | 15 |
| 6.1.4.1 m_chipid | 15 |
| 6.1.4.2 m_chipinfo | 15 |
| 6.2 tBoatData Strukturreferenz | 16 |
| 6.2.1 Ausführliche Beschreibung | 16 |
| 6.2.2 Beschreibung der Konstruktoren und Destruktoren | 17 |
| 6.2.2.1 tBoatData() | 17 |
| 6.2.3 Dokumentation der Datenelemente | 17 |
| 6.2.3.1 DaysSince1970 | 17 |
| 6.2.3.2 TrueHeading | 17 |

| | |
|---------------------------------------|-----------|
| 6.2.3.3 SOG | 17 |
| 6.2.3.4 COG | 17 |
| 6.2.3.5 Variation | 18 |
| 6.2.3.6 GPSTime | 18 |
| 6.2.3.7 Latitude | 18 |
| 6.2.3.8 Longitude | 18 |
| 6.2.3.9 Altitude | 18 |
| 6.2.3.10 HDOP | 18 |
| 6.2.3.11 GeoidalSeparation | 18 |
| 6.2.3.12 DGPSAge | 18 |
| 6.2.3.13 WaterTemperature | 19 |
| 6.2.3.14 WaterDepth | 19 |
| 6.2.3.15 Offset | 19 |
| 6.2.3.16 WindDirectionT | 19 |
| 6.2.3.17 WindDirectionM | 19 |
| 6.2.3.18 WindSpeedK | 19 |
| 6.2.3.19 WindSpeedM | 19 |
| 6.2.3.20 WindAngle | 19 |
| 6.2.3.21 GPSQualityIndicator | 20 |
| 6.2.3.22 SatelliteCount | 20 |
| 6.2.3.23 DGPSReferenceStationID | 20 |
| 6.2.3.24 MOBActivated | 20 |
| 6.2.3.25 Status | 20 |
| 6.3 Web_Config Strukturreferenz | 20 |
| 6.3.1 Ausführliche Beschreibung | 21 |
| 6.3.2 Dokumentation der Datenelemente | 21 |
| 6.3.2.1 wAP_IP | 21 |
| 6.3.2.2 wAP_SSID | 21 |
| 6.3.2.3 wAP_Password | 21 |
| 6.3.2.4 wTemp1_Offset | 21 |
| 6.3.2.5 wTemp2_Offset | 21 |
| 6.3.2.6 wFuellstandmax | 21 |
| 7 Datei-Dokumentation | 23 |
| 7.1 data/index.html-Dateireferenz | 23 |
| 7.2 index.html | 23 |
| 7.3 data/reboot.html-Dateireferenz | 25 |
| 7.4 reboot.html | 25 |
| 7.5 data/settings.html-Dateireferenz | 26 |
| 7.6 settings.html | 26 |
| 7.7 data/system.html-Dateireferenz | 27 |
| 7.8 system.html | 27 |

| | |
|--|----|
| 7.9 data/ueber.html-Dateireferenz | 27 |
| 7.10 ueber.html | 27 |
| 7.11 data/werte.html-Dateireferenz | 27 |
| 7.12 wert.html | 27 |
| 7.13 README.md-Dateireferenz | 28 |
| 7.14 replace_fs.py-Dateireferenz | 28 |
| 7.15 replace_fs.py | 28 |
| 7.16 src/BoardInfo.cpp-Dateireferenz | 29 |
| 7.16.1 Ausführliche Beschreibung | 29 |
| 7.16.2 Makro-Dokumentation | 30 |
| 7.16.2.1 BUF | 30 |
| 7.16.3 Dokumentation der Funktionen | 30 |
| 7.16.3.1 temprature_sens_read() | 30 |
| 7.17 BoardInfo.cpp | 30 |
| 7.18 src/BoardInfo.h-Dateireferenz | 32 |
| 7.18.1 Ausführliche Beschreibung | 32 |
| 7.19 BoardInfo.h | 33 |
| 7.20 src/BoatData.h-Dateireferenz | 33 |
| 7.21 BoatData.h | 34 |
| 7.22 src/configuration.h-Dateireferenz | 34 |
| 7.22.1 Ausführliche Beschreibung | 37 |
| 7.22.2 Makro-Dokumentation | 37 |
| 7.22.2.1 Version | 37 |
| 7.22.2.2 ESP32_CAN_TX_PIN | 37 |
| 7.22.2.3 ESP32_CAN_RX_PIN | 38 |
| 7.22.2.4 N2K_SOURCE | 38 |
| 7.22.2.5 EngineSendOffset | 38 |
| 7.22.2.6 TankSendOffset | 38 |
| 7.22.2.7 RPMSendOffset | 38 |
| 7.22.2.8 BatteryDCSendOffset | 38 |
| 7.22.2.9 BatteryDCStatusSendOffset | 38 |
| 7.22.2.10 SlowDataUpdatePeriod | 38 |
| 7.22.2.11 PAGE_REFRESH | 39 |
| 7.22.2.12 WEB_TITEL | 39 |
| 7.22.2.13 HostName | 39 |
| 7.22.2.14 CL_SSID | 39 |
| 7.22.2.15 CL_PASSWORD | 39 |
| 7.22.2.16 I2C_SDA | 39 |
| 7.22.2.17 I2C_SCL | 39 |
| 7.22.2.18 SEALEVELPRESSURE_HPA | 39 |
| 7.22.2.19 RPM_Calibration_Value | 40 |
| 7.22.2.20 Engine_RPM_Pin | 40 |

| | |
|---|----|
| 7.22.2.21 ONE_WIRE_BUS | 40 |
| 7.22.2.22 SERVER_HOST_NAME | 40 |
| 7.22.2.23 TCP_PORT | 40 |
| 7.22.2.24 DNS_PORT | 40 |
| 7.22.3 Dokumentation der Aufzählungstypen | 40 |
| 7.22.3.1 EngineStatus | 40 |
| 7.22.4 Variablen-Dokumentation | 41 |
| 7.22.4.1 NodeAddress | 41 |
| 7.22.4.2 preferences | 41 |
| 7.22.4.3 chipid | 41 |
| 7.22.4.4 id | 41 |
| 7.22.4.5 i | 41 |
| 7.22.4.6 tAP_Config | 41 |
| 7.22.4.7 channel | 41 |
| 7.22.4.8 hide_SSID | 42 |
| 7.22.4.9 max_connection | 42 |
| 7.22.4.10 IP | 42 |
| 7.22.4.11 Gateway | 42 |
| 7.22.4.12 NMask | 42 |
| 7.22.4.13 AP_SSID | 42 |
| 7.22.4.14 AP_PASSWORD | 42 |
| 7.22.4.15 AP_IP | 42 |
| 7.22.4.16 CL_IP | 43 |
| 7.22.4.17 SELF_IP | 43 |
| 7.22.4.18 sAP_Station | 43 |
| 7.22.4.19 iSTA_on | 43 |
| 7.22.4.20 bConnect_CL | 43 |
| 7.22.4.21 bClientConnected | 43 |
| 7.22.4.22 fbmp_temperature | 43 |
| 7.22.4.23 fbmp_pressure | 43 |
| 7.22.4.24 fbmp_altitude | 44 |
| 7.22.4.25 sl2C_Status | 44 |
| 7.22.4.26 bl2C_Status | 44 |
| 7.22.4.27 iMaxSonar | 44 |
| 7.22.4.28 iDistance | 44 |
| 7.22.4.29 FuelLevel | 44 |
| 7.22.4.30 FuelLevelMax | 44 |
| 7.22.4.31 OilTemp | 44 |
| 7.22.4.32 MotTemp | 45 |
| 7.22.4.33 EngineRPM | 45 |
| 7.22.4.34 BordSpannung | 45 |
| 7.22.4.35 EngineOn | 45 |

| | |
|---|----|
| 7.22.4.36 Counter | 45 |
| 7.22.4.37 Bat1Capacity | 45 |
| 7.22.4.38 Bat2Capacity | 45 |
| 7.22.4.39 SoCError | 45 |
| 7.22.4.40 BatSoC | 46 |
| 7.22.4.41 sOneWire_Status | 46 |
| 7.22.4.42 fDrehzahl | 46 |
| 7.22.4.43 fGaugeDrehzahl | 46 |
| 7.22.4.44 fBordSpannung | 46 |
| 7.22.4.45 fOilTemp1 | 46 |
| 7.22.4.46 fMotTemp2 | 46 |
| 7.22.4.47 fTemp1Offset | 46 |
| 7.22.4.48 fTemp2Offset | 47 |
| 7.22.4.49 sSTBB | 47 |
| 7.22.4.50 sOrient | 47 |
| 7.22.4.51 dMWV_WindDirectionT | 47 |
| 7.22.4.52 dMWV_WindSpeedM | 47 |
| 7.22.4.53 dVWR_WindDirectionM | 47 |
| 7.22.4.54 dVWR_WindAngle | 47 |
| 7.22.4.55 dVWR_WindSpeedkn | 47 |
| 7.22.4.56 dVWR_WindSpeedms | 48 |
| 7.22.4.57 udpAddress | 48 |
| 7.22.4.58 udpPort | 48 |
| 7.23 configuration.h | 48 |
| 7.24 src/helper.h-Dateireferenz | 50 |
| 7.24.1 Ausführliche Beschreibung | 51 |
| 7.24.2 Dokumentation der Funktionen | 51 |
| 7.24.2.1 ShowTime() | 51 |
| 7.24.2.2 freeHeapSpace() | 52 |
| 7.24.2.3 WiFiDiag() | 52 |
| 7.24.2.4 listDir() | 53 |
| 7.24.2.5 readConfig() | 54 |
| 7.24.2.6 writeConfig() | 55 |
| 7.24.2.7 I2C_scan() | 56 |
| 7.24.2.8 sWifiStatus() | 57 |
| 7.24.2.9 toChar() | 57 |
| 7.25 helper.h | 58 |
| 7.26 src/hourmeter.h-Dateireferenz | 61 |
| 7.26.1 Ausführliche Beschreibung | 62 |
| 7.26.2 Dokumentation der Funktionen | 63 |
| 7.26.2.1 EngineHours() | 63 |
| 7.26.3 Variablen-Dokumentation | 64 |

| | | |
|-----------|------------------------------------|----|
| 7.26.3.1 | bsz1 | 64 |
| 7.26.3.2 | lastRun | 65 |
| 7.26.3.3 | CounterOld | 65 |
| 7.26.3.4 | milliRest | 65 |
| 7.26.3.5 | state1 | 65 |
| 7.26.3.6 | laststate1 | 65 |
| 7.27 | hourmeter.h | 65 |
| 7.28 | src/LED.h-Dateireferenz | 66 |
| 7.28.1 | Ausführliche Beschreibung | 67 |
| 7.28.2 | Dokumentation der Aufzählungstypen | 67 |
| 7.28.2.1 | LED | 67 |
| 7.28.3 | Dokumentation der Funktionen | 68 |
| 7.28.3.1 | LEDblink() | 68 |
| 7.28.3.2 | LEDflash() | 68 |
| 7.28.3.3 | flashLED() | 68 |
| 7.28.3.4 | LEDInit() | 69 |
| 7.28.3.5 | LEDon() | 69 |
| 7.28.3.6 | LEDOff() | 70 |
| 7.28.3.7 | LEDOff_RGB() | 70 |
| 7.29 | LED.h | 70 |
| 7.30 | src/Motordaten.ino-Dateireferenz | 71 |
| 7.30.1 | Ausführliche Beschreibung | 73 |
| 7.30.2 | Makro-Dokumentation | 73 |
| 7.30.2.1 | ENABLE_DEBUG_LOG | 73 |
| 7.30.2.2 | ADC_Calibration_Value1 | 74 |
| 7.30.2.3 | ADC_Calibration_Value2 | 74 |
| 7.30.3 | Dokumentation der Funktionen | 74 |
| 7.30.3.1 | oneWire() | 74 |
| 7.30.3.2 | debug_log() | 74 |
| 7.30.3.3 | handleInterrupt() | 75 |
| 7.30.3.4 | setup() | 75 |
| 7.30.3.5 | GetTemperature() | 78 |
| 7.30.3.6 | ReadRPM() | 79 |
| 7.30.3.7 | IsTimeToUpdate() | 80 |
| 7.30.3.8 | InitNextUpdate() | 80 |
| 7.30.3.9 | SetNextUpdate() | 81 |
| 7.30.3.10 | SendN2kDCStatus() | 82 |
| 7.30.3.11 | SendN2kBattery() | 83 |
| 7.30.3.12 | SendN2kTankLevel() | 84 |
| 7.30.3.13 | SendN2kEngineData() | 85 |
| 7.30.3.14 | SendN2kEngineRPM() | 86 |
| 7.30.3.15 | ReadVoltage() | 87 |

| | |
|---|-----|
| 7.30.3.16 loop() | 88 |
| 7.30.4 Variablen-Dokumentation | 89 |
| 7.30.4.1 PROGMEM | 89 |
| 7.30.4.2 StartValue | 90 |
| 7.30.4.3 PeriodCount | 90 |
| 7.30.4.4 Last_int_time | 90 |
| 7.30.4.5 timer | 90 |
| 7.30.4.6 mux | 90 |
| 7.30.4.7 oneWire | 90 |
| 7.30.4.8 MotorCoolant | 91 |
| 7.30.4.9 MotorOil | 91 |
| 7.30.4.10 ADCpin2 | 91 |
| 7.30.4.11 ADCpin1 | 91 |
| 7.30.4.12 Task1 | 91 |
| 7.30.4.13 baudrate | 91 |
| 7.30.4.14 rs_config | 92 |
| 7.31 Motordaten.ino | 92 |
| 7.32 src/NMEA0183Telegram.h-Dateireferenz | 98 |
| 7.32.1 Ausführliche Beschreibung | 98 |
| 7.32.2 Dokumentation der Funktionen | 99 |
| 7.32.2.1 CheckSum() | 99 |
| 7.32.2.2 sendXDR() | 99 |
| 7.33 NMEA0183Telegram.h | 100 |
| 7.34 src/task.h-Dateireferenz | 101 |
| 7.34.1 Makro-Dokumentation | 102 |
| 7.34.1.1 taskBegin | 102 |
| 7.34.1.2 taskEnd | 102 |
| 7.34.1.3 taskSwitch | 102 |
| 7.34.1.4 taskPause | 102 |
| 7.34.1.5 taskWaitFor | 102 |
| 7.34.1.6 taskStepName | 103 |
| 7.34.1.7 taskJumpTo | 103 |
| 7.35 task.h | 103 |
| 7.36 src/web.h-Dateireferenz | 103 |
| 7.36.1 Ausführliche Beschreibung | 104 |
| 7.36.2 Dokumentation der Funktionen | 105 |
| 7.36.2.1 server() | 105 |
| 7.36.2.2 processor() | 105 |
| 7.36.2.3 replaceVariable() | 106 |
| 7.36.2.4 website() | 107 |
| 7.36.3 Variablen-Dokumentation | 108 |
| 7.36.3.1 sBoardInfo | 108 |

| | |
|-------------------------------------|------------|
| 7.36.3.2 boardInfo | 108 |
| 7.36.3.3 IsRebootRequired | 109 |
| 7.36.3.4 sCL_Status | 109 |
| 7.37 web.h | 109 |
| Index | 111 |

Kapitel 1

MotorData NMEA2000

1.1 Description

This repository shows how to measure the

- Battery Voltage
- Engine RPM
- Fuel Level
- Oil and Motor Temperature
- Alarms engine stop and tempertur high
- Enginehours

and send it as NNMEA2000 meassage.

- PGN 127488 // Engine Rapid / RPM
- PGN 127489 // Engine parameters dynamic
- PGN 127505 // Fluid Level
- PGN 127506 // Battery
- PGN 127508 // Battery Status

In addition, all data and part of the configuration are displayed as a website.

[Doxygen Documentation](#)

1.2 Based on the work of

[NMEA2000-Data-Sender](#) @AK-Homberger

[NMEA 2000](#) @ttlappalainen

This project is part of [OpenBoatProject](#)

1.3 Website

1.4 Plotter

1.5 Wiring diagram

1.6 PCB Layout

The project requires the NMEA2000 and the NMEA2000_esp32 libraries from Timo Lappalainen: <https://github.com/ttlappalainen>. Both libraries have to be downloaded and installed.

The ESP32 in this project is an Adafruit Huzzah! ESP32. Pin layout for other ESP32 devices might differ.

For the ESP32 CAN bus, I used the "SN65HVD230 Chip from TI" as transceiver. It works well with the ESP32. The correct GPIO ports are defined in the main sketch. For this project, I use the pins GPIO4 for CAN RX and GPIO5 for CAN TX.

The 12 Volt is reduced to 5 Volt with a DC Step-Down_Converter. 12V DC comes from the N2k Bus Connector with the M12 Connector.

The Website use LittleFS Filesystem. You must use Partition Schemes "Minimal SPIFFS with APPS and OTA". The HTML Data upload separately with

- "ESP 32 Skcetch Data upload" (Arduino IDE) or
- PlatformIO > Build Filesystem and Upload Filesystem Image (PlatformIO) from /data directory.

1.7 Partlist:

- PCB by Aisler [Link](#)

Assembly: [MD N2k__Assembly.pdf](#)

- 1 C1 10μ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 2 C2 22μ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 3 R1 100k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 4 R2 27k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 5 R3 300R R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 6 R4 10k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 7 R5 1k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 8 R6 4k7 R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 9 R7 2k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 10 D1 B360 B 360 F Schottkydiode, 60 V, 3 A, DO-214AB/SMC 1

- 11 D2 LED_RBKG RGB [LED](#) Kingbright 1
- 12 D3 PESD1CAN SOT-23 Dual bidirectional TVS diode 1
- 13 D4 ZPD3.3 D_DO-35_SOD27_P10.16mm_Horizontal 1 [Link](#)
- 14 D5 1N4148 D_DO-35_SOD27_P7.62mm_Horizontal 1 [Link](#)
- 15 D6 P4SMAJ26CA D_SMA_TVS 1
- 16 U1 TSR_1-2450 Converter_DCDC_TRACO_TSR-1_THT 1 [Link](#)
- 17 U2 ESP32-Huzzah Adafruit_ESP32 1
- 18 U3 SN65HVD230 SOIC-8_3.9x4.9mm_P1.27mm 1 [Link](#)
- 19 U4 H11L1 DIP-6_W7.62mm 1 [Link](#)
- 20 FL1 EPCO B82789C0513 B82789C0113N002 1
- 21 J2, J3 Conn_01x04_Pin PinHeader_1x04_P2.54mm_Vertical 2
- 22 J1 Conn_01x03_Pin PinHeader_1x03_P2.54mm_Vertical 1
- 23 Wago-Case: [Link](#)

1.8 Changes

- Version 2.4 add Doxygen
- Version 2.3 add Temperatur: Motor(Water)temp and OilTemp (2x OneWire), add Alarm Watertemp
- Version 2.2 add Motorparameter: EngineHours and Alarms (Oiltemp max / Engine Stop)
- Version 2.1 Minor updates website, change Engine Parameter to PGN127489 (Oil Temp)
- Version 2.0
 - update Website (code and html files)
 - change Hardware layout, add protection's and C's on Voltage input, add protection's for CanBus
 - change Webinterface, add calibration-offset for temperature

Kapitel 2

Verzeichnis der Namensbereiche

2.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

| | |
|-----------------------------------|----|
| <code>replace_fs</code> | 11 |
|-----------------------------------|----|

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

| | | |
|----------------------------|-------|----|
| BoardInfo | | 13 |
| tBoatData | | 16 |
| Web_Config | | 20 |

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

| | |
|---|-----|
| replace_fs.py | 28 |
| data/index.html | 23 |
| data/reboot.html | 25 |
| data/settings.html | 26 |
| data/system.html | 27 |
| data/ueber.html | 27 |
| data/werte.html | 27 |
| src/BoardInfo.cpp | |
| Boardinfo | 29 |
| src/BoardInfo.h | |
| Hardwareinfo from ESP Board | 32 |
| src/BoatData.h | 33 |
| src/configuration.h | |
| Konfiguration für GPIO und Variable | 34 |
| src/helper.h | |
| Hilfsfunktionen | 50 |
| src/hourmeter.h | |
| Betriebsstundenzähler | 61 |
| src/LED.h | |
| LED Ansteuerung | 66 |
| src/Motordaten.ino | |
| Motordaten NMEA2000 | 71 |
| src/NMEA0183Telegram.h | |
| NMEA0183 Telegramme senden | 98 |
| src/task.h | 101 |
| src/web.h | |
| Webseite Variablen lesen und schreiben, Webseiten erstellen | 103 |

Kapitel 5

Dokumentation der Namensbereiche

5.1 replace_fs-Namensbereichsreferenz

Variablen

- [MKSPIFFSTOOL](#)

5.1.1 Variablen-Dokumentation

5.1.1.1 MKSPIFFSTOOL

`replace_fs.MKSPIFFSTOOL`

Definiert in Zeile [3](#) der Datei [replace_fs.py](#).

Kapitel 6

Klassen-Dokumentation

6.1 BoardInfo Klassenreferenz

```
#include <BoardInfo.h>
```

Öffentliche Methoden

- [BoardInfo](#) ()
Construct a new Board Info:: Board Info object.
- void [ShowChipID](#) ()
- void [ShowChipInfo](#) ()
- void [ShowChipTemperature](#) ()
- String [ShowChipIDtoString](#) ()

Geschützte Attribute

- uint64_t [m_chipid](#)
- esp_chip_info_t [m_chipinfo](#)

6.1.1 Ausführliche Beschreibung

Definiert in Zeile [16](#) der Datei [BoardInfo.h](#).

6.1.2 Beschreibung der Konstruktoren und Destruktoren

6.1.2.1 BoardInfo()

```
BoardInfo::BoardInfo ()
```

Construct a new Board Info:: Board Info object.

Definiert in Zeile [36](#) der Datei [BoardInfo.cpp](#).

```
00037 {  
00038     // Konstruktor der Klasse  
00039     // ChipID auslesen  
00040     //The chip ID is essentially its MAC address(length: 6 bytes).  
00041     m_chipid = 0;  
00042     m_chipid = ESP.getEfuseMac(); //The chip ID is essentially its MAC address(length: 6 bytes).  
00043     // Chip - Info auslesen  
00044     esp_chip_info(&m_chipinfo);  
00045 }
```

6.1.3 Dokumentation der Elementfunktionen

6.1.3.1 ShowChipID()

void BoardInfo::ShowChipID ()

Definiert in Zeile 47 der Datei BoardInfo.cpp.

```
00048 {
00049     if (m_chipid != 0)
00050     {
00051         Serial.printf("ESP32 Chip ID = %04X", (uint16_t) (m_chipid >> 32)); //print High 2 bytes
00052         Serial.printf("%08X\n", (uint32_t) m_chipid); //print Low 4bytes.
00053     }
00054     else
00055     {
00056         // Fehler beim Lesen der ID....
00057         Serial.println("ESP32 Chip ID konnte nicht ausgelesen werden");
00058     }
00059 }
```

6.1.3.2 ShowChipInfo()

void BoardInfo::ShowChipInfo ()

Definiert in Zeile 100 der Datei BoardInfo.cpp.

```
00101 {
00102     // Infos zum Board
00103     Serial.printf("Das ist ein Chip mit %d CPU - Kernen\nWLAN: %s\nBluetooth: %s\n",
00104         m_chipinfo.cores,
00105         (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
00106         (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
00107         (m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00108
00109     Serial.printf("Silicon revision %d\n", m_chipinfo.revision);
00110
00111     Serial.printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
00112         (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");
00113
00114     Serial.printf("(Freier Speicher: %d bytes)\n", esp_get_free_heap_size());
00115     Serial.printf("Freier Speicher: %d bytes\n", ESP.getFreeHeap());
00116     Serial.printf("Minimum freier Speicher: %d bytes\n", esp_get_minimum_free_heap_size());
00117 }
```

6.1.3.3 ShowChipTemperature()

void BoardInfo::ShowChipTemperature ()

Definiert in Zeile 119 der Datei BoardInfo.cpp.

```
00120 {
00121     uint8_t temp_fahrenheit;
00122     float temp_celsius;
00123     temp_fahrenheit = temprature_sens_read();
00124     if (128 == temp_fahrenheit)
00125     {
00126         Serial.println("Kein Temperatur - Sensor vorhanden.");
00127         return;
00128     }
00129     temp_celsius = (temp_fahrenheit - 32) / 1.8;
00130     Serial.printf("Temperatur Board: %i Fahrenheit\n", temp_fahrenheit);
00131     Serial.printf("Temperatur Board: %.1f °C\n", temp_celsius);
00132 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



6.1.3.4 ShowChipIDtoString()

String BoardInfo::ShowChipIDtoString ()

Definiert in Zeile 61 der Datei [BoardInfo.cpp](#).

```

00062 {
00063     String msg;
00064     if (m_chipid != 0)
00065     {
00066         char string1[BUF];
00067         sprintf(string1, "ESP32 Chip ID = %04X%08X<br>", (uint16_t) (m_chipid>>32), (uint32_t)m_chipid);
00068         msg = (char*)string1;
00069         msg += "<br>";
00070         sprintf(string1, "%d CPU - Kerne<br>WLAN: %s<br>Bluetooth: %s%s",
00071             m_chipinfo.cores,
00072             (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
00073             (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
00074             (m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00075         msg += (char*)string1;
00076         msg += "<br>";
00077         sprintf(string1, "Silicon revision: %d", m_chipinfo.revision);
00078         msg += (char*)string1;
00079         msg += "<br>";
00080         sprintf(string1, "%s Speicher %dMB", (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ?
"embedded" : "external",
                                spi_flash_get_chip_size() / (1024 * 1024));
00081
00082         msg += (char*)string1;
00083         msg += "<br>";
00084         sprintf(string1, "Freier Speicher: %d bytes", ESP.getFreeHeap());
00085         msg += (char*)string1;
00086         msg += "<br>";
00087         sprintf(string1, "Min freier Speicher: %d bytes", esp_get_minimum_free_heap_size());
00088         msg += (char*)string1;
00089         msg += "<br>";
00090     }
00091     else
00092     {
00093         // Fehler beim Lesen der ID....
00094         msg = "ESP32 Chip ID konnte nicht ausgelesen werden";
00095     }
00096     return msg;
00097 }
00098 }
```

6.1.4 Dokumentation der Datenelemente

6.1.4.1 m_chipid

uint64_t BoardInfo::m_chipid [protected]

Definiert in Zeile 28 der Datei [BoardInfo.h](#).

6.1.4.2 m_chipinfo

esp_chip_info_t BoardInfo::m_chipinfo [protected]

Definiert in Zeile 29 der Datei [BoardInfo.h](#).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- [src/BoardInfo.h](#)
- [src/BoardInfo.cpp](#)

6.2 tBoatData Strukturreferenz

```
#include <BoatData.h>
```

Öffentliche Methoden

- [tBoatData](#) ()

Öffentliche Attribute

- unsigned long [DaysSince1970](#)
- double [TrueHeading](#)
- double [SOG](#)
- double [COG](#)
- double [Variation](#)
- double [GPSTime](#)
- double [Latitude](#)
- double [Longitude](#)
- double [Altitude](#)
- double [HDOP](#)
- double [GeoidalSeparation](#)
- double [DGPSAge](#)
- double [WaterTemperature](#)
- double [WaterDepth](#)
- double [Offset](#)
- double [WindDirectionT](#)
- double [WindDirectionM](#)
- double [WindSpeedK](#)
- double [WindSpeedM](#)
- double [WindAngle](#)
- int [GPSQualityIndicator](#)
- int [SatelliteCount](#)
- int [DGPSReferenceStationID](#)
- bool [MOBActivated](#)
- char [Status](#)

6.2.1 Ausführliche Beschreibung

Definiert in Zeile 4 der Datei [BoatData.h](#).

6.2.2 Beschreibung der Konstruktoren und Destruktoren

6.2.2.1 tBoatData()

```
tBoatData::tBoatData () [inline]
```

Definiert in Zeile 18 der Datei [BoatData.h](#).

```
00018     {  
00019     TrueHeading=0;  
00020     SOG=0;  
00021     COG=0;  
00022     Variation=7.0;  
00023     GPSTime=0;  
00024     Latitude = 0;  
00025     Longitude = 0;  
00026     Altitude=0;  
00027     HDOP=100000;  
00028     DGPSAge=100000;  
00029     WaterTemperature = 0;  
00030     DaysSince1970=0;  
00031     MOBActivated=false;  
00032     SatelliteCount=0;  
00033     DGPSReferenceStationID=0;  
00034 };
```

6.2.3 Dokumentation der Datenelemente

6.2.3.1 DaysSince1970

```
unsigned long tBoatData::DaysSince1970
```

Definiert in Zeile 5 der Datei [BoatData.h](#).

6.2.3.2 TrueHeading

```
double tBoatData::TrueHeading
```

Definiert in Zeile 7 der Datei [BoatData.h](#).

6.2.3.3 SOG

```
double tBoatData::SOG
```

Definiert in Zeile 7 der Datei [BoatData.h](#).

6.2.3.4 COG

```
double tBoatData::COG
```

Definiert in Zeile 7 der Datei [BoatData.h](#).

6.2.3.5 Variation

```
double tBoatData::Variation
```

Definiert in Zeile 7 der Datei [BoatData.h](#).

6.2.3.6 GPSTime

```
double tBoatData::GPSTime
```

Definiert in Zeile 8 der Datei [BoatData.h](#).

6.2.3.7 Latitude

```
double tBoatData::Latitude
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.8 Longitude

```
double tBoatData::Longitude
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.9 Altitude

```
double tBoatData::Altitude
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.10 HDOP

```
double tBoatData::HDOP
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.11 GeoidalSeparation

```
double tBoatData::GeoidalSeparation
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.12 DGPSAge

```
double tBoatData::DGPSAge
```

Definiert in Zeile 9 der Datei [BoatData.h](#).

6.2.3.13 WaterTemperature

```
double tBoatData::WaterTemperature
```

Definiert in Zeile 10 der Datei [BoatData.h](#).

6.2.3.14 WaterDepth

```
double tBoatData::WaterDepth
```

Definiert in Zeile 10 der Datei [BoatData.h](#).

6.2.3.15 Offset

```
double tBoatData::Offset
```

Definiert in Zeile 10 der Datei [BoatData.h](#).

6.2.3.16 WindDirectionT

```
double tBoatData::WindDirectionT
```

Definiert in Zeile 11 der Datei [BoatData.h](#).

6.2.3.17 WindDirectionM

```
double tBoatData::WindDirectionM
```

Definiert in Zeile 11 der Datei [BoatData.h](#).

6.2.3.18 WindSpeedK

```
double tBoatData::WindSpeedK
```

Definiert in Zeile 11 der Datei [BoatData.h](#).

6.2.3.19 WindSpeedM

```
double tBoatData::WindSpeedM
```

Definiert in Zeile 11 der Datei [BoatData.h](#).

6.2.3.20 WindAngle

```
double tBoatData::WindAngle
```

Definiert in Zeile 12 der Datei [BoatData.h](#).

6.2.3.21 GPSQualityIndicator

```
int tBoatData::GPSQualityIndicator
```

Definiert in Zeile 13 der Datei [BoatData.h](#).

6.2.3.22 SatelliteCount

```
int tBoatData::SatelliteCount
```

Definiert in Zeile 13 der Datei [BoatData.h](#).

6.2.3.23 DGPSReferenceStationID

```
int tBoatData::DGPSReferenceStationID
```

Definiert in Zeile 13 der Datei [BoatData.h](#).

6.2.3.24 MOBActivated

```
bool tBoatData::MOBActivated
```

Definiert in Zeile 14 der Datei [BoatData.h](#).

6.2.3.25 Status

```
char tBoatData::Status
```

Definiert in Zeile 15 der Datei [BoatData.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [src/BoatData.h](#)

6.3 Web_Config Strukturreferenz

```
#include <configuration.h>
```

Öffentliche Attribute

- char [wAP_IP](#) [20]
- char [wAP_SSID](#) [64]
- char [wAP_Password](#) [12]
- char [wTemp1_Offset](#) [5]
- char [wTemp2_Offset](#) [5]
- char [wFuellstandmax](#) [3]

6.3.1 Ausführliche Beschreibung

Definiert in Zeile 46 der Datei [configuration.h](#).

6.3.2 Dokumentation der Datenelemente

6.3.2.1 wAP_IP

```
char Web_Config::wAP_IP[20]
```

Definiert in Zeile 48 der Datei [configuration.h](#).

6.3.2.2 wAP_SSID

```
char Web_Config::wAP_SSID[64]
```

Definiert in Zeile 49 der Datei [configuration.h](#).

6.3.2.3 wAP_Password

```
char Web_Config::wAP_Password[12]
```

Definiert in Zeile 50 der Datei [configuration.h](#).

6.3.2.4 wTemp1_Offset

```
char Web_Config::wTemp1_Offset[5]
```

Definiert in Zeile 51 der Datei [configuration.h](#).

6.3.2.5 wTemp2_Offset

```
char Web_Config::wTemp2_Offset[5]
```

Definiert in Zeile 52 der Datei [configuration.h](#).

6.3.2.6 wFuellstandmax

```
char Web_Config::wFuellstandmax[3]
```

Definiert in Zeile 53 der Datei [configuration.h](#).

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- [src/configuration.h](#)

Kapitel 7

Datei-Dokumentation

7.1 data/index.html-Dateireferenz

7.2 index.html

[gehe zur Dokumentation dieser Datei](#)

```
00001 <!DOCTYPE html>
00002 <html>
00003 <head>
00004     <title>Motordaten</title>
00005     <meta name="viewport" content="width=device-width, initial-scale=1">
00006     <link rel="shortcut icon" type="image/x-icon" href="favicon.ico">
00007     <link rel="icon" href="data:,">
00008     <link rel="stylesheet" type="text/css" href="style.css">
00009     <script src='gauge.min.js'></script>
00010     <meta http-equiv="refresh" content="5">
00011 </head>
00012 <body>
00013     <canvas data-type="radial-gauge"
00014         data-width="200"
00015         data-height="200"
00016         data-units="U &frasl; min"
00017         data-title="Drehzahl"
00018         data-min-value="0"
00019         data-start-angle="70"
00020         data-ticks-angle="220"
00021         data-value-box="true"
00022         data-max-value="5000"
00023         data-major-ticks="0,1000,2000,3000,4000,5000"
00024         data-minor-ticks="5"
00025         data-stroke-ticks="true"
00026         data-highlights=[
00027     {"from": 0, "to": 800, "color": "rgba(255, 165, 0, .75)"},
00028     {"from": 800, "to": 3000, "color": "rgba(0, 255, 0, .75)"},
00029     {"from": 3000, "to": 5000, "color": "rgba(255, 50, 50, .75)"}
00030     ]'
00031         data-color-plate="#fff"
00032         data-border-shadow-width="0"
00033         data-borders="false"
00034         data-needle-type="arrow"
00035         data-needle-width="4"
00036         data-needle-circle-size="7"
00037         data-needle-circle-outer="true"
00038         data-needle-circle-inner="false"
00039         data-animation-duration="1500"
00040         data-animation-rule="linear"
00041         data-value-text='%sDrehzahl% U &frasl; min'
00042         data-value='%sDrehzahl%'
00043     ></canvas>
00044
00045     <canvas data-type="radial-gauge"
00046         data-width="200"
00047         data-height="200"
00048         data-units="%deg;C"
00049         data-title="Oil Temperatur"
```

```

00050         data-min-value="0"
00051         data-start-angle="70"
00052         data-ticks-angle="220"
00053         data-value-box="true"
00054         data-max-value="80"
00055         data-major-ticks="0,10,20,30,40,50,60,70,80"
00056         data-minor-ticks="2"
00057         data-stroke-ticks="true"
00058         data-highlights=' [
00059 {"from": 0, "to": 50, "color": "rgba(0, 191, 255, .75)"},
00060 {"from": 50, "to": 70, "color": "rgba(0, 255, 0, .75)"},
00061 {"from": 70, "to": 80, "color": "rgba(255, 50, 50, .75)"}
00062 ]',
00063         data-color-plate="#fff"
00064         data-border-shadow-width="0"
00065         data-borders="false"
00066         data-needle-type="arrow"
00067         data-needle-width="4"
00068         data-needle-circle-size="7"
00069         data-needle-circle-outer="true"
00070         data-needle-circle-inner="false"
00071         data-animation-duration="1500"
00072         data-animation-rule="linear"
00073         data-value-text='%sOilTemp1% &deg;C'
00074         data-value='%sOilTemp1%'
00075     ></canvas>
00076     <canvas data-type="radial-gauge"
00077         data-width="200"
00078         data-height="200"
00079         data-units="%deg;C"
00080         data-title="Mot Temperatur"
00081         data-min-value="0"
00082         data-start-angle="70"
00083         data-ticks-angle="220"
00084         data-value-box="true"
00085         data-max-value="80"
00086         data-major-ticks="0,10,20,30,40,50,60,70,80"
00087         data-minor-ticks="2"
00088         data-stroke-ticks="true"
00089         data-highlights=' [
00090 {"from": 0, "to": 50, "color": "rgba(0, 191, 255, .75)"},
00091 {"from": 50, "to": 70, "color": "rgba(0, 255, 0, .75)"},
00092 {"from": 70, "to": 80, "color": "rgba(255, 50, 50, .75)"}
00093 ]',
00094         data-color-plate="#fff"
00095         data-border-shadow-width="0"
00096         data-borders="false"
00097         data-needle-type="arrow"
00098         data-needle-width="4"
00099         data-needle-circle-size="7"
00100         data-needle-circle-outer="true"
00101         data-needle-circle-inner="false"
00102         data-animation-duration="1500"
00103         data-animation-rule="linear"
00104         data-value-text='%sMotTemp2% &deg;C'
00105         data-value='%sMotTemp2%'
00106     ></canvas>
00107     <br>
00108     <canvas data-type="radial-gauge"
00109         data-width="300"
00110         data-height="300"
00111         data-units="V"
00112         data-title="Bordspannung"
00113         data-min-value="7"
00114         data-start-angle="70"
00115         data-ticks-angle="220"
00116         data-value-box="true"
00117         data-max-value="15"
00118         data-major-ticks="7,8,9,10,11,12,13,14,15"
00119         data-minor-ticks="10"
00120         data-stroke-ticks="true"
00121         data-highlights=' [
00122 {"from": 7, "to": 11, "color": "rgba(255, 50, 50, .75)"},
00123 {"from": 11, "to": 13, "color": "rgba(0, 255, 0, .75)"},
00124 {"from": 13, "to": 15, "color": "rgba(255, 165, 0, .75)"}
00125 ]',
00126         data-color-plate="#fff"
00127         data-border-shadow-width="0"
00128         data-borders="false"
00129         data-needle-type="arrow"
00130         data-needle-width="4"
00131         data-needle-circle-size="7"
00132         data-needle-circle-outer="true"
00133         data-needle-circle-inner="false"
00134         data-animation-duration="1500"
00135         data-animation-rule="linear"
00136         data-value-text='%sBordspannung% V'

```

```

00137         data-value='%sBordspannung%'
00138     ></canvas>
00139
00140     <canvas data-type="radial-gauge"
00141         data-width="300"
00142         data-height="300"
00143         data-units="&#37;"
00144         data-title="F&uuml;llstand"
00145         data-min-value="0"
00146         data-start-angle="70"
00147         data-ticks-angle="220"
00148         data-value-box="true"
00149         data-max-value="100"
00150         data-major-ticks="0,10,20,30,40,50,60,70,80,90,100"
00151         data-minor-ticks="2"
00152         data-stroke-ticks="true"
00153         data-highlights=[
00154             {"from": 0, "to": 10, "color": "rgba(255, 50, 50, .75)"},
00155             {"from": 10, "to": 20, "color": "rgba(255, 165, 0, .75)"},
00156             {"from": 20, "to": 100, "color": "rgba(0, 255, 0, .75)"}
00157         ],
00158         data-color-plate="#fff"
00159         data-border-shadow-width="0"
00160         data-borders="false"
00161         data-needle-type="arrow"
00162         data-needle-width="4"
00163         data-needle-circle-size="7"
00164         data-needle-circle-outer="true"
00165         data-needle-circle-inner="false"
00166         data-animation-duration="1500"
00167         data-animation-rule="linear"
00168         data-value-text='%sFuellstand% &#37;'
00169         data-value='%sFuellstand%'
00170     ></canvas>
00171     <ul class="bottomnav">
00172         <li><a class="active" href="/">Home</a></li>
00173         <li><a href="werte.html">Werte</a></li>
00174         <li><a href="settings.html">Setting</a></li>
00175         <li><a href="system.html">System</a></li>
00176         <li class="right"><a href="ueber.html">About</a></li>
00177     </ul>
00178 </body>
00179 </html>

```

7.3 data/reboot.html-Dateireferenz

7.4 reboot.html

[gehe zur Dokumentation dieser Datei](#)

```

00001 <!DOCTYPE HTML>
00002 <html lang="de">
00003 <head>
00004     <meta charset="UTF-8">
00005     <link rel="stylesheet" type="text/css" href="style.css">
00006 </head>
00007 <body>
00008     <h1>
00009         Wartezeit für Reboot, WiFi und Webserver Initialisierung<br>Aufruf der home page in <span
00010         id="countdown">15</span> Sekunden...
00011     </h1>
00011     <script type="text/javascript">
00012         var seconds = 15;
00013         function countdown() {
00014             seconds = seconds - 1;
00015             if (seconds <= 0) {
00016                 window.location = "/";
00017             } else {
00018                 document.getElementById("countdown").innerHTML = seconds;
00019                 window.setTimeout("countdown()", 1000);
00020             }
00021         }
00022         countdown();
00023     </script>
00024     <ul class="bottomnav">
00025         <li><a href="/">Home</a></li>
00026         <li><a href="werte.html">Werte</a></li>
00027         <li><a class="active" href="settings.html">Settings</a></li>
00028         <li><a href="system.html">System</a></li>

```

```

00029     <li class="right"><a href="ueber.html">About</a></li>
00030 </ul>
00031 </body>
00032 </html>

```

7.5 data/settings.html-Dateireferenz

7.6 settings.html

[gehe zur Dokumentation dieser Datei](#)

```

00001 <!DOCTYPE html>
00002 <html>
00003 <head>
00004     <title>Settings</title>
00005     <meta name="viewport" content="width=device-width, initial-scale=1">
00006     <link rel="icon" href="data:,">
00007     <link rel="stylesheet" type="text/css" href="style.css">
00008 </head>
00009 <body>
00010     <br />
00011     <p class="label"> Oiltemperatur: %sOilTemp1% &deg;C</br>
00012         Offset: %sTemp1Offset% &deg;C</br>
00013         Motortemperatur: %sMotTemp2% &deg;C</br>
00014         Offset: %sTemp2Offset% &deg;C</p>
00015     %CONFIGPLACEHOLDER%
00016     <script>
00017         function formToJson(form) {
00018             var xhr = new XMLHttpRequest();
00019             var SSID = form.SSID.value;
00020             var IP = form.IP.value;
00021             var Password = form.Password.value;
00022             var Temp1Offset = form.Temp1Offset.value;
00023             var Temp2Offset = form.Temp2Offset.value;
00024             var Fuellstandmax = form.Fuellstandmax.value;
00025
00026             var jsonFormInfo = JSON.stringify({
00027                 SSID: SSID,
00028                 IP: IP,
00029                 Password: Password,
00030                 Temp1Offset: Temp1Offset,
00031                 Temp2Offset: Temp2Offset,
00032                 Fuellstandmax: Fuellstandmax
00033             });
00034
00035             xhr.open("POST", "/settings.html?save=" + jsonFormInfo, true);
00036             /* window.alert("Json function send end"); */
00037             xhr.send();
00038             window.alert("Gespeichert!");
00039         }
00040     </script>
00041
00042     <p class="label">Nach &Auml;nderungen neu starten!</p>
00043
00044     <button class="button" onclick="reboot_handler()">Neustart</button>
00045     </p>
00046
00047     <p id="status"></p>
00048     <script>
00049         function reboot_handler()
00050         {
00051             document.getElementById("status").innerHTML = "Starte Reboot ...";
00052             var xhr = new XMLHttpRequest();
00053             xhr.open("GET", "/reboot", true);
00054             xhr.send();
00055             setTimeout(function(){ window.open("/reboot","_self"); }, 500);
00056         }
00057     </script>
00058
00059
00060     <ul class="bottomnav">
00061         <li><a href="/">Home</a></li>
00062         <li><a href="werte.html">Werte</a></li>
00063         <li><a class="active" href="settings.html">Settings</a></li>
00064         <li><a href="system.html">System</a></li>
00065         <li class="right"><a href="ueber.html">About</a></li>
00066     </ul>
00067 </body>
00068 </html>

```

7.7 data/system.html-Dateireferenz

7.8 system.html

[gehe zur Dokumentation dieser Datei](#)

```

00001 <HTML>
00002 <HEAD>
00003     <TITLE>Systeminfo</TITLE>
00004     <meta name="viewport" content="width=device-width, initial-scale=1">
00005     <link rel="icon" href="data:,">
00006     <link rel="stylesheet" type="text/css" href="style.css">
00007 </HEAD>
00008 <BODY>
00009     <br />
00010     <p class="label">Eigene IP-Adresse - AP: %sAP_IP%<br />
00011         Clients am AP: %sAP_Clients%</p>
00012     <p class="label">OneWire %sOneWire_Status% Sensor gefunden</p>
00013     <p class="label">Informationen zum ESP32 - Board:<br />%sBoardInfo%</p>
00014     <p class="label">LittleFS, benutzte Bytes: %sFS_USpace% <br />
00015         LittleFS, gesamte Bytes: %sFS_TSpace% </p>
00016     <br />
00017     <br />
00018     <ul class="bottomnav">
00019         <li><a href="/">Home</a></li>
00020         <li><a href="werte.html">Werte</a></li>
00021         <li><a href="settings.html">Setting</a></li>
00022         <li><a class="active" href="system.html">System</a></li>
00023         <li class="right"><a href="ueber.html">About</a></li>
00024     </ul>
00025 </BODY>
00026 </HTML>

```

7.9 data/ueber.html-Dateireferenz

7.10 ueber.html

[gehe zur Dokumentation dieser Datei](#)

```

00001 <HTML>
00002 <HEAD>
00003     <TITLE>Wer steckt dahinter</TITLE>
00004     <meta name="viewport" content="width=device-width, initial-scale=1">
00005     <link rel="icon" href="data:,">
00006     <link rel="stylesheet" type="text/css" href="style.css">
00007 </HEAD>
00008 <BODY>
00009     <p class="label">%sVersion%</p>
00010     <br />
00011     <p class="label">Autor: Gerry Sebb</br>
00012     <a href="mailto:gerry@sebb.de">gerry@sebb.de</a></p>
00013     <br />
00014     
00015     <br />
00016     <ul class="bottomnav">
00017         <li><a href="/">Home</a></li>
00018         <li><a href="werte.html">Werte</a></li>
00019         <li><a href="settings.html">Setting</a></li>
00020         <li><a href="system.html">System</a></li>
00021         <li class="right"><a class="active" href="ueber.html">About</a></li>
00022     </ul>
00023 </BODY>
00024 </HTML>

```

7.11 data/werte.html-Dateireferenz

7.12 werte.html

[gehe zur Dokumentation dieser Datei](#)

```

00001 <!DOCTYPE html>
00002 <html>
00003 <head>
00004     <title>Motordaten</title>
00005     <meta name="viewport" content="width=device-width, initial-scale=1">
00006     <link rel="shortcut icon" type="image/x-icon" href="favicon.ico">
00007     <link rel="icon" href="data:,">
00008     <link rel="stylesheet" type="text/css" href="style.css">
00009     <script src='gauge.min.js'></script>
00010     <meta http-equiv="refresh" content="5">
00011 </head>
00012 <body>
00013     <p class="label">Bordspannung: %sBordspannung% V</p>
00014     <p class="label">Oil Temperatur: %sOilTemp1% &deg;C</br>
00015         Offset: %sTemp1Offset% &deg;C
00016     </p>
00017     <p class="label">Motor Temperatur: %sMotTemp2% &deg;C</br>
00018         Offset: %sTemp2Offset% &deg;C
00019     </p>
00020     <p class="label">Motor Drehzahl: %sDrehzahl% U &frac{1}{min}</p>
00021     <p class="label">Tank F&uuml;llstand: %sFuellstand% &#37;</br>max. F&uuml;llstand:
00022     %sFuellstandmax% l</p>
00023     <p class="label">Maschinenstunden: %sCounter% h</p>
00024
00025
00026     <ul class="bottomnav">
00027         <li><a href="/">Home</a></li>
00028         <li><a class="active" href="/">Werte</a></li>
00029         <li><a href="settings.html">Setting</a></li>
00030         <li><a href="system.html">System</a></li>
00031         <li class="right"><a href="ueber.html">About</a></li>
00032     </ul>
00033 </body>
00034 </html>

```

7.13 README.md-Dateireferenz

7.14 replace_fs.py-Dateireferenz

Namensbereiche

- namespace [replace_fs](#)

Variablen

- [replace_fs.MKSPIFFSTOOL](#)

7.15 replace_fs.py

[gehe zur Dokumentation dieser Datei](#)

```

00001 Import ("env")
00002 print ("Replace MKSPIFFSTOOL with mklittlefs.exe")
00003 env.Replace (MKSPIFFSTOOL = "mklittlefs.exe")

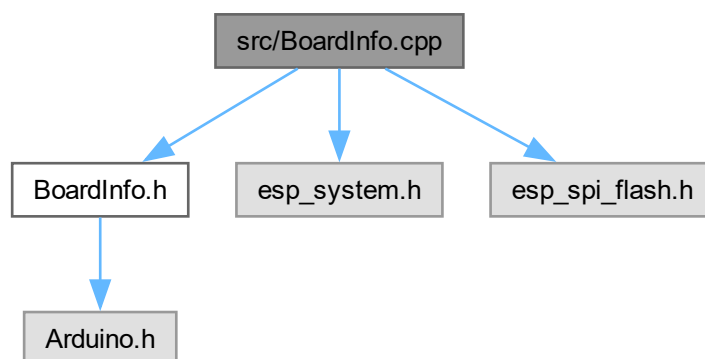
```

7.16 src/BoardInfo.cpp-Dateireferenz

Boardinfo.

```
#include "BoardInfo.h"  
#include <esp_system.h>  
#include <esp_spi_flash.h>
```

Include-Abhängigkeitsdiagramm für BoardInfo.cpp:



Makrodefinitionen

- `#define BUF 255`

Funktionen

- `uint8_t temprature_sens_read ()`

7.16.1 Ausführliche Beschreibung

Boardinfo.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei `BoardInfo.cpp`.

7.16.2 Makro-Dokumentation

7.16.2.1 BUF

```
#define BUF 255
```

Definiert in Zeile 29 der Datei [BoardInfo.cpp](#).

7.16.3 Dokumentation der Funktionen

7.16.3.1 temprature_sens_read()

```
uint8_t temprature_sens_read ()
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.17 BoardInfo.cpp

[gehe zur Dokumentation dieser Datei](#)

```

00001
00011
00012
00013 #include "BoardInfo.h"
00014 #include <esp_system.h>
00015 #include <esp_spi_flash.h>
00016
00017 #ifdef __cplusplus
00018     extern "C" {
00019 #endif
00020
00021     uint8_t temprature_sens_read();
00022
00023 #ifdef __cplusplus
00024 }
00025 #endif
00026
00027 uint8_t temprature_sens_read();
00028
00029 #define BUF 255
00030
00035
00036 BoardInfo::BoardInfo()
00037 {
00038     // Konstruktor der Klasse
00039     // ChipID auslesen
00040     //The chip ID is essentially its MAC address(length: 6 bytes).
00041     m_chipid = 0;
00042     m_chipid = ESP.getEfuseMac(); //The chip ID is essentially its MAC address(length: 6 bytes).
00043     // Chip - Info auslesen
00044     esp_chip_info(&m_chipinfo);
00045 }
00046
00047 void BoardInfo::ShowChipID()
00048 {

```



```

00049     if (m_chipid != 0)
00050     {
00051         Serial.printf("ESP32 Chip ID = %04X", (uint16_t) (m_chipid>>32));           //print High 2 bytes
00052         Serial.printf("%08X\n", (uint32_t)m_chipid);                             //print Low 4bytes.
00053     }
00054     else
00055     {
00056         // Fehler beim Lesen der ID....
00057         Serial.println("ESP32 Chip ID konnte nicht ausgelesen werden");
00058     }
00059 }
00060
00061 String BoardInfo::ShowChipIDtoString()
00062 {
00063     String msg;
00064     if (m_chipid != 0)
00065     {
00066         char string1[BUF];
00067         sprintf(string1, "ESP32 Chip ID = %04X%08X<br>", (uint16_t) (m_chipid>>32), (uint32_t)m_chipid);
00068         msg = (char*)string1;
00069         msg += "<br>";
00070         sprintf(string1, "%d CPU - Kerne<br>WLAN: %s<br>Bluetooth: %s%s",
00071             m_chipinfo.cores,
00072             (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
00073             (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
00074             (m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00075         msg += (char*)string1;
00076         msg += "<br>";
00077         sprintf(string1, "Silicon revision: %d", m_chipinfo.revision);
00078         msg += (char*)string1;
00079         msg += "<br>";
00080         sprintf(string1, "%s Speicher %dMB", (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ?
"embedded" : "external",
00081             spi_flash_get_chip_size() / (1024 * 1024));
00082
00083         msg += (char*)string1;
00084         msg += "<br>";
00085         sprintf(string1, "Freier Speicher: %d bytes", ESP.getFreeHeap());
00086         msg += (char*)string1;
00087         msg += "<br>";
00088         sprintf(string1, "Min freier Speicher: %d bytes", esp_get_minimum_free_heap_size());
00089         msg += (char*)string1;
00090         msg += "<br>";
00091     }
00092     else
00093     {
00094         // Fehler beim Lesen der ID....
00095         msg = "ESP32 Chip ID konnte nicht ausgelesen werden";
00096     }
00097     return msg;
00098 }
00099
00100 void BoardInfo::ShowChipInfo()
00101 {
00102     // Infos zum Board
00103     Serial.printf("Das ist ein Chip mit %d CPU - Kernen\nWLAN: %s\nBluetooth: %s%s\n",
00104         m_chipinfo.cores,
00105         (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
00106         (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
00107         (m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00108
00109     Serial.printf("Silicon revision %d\n", m_chipinfo.revision);
00110
00111     Serial.printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
00112         (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");
00113
00114     Serial.printf("(Freier Speicher: %d bytes)\n", esp_get_free_heap_size());
00115     Serial.printf("Freier Speicher: %d bytes\n", ESP.getFreeHeap());
00116     Serial.printf("Minimum freier Speicher: %d bytes\n", esp_get_minimum_free_heap_size());
00117 }
00118
00119 void BoardInfo::ShowChipTemperature()
00120 {
00121     uint8_t temp_fahrenheit;
00122     float temp_celsius;
00123     temp_fahrenheit = temprature_sens_read();
00124     if (128 == temp_fahrenheit)
00125     {
00126         Serial.println("Kein Temperatur - Sensor vorhanden.");
00127         return;
00128     }
00129     temp_celsius = (temp_fahrenheit - 32) / 1.8;
00130     Serial.printf("Temperatur Board: %i Fahrenheit\n", temp_fahrenheit);
00131     Serial.printf("Temperatur Board: %.1f °C\n", temp_celsius);
00132 }

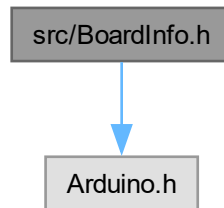
```

7.18 src/BoardInfo.h-Dateireferenz

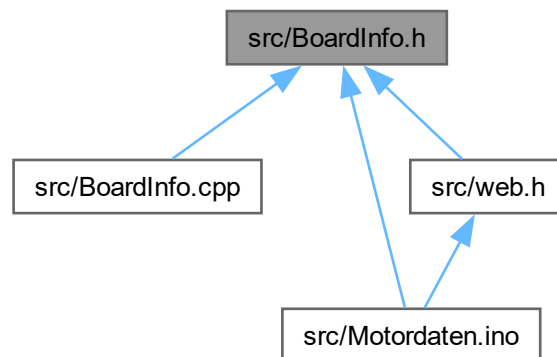
Hardwareinfo from ESP Board.

```
#include <Arduino.h>
```

Include-Abhängigkeitsdiagramm für BoardInfo.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class [BoardInfo](#)

7.18.1 Ausführliche Beschreibung

Hardwareinfo from ESP Board.

Autor

Gerry Sebb

Version

1.0

Datum

2024-01-22

Copyright

Copyright (c) 2024

Definiert in Datei [BoardInfo.h](#).

7.19 BoardInfo.h

[gehe zur Dokumentation dieser Datei](#)

```
00001 #ifndef _BoardInfo_H_
00002 #define _BoardInfo_H_
00003
00014 #include <Arduino.h>
00015
00016 class BoardInfo
00017 {
00018 public:
00019     BoardInfo();
00020
00021     void ShowChipID();
00022     void ShowChipInfo();
00023     void ShowChipTemperature();
00024
00025     String ShowChipIDtoString();
00026
00027 protected:
00028     uint64_t m_chipid;
00029     esp_chip_info_t m_chipinfo;
00030 };
00031
00032 #endif
```

7.20 src/BoatData.h-Dateireferenz

Klassen

- struct [tBoatData](#)

7.21 BoatData.h

[gehe zur Dokumentation dieser Datei](#)

```

00001 #ifndef _BoatData_H_
00002 #define _BoatData_H_
00003
00004 struct tBoatData {
00005     unsigned long DaysSince1970;    // Days since 1970-01-01
00006
00007     double TrueHeading, SOG, COG, Variation,
00008           GPSTime, // Secs since midnight,
00009           Latitude, Longitude, Altitude, HDOP, GeoidalSeparation, DGPSAge,
00010           WaterTemperature, WaterDepth, Offset,
00011           WindDirectionT, WindDirectionM, WindSpeedK, WindSpeedM,
00012           WindAngle ;
00013     int GPSQualityIndicator, SatelliteCount, DGPSReferenceStationID;
00014     bool MOBActivated;
00015     char Status;
00016
00017 public:
00018     tBoatData() {
00019         TrueHeading=0;
00020         SOG=0;
00021         COG=0;
00022         Variation=7.0;
00023         GPSTime=0;
00024         Latitude = 0;
00025         Longitude = 0;
00026         Altitude=0;
00027         HDOP=100000;
00028         DGPSAge=100000;
00029         WaterTemperature = 0;
00030         DaysSince1970=0;
00031         MOBActivated=false;
00032         SatelliteCount=0;
00033         DGPSReferenceStationID=0;
00034     };
00035 };
00036
00037 #endif // _BoatData_H_

```

7.22 src/configuration.h-Dateireferenz

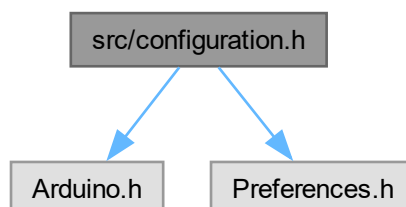
Konfiguration für GPIO und Variable.

```

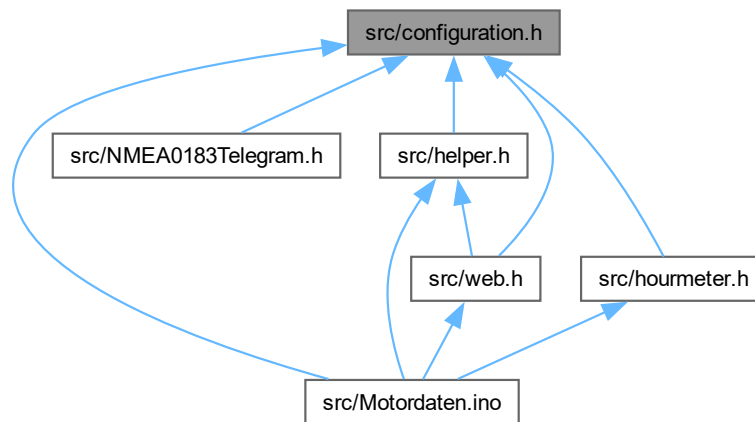
#include <Arduino.h>
#include <Preferences.h>

```

Include-Abhängigkeitsdiagramm für configuration.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- struct `Web_Config`

Makrodefinitionen

- `#define Version "V2.4 vom 21.01.2025"`
- `#define ESP32_CAN_TX_PIN GPIO_NUM_4`
Config NMEA2000.
- `#define ESP32_CAN_RX_PIN GPIO_NUM_5`
- `#define N2K_SOURCE 15`
- `#define EngineSendOffset 0`
- `#define TankSendOffset 40`
- `#define RPMSendOffset 80`
- `#define BatteryDCSendOffset 120`
- `#define BatteryDCStatusSendOffset 160`
- `#define SlowDataUpdatePeriod 1000`
- `#define PAGE_REFRESH 10`
- `#define WEB_TITEL "Motordaten"`
- `#define HostName "Motordaten"`
- `#define CL_SSID "NoWa"`
- `#define CL_PASSWORD "12345678"`
- `#define I2C_SDA 21`
- `#define I2C_SCL 22`
- `#define SEALEVELPRESSURE_HPA (1013.25)`
- `#define RPM_Calibration_Value 4.0`
- `#define Engine_RPM_Pin 19`
- `#define ONE_WIRE_BUS 13`
- `#define SERVER_HOST_NAME "192.168.30.15"`
- `#define TCP_PORT 6666`
- `#define DNS_PORT 53`

Aufzählungen

- enum [EngineStatus](#) { [Off](#) = 0 , [On](#) = 1 }

Variablen

- int [NodeAddress](#)
- Preferences [preferences](#)
- uint8_t [chipid](#) [6]
- uint32_t [id](#) = 0
- int [i](#) = 0
- [Web_Config](#) tAP_Config
- const int [channel](#) = 10
- const bool [hide_SSID](#) = false
- const int [max_connection](#) = 2
- IPAddress [IP](#) = IPAddress(192, 168, 15, 30)
- IPAddress [Gateway](#) = IPAddress(192, 168, 15, 30)
- IPAddress [NMask](#) = IPAddress(255, 255, 255, 0)
- const char * [AP_SSID](#) = "Motordaten"
- const char * [AP_PASSWORD](#) = "12345678"
- IPAddress [AP_IP](#)
- IPAddress [CL_IP](#)
- IPAddress [SELF_IP](#)
- String [sAP_Station](#) = ""
- int [iSTA_on](#) = 0
- int [bConnect_CL](#) = 0
- bool [bClientConnected](#) = 0
- float [fbmp_temperature](#) = 0
- float [fbmp_pressure](#) = 0
- float [fbmp_altitude](#) = 0
- String [sI2C_Status](#) = ""
- bool [bI2C_Status](#) = 0
- const int [iMaxSonar](#) = 35
- int [iDistance](#) = 0
- float [FuelLevel](#) = 0
- float [FuelLevelMax](#) = 30
- float [OilTemp](#) = 0
- float [MotTemp](#) = 0
- float [EngineRPM](#) = 0
- float [BordSpannung](#) = 0
- bool [EngineOn](#) = false
- static unsigned long [Counter](#)
- int [Bat1Capacity](#) = 55
- int [Bat2Capacity](#) = 90
- int [SoCError](#) = 0
- float [BatSoC](#) = 0
- String [sOneWire_Status](#) = ""
- float [fDrehzahl](#) = 0
- float [fGaugeDrehzahl](#) = 0
- float [fBordSpannung](#) = 0
- float [fOilTemp1](#) = 0
- float [fMotTemp2](#) = 0
- float [fTemp1Offset](#) = 0
- float [fTemp2Offset](#) = 0

- String `sSTBB` = ""
- String `sOrient` = ""
- double `dMWV_WindDirectionT` = 0
- double `dMWV_WindSpeedM` = 0
- double `dVWR_WindDirectionM` = 0
- double `dVWR_WindAngle` = 0
- double `dVWR_WindSpeedkn` = 0
- double `dVWR_WindSpeedms` = 0
- const char * `udpAddress` = "192.168.30.255"
- const int `udpPort` = 4444

7.22.1 Ausführliche Beschreibung

Konfiguration für GPIO und Variable.

Autor

Gerry Sebb

Version

2.3

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [configuration.h](#).

7.22.2 Makro-Dokumentation

7.22.2.1 Version

```
#define Version "V2.4 vom 21.01.2025"
```

Definiert in Zeile 19 der Datei [configuration.h](#).

7.22.2.2 ESP32_CAN_TX_PIN

```
#define ESP32_CAN_TX_PIN GPIO_NUM_4
```

Config NMEA2000.

Definiert in Zeile 25 der Datei [configuration.h](#).

7.22.2.3 ESP32_CAN_RX_PIN

```
#define ESP32_CAN_RX_PIN GPIO_NUM_5
```

Definiert in Zeile 26 der Datei [configuration.h](#).

7.22.2.4 N2K_SOURCE

```
#define N2K_SOURCE 15
```

Definiert in Zeile 27 der Datei [configuration.h](#).

7.22.2.5 EngineSendOffset

```
#define EngineSendOffset 0
```

Definiert in Zeile 33 der Datei [configuration.h](#).

7.22.2.6 TankSendOffset

```
#define TankSendOffset 40
```

Definiert in Zeile 34 der Datei [configuration.h](#).

7.22.2.7 RPMsSendOffset

```
#define RPMsSendOffset 80
```

Definiert in Zeile 35 der Datei [configuration.h](#).

7.22.2.8 BatteryDCSendOffset

```
#define BatteryDCSendOffset 120
```

Definiert in Zeile 36 der Datei [configuration.h](#).

7.22.2.9 BatteryDCStatusSendOffset

```
#define BatteryDCStatusSendOffset 160
```

Definiert in Zeile 37 der Datei [configuration.h](#).

7.22.2.10 SlowDataUpdatePeriod

```
#define SlowDataUpdatePeriod 1000
```

Definiert in Zeile 38 der Datei [configuration.h](#).

7.22.2.11 PAGE_REFRESH

```
#define PAGE_REFRESH 10
```

Definiert in Zeile 42 der Datei [configuration.h](#).

7.22.2.12 WEB_TITEL

```
#define WEB_TITEL "Motordaten"
```

Definiert in Zeile 43 der Datei [configuration.h](#).

7.22.2.13 HostName

```
#define HostName "Motordaten"
```

Definiert in Zeile 58 der Datei [configuration.h](#).

7.22.2.14 CL_SSID

```
#define CL_SSID "NoWa"
```

Definiert in Zeile 75 der Datei [configuration.h](#).

7.22.2.15 CL_PASSWORD

```
#define CL_PASSWORD "12345678"
```

Definiert in Zeile 76 der Datei [configuration.h](#).

7.22.2.16 I2C_SDA

```
#define I2C_SDA 21
```

Definiert in Zeile 82 der Datei [configuration.h](#).

7.22.2.17 I2C_SCL

```
#define I2C_SCL 22
```

Definiert in Zeile 83 der Datei [configuration.h](#).

7.22.2.18 SEALEVELPRESSURE_HPA

```
#define SEALEVELPRESSURE_HPA (1013.25)
```

Definiert in Zeile 84 der Datei [configuration.h](#).

7.22.2.19 RPM_Calibration_Value

```
#define RPM_Calibration_Value 4.0
```

Definiert in Zeile 104 der Datei [configuration.h](#).

7.22.2.20 Eengine_RPM_Pin

```
#define Eengine_RPM_Pin 19
```

Definiert in Zeile 105 der Datei [configuration.h](#).

7.22.2.21 ONE_WIRE_BUS

```
#define ONE_WIRE_BUS 13
```

Definiert in Zeile 114 der Datei [configuration.h](#).

7.22.2.22 SERVER_HOST_NAME

```
#define SERVER_HOST_NAME "192.168.30.15"
```

Definiert in Zeile 137 der Datei [configuration.h](#).

7.22.2.23 TCP_PORT

```
#define TCP_PORT 6666
```

Definiert in Zeile 138 der Datei [configuration.h](#).

7.22.2.24 DNS_PORT

```
#define DNS_PORT 53
```

Definiert in Zeile 139 der Datei [configuration.h](#).

7.22.3 Dokumentation der Aufzählungstypen

7.22.3.1 EngineStatus

```
enum EngineStatus
```

Aufzählungswerte

| | |
|-----|--|
| Off | |
| On | |

Definiert in Zeile 103 der Datei [configuration.h](#).

```
00103 { Off = 0, On = 1, };
```

7.22.4 Variablen-Dokumentation

7.22.4.1 NodeAddress

```
int NodeAddress
```

Definiert in Zeile 28 der Datei [configuration.h](#).

7.22.4.2 preferences

```
Preferences preferences
```

Definiert in Zeile 29 der Datei [configuration.h](#).

7.22.4.3 chipid

```
uint8_t chipid[6]
```

Definiert in Zeile 30 der Datei [configuration.h](#).

7.22.4.4 id

```
uint32_t id = 0
```

Definiert in Zeile 31 der Datei [configuration.h](#).

7.22.4.5 i

```
int i = 0
```

Definiert in Zeile 32 der Datei [configuration.h](#).

7.22.4.6 tAP_Config

```
Web_Config tAP_Config
```

Definiert in Zeile 55 der Datei [configuration.h](#).

7.22.4.7 channel

```
const int channel = 10
```

Definiert in Zeile 59 der Datei [configuration.h](#).

7.22.4.8 hide_SSID

```
const bool hide_SSID = false
```

Definiert in Zeile 60 der Datei [configuration.h](#).

7.22.4.9 max_connection

```
const int max_connection = 2
```

Definiert in Zeile 61 der Datei [configuration.h](#).

7.22.4.10 IP

```
IPAddress IP = IPAddress(192, 168, 15, 30)
```

Definiert in Zeile 64 der Datei [configuration.h](#).

7.22.4.11 Gateway

```
IPAddress Gateway = IPAddress(192, 168, 15, 30)
```

Definiert in Zeile 65 der Datei [configuration.h](#).

7.22.4.12 NMask

```
IPAddress NMask = IPAddress(255, 255, 255, 0)
```

Definiert in Zeile 66 der Datei [configuration.h](#).

7.22.4.13 AP_SSID

```
const char* AP_SSID = "Motordaten"
```

Definiert in Zeile 67 der Datei [configuration.h](#).

7.22.4.14 AP_PASSWORD

```
const char* AP_PASSWORD = "12345678"
```

Definiert in Zeile 68 der Datei [configuration.h](#).

7.22.4.15 AP_IP

```
IPAddress AP_IP
```

Definiert in Zeile 69 der Datei [configuration.h](#).

7.22.4.16 CL_IP

```
IPAddress CL_IP
```

Definiert in Zeile 70 der Datei [configuration.h](#).

7.22.4.17 SELF_IP

```
IPAddress SELF_IP
```

Definiert in Zeile 71 der Datei [configuration.h](#).

7.22.4.18 sAP_Station

```
String sAP_Station = ""
```

Definiert in Zeile 72 der Datei [configuration.h](#).

7.22.4.19 iSTA_on

```
int iSTA_on = 0
```

Definiert in Zeile 77 der Datei [configuration.h](#).

7.22.4.20 bConnect_CL

```
int bConnect_CL = 0
```

Definiert in Zeile 78 der Datei [configuration.h](#).

7.22.4.21 bClientConnected

```
bool bClientConnected = 0
```

Definiert in Zeile 79 der Datei [configuration.h](#).

7.22.4.22 fbmp_temperature

```
float fbmp_temperature = 0
```

Definiert in Zeile 85 der Datei [configuration.h](#).

7.22.4.23 fbmp_pressure

```
float fbmp_pressure = 0
```

Definiert in Zeile 86 der Datei [configuration.h](#).

7.22.4.24 fbmp_altitude

```
float fbmp_altitude = 0
```

Definiert in Zeile [87](#) der Datei [configuration.h](#).

7.22.4.25 sI2C_Status

```
String sI2C_Status = ""
```

Definiert in Zeile [88](#) der Datei [configuration.h](#).

7.22.4.26 bI2C_Status

```
bool bI2C_Status = 0
```

Definiert in Zeile [89](#) der Datei [configuration.h](#).

7.22.4.27 iMaxSonar

```
const int iMaxSonar = 35
```

Definiert in Zeile [91](#) der Datei [configuration.h](#).

7.22.4.28 iDistance

```
int iDistance = 0
```

Definiert in Zeile [92](#) der Datei [configuration.h](#).

7.22.4.29 FuelLevel

```
float FuelLevel = 0
```

Definiert in Zeile [95](#) der Datei [configuration.h](#).

7.22.4.30 FuelLevelMax

```
float FuelLevelMax = 30
```

Definiert in Zeile [96](#) der Datei [configuration.h](#).

7.22.4.31 OilTemp

```
float OilTemp = 0
```

Definiert in Zeile [97](#) der Datei [configuration.h](#).

7.22.4.32 MotTemp

```
float MotTemp = 0
```

Definiert in Zeile 98 der Datei [configuration.h](#).

7.22.4.33 EngineRPM

```
float EngineRPM = 0
```

Definiert in Zeile 99 der Datei [configuration.h](#).

7.22.4.34 BordSpannung

```
float BordSpannung = 0
```

Definiert in Zeile 100 der Datei [configuration.h](#).

7.22.4.35 EngineOn

```
bool EngineOn = false
```

Definiert in Zeile 101 der Datei [configuration.h](#).

7.22.4.36 Counter

```
unsigned long Counter [static]
```

Definiert in Zeile 102 der Datei [configuration.h](#).

7.22.4.37 Bat1Capacity

```
int Bat1Capacity = 55
```

Definiert in Zeile 108 der Datei [configuration.h](#).

7.22.4.38 Bat2Capacity

```
int Bat2Capacity = 90
```

Definiert in Zeile 109 der Datei [configuration.h](#).

7.22.4.39 SoCError

```
int SoCError = 0
```

Definiert in Zeile 110 der Datei [configuration.h](#).

7.22.4.40 BatSoC

```
float BatSoC = 0
```

Definiert in Zeile 111 der Datei [configuration.h](#).

7.22.4.41 sOneWire_Status

```
String sOneWire_Status = ""
```

Definiert in Zeile 115 der Datei [configuration.h](#).

7.22.4.42 fDrehzahl

```
float fDrehzahl = 0
```

Definiert in Zeile 118 der Datei [configuration.h](#).

7.22.4.43 fGaugeDrehzahl

```
float fGaugeDrehzahl = 0
```

Definiert in Zeile 119 der Datei [configuration.h](#).

7.22.4.44 fBordSpannung

```
float fBordSpannung = 0
```

Definiert in Zeile 120 der Datei [configuration.h](#).

7.22.4.45 fOilTemp1

```
float fOilTemp1 = 0
```

Definiert in Zeile 121 der Datei [configuration.h](#).

7.22.4.46 fMotTemp2

```
float fMotTemp2 = 0
```

Definiert in Zeile 122 der Datei [configuration.h](#).

7.22.4.47 fTemp1Offset

```
float fTemp1Offset = 0
```

Definiert in Zeile 123 der Datei [configuration.h](#).

7.22.4.48 fTemp2Offset

```
float fTemp2Offset = 0
```

Definiert in Zeile 124 der Datei [configuration.h](#).

7.22.4.49 sSTBB

```
String sSTBB = ""
```

Definiert in Zeile 125 der Datei [configuration.h](#).

7.22.4.50 sOrient

```
String sOrient = ""
```

Definiert in Zeile 126 der Datei [configuration.h](#).

7.22.4.51 dMWV_WindDirectionT

```
double dMWV_WindDirectionT = 0
```

Definiert in Zeile 129 der Datei [configuration.h](#).

7.22.4.52 dMWV_WindSpeedM

```
double dMWV_WindSpeedM = 0
```

Definiert in Zeile 130 der Datei [configuration.h](#).

7.22.4.53 dVWR_WindDirectionM

```
double dVWR_WindDirectionM = 0
```

Definiert in Zeile 131 der Datei [configuration.h](#).

7.22.4.54 dVWR_WindAngle

```
double dVWR_WindAngle = 0
```

Definiert in Zeile 132 der Datei [configuration.h](#).

7.22.4.55 dVWR_WindSpeedkn

```
double dVWR_WindSpeedkn = 0
```

Definiert in Zeile 133 der Datei [configuration.h](#).

7.22.4.56 dVWR_WindSpeedms

```
double dVWR_WindSpeedms = 0
```

Definiert in Zeile 134 der Datei [configuration.h](#).

7.22.4.57 udpAddress

```
const char* udpAddress = "192.168.30.255"
```

Definiert in Zeile 142 der Datei [configuration.h](#).

7.22.4.58 udpPort

```
const int udpPort = 4444
```

Definiert in Zeile 143 der Datei [configuration.h](#).

7.23 configuration.h

[gehe zur Dokumentation dieser Datei](#)

```
00001 #ifndef __configuration_H__
00002 #define __configuration_H__
00003
00014
00015 #include <Arduino.h>
00016 #include <Preferences.h>
00017
00018 // Versionierung
00019 #define Version "V2.4 vom 21.01.2025" // Version
00020
00025 #define ESP32_CAN_TX_PIN GPIO_NUM_4 // Set CAN TX port to 4
00026 #define ESP32_CAN_RX_PIN GPIO_NUM_5 // Set CAN RX port to 5
00027 #define N2K_SOURCE 15
00028 int NodeAddress; // To store Last Node Address
00029 Preferences preferences; // Nonvolatile storage on ESP32 - To store LastDeviceAddress
00030 uint8_t chipid[6];
00031 uint32_t id = 0;
00032 int i = 0;
00033 #define EngineSendOffset 0
00034 #define TankSendOffset 40
00035 #define RPMSendOffset 80
00036 #define BatteryDCSendOffset 120
00037 #define BatteryDCStatusSendOffset 160
00038 #define SlowDataUpdatePeriod 1000 // Time between CAN Messages sent
00039
00040
00041 //Configuration Website
00042 #define PAGE_REFRESH 10 // x Sec.
00043 #define WEB_TITEL "Motordaten"
00044
00045 //Configuration mit Webinterface
00046 struct Web_Config
00047 {
00048     char wAP_IP[20];
00049     char wAP_SSID[64];
00050     char wAP_Password[12];
00051     char wTempl_Offset[5];
00052     char wTemp2_Offset[5];
00053     char wFuellstandmax[3];
00054 };
00055 Web_Config tAP_Config;
00056
00057 //Configuration AP
00058 #define HostName "Motordaten"
00059 const int channel = 10; // WiFi Channel number between 1 and 13
```

```

00060 const bool   hideSSID      = false;           // To disable SSID broadcast -> SSID will not appear
        in a basic WiFi scan
00061 const int     max_connection = 2;              // Maximum simultaneous connected clients on the AP
00062
00063 // Variables for WIFI-AP
00064 IPAddress IP = IPAddress(192, 168, 15, 30);
00065 IPAddress Gateway = IPAddress(192, 168, 15, 30);
00066 IPAddress NMask = IPAddress(255, 255, 255, 0);
00067 const char* AP_SSID = "Motordaten";
00068 const char* AP_PASSWORD = "12345678";
00069 IPAddress AP_IP;
00070 IPAddress CL_IP;
00071 IPAddress SELF_IP;
00072 String sAP_Station = "";
00073
00074 //Configuration Client (Network Data Windsensor)
00075 #define CL_SSID "NoWa" //Windmesser
00076 #define CL_PASSWORD "12345678"
00077 int iSTA_on = 0; // Status STA-Mode
00078 int bConnect_CL = 0;
00079 bool bClientConnected = 0;
00080
00081 //Confuration Sensors I2C
00082 #define I2C_SDA 21 //Standard 21
00083 #define I2C_SCL 22 //Standard 22
00084 #define SEALEVELPRESSURE_HPA (1013.25) //1013.25
00085 float fbmp_temperature = 0;
00086 float fbmp_pressure = 0;
00087 float fbmp_altitude = 0;
00088 String sI2C_Status = "";
00089 bool bI2C_Status = 0;
00090 // Global Data Sonar
00091 const int iMaxSonar = 35; //Analoginput
00092 int iDistance = 0;
00093
00094 // Global Data Motordata Sensor
00095 float FuelLevel = 0;
00096 float FuelLevelMax = 30;
00097 float OilTemp = 0;
00098 float MotTemp = 0;
00099 float EngineRPM = 0;
00100 float BordSpannung = 0;
00101 bool EngineOn = false;
00102 static unsigned long Counter; // Enginehours
00103 enum EngineStatus { Off = 0, On = 1, };
00104 #define RPM_Calibration_Value 4.0 // Translates Generator RPM to Engine RPM
00105 #define Eingine_RPM_Pin 19 // Engine RPM is measured as interrupt on GPIO 23
00106
00107 // Global Data Battery
00108 int Bat1Capacity = 55; // Starterbatterie
00109 int Bat2Capacity = 90; // Versorgerbatterie
00110 int SoCError = 0;
00111 float BatSoC = 0;
00112
00113 // Data wire for teperature (Dallas DS18B20)
00114 #define ONE_WIRE_BUS 13 // Data wire for teperature (Dallas DS18B20) is plugged into GPIO 13
00115 String sOneWire_Status = "";
00116
00117 // Variables Website
00118 float fDrehzahl = 0;
00119 float fGaugeDrehzahl = 0;
00120 float fBordSpannung = 0;
00121 float fOilTemp1 = 0;
00122 float fMotTemp2 = 0;
00123 float fTemp1Offset = 0;
00124 float fTemp2Offset = 0;
00125 String sSTBB = "";
00126 String sOrient = "";
00127
00128 //Definiton NMEA0183 MWV
00129 double dMWV_WindDirectionT = 0;
00130 double dMWV_WindSpeedM = 0;
00131 double dVWR_WindDirectionM = 0;
00132 double dVWR_WindAngle = 0;
00133 double dVWR_WindSpeedkn = 0;
00134 double dVWR_WindSpeedms = 0;
00135
00136 //Configuration NMEA0183
00137 #define SERVER_HOST_NAME "192.168.30.15" // "192.168.76.34"
00138 #define TCP_PORT 6666 //6666
00139 #define DNS_PORT 53
00140
00141 //Variable NMEA 0183 Stream
00142 const char *udpAddress = "192.168.30.255"; // Set network address for broadcast
00143 const int udpPort = 4444; // UDP port
00144
00145 #endif

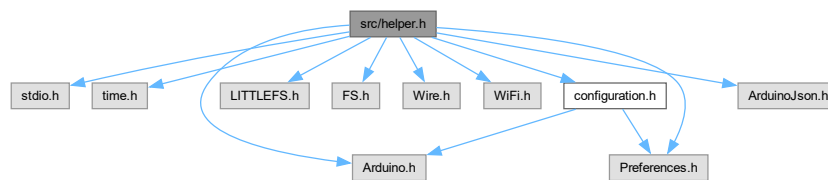
```

7.24 src/helper.h-Dateireferenz

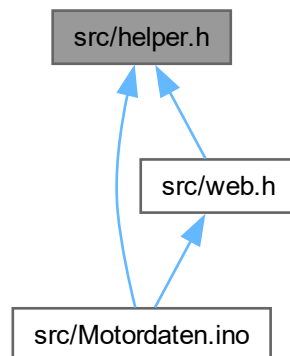
Hilfsfunktionen.

```
#include <stdio.h>
#include <time.h>
#include <Arduino.h>
#include <LITTLEFS.h>
#include <FS.h>
#include <Wire.h>
#include <WiFi.h>
#include "configuration.h"
#include <ArduinoJson.h>
#include <Preferences.h>
```

Include-Abhängigkeitsdiagramm für helper.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void `ShowTime` ()
- void `freeHeapSpace` ()
- void `WiFiDiag` (void)
- void `listDir` (fs::FS &fs, const char *dirname, uint8_t levels)

LittleFS, Dateien auflisten.

- void [readConfig](#) (String filename)
Konfiguration aus Json-Datei lesen.
- bool [writeConfig](#) (String json)
Webseiten Eingabe in Json-Datei schreiben.
- void [l2C_scan](#) (void)
- String [sWifiStatus](#) (int Status)
WIFI Status lesen.
- char * [toChar](#) (String command)
Convert string to char.

7.24.1 Ausführliche Beschreibung

Hilfsfunktionen.

Autor

Gerry Sebb

Version

1.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [helper.h](#).

7.24.2 Dokumentation der Funktionen

7.24.2.1 ShowTime()

```
void ShowTime ()
```

Definiert in Zeile [27](#) der Datei [helper.h](#).

```
00027     {
00028     time_t now = time(NULL);
00029     struct tm tm_now;
00030     localtime_r(&now, &tm_now);
00031     char buff[100];
00032     strftime(buff, sizeof(buff), "%d-%m-%Y %H:%M:%S", &tm_now);
00033     printf("Zeit: %s\n", buff);
00034 }
```

7.24.2.2 freeHeapSpace()

```
void freeHeapSpace ()
```

Freie Speichergroesse aller 5s lesen

Definiert in Zeile 37 der Datei [helper.h](#).

```
00037     {
00038         static unsigned long last = millis();
00039         if (millis() - last > 5000) {
00040             last = millis();
00041             Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00042         }
00043     }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.3 WiFiDiag()

```
void WiFiDiag (
    void )
```

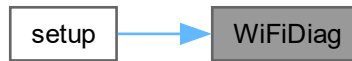
Ausgabe WIFI Parameter und Netzwerk scannen

Definiert in Zeile 46 der Datei [helper.h](#).

```
00046     {
00047         Serial.println("\nWifi-Diag:");
00048         AP_IP = WiFi.softAPIP();
00049         CL_IP = WiFi.localIP();
00050         Serial.print("AP IP address: ");
00051         Serial.println(AP_IP.toString());
00052         Serial.print("Client IP address: ");
00053         Serial.println(CL_IP.toString());
00054         WiFi.printDiag(Serial);
00055         Serial.print("\nScan AP's ");
00056         {
00057             // WiFi.scanNetworks will return the number of networks found
00058             int n = WiFi.scanNetworks();
00059             Serial.println("scan done");
00060             if (n == 0) {
00061                 Serial.println("no networks found");
00062             } else {
00063                 Serial.print(n);
00064                 Serial.println(" networks found");
00065                 for (int i = 0; i < n; ++i)
00066                 {
00067                     // Print SSID and RSSI for each network found
00068                     Serial.print(i + 1);
00069                     Serial.print(": ");
00070                     Serial.print(WiFi.SSID(i));
00071                     Serial.print(" ");
00072                     Serial.print(WiFi.RSSI(i));
00073                     Serial.print(" ");
00074                     Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? " " : "*");
00075                     delay(10);
00076                 }
00077             }
```

```
00078     }
00079 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.4 listDir()

```
void listDir (
    fs::FS & fs,
    const char * dirname,
    uint8_t levels)
```

LittleFS, Dateien auflisten.

Parameter

| | |
|----------------|--|
| <i>fs</i> | |
| <i>dirname</i> | |
| <i>levels</i> | |

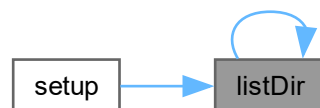
Definiert in Zeile 91 der Datei [helper.h](#).

```
00091                                     {
00092     Serial.printf("Listing directory: %s\r\n", dirname);
00093
00094     File root = fs.open(dirname);
00095     if(!root){
00096         Serial.println("- failed to open directory");
00097         return;
00098     }
00099     if(!root.isDirectory()){
00100         Serial.println(" - not a directory");
00101         return;
00102     }
00103
00104     File file = root.openNextFile();
00105     while(file){
00106         if(file.isDirectory()){
00107             Serial.print("  DIR : ");
00108             Serial.println(file.name());
00109             if(levels){
00110                 listDir(fs, file.path(), levels -1);
00111             }
00112         } else {
00113             Serial.print("  FILE: ");
00114             Serial.print(file.name());
00115             Serial.print("\tSIZE: ");
00116             Serial.println(file.size());
00117         }
00118         file = root.openNextFile();
00119     }
00120 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.5 readConfig()

```
void readConfig (
    String filename)
```

Konfiguration aus Json-Datei lesen.

Parameter

| | |
|-----------------|--|
| <i>filename</i> | |
|-----------------|--|

Definiert in Zeile 128 der Datei [helper.h](#).

```

00128     {
00129         JsonDocument testDocument;
00130         File configFile = LittleFS.open(filename);
00131         if (configFile)
00132         {
00133             Serial.println("opened config file");
00134             DeserializationError error = deserializeJson(testDocument, configFile);
00135
00136             // Test if parsing succeeds.
00137             if (error)
00138             {
00139                 Serial.print(F("deserializeJson() failed: "));
00140                 Serial.println(error.f_str());
00141                 return;
00142             }
00143
00144             Serial.println("deserializeJson ok");
00145             {
00146                 Serial.println("Lese Daten aus Config - Datei");
00147                 strcpy(tAP_Config.wAP_SSID, testDocument["SSID"] | "Motordaten");
  
```



```

00148         strcpy(tAP_Config.wAP_IP, testDocument["IP"] | "192.168.15.30");
00149         strcpy(tAP_Config.wAP_Password, testDocument["Password"] | "12345678");
00150         strcpy(tAP_Config.wTemp1_Offset, testDocument["Temp1Offset"] | "0.0");
00151         strcpy(tAP_Config.wTemp2_Offset, testDocument["Temp2Offset"] | "0.0");
00152         strcpy(tAP_Config.wFuellstandmax, testDocument["Fuellstandmax"] | "0.0");
00153     }
00154     configFile.close();
00155     Serial.println("Config - Datei geschlossen");
00156 }
00157
00158 else
00159 {
00160     Serial.println("failed to load json config");
00161 }
00162 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.6 writeConfig()

```

bool writeConfig (
    String json)

```

Webseiten Eingabe in Json-Datei schreiben.

Parameter

| | |
|------|--|
| json | |
|------|--|

Rückgabe

true

false

Definiert in Zeile 172 der Datei [helper.h](#).

```

00173 {
00174     Serial.println(json);
00175
00176     Serial.println("neue Konfiguration speichern");
00177
00178     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00179     if (configFile)
00180     {
00181         Serial.println("Config - Datei öffnen");
00182         File configFile = LittleFS.open("/config.json", FILE_WRITE);
00183         if (configFile)
00184         {
00185             Serial.println("Config - Datei zum Schreiben geöffnet");
00186             JsonDocument testDocument;
00187             Serial.println("JSON - Daten übergeben");
00188             DeserializationError error = deserializeJson(testDocument, json);
00189             // Test if parsing succeeds.
00190             if (error)
00191             {

```

```

00192         Serial.print(F("deserializeJson() failed: "));
00193         Serial.println(error.f_str());
00194         // bei Memory - Fehler den <Wert> in StaticJsonDocument<200> testDocument; erhöhen
00195         return false;
00196     }
00197     Serial.println("Konfiguration schreiben...");
00198     serializeJson(testDocument, configFile);
00199     Serial.println("Konfiguration geschrieben...");
00200
00201     // neue Config in Serial ausgeben zur Kontrolle
00202     serializeJsonPretty(testDocument, Serial);
00203
00204     Serial.println("Config - Datei geschlossen");
00205     configFile.close();
00206 }
00207 }
00208 return true;
00209 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.7 I2C_scan()

```

void I2C_scan (
    void )

```

I2C Bus auslesen, alle Geräte mit Adresse ausgegeben

Definiert in Zeile 214 der Datei [helper.h](#).

```

00214     {
00215     byte error, address;
00216     int nDevices;
00217     Serial.println("Scanning...");
00218     nDevices = 0;
00219     for(address = 1; address < 127; address++ )
00220     {
00221         Wire.beginTransmission(address);
00222         error = Wire.endTransmission();
00223         if (error == 0)
00224         {
00225             Serial.print("I2C device found at address 0x");
00226             if (address<16)
00227             {
00228                 Serial.print("0");
00229             }
00230             Serial.println(address,HEX);
00231             nDevices++;
00232         }
00233         else if (error==4)
00234         {
00235             Serial.print("Unknow error at address 0x");
00236             if (address<16)
00237             {
00238                 Serial.print("0");
00239             }
00240             Serial.println(address,HEX);
00241             nDevices++;
00242         }
00243         else if (error==4) {
00244             Serial.print("Unknow error at address 0x");
00245             if (address<16) {
00246                 Serial.print("0");

```

```

00247     }
00248     Serial.println(address, HEX);
00249 }
00250 }
00251 if (nDevices == 0) {
00252     Serial.println("No I2C devices found\n");
00253 }
00254 else {
00255     Serial.println("done\n");
00256 }
00257 }

```

7.24.2.8 sWifiStatus()

```

String sWifiStatus (
    int Status)

```

WIFI Status lesen.

Parameter

| | |
|--------|--|
| Status | |
|--------|--|

Rückgabe

String

Definiert in Zeile 266 der Datei [helper.h](#).

```

00267 {
00268     switch(Status){
00269         case WL_IDLE_STATUS: return "Warten";
00270         case WL_NO_SSID_AVAIL: return "Keine SSID vorhanden";
00271         case WL_SCAN_COMPLETED: return "Scan komplett";
00272         case WL_CONNECTED: return "Verbunden";
00273         case WL_CONNECT_FAILED: return "Verbindung fehlerhaft";
00274         case WL_CONNECTION_LOST: return "Verbindung verloren";
00275         case WL_DISCONNECTED: return "Nicht verbunden";
00276         default: return "unbekannt";
00277     }
00278 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.24.2.9 toChar()

```

char * toChar (
    String command)

```

Convert string to char.

Parameter

| | |
|----------------|--|
| <i>command</i> | |
|----------------|--|

Rückgabe

char*

Definiert in Zeile 287 der Datei [helper.h](#).

```

00287     {
00288         if(command.length() != 0) {
00289             char *p = const_cast<char*>(command.c_str());
00290             return p;
00291         }
00292         else{
00293             return 0;
00294         }
00295     }

```

7.25 helper.h

[gehe zur Dokumentation dieser Datei](#)

```

00001 #ifndef _HELPER_H_
00002 #define _HELPER_H_
00003
00014
00015
00016 #include <stdio.h>
00017 #include <time.h>
00018 #include <Arduino.h>
00019 #include <LITTLEFS.h>
00020 #include <FS.h>
00021 #include <Wire.h>
00022 #include <WiFi.h>
00023 #include "configuration.h"
00024 #include <ArduinoJson.h>
00025 #include <Preferences.h>
00026
00027 void ShowTime() {
00028     time_t now = time(NULL);
00029     struct tm tm_now;
00030     localtime_r(&now, &tm_now);
00031     char buff[100];
00032     strftime(buff, sizeof(buff), "%d-%m-%Y %H:%M:%S", &tm_now);
00033     printf("Zeit: %s\n", buff);
00034 }
00035
00037 void freeHeapSpace() {
00038     static unsigned long last = millis();
00039     if (millis() - last > 5000) {
00040         last = millis();
00041         Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00042     }
00043 }
00044
00046 void WiFiDiag(void) {
00047     Serial.println("\nWifi-Diag:");
00048     AP_IP = WiFi.softAPIP();
00049     CL_IP = WiFi.localIP();
00050     Serial.print("AP IP address: ");
00051     Serial.println(AP_IP.toString());
00052     Serial.print("Client IP address: ");
00053     Serial.println(CL_IP.toString());
00054     WiFi.printDiag(Serial);
00055     Serial.print("\nScan AP's ");
00056     {
00057         // WiFi.scanNetworks will return the number of networks found
00058         int n = WiFi.scanNetworks();
00059         Serial.println("scan done");
00060         if (n == 0) {
00061             Serial.println("no networks found");
00062         } else {
00063             Serial.print(n);

```

```

00064     Serial.println(" networks found");
00065     for (int i = 0; i < n; ++i)
00066     {
00067         // Print SSID and RSSI for each network found
00068         Serial.print(i + 1);
00069         Serial.print(": ");
00070         Serial.print(WiFi.SSID(i));
00071         Serial.print(" ");
00072         Serial.print(WiFi.RSSI(i));
00073         Serial.print(" ");
00074         Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
00075         delay(10);
00076     }
00077 }
00078 }
00079 }
00080
00081 /***** Filesystem *****/
00082
00083
00091 void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
00092     Serial.printf("Listing directory: %s\r\n", dirname);
00093
00094     File root = fs.open(dirname);
00095     if(!root){
00096         Serial.println("- failed to open directory");
00097         return;
00098     }
00099     if(!root.isDirectory()){
00100         Serial.println(" - not a directory");
00101         return;
00102     }
00103
00104     File file = root.openNextFile();
00105     while(file){
00106         if(file.isDirectory()){
00107             Serial.print(" DIR : ");
00108             Serial.println(file.name());
00109             if(levels){
00110                 listDir(fs, file.path(), levels -1);
00111             }
00112         } else {
00113             Serial.print(" FILE: ");
00114             Serial.print(file.name());
00115             Serial.print("\tSIZE: ");
00116             Serial.println(file.size());
00117         }
00118         file = root.openNextFile();
00119     }
00120 }
00121
00122
00128 void readConfig(String filename) {
00129     JsonDocument testDocument;
00130     File configFile = LittleFS.open(filename);
00131     if (configFile)
00132     {
00133         Serial.println("opened config file");
00134         DeserializationError error = deserializeJson(testDocument, configFile);
00135
00136         // Test if parsing succeeds.
00137         if (error)
00138         {
00139             Serial.print(F("deserializeJson() failed: "));
00140             Serial.println(error.f_str());
00141             return;
00142         }
00143
00144         Serial.println("deserializeJson ok");
00145         {
00146             Serial.println("Lese Daten aus Config - Datei");
00147             strcpy(tAP_Config.wAP_SSID, testDocument["SSID"] | "Motordaten");
00148             strcpy(tAP_Config.wAP_IP, testDocument["IP"] | "192.168.15.30");
00149             strcpy(tAP_Config.wAP_Password, testDocument["Password"] | "12345678");
00150             strcpy(tAP_Config.wTemp_Offset, testDocument["TempOffset"] | "0.0");
00151             strcpy(tAP_Config.wTemp2_Offset, testDocument["Temp2Offset"] | "0.0");
00152             strcpy(tAP_Config.wFuellstandmax, testDocument["Fuellstandmax"] | "0.0");
00153         }
00154         configFile.close();
00155         Serial.println("Config - Datei geschlossen");
00156     }
00157
00158     else
00159     {
00160         Serial.println("failed to load json config");
00161     }
00162 }

```

```

00163
00171
00172 bool writeConfig(String json)
00173 {
00174     Serial.println(json);
00175
00176     Serial.println("neue Konfiguration speichern");
00177
00178     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00179     if (configFile)
00180     {
00181         Serial.println("Config - Datei öffnen");
00182         File configFile = LittleFS.open("/config.json", FILE_WRITE);
00183         if (configFile)
00184         {
00185             Serial.println("Config - Datei zum Schreiben geöffnet");
00186             JsonDocument testDocument;
00187             Serial.println("JSON - Daten übergeben");
00188             DeserializationError error = deserializeJson(testDocument, json);
00189             // Test if parsing succeeds.
00190             if (error)
00191             {
00192                 Serial.print(F("deserializeJson() failed: "));
00193                 Serial.println(error.f_str());
00194                 // bei Memory - Fehler den <Wert> in StaticJsonDocument<200> testDocument; erhöhen
00195                 return false;
00196             }
00197             Serial.println("Konfiguration schreiben...");
00198             serializeJson(testDocument, configFile);
00199             Serial.println("Konfiguration geschrieben...");
00200
00201             // neue Config in Serial ausgeben zur Kontrolle
00202             serializeJsonPretty(testDocument, Serial);
00203
00204             Serial.println("Config - Datei geschlossen");
00205             configFile.close();
00206         }
00207     }
00208     return true;
00209 }
00210
00211 /***** I2C Bus *****/
00213
00214 void I2C_scan(void){
00215     byte error, address;
00216     int nDevices;
00217     Serial.println("Scanning...");
00218     nDevices = 0;
00219     for(address = 1; address < 127; address++ )
00220     {
00221         Wire.beginTransmission(address);
00222         error = Wire.endTransmission();
00223         if (error == 0)
00224         {
00225             Serial.print("I2C device found at address 0x");
00226             if (address<16)
00227             {
00228                 Serial.print("0");
00229             }
00230             Serial.println(address,HEX);
00231             nDevices++;
00232         }
00233         else if (error==4)
00234         {
00235             Serial.print("Unknow error at address 0x");
00236             if (address<16)
00237             {
00238                 Serial.print("0");
00239             }
00240             Serial.println(address,HEX);
00241             nDevices++;
00242         }
00243         else if (error==4) {
00244             Serial.print("Unknow error at address 0x");
00245             if (address<16) {
00246                 Serial.print("0");
00247             }
00248             Serial.println(address,HEX);
00249         }
00250     }
00251     if (nDevices == 0) {
00252         Serial.println("No I2C devices found\n");
00253     }
00254     else {
00255         Serial.println("done\n");
00256     }
00257 }

```

```

00258
00265
00266 String sWifiStatus(int Status)
00267 {
00268     switch(Status){
00269         case WL_IDLE_STATUS: return "Warten";
00270         case WL_NO_SSID_AVAIL: return "Keine SSID vorhanden";
00271         case WL_SCAN_COMPLETED: return "Scan komplett";
00272         case WL_CONNECTED: return "Verbunden";
00273         case WL_CONNECT_FAILED: return "Verbindung fehlerhaft";
00274         case WL_CONNECTION_LOST: return "Verbindung verloren";
00275         case WL_DISCONNECTED: return "Nicht verbunden";
00276         default: return "unbekannt";
00277     }
00278 }
00279
00286
00287 char* toChar(String command){
00288     if(command.length() != 0){
00289         char *p = const_cast<char*>(command.c_str());
00290         return p;
00291     }
00292     else{
00293         return 0;
00294     }
00295 }
00296
00297
00298 #endif

```

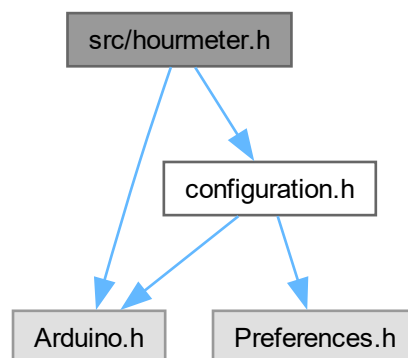
7.26 src/hourmeter.h-Dateireferenz

Betriebstundenzähler.

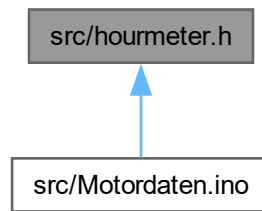
```
#include <Arduino.h>
```

```
#include "configuration.h"
```

Include-Abhängigkeitsdiagramm für hourmeter.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- unsigned long [EngineHours](#) (bool CountOn=0)
Betriebstundenzähler Berechnet Betriebstunden, wenn Anlage eingeschaltet ist.

Variablen

- Preferences [bsz1](#)
- static unsigned long [lastRun](#)
- static unsigned long [CounterOld](#)
- static unsigned long [milliRest](#)
- int [state1](#) = LOW
- int [laststate1](#) = LOW

7.26.1 Ausführliche Beschreibung

Betriebstundenzähler.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [hourmeter.h](#).

7.26.2 Dokumentation der Funktionen

7.26.2.1 EngineHours()

```
unsigned long EngineHours (  
    bool CountOn = 0)
```

Betriebsstundenzähler Berechnet Betriebsstunden, wenn Anlage eingeschaltet ist.

Parameter

| | |
|---------|--|
| CountOn | |
|---------|--|

Rückgabe

unsigned long

< speichern bei Flanke negativ

< NVS nutzen, BSZ erstellen, lesen und schreiben (false)

< Speicher auslesen

< Laufzeit alt + aktuell

< Speicher schreiben

< Preferences beenden

Definiert in Zeile 30 der Datei [hourmeter.h](#).

```

00030                                     {
00031     {
00032         long now = millis();
00033         milliRest += now - lastRun;
00034         if (CountOn == 1)
00035         {
00036             while (milliRest>=1000){
00037                 Counter++;
00038                 milliRest-=1000;
00039             }
00040         }
00041         else milliRest=0;
00042         lastRun = now;
00043         return Counter;
00044     }
00045     statel = CountOn;
00046     if (laststatel == HIGH && statel == LOW)
00047     {
00048         bsz1.begin("bsz", false);
00049         CounterOld = preferences.getUInt("Start", 0);
00050         Counter = CounterOld + Counter;
00051         bsz1.putUInt("Start", Counter);
00052         bsz1.end();
00053         statel = LOW;
00054     }
00055 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.26.3 Variablen-Dokumentation

7.26.3.1 bsz1

Preferences bsz1

Definiert in Zeile 18 der Datei [hourmeter.h](#).

7.26.3.2 lastRun

```
unsigned long lastRun [static]
```

Definiert in Zeile 20 der Datei [hourmeter.h](#).

7.26.3.3 CounterOld

```
unsigned long CounterOld [static]
```

Definiert in Zeile 20 der Datei [hourmeter.h](#).

7.26.3.4 milliRest

```
unsigned long milliRest [static]
```

Definiert in Zeile 20 der Datei [hourmeter.h](#).

7.26.3.5 state1

```
int state1 = LOW
```

Definiert in Zeile 21 der Datei [hourmeter.h](#).

7.26.3.6 laststate1

```
int laststate1 = LOW
```

Definiert in Zeile 21 der Datei [hourmeter.h](#).

7.27 hourmeter.h

[gehe zur Dokumentation dieser Datei](#)

```
00001 #ifndef _HOURMETER_H_
00002 #define _HOURMETER_H_
00003
00014
00015 #include <Arduino.h>
00016 #include "configuration.h"
00017
00018 Preferences bsz1;
00019
00020 static unsigned long lastRun, CounterOld, milliRest;
00021 int state1 = LOW, laststate1 = LOW;
00022
00029
00030 unsigned long EngineHours(bool CountOn = 0){
00031     {
00032         long now = millis();
00033         milliRest += now - lastRun;
00034         if (CountOn == 1)
00035             {
00036                 while (milliRest>=1000){
00037                     Counter++;
00038                     milliRest-=1000;
00039                 }
```

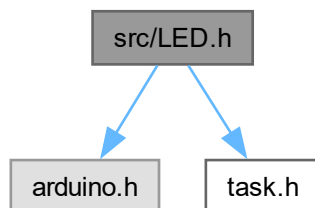
```
00040     }  
00041     else milliRest=0;  
00042     lastRun = now;  
00043     return Counter;  
00044 }  
00045 statel = CountOn;  
00046 if (laststatel == HIGH && statel == LOW)  
00047 {  
00048     bsz1.begin("bsz", false);  
00049     CounterOld = preferences.getUInt("Start", 0);  
00050     Counter = CounterOld + Counter;  
00051     bsz1.putUInt("Start", Counter);  
00052     bsz1.end();  
00053     statel = LOW;  
00054 }  
00055 }  
00056  
00057 #endif
```

7.28 src/LED.h-Dateireferenz

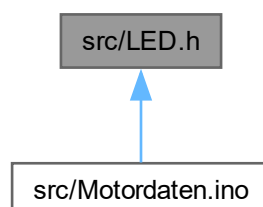
LED Ansteuerung.

```
#include <arduino.h>  
#include "task.h"
```

Include-Abhängigkeitsdiagramm für LED.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Aufzählungen

- enum `LED` { `Red` = 25 , `Green` = 26 , `Blue` = 33 , `LEDBoard` = 13 }

Funktionen

- void `LEDblink` (int PIN=`LED`())
- void `LEDflash` (int PIN=`LED`())
- void `flashLED` (int PIN=`LED`())
- void `LEDInit` ()
Start Initialisierung LEDtest.
- void `LEDon` (int PIN=`LED`())
- void `LEDOff` (int PIN=`LED`())
- void `LEDOff_RGB` ()

7.28.1 Ausführliche Beschreibung

`LED` Ansteuerung.

Autor

Gerry Sebb

Version

2.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei `LED.h`.

7.28.2 Dokumentation der Aufzählungstypen

7.28.2.1 LED

enum `LED`

Aufzählungswerte

| | |
|----------|--|
| Red | |
| Green | |
| Blue | |
| LEDBoard | |

Definiert in Zeile 19 der Datei `LED.h`.

```
00019     {
00020     Red = 25,
00021     Green = 26,
00022     Blue = 33,
00023     LEDBoard = 13 //Adafruit Huzzah32
00024     };
```

7.28.3 Dokumentation der Funktionen

7.28.3.1 LEDblink()

```
void LEDblink (
    int PIN = LED())
```

Definiert in Zeile 26 der Datei LED.h.

```
00026                                     {
00027     taskBegin();
00028     while(1)    // blockiert dank der TaskPause nicht
00029     {
00030         digitalWrite(PIN,HIGH); // LED ein
00031         taskPause(250);    // gibt Rechenzeit ab
00032         digitalWrite(PIN,LOW); // LED aus
00033         taskPause(1000);    // gibt Rechenzeit ab
00034     }
00035     taskEnd();
00036 }
```

7.28.3.2 LEDflash()

```
void LEDflash (
    int PIN = LED())
```

Definiert in Zeile 38 der Datei LED.h.

```
00038                                     {
00039     taskBegin();
00040     while(1)    // blockiert dank der TaskPause nicht
00041     {
00042         digitalWrite(PIN,HIGH); // LED ein
00043         delay (5);
00044         //taskPause(2);    // gibt Rechenzeit ab
00045         digitalWrite(PIN,LOW); // LED aus
00046         taskPause(3000);    // gibt Rechenzeit ab
00047     }
00048     taskEnd();
00049 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.28.3.3 flashLED()

```
void flashLED (
    int PIN = LED())
```

Definiert in Zeile 51 der Datei LED.h.

```
00051                                     {
00052     if (millis() % 1000 > 500) {
00053         digitalWrite(PIN, HIGH);
00054     } else {
00055         digitalWrite(PIN, LOW);
00056     }
00057 }
```

7.28.3.4 LEDInit()

```
void LEDInit ()
```

Start Initialisierung LEDtest.

Definiert in Zeile 63 der Datei LED.h.

```
00063     {  
00064     pinMode(LED(Red),   OUTPUT);  
00065     pinMode(LED(Blue),  OUTPUT);  
00066     pinMode(LED(Green), OUTPUT);  
00067     digitalWrite(LED(Red), HIGH);  
00068     delay(250);  
00069     digitalWrite(LED(Red), LOW);  
00070     digitalWrite(LED(Blue), HIGH);  
00071     delay(250);  
00072     digitalWrite(LED(Blue), LOW);  
00073     digitalWrite(LED(Green), HIGH);  
00074     delay(250);  
00075     digitalWrite(LED(Green), LOW);  
00076 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.28.3.5 LEDon()

```
void LEDon (  
    int PIN = LED())
```

Definiert in Zeile 78 der Datei LED.h.

```
00078     {  
00079     digitalWrite(PIN, HIGH);  
00080 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.28.3.6 LEDoff()

```
void LEDoff (
    int PIN = LED())
```

Definiert in Zeile [82](#) der Datei [LED.h](#).

```
00082     {
00083     digitalWrite(PIN, LOW);
00084 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.28.3.7 LEDoff_RGB()

```
void LEDoff_RGB ()
```

Definiert in Zeile [86](#) der Datei [LED.h](#).

```
00086     {
00087     digitalWrite(LED(Blue), LOW);
00088     digitalWrite(LED(Green), LOW);
00089     digitalWrite(LED(Red), LOW);
00090 }
```

7.29 LED.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00011
00012 #include <arduino.h>
00013 #include "task.h"
00014
00015 //Configuration LED
00016 //const int LEDBoard = 2; //DevModule
00017 //const int LEDBoard = 13; //Adafruit Huzzah32
00018
00019 enum LED {
00020     Red = 25,
00021     Green = 26,
00022     Blue = 33,
00023     LEDBoard = 13 //Adafruit Huzzah32
00024 };
00025
00026 void LEDblink(int PIN = LED()){
00027     taskBegin();
00028     while(1) // blockiert dank der TaskPause nicht
00029     {
00030         digitalWrite(PIN,HIGH); // LED ein
00031         taskPause(250); // gibt Rechenzeit ab
00032         digitalWrite(PIN,LOW); // LED aus
00033         taskPause(1000); // gibt Rechenzeit ab
00034     }
00035     taskEnd();
00036 }
```



```

00037
00038 void LEDflash(int PIN = LED()){
00039     taskBegin();
00040     while(1)    // blockiert dank der TaskPause nicht
00041     {
00042         digitalWrite(PIN,HIGH); // LED ein
00043         delay (5);
00044         //taskPause(2); // gibt Rechenzeit ab
00045         digitalWrite(PIN,LOW); // LED aus
00046         taskPause(3000); // gibt Rechenzeit ab
00047     }
00048     taskEnd();
00049 }
00050
00051 void flashLED(int PIN = LED()) {
00052     if (millis() % 1000 > 500) {
00053         digitalWrite(PIN, HIGH);
00054     } else {
00055         digitalWrite(PIN, LOW);
00056     }
00057 }
00058
00063 void LEDInit() {
00064     pinMode(LED(Red), OUTPUT);
00065     pinMode(LED(Blue), OUTPUT);
00066     pinMode(LED(Green), OUTPUT);
00067     digitalWrite(LED(Red), HIGH);
00068     delay(250);
00069     digitalWrite(LED(Red), LOW);
00070     digitalWrite(LED(Blue), HIGH);
00071     delay(250);
00072     digitalWrite(LED(Blue), LOW);
00073     digitalWrite(LED(Green), HIGH);
00074     delay(250);
00075     digitalWrite(LED(Green), LOW);
00076 }
00077
00078 void LEDon(int PIN = LED()) {
00079     digitalWrite(PIN, HIGH);
00080 }
00081
00082 void LEDoff(int PIN = LED()) {
00083     digitalWrite(PIN, LOW);
00084 }
00085
00086 void LEDoff_RGB() {
00087     digitalWrite(LED(Blue), LOW);
00088     digitalWrite(LED(Green),LOW);
00089     digitalWrite(LED(Red), LOW);
00090 }
00091

```

7.30 src/Motordaten.ino-Dateireferenz

Motordaten NMEA2000.

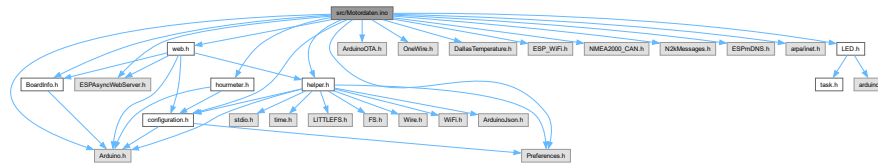
```

#include <Arduino.h>
#include "configuration.h"
#include <Preferences.h>
#include <ArduinoOTA.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <ESP_WiFi.h>
#include <ESPAsyncWebServer.h>
#include <NMEA2000_CAN.h>
#include <N2kMessages.h>
#include <ESPmDNS.h>
#include <arpa/inet.h>
#include "BoardInfo.h"
#include "helper.h"
#include "LED.h"
#include "web.h"

```

```
#include "hourmeter.h"
```

Include-Abhängigkeitsdiagramm für Motordaten.ino:



Makrodefinitionen

- #define `ENABLE_DEBUG_LOG` 0
- #define `ADC_Calibration_Value1` 250.0
ADC calibration Calibration data variable definition for ADC1 and ADC2 Input.
- #define `ADC_Calibration_Value2` 19.0

Funktionen

- OneWire `oneWire` (`ONE_WIRE_BUS`)
- void `debug_log` (char *str)
- void IRAM_ATTR `handleInterrupt` ()
RPM Event Interrupt Enters on falling edge.
- void `setup` ()
- void `GetTemperature` (void *parameter)
Get the Temperature object This task runs isolated on core 0 because `sensors.requestTemperatures()` is slow and blocking for about 750 ms.
- double `ReadRPM` ()
Calculate engine RPM from number of interrupts per time.
- bool `IsTimeToUpdate` (unsigned long NextUpdate)
- unsigned long `InitNextUpdate` (unsigned long Period, unsigned long Offset=0)
- void `SetNextUpdate` (unsigned long &NextUpdate, unsigned long Period)
- void `SendN2kDCStatus` (double BatteryVoltage, double SoC, double BatCapacity)
Send PGN127506.
- void `SendN2kBattery` (double BatteryVoltage)
Send PGN127508.
- void `SendN2kTankLevel` (double level, double capacity)
Send PGN 127505.
- void `SendN2kEngineData` (double Oiltemp, double Watertemp, double rpm, double hours, double voltage)
Send PGN 127489.
- void `SendN2kEngineRPM` (double RPM)
Send PGN 127488.
- double `ReadVoltage` (byte pin)
ReadVoltage is used to improve the linearity of the ESP32 ADC see: <https://github.com/G6EJD/ESP32-ADC-Accuracy-Improvement-function>.
- void `loop` ()

Variablen

- const unsigned long TransmitMessages[] [PROGMEM](#)
- volatile uint64_t [StartValue](#) = 0
- volatile uint64_t [PeriodCount](#) = 0
- unsigned long [Last_int_time](#) = 0
- hw_timer_t * [timer](#) = NULL
- portMUX_TYPE [mux](#) = portMUX_INITIALIZER_UNLOCKED
- DallasTemperature sensors & [oneWire](#)
- uint8_t [MotorCoolant](#) [8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 }
- uint8_t [MotorOil](#) [8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 }
- const int [ADCpin2](#) = 35
- const int [ADCpin1](#) = 34
- TaskHandle_t [Task1](#)
- const int [baudrate](#) = 38400
- const int [rs_config](#) = SERIAL_8N1

7.30.1 Ausführliche Beschreibung

Motordaten NMEA2000.

Autor

Gerry Sebb

Version

2.3

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [Motordaten.ino](#).

7.30.2 Makro-Dokumentation

7.30.2.1 ENABLE_DEBUG_LOG

```
#define ENABLE_DEBUG_LOG 0
```

Definiert in Zeile [45](#) der Datei [Motordaten.ino](#).

7.30.2.2 ADC_Calibration_Value1

```
#define ADC_Calibration_Value1 250.0
```

ADC calibration Calibration data variable definition for ADC1 and ADC2 Input.

For resistor measure 5 Volt and 180 Ohm equals 100% plus 1K resistor.

Definiert in Zeile 51 der Datei [Motordaten.ino](#).

7.30.2.3 ADC_Calibration_Value2

```
#define ADC_Calibration_Value2 19.0
```

The real value depends on the true resistor values for the ADC input (100K / 27 K). Old value 34.3

Definiert in Zeile 52 der Datei [Motordaten.ino](#).

7.30.3 Dokumentation der Funktionen

7.30.3.1 oneWire()

```
OneWire oneWire (  
    ONE_WIRE_BUS )
```

Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)

7.30.3.2 debug_log()

```
void debug_log (  
    char * str)
```

Definiert in Zeile 95 der Datei [Motordaten.ino](#).

```
00095 {  
00096 #if ENABLE_DEBUG_LOG == 1  
00097     Serial.println(str);  
00098 #endif  
00099 }
```

7.30.3.3 handleInterrupt()

```
void IRAM_ATTR handleInterrupt ()
```

RPM Event Interrupt Enters on falling edge.

Rückgabe

* void

Definiert in Zeile 107 der Datei [Motordaten.ino](#).

```
00108 {
00109     portENTER_CRITICAL_ISR(&mux);
00110     uint64_t TempVal = timerRead(timer);           // value of timer at interrupt
00111     PeriodCount = TempVal - StartValue;           // period count between rising edges in 0.000001 of a
second
00112     StartValue = TempVal;                         // puts latest reading as start for next calculation
00113     portEXIT_CRITICAL_ISR(&mux);
00114     Last_int_time = millis();
00115 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.4 setup()

```
void setup ()
```

Filesystem prepare for Webfiles

file exists, reading and loading config file

Read Boardinfo for output

Construct a new pin Mode object

Start OneWire

Set NMEA2000 product information

OTA

Definiert in Zeile 118 der Datei [Motordaten.ino](#).

```
00118     {
00119
00120     // Init USB serial port
00121     Serial.begin(115200);
00122
00123     Serial.printf("Motordaten setup %s start\n", Version);
00124 }
```

```

00129     if (!LittleFS.begin(true)) {
00130         Serial.println("An Error has occurred while mounting LittleFS");
00131         return;
00132     }
00133     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00134
00135     File root = LittleFS.open("/");
00136     listDir(LittleFS, "/", 3);
00137
00142     readConfig("/config.json");
00143     IP = inet_addr(tAP_Config.wAP_IP);
00144     AP_SSID = tAP_Config.wAP_SSID;
00145     AP_PASSWORD = tAP_Config.wAP_Password;
00146     fTemp1Offset = atof(tAP_Config.wTemp1_Offset);
00147     fTemp2Offset = atof(tAP_Config.wTemp2_Offset);
00148     FuelLevelMax = atof(tAP_Config.wFuelstandmax);
00149     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + ", Passwort:
" + AP_PASSWORD + ", Temp1Offset: " + fTemp1Offset + ", Temp2Offset: " + fTemp2Offset + " read from
file");
00150
00151     // LED
00152     LEDInit();
00153
00154     // Boardinfo
00159     sBoardInfo = boardInfo.ShowChipIDtoString();
00160
00161     //Wifi
00162     WiFi.mode(WIFI_AP_STA);
00163     WiFi.softAPdisconnect();
00164     if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00165         WiFi.softAPConfig(IP, Gateway, NMask);
00166         Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00167         Serial.println("\nSet IP " + IP.toString() + ", Gateway: " + Gateway.toString() + ", NetMask: " +
NMask.toString() + " ready");
00168         LEDon(LED(Green));
00169         delay(1000);
00170     } else {
00171         Serial.println("Starting AP failed.");
00172         LEDoff(LED(Green));
00173         LEDon(LED(Red));
00174         delay(1000);
00175         ESP.restart();
00176     }
00177
00178     WiFi.setHostname(HostName);
00179     Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00180
00181     delay(1000);
00182     WiFiDiag();
00183
00184     if (!MDNS.begin(AP_SSID)) {
00185         Serial.println("Error setting up MDNS responder!");
00186         while (1) {
00187             delay(1000);
00188         }
00189     }
00190     Serial.println("mDNS responder started\n");
00191
00192     // Start TCP (HTTP) server
00193     server.begin();
00194     Serial.println("TCP server started\n");
00195
00196     // Add service to MDNS-SD
00197     MDNS.addService("http", "tcp", 80);
00198
00199     // Webconfig laden
00200     website();
00201
00206     pinMode(Eingine_RPM_Pin, INPUT_PULLUP); // sets pin high
00207     attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
to interrupt on Falling Edge
00208     timer = timerBegin(0, 80, true); // this returns a
pointer to the hw_timer_t global variable
00209     // 0 = first timer
00210     // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00211     // true - counts up
00212     timerStart(timer); // starts the timer
00213
00218     sensors.begin();
00219     oneWire.reset();
00220     Serial.print("OneWire: Found ");
00221     Serial.print(sensors.getDeviceCount(), DEC);
00222     Serial.println(" devices.");
00223     Serial.print("Parasite power is: ");
00224     if (sensors.isParasitePowerMode()) Serial.println("ON");
00225     else Serial.println("OFF");
00226     sOneWire_Status = String(sensors.getDeviceCount(), DEC);

```

```

00227
00228   byte ow;
00229   byte addr[8];
00230
00231   if (!oneWire.search(addr)) {
00232       Serial.println("No more OneWire addresses.");
00233       Serial.println();
00234       oneWire.reset_search();
00235       delay(250);
00236       return;
00237   }
00238   Serial.print("ROM =");
00239   for (ow = 0; ow < 8; ow++) {
00240       Serial.write(' ');
00241       Serial.print(addr[ow], HEX);
00242   }
00243   Serial.print("\n");
00244   // search for devices on the bus and assign based on an index
00245   if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00246   if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00247
00248   // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
   or Mega
00249   NMEA2000.SetN2kCANMsgBufSize(8);
00250   NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00251   NMEA2000.SetN2kCANSendFrameBufSize(250);
00252
00253   esp_efuse_mac_get_default(chipid);
00254   for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00255
00260   NMEA2000.SetProductInformation("MD01", // Manufacturer's Model serial code
00261                                   100, // Manufacturer's product code
00262                                   "MD Sensor Module", // Manufacturer's Model ID
00263                                   "2.3.0.0 (2024-12-20)", // Manufacturer's Software version code
00264                                   "2.0.0.0 (2023-05-30)" // Manufacturer's Model version
00265                                   );
00266   // Set device information
00267   NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00268                                   132, // Device function=Analog to NMEA 2000 Gateway. See codes on
http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20&%20function%20codes%20v%202.00.pdf
00269                                   25, // Device class=Inter/Intranetwork Device. See codes on
http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20&%20function%20codes%20v%202.00.pdf
00270                                   2046 // Just choosen free from code list on
http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00271                                   );
00272
00273   // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
   below
00274
00275   NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
   Actisense format.
00276
00277   preferences.begin("nvs", false); // Open nonvolatile storage (nvs)
00278   NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
   33
00279   preferences.end();
00280   Serial.printf("NodeAddress=%d\n", NodeAddress);
00281
00282   NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00283   NMEA2000.ExtendTransmitMessages(TransmitMessages);
00284   NMEA2000.Open();
00285
00286   xTaskCreatePinnedToCore(
00287       GetTemperature, /* Function to implement the task */
00288       "Task1", /* Name of the task */
00289       10000, /* Stack size in words */
00290       NULL, /* Task input parameter */
00291       0, /* Priority of the task */
00292       &Task1, /* Task handle. */
00293       0); /* Core where the task should run */
00294
00295   delay(200);
00296
00301   ArduinoOTA
00302   .onStart([]() {
00303       String type;
00304       if (ArduinoOTA.getCommand() == U_FLASH)
00305           type = "sketch";
00306       else // U_SPIFFS
00307           type = "filesystem";
00308
00309       // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00310       Serial.println("Start updating " + type);
00311   })
00312   .onEnd([]() {
00313       Serial.println("\nEnd");
00314   })

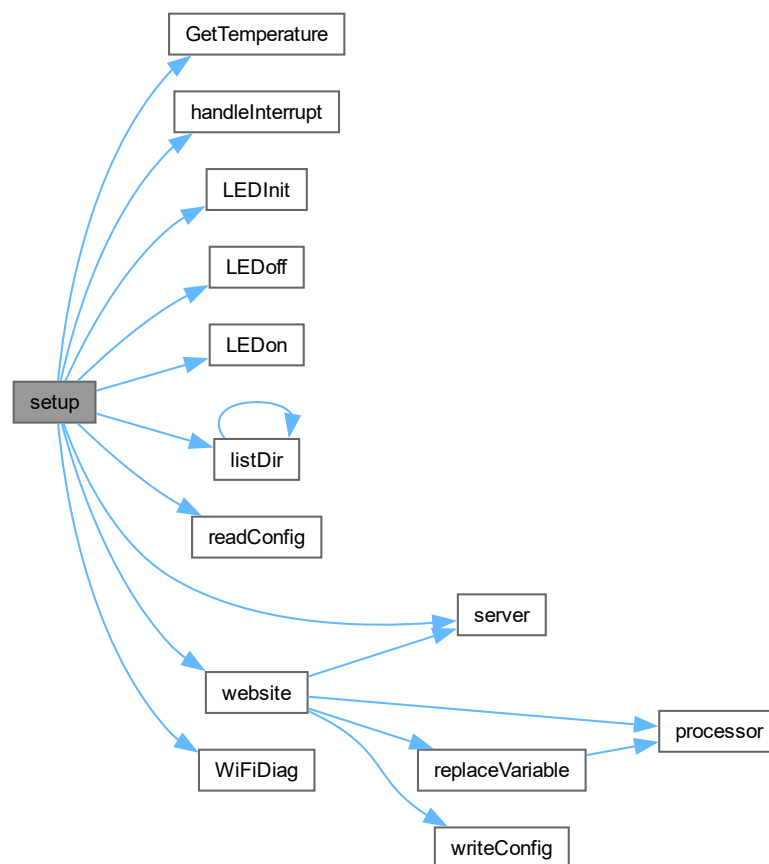
```

```

00315     .onProgress([](unsigned int progress, unsigned int total) {
00316         Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00317     })
00318     .onError([](ota_error_t error) {
00319         Serial.printf("Error[%u]: ", error);
00320         if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00321         else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00322         else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00323         else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00324         else if (error == OTA_END_ERROR) Serial.println("End Failed");
00325     });
00326
00327     ArduinoOTA.begin();
00328
00329     printf("Setup end\n");
00330 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



7.30.3.5 GetTemperature()

```

void GetTemperature (
    void * parameter)

```

Get the Temperature object This task runs isolated on core 0 because sensors.requestTemperatures() is slow and blocking for about 750 ms.

Parameter

| | |
|------------------|--|
| <i>parameter</i> | |
|------------------|--|

Definiert in Zeile 337 der Datei [Motordaten.ino](#).

```

00337                                     {
00338     float tmp0 = 0;
00339     float tmp1 = 0;
00340     for (;;) {
00341         sensors.requestTemperatures(); // Send the command to get temperatures
00342         vTaskDelay(100);
00343         tmp0 = sensors.getTempCByIndex(0) + fTemp1Offset;
00344         if (tmp0 != -127) OilTemp = tmp0;
00345         vTaskDelay(100);
00346         tmp1 = sensors.getTempCByIndex(1) + fTemp2Offset;
00347         if (tmp1 != -127) MotTemp = tmp1;
00348         vTaskDelay(100);
00349     }
00350 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.6 ReadRPM()

```
double ReadRPM ()
```

Calculate engine RPM from number of interrupts per time.

Rückgabe

double

Definiert in Zeile 357 der Datei [Motordaten.ino](#).

```

00357     {
00358     double RPM = 0;
00359
00360     portENTER_CRITICAL(&mux);
00361     if (PeriodCount != 0) { // 0 means no signals measured
00362         RPM = 1000000.00 / PeriodCount; // PeriodCount in 0.000001 of a second
00363     }
00364     portEXIT_CRITICAL(&mux);
00365     if (millis() > Last_int_time + 200) RPM = 0; // No signals RPM=0;
00366     return (RPM);
00367 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



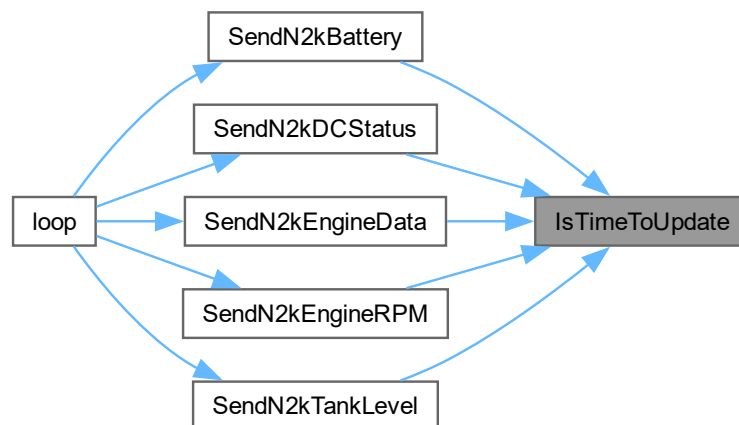
7.30.3.7 IsTimeToUpdate()

```
bool IsTimeToUpdate (
    unsigned long NextUpdate)
```

Definiert in Zeile 370 der Datei [Motordaten.ino](#).

```
00370 {
00371     return (NextUpdate < millis());
00372 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



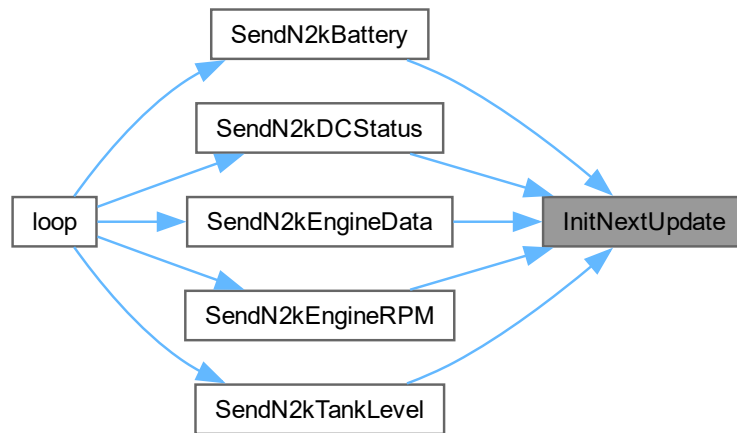
7.30.3.8 InitNextUpdate()

```
unsigned long InitNextUpdate (
    unsigned long Period,
    unsigned long Offset = 0)
```

Definiert in Zeile 373 der Datei [Motordaten.ino](#).

```
00373 {
00374     return millis() + Period + Offset;
00375 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



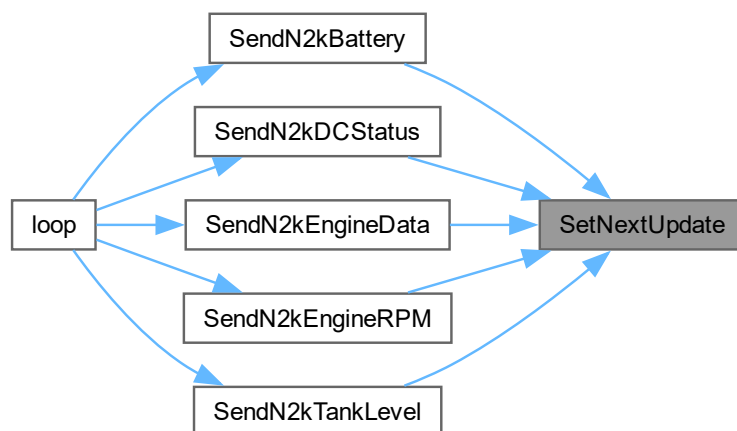
7.30.3.9 SetNextUpdate()

```
void SetNextUpdate (
    unsigned long & NextUpdate,
    unsigned long Period)
```

Definiert in Zeile 377 der Datei [Motordaten.ino](#).

```
00377 {
00378     while ( NextUpdate < millis() ) NextUpdate += Period;
00379 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.10 SendN2kDCStatus()

```
void SendN2kDCStatus (
    double BatteryVoltage,
    double SoC,
    double BatCapacity)
```

Send PGN127506.

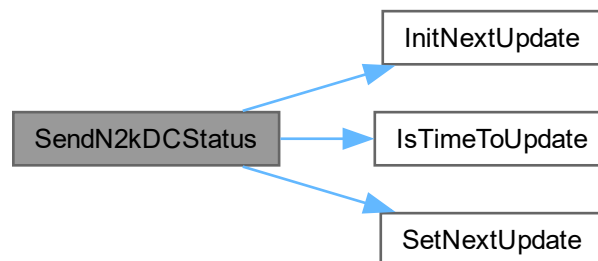
Parameter

| | |
|-----------------------|--|
| <i>BatteryVoltage</i> | |
| <i>SoC</i> | |
| <i>BatCapacity</i> | |

Definiert in Zeile 389 der Datei [Motordaten.ino](#).

```
00389
00390 static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
00391               BatteryDCStatusSendOffset);
00392
00393 if ( IsTimeToUpdate(SlowDataUpdated) ) {
00394     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00395
00396     Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00397     Serial.printf("SoC          : %3.1f %\n", SoC);
00398     Serial.printf("Capacity    : %3.1f Ah\n", BatCapacity);
00399     // SetN2kDCStatus(N2kMsg, 1, 1, N2kDct_Battery, 56, 92, 38500, 0.012, AhToCoulomb(420));
00400     SetN2kDCStatus(N2kMsg, 1, 2, N2kDct_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
00401               AhToCoulomb(55));
00401     NMEA2000.SendMessage(N2kMsg);
00402 }
00403 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.11 SendN2kBattery()

```
void SendN2kBattery (  
    double BatteryVoltage)
```

Send PGN127508.

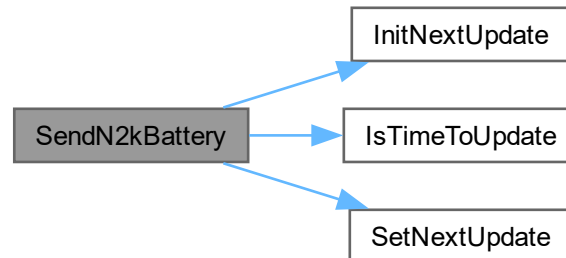
Parameter

| | |
|----------------|--|
| BatteryVoltage | |
|----------------|--|

Definiert in Zeile 410 der Datei [Motordaten.ino](#).

```
00410 {  
00411     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);  
00412     tN2kMsg N2kMsg;  
00413  
00414     if ( IsTimeToUpdate(SlowDataUpdated) ) {  
00415         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);  
00416  
00417         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);  
00418  
00419         SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);  
00420         NMEA2000.SendMessage(N2kMsg);  
00421     }  
00422 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.12 SendN2kTankLevel()

```
void SendN2kTankLevel (
    double level,
    double capacity)
```

Send PGN 127505.

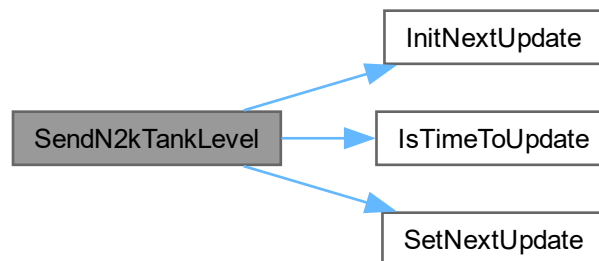
Parameter

| | |
|-----------------|--|
| <i>level</i> | |
| <i>capacity</i> | |

Definiert in Zeile 430 der Datei [Motordaten.ino](#).

```
00430 {
00431     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00432     tN2kMsg N2kMsg;
00433
00434     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00435         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00436
00437         Serial.printf("Fuel Level   : %3.1f %%\n", level);
00438         Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00439
00440         SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00441         NMEA2000.SendMessage(N2kMsg);
00442     }
00443 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.13 SendN2kEngineData()

```
void SendN2kEngineData (
    double Oiltemp,
    double Watertemp,
    double rpm,
    double hours,
    double voltage)
```

Send PGN 127489.

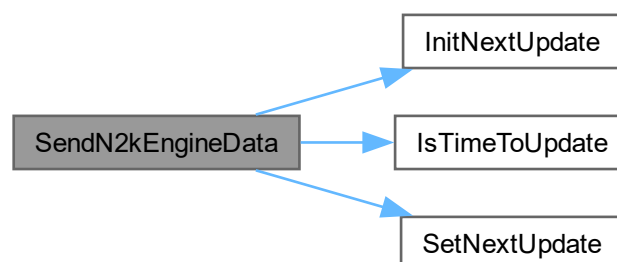
Parameter

| | |
|------------------|--|
| <i>Oiltemp</i> | |
| <i>Watertemp</i> | |
| <i>rpm</i> | |
| <i>hours</i> | |
| <i>voltage</i> | |

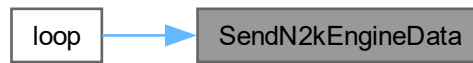
Definiert in Zeile 454 der Datei [Motordaten.ino](#).

```
00454
00455     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00456     tN2kMsg N2kMsg;
00457     tN2kEngineDiscreteStatus1 Status1;
00458     tN2kEngineDiscreteStatus2 Status2;
00459     Status1.Bits.OverTemperature = Oiltemp > 90;           // Alarm Motor over temp
00460     Status1.Bits.LowCoolantLevel = Watertemp > 90;        // Alarm low cooling
00461     Status1.Bits.LowSystemVoltage = voltage < 11;
00462     Status2.Bits.EngineShuttingDown = rpm < 100;          // Alarm Motor off
00463     EngineOn = !Status2.Bits.EngineShuttingDown;
00464
00465     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00466         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00467
00468         Serial.printf("Oil Temp      : %3.1f °C \n", Oiltemp);
00469         Serial.printf("Coolant Temp: %3.1f °C \n", Watertemp);
00470         Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00471         Serial.printf("Overtemp Oil: %s \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00472         Serial.printf("Overtemp Mot: %s \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00473         Serial.printf("Engine Off  : %s \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00474
00475         // SetN2kTemperatureExt (N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
00476         // PGN130312, uncomment the PGN to be used
00477         SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Watertemp),
00478             N2kDoubleNA, N2kDoubleNA, hours, N2kDoubleNA, N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00479         NMEA2000.SendMessage(N2kMsg);
00480     }
00481 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.14 SendN2kEngineRPM()

```
void SendN2kEngineRPM (
    double RPM)
```

Send PGN 127488.

Parameter

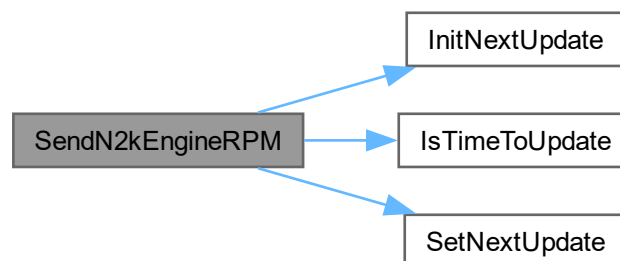
| | |
|------------|--|
| <i>RPM</i> | |
|------------|--|

Definiert in Zeile 488 der Datei [Motordaten.ino](#).

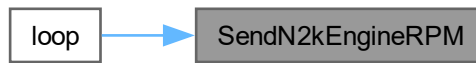
```

00488                                     {
00489     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMsSendOffset);
00490     tN2kMsg N2kMsg;
00491
00492     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00493         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00494
00495         Serial.printf("Engine RPM   : %4.0f RPM \n", RPM);
00496
00497         SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00498
00499         NMEA2000.SendMsg(N2kMsg);
00500     }
00501 }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.15 ReadVoltage()

```
double ReadVoltage (  
    byte pin)
```

ReadVoltage is used to improve the linearity of the ESP32 ADC see: <https://github.com/G6EJD/ESP32-ADC-Accuracy-Improvement-function>.

Parameter

| | |
|------------|--|
| <i>pin</i> | |
|------------|--|

Rückgabe

double

Definiert in Zeile 509 der Datei [Motordaten.ino](#).

```
00509 {  
00510     double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in  
        range 0 to 4095  
00511     if (reading < 1 || reading > 4095) return 0;  
00512     // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +  
        0.000854596860691 * reading + 0.065440348345433;  
00513     return (-0.000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -  
        0.0000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;  
00514 } // Added an improved polynomial, use either, comment out as required
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.30.3.16 loop()

```
void loop ()
```

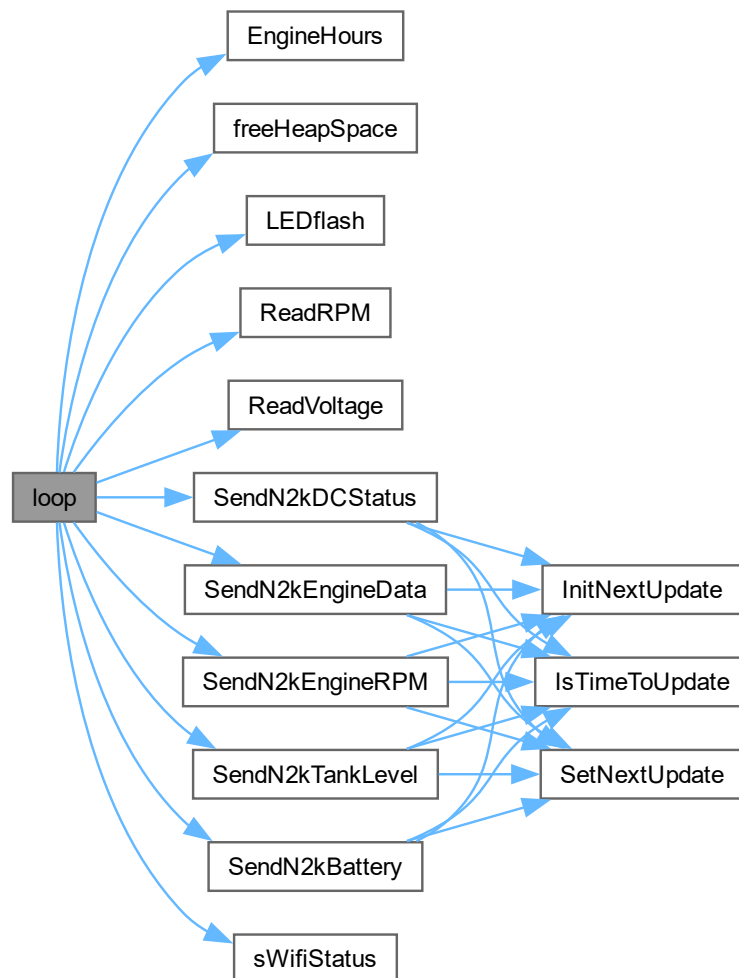
Actual Website Data

Construct a new if object Reboot from Website

Definiert in Zeile 517 der Datei [Motordaten.ino](#).

```
00517     {
00518
00519         // LED
00520         LEDflash(LED(Green)); // flash for loop run
00521
00522         // if (!sensors.getAddress(MotorThermometer, 0)) LEDflash(LED(Red)); // search for device on the
        bus and unable to find
00523         // sensors.requestTemperatures(); // Send the command to get temperatures
00524         // ExhaustTemp = sensors.getTempCByIndex(0) + fTempOffset;
00525
00526         //Wifi variables
00527         bConnect_CL = WiFi.status() == WL_CONNECTED ? 1 : 0;
00528
00529         // unsigned int size;
00530
00531         BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
        // This implements a low pass filter to eliminate spike for ADC readings
00532
00533         FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
        This implements a low pass filter to eliminate spike for ADC readings
00534
00535         EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
        pass filter to eliminate spike for RPM measurements
00536
00537         BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0;
00538         // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00539
00540         EngineHours(EngineOn);
00541
00542         SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00543         SendN2kEngineData(OilTemp, MotTemp, EngineRPM, Counter, BordSpannung);
00544         SendN2kEngineRPM(EngineRPM);
00545         SendN2kBattery(BordSpannung);
00546         SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00547
00548         NMEA2000.ParseMessages();
00549         int SourceAddress = NMEA2000.GetN2kSource();
00550         if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00551             NodeAddress = SourceAddress; // Set new Node Address (to save only once)
00552             preferences.begin("nvs", false);
00553             preferences.putInt("LastNodeAddress", SourceAddress);
00554             preferences.end();
00555             Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00556         }
00557
00558         // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00559         if ( Serial.available() ) {
00560             Serial.read();
00561         }
00562
00563
00564         // OTA
00565         ArduinoOTA.handle();
00566
00567         fOilTemp1 = OilTemp;
00568         fMotTemp2 = MotTemp;
00569         fBordSpannung = BordSpannung;
00570         fDrehzahl = EngineRPM;
00571         sCL_Status = sWifiStatus(WiFi.status());
00572         sAP_Station = WiFi.softAPgetStationNum();
00573         freeHeapSpace();
00574
00575         if (IsRebootRequired) {
00576             Serial.println("Rebooting ESP32: ");
00577             delay(1000); // give time for reboot page to load
00578             ESP.restart();
00579         }
00580
00581     }
```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



7.30.4 Variablen-Dokumentation

7.30.4.1 PROGRAMM

```
const unsigned long TransmitMessages [ ] PROGMEM
```

Initialisierung:

```
= {127488L,
    127489L,
    127505L,
    127506L,
    127508L,
    0
}
```

Set the information for other bus devices, which PGN messages we support

Definiert in Zeile 57 der Datei [Motordaten.ino](#).

```

00057                                     {127488L, // Engine Rapid / RPM
00058                                     127489L, // Engine parameters dynamic
00059                                     127505L, // Fluid Level
00060                                     127506L, // Battery
00061                                     127508L, // Battery Status
00062                                     0
00063                                     };

```

7.30.4.2 StartValue

```
volatile uint64_t StartValue = 0
```

RPM data. Generator RPM is measured on connector "W" First interrupt value

Definiert in Zeile [70](#) der Datei [Motordaten.ino](#).

7.30.4.3 PeriodCount

```
volatile uint64_t PeriodCount = 0
```

period in counts of 0.000001 of a second

Definiert in Zeile [71](#) der Datei [Motordaten.ino](#).

7.30.4.4 Last_int_time

```
unsigned long Last_int_time = 0
```

Definiert in Zeile [72](#) der Datei [Motordaten.ino](#).

7.30.4.5 timer

```
hw_timer_t* timer = NULL
```

pointer to a variable of type hw_timer_t

Definiert in Zeile [73](#) der Datei [Motordaten.ino](#).

7.30.4.6 mux

```
portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED
```

synchs between maon cose and interrupt?

Definiert in Zeile [74](#) der Datei [Motordaten.ino](#).

7.30.4.7 oneWire

```
DallasTemperature sensors& oneWire
```

Pass our oneWire reference to Dallas Temperature.

Definiert in Zeile [80](#) der Datei [Motordaten.ino](#).

7.30.4.8 MotorCoolant

```
uint8_t MotorCoolant[8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 }
```

DeviceAddress Coolant

Definiert in Zeile [82](#) der Datei [Motordaten.ino](#).

```
00082 { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
```

7.30.4.9 MotorOil

```
uint8_t MotorOil[8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 }
```

DeviceAddress Engine Oil

Definiert in Zeile [83](#) der Datei [Motordaten.ino](#).

```
00083 { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
```

7.30.4.10 ADCpin2

```
const int ADCpin2 = 35
```

Voltage measure is connected GPIO 35 (Analog ADC1_CH7)

Definiert in Zeile [85](#) der Datei [Motordaten.ino](#).

7.30.4.11 ADCpin1

```
const int ADCpin1 = 34
```

Tank fluid level measure is connected GPIO 34 (Analog ADC1_CH6)

Definiert in Zeile [86](#) der Datei [Motordaten.ino](#).

7.30.4.12 Task1

```
TaskHandle_t Task1
```

Task handle for OneWire read (Core 0 on ESP32)

Definiert in Zeile [89](#) der Datei [Motordaten.ino](#).

7.30.4.13 baudrate

```
const int baudrate = 38400
```

Serial port 2 config (GPIO 16)

Definiert in Zeile [92](#) der Datei [Motordaten.ino](#).

7.30.4.14 rs_config

```
const int rs_config = SERIAL_8N1
```

Definiert in Zeile 93 der Datei [Motordaten.ino](#).

7.31 Motordaten.ino

[gehe zur Dokumentation dieser Datei](#)

```
00001 /*
00002  * This code is free software; you can redistribute it and/or
00003  * modify it under the terms of the GNU Lesser General Public
00004  * License as published by the Free Software Foundation; either
00005  * version 2.1 of the License, or (at your option) any later version.
00006  * This code is distributed in the hope that it will be useful,
00007  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00008  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
00009  * Lesser General Public License for more details.
00010  * You should have received a copy of the GNU Lesser General Public
00011  * License along with this library; if not, write to the Free Software
00012  * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00013  */
00014
00015
00016
00017 #include <Arduino.h>
00018 #include "configuration.h"
00019 #include <Preferences.h>
00020 #include <ArduinoOTA.h>
00021 #include <OneWire.h>
00022 #include <DallasTemperature.h>
00023 #include <ESP_WiFi.h>
00024 #include <ESPAsyncWebServer.h>
00025 #include <NMEA2000_CAN.h> // This will automatically choose right CAN library and create suitable
00026     NMEA2000 object
00027 #include <N2kMessages.h>
00028 #include <ESPmDNS.h>
00029 #include <arpa/inet.h>
00030 #include "BoardInfo.h"
00031 #include "helper.h"
00032 #include "LED.h"
00033 #include "web.h"
00034 #include "hourmeter.h"
00035
00036 #define ENABLE_DEBUG_LOG 0 // Debug log
00037
00038 #define ADC_Calibration_Value1 250.0
00039 #define ADC_Calibration_Value2 19.0
00040
00041 const unsigned long TransmitMessages[] PROGMEM = {127488L, // Engine Rapid / RPM
00042     127489L, // Engine parameters dynamic
00043     127505L, // Fluid Level
00044     127506L, // Battery
00045     127508L, // Battery Status
00046     0
00047 };
00048
00049 volatile uint64_t StartValue = 0;
00050 volatile uint64_t PeriodCount = 0;
00051 unsigned long Last_int_time = 0;
00052 hw_timer_t * timer = NULL;
00053 portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
00054
00055 OneWire oneWire(ONE_WIRE_BUS);
00056 DallasTemperature sensors(&oneWire);
00057 // DeviceAddress MotorThermometer; //**< arrays to hold device addresses
00058 uint8_t MotorCoolant[8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
00059 uint8_t MotorOil[8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
00060
00061 const int ADCpin2 = 35;
00062 const int ADCpin1 = 34;
00063
00064 TaskHandle_t Task1;
00065
00066 const int baudrate = 38400;
00067 const int rs_config = SERIAL_8N1;
```

```

00094
00095 void debug_log(char* str) {
00096 #if ENABLE_DEBUG_LOG == 1
00097     Serial.println(str);
00098 #endif
00099 }
00100
00106 //=====
00107 void IRAM_ATTR handleInterrupt()
00108 {
00109     portENTER_CRITICAL_ISR(&mux);
00110     uint64_t TempVal = timerRead(timer); // value of timer at interrupt
00111     PeriodCount = TempVal - StartValue; // period count between rising edges in 0.000001 of a
second
00112     StartValue = TempVal; // puts latest reading as start for next calculation
00113     portEXIT_CRITICAL_ISR(&mux);
00114     Last_int_time = millis();
00115 }
00116
00117 /***** Setup
*****
00118 void setup() {
00119
00120     // Init USB serial port
00121     Serial.begin(115200);
00122
00123     Serial.printf("Motordaten setup %s start\n", Version);
00124
00129     if (!LittleFS.begin(true)) {
00130         Serial.println("An Error has occurred while mounting LittleFS");
00131         return;
00132     }
00133     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00134
00135     File root = LittleFS.open("/");
00136     listDir(LittleFS, "/", 3);
00137
00142     readConfig("/config.json");
00143     IP = inet_addr(tAP_Config.wAP_IP);
00144     AP_SSID = tAP_Config.wAP_SSID;
00145     AP_PASSWORD = tAP_Config.wAP_Password;
00146     fTemp1Offset = atof(tAP_Config.wTemp1_Offset);
00147     fTemp2Offset = atof(tAP_Config.wTemp2_Offset);
00148     FuelLevelMax = atof(tAP_Config.wFuelstandmax);
00149     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + ", Passwort: " + AP_PASSWORD + ", Temp1Offset: " + fTemp1Offset + ", Temp2Offset: " + fTemp2Offset + " read from
file");
00150
00151     // LED
00152     LEDInit();
00153
00154     // Boardinfo
00159     sBoardInfo = boardInfo.ShowChipIDtoString();
00160
00161     //Wifi
00162     WiFi.mode(WIFI_AP_STA);
00163     WiFi.softAPdisconnect();
00164     if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00165         WiFi.softAPConfig(IP, Gateway, NMask);
00166         Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00167         Serial.println("\nSet IP " + IP.toString() + ", Gateway: " + Gateway.toString() + ", NetMask: " +
NMask.toString() + " ready");
00168         LEDon(LED.Green);
00169         delay(1000);
00170     } else {
00171         Serial.println("Starting AP failed.");
00172         LEDoff(LED.Green);
00173         LEDon(LED.Red);
00174         delay(1000);
00175         ESP.restart();
00176     }
00177
00178     WiFi.setHostname(HostName);
00179     Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00180
00181     delay(1000);
00182     WiFiDiag();
00183
00184     if (!MDNS.begin(AP_SSID)) {
00185         Serial.println("Error setting up MDNS responder!");
00186         while (1) {
00187             delay(1000);
00188         }
00189     }
00190     Serial.println("mDNS responder started\n");
00191
00192 // Start TCP (HTTP) server

```

```

00193     server.begin();
00194     Serial.println("TCP server started\n");
00195
00196 // Add service to MDNS-SD
00197     MDNS.addService("http", "tcp", 80);
00198
00199 // Webconfig laden
00200     website();
00201
00206     pinMode(Eingine_RPM_Pin, INPUT_PULLUP); // sets pin high
00207     attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
to interrupt on Falling Edge
00208     timer = timerBegin(0, 80, true); // this returns a
pointer to the hw_timer_t global variable
00209     // 0 = first timer
00210     // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00211     // true - counts up
00212     timerStart(timer); // starts the timer
00213
00218     sensors.begin();
00219     oneWire.reset();
00220     Serial.print("OneWire: Found ");
00221     Serial.print(sensors.getDeviceCount(), DEC);
00222     Serial.println(" devices.");
00223     Serial.print("Parasite power is: ");
00224     if (sensors.isParasitePowerMode()) Serial.println("ON");
00225     else Serial.println("OFF");
00226     sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00227
00228     byte ow;
00229     byte addr[8];
00230
00231     if (!oneWire.search(addr)) {
00232         Serial.println("No more OneWire addresses.");
00233         Serial.println();
00234         oneWire.reset_search();
00235         delay(250);
00236         return;
00237     }
00238     Serial.print("ROM =");
00239     for (ow = 0; ow < 8; ow++) {
00240         Serial.write(' ');
00241         Serial.print(addr[ow], HEX);
00242     }
00243     Serial.print("\n");
00244 // search for devices on the bus and assign based on an index
00245     if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00246     if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00247
00248 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
or Mega
00249     NMEA2000.SetN2kCANMsgBufSize(8);
00250     NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00251     NMEA2000.SetN2kCANSendFrameBufSize(250);
00252
00253     esp_efuse_mac_get_default(chipid);
00254     for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00255
00260     NMEA2000.SetProductInformation("MD01", // Manufacturer's Model serial code
00261                                     100, // Manufacturer's product code
00262                                     "MD Sensor Module", // Manufacturer's Model ID
00263                                     "2.3.0.0 (2024-12-20)", // Manufacturer's Software version code
00264                                     "2.0.0.0 (2023-05-30)" // Manufacturer's Model version
00265                                     );
00266 // Set device information
00267     NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00268                                   132, // Device function=Analog to NMEA 2000 Gateway. See codes on
http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20%20function%20codes%20v%202.00.pdf
00269                                   25, // Device class=Inter/Intranetwork Device. See codes on
http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20%20function%20codes%20v%202.00.pdf
00270                                   2046 // Just choosen free from code list on
http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00271                                   );
00272
00273 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
below
00274
00275     NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
Actisense format.
00276
00277     preferences.begin("nvs", false); // Open nonvolatile storage (nvs)
00278     NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
33
00279     preferences.end();
00280     Serial.printf("NodeAddress=%d\n", NodeAddress);
00281
00282     NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);

```



```

00283 NMEA2000.ExtendTransmitMessages(TransmitMessages);
00284 NMEA2000.Open();
00285
00286 xTaskCreatePinnedToCore(
00287     GetTemperature, /* Function to implement the task */
00288     "Task1", /* Name of the task */
00289     10000, /* Stack size in words */
00290     NULL, /* Task input parameter */
00291     0, /* Priority of the task */
00292     &Task1, /* Task handle. */
00293     0); /* Core where the task should run */
00294
00295 delay(200);
00296
00301 ArduinoOTA
00302     .onStart([]() {
00303         String type;
00304         if (ArduinoOTA.getCommand() == U_FLASH)
00305             type = "sketch";
00306         else // U_SPIFFS
00307             type = "filesystem";
00308
00309         // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00310         Serial.println("Start updating " + type);
00311     })
00312     .onEnd([]() {
00313         Serial.println("\nEnd");
00314     })
00315     .onProgress([](unsigned int progress, unsigned int total) {
00316         Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00317     })
00318     .onError([](ota_error_t error) {
00319         Serial.printf("Error[%u]: ", error);
00320         if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00321         else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00322         else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00323         else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00324         else if (error == OTA_END_ERROR) Serial.println("End Failed");
00325     });
00326
00327 ArduinoOTA.begin();
00328
00329 printf("Setup end\n");
00330 }
00331
00337 void GetTemperature( void * parameter) {
00338     float tmp0 = 0;
00339     float tmp1 = 0;
00340     for (;;) {
00341         sensors.requestTemperatures(); // Send the command to get temperatures
00342         vTaskDelay(100);
00343         tmp0 = sensors.getTempCByIndex(0) + fTemp1Offset;
00344         if (tmp0 != -127) OilTemp = tmp0;
00345         vTaskDelay(100);
00346         tmp1 = sensors.getTempCByIndex(1) + fTemp2Offset;
00347         if (tmp1 != -127) MotTemp = tmp1;
00348         vTaskDelay(100);
00349     }
00350 }
00351
00357 double ReadRPM() {
00358     double RPM = 0;
00359
00360     portENTER_CRITICAL(&mux);
00361     if (PeriodCount != 0) {
00362         RPM = 1000000.00 / PeriodCount;
00363     }
00364     portEXIT_CRITICAL(&mux);
00365     if (millis() > Last_int_time + 200) RPM = 0; // No signals RPM=0;
00366     return (RPM);
00367 }
00368
00369
00370 bool IsTimeToUpdate(unsigned long NextUpdate) {
00371     return (NextUpdate < millis());
00372 }
00373
00373 unsigned long InitNextUpdate(unsigned long Period, unsigned long Offset = 0) {
00374     return millis() + Period + Offset;
00375 }
00376
00377 void SetNextUpdate(unsigned long &NextUpdate, unsigned long Period) {
00378     while ( NextUpdate < millis() ) NextUpdate += Period;
00379 }
00380
00381 /***** n2k Datenfunktionen *****/
00389 void SendN2kDCStatus(double BatteryVoltage, double SoC, double BatCapacity) {
00390     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,

```

```

        BatteryDCStatusSendOffset);
00391     tN2kMsg N2kMsg;
00392
00393     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00394         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00395
00396         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00397         Serial.printf("SoC        : %3.1f %\n", SoC);
00398         Serial.printf("Capacity   : %3.1f Ah\n", BatCapacity);
00399         // SetN2kDCStatus(N2kMsg, 1, 1, N2kDcT_Battery, 56, 92, 38500, 0.012, AhToCoulomb(420));
00400         SetN2kDCStatus(N2kMsg, 1, 2, N2kDcT_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
        AhToCoulomb(55));
00401         NMEA2000.SendMessage(N2kMsg);
00402     }
00403 }
00404
00410 void SendN2kBattery(double BatteryVoltage) {
00411     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00412     tN2kMsg N2kMsg;
00413
00414     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00415         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00416
00417         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00418
00419         SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00420         NMEA2000.SendMessage(N2kMsg);
00421     }
00422 }
00423
00430 void SendN2kTankLevel(double level, double capacity) {
00431     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00432     tN2kMsg N2kMsg;
00433
00434     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00435         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00436
00437         Serial.printf("Fuel Level   : %3.1f %\n", level);
00438         Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00439
00440         SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity);
00441         NMEA2000.SendMessage(N2kMsg);
00442     }
00443 }
00444
00454 void SendN2kEngineData(double Oiltemp, double Watertemp, double rpm, double hours, double voltage) {
00455     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00456     tN2kMsg N2kMsg;
00457     tN2kEngineDiscreteStatus1 Status1;
00458     tN2kEngineDiscreteStatus2 Status2;
00459     Status1.Bits.OverTemperature = Oiltemp > 90;           // Alarm Motor over temp
00460     Status1.Bits.LowCoolantLevel = Watertemp > 90;         // Alarm low cooling
00461     Status1.Bits.LowSystemVoltage = voltage < 11;
00462     Status2.Bits.EngineShuttingDown = rpm < 100;          // Alarm Motor off
00463     EngineOn = !Status2.Bits.EngineShuttingDown;
00464
00465     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00466         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00467
00468         Serial.printf("Oil Temp      : %3.1f °C \n", Oiltemp);
00469         Serial.printf("Coolant Temp: %3.1f °C \n", Watertemp);
00470         Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00471         Serial.printf("Overtemp Oil: %s \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00472         Serial.printf("Overtemp Mot: %s \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00473         Serial.printf("Engine Off  : %s \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00474
00475         // SetN2kTemperatureExt(N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
00476         // PGN130312, uncomment the PGN to be used
00477
00478         SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Watertemp),
        N2kDoubleNA, N2kDoubleNA, hours, N2kDoubleNA, N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00479         NMEA2000.SendMessage(N2kMsg);
00480     }
00481 }
00482
00488 void SendN2kEngineRPM(double RPM) {
00489     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMsSendOffset);
00490     tN2kMsg N2kMsg;
00491
00492     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00493         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00494
00495         Serial.printf("Engine RPM   : %4.0f RPM \n", RPM);
00496
00497         SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00498     }

```

```

00499     NMEA2000.SendMessage(N2kMsg);
00500 }
00501 }
00502
00509 double ReadVoltage(byte pin) {
00510     double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in
    range 0 to 4095
00511     if (reading < 1 || reading > 4095) return 0;
00512     // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +
    0.000854596860691 * reading + 0.065440348345433;
00513     return (-0.000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -
    0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;
00514 } // Added an improved polynomial, use either, comment out as required
00515
00516 /***** Loop *****/
00517 void loop() {
00518     // LED
00519     LEDflash(LED(Green)); // flash for loop run
00520
00521     // if (!sensors.getAddress(MotorThermometer, 0)) LEDflash(LED(Red)); // search for device on the
    bus and unable to find
00523     // sensors.requestTemperatures(); // Send the command to get temperatures
00524     // ExhaustTemp = sensors.getTempCByIndex(0) + fTempOffset;
00525
00526     //Wifi variables
00527     bConnect_CL = WiFi.status() == WL_CONNECTED ? 1 : 0;
00528
00529     // unsigned int size;
00530
00531     BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
    // This implements a low pass filter to eliminate spike for ADC readings
00532
00533     FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
    This implements a low pass filter to eliminate spike for ADC readings
00534
00535     EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
    pass filter to eliminate spike for RPM measurements
00536
00537     BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0;
00538     // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00539
00540     EngineHours(EngineOn);
00541
00542     SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00543     SendN2kEngineData(OilTemp, MotTemp, EngineRPM, Counter, BordSpannung);
00544     SendN2kEngineRPM(EngineRPM);
00545     SendN2kBattery(BordSpannung);
00546     SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00547
00548     NMEA2000.ParseMessages();
00549     int SourceAddress = NMEA2000.GetN2kSource();
00550     if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00551         NodeAddress = SourceAddress; // Set new Node Address (to save only once)
00552         preferences.begin("nvs", false);
00553         preferences.putInt("LastNodeAddress", SourceAddress);
00554         preferences.end();
00555         Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00556     }
00557
00558     // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00559     if (Serial.available() ) {
00560         Serial.read();
00561     }
00562
00563
00564     // OTA
00565     ArduinoOTA.handle();
00566
00571     fOilTemp1 = OilTemp;
00572     fMotTemp2 = MotTemp;
00573     fBordSpannung = BordSpannung;
00574     fDrehzahl = EngineRPM;
00575     sCL_Status = sWifiStatus(WiFi.status());
00576     sAP_Station = WiFi.softAPgetStationNum();
00577     freeHeapSpace();
00582     if (IsRebootRequired) {
00583         Serial.println("Rebooting ESP32: ");
00584         delay(1000); // give time for reboot page to load
00585         ESP.restart();
00586     }
00587
00588
00589 }

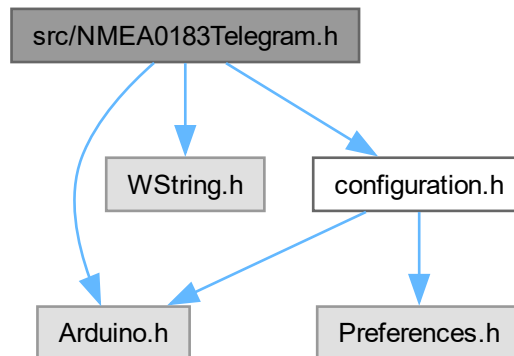
```

7.32 src/NMEA0183Telegram.h-Dateireferenz

NMEA0183 Telegramme senden.

```
#include <Arduino.h>
#include <WString.h>
#include "configuration.h"
```

Include-Abhängigkeitsdiagramm für NMEA0183Telegram.h:



Funktionen

- char `Checksum` (String NMEADData)
Checksum calculation for NMEA.
- String `sendXDR` ()
Send NMEA0183 Send Sensor data.

7.32.1 Ausführliche Beschreibung

NMEA0183 Telegramme senden.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [NMEA0183Telegram.h](#).

7.32.2 Dokumentation der Funktionen

7.32.2.1 CheckSum()

```
char CheckSum (  
    String NMEADData)
```

Checksum calculation for NMEA.

Parameter

| | |
|-----------|--|
| NMEADData | |
|-----------|--|

Rückgabe

char

Definiert in Zeile 23 der Datei [NMEA0183Telegram.h](#).

```
00023                                     {  
00024     char checksum = 0;  
00025     // Iterate over the string, XOR each byte with the total sum  
00026     for (int c = 0; c < NMEADData.length(); c++) {  
00027         checksum = char(checksum ^ NMEADData.charAt(c));  
00028     }  
00029     // Return the result  
00030     return checksum;  
00031 }
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.32.2.2 sendXDR()

```
String sendXDR ()
```

Send NMEA0183 Send Sensor data.

Rückgabe

String

Definiert in Zeile 60 der Datei [NMEA0183Telegram.h](#).

```

00061 {
00062     String HexChecksum;
00063     String NMEASensor;
00064     String SendSensor;
00065
00066     NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);
00067     //NMEASensorKraeng += ",";
00068     NMEASensor += String(fGaugeDrehzahl);
00069     NMEASensor += ",D,ROLL";
00070
00071     // Build CheckSum
00072     HexChecksum = String(CheckSum(NMEASensor), HEX);
00073     // Build complete NMEA string
00074     SendSensor = "$" + NMEASensor;
00075     SendSensor += "*";
00076     SendSensor += HexChecksum;
00077
00078     Serial.println(SendSensor);
00079
00080     return SendSensor;
00081 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



7.33 NMEA0183Telegram.h

[gehe zur Dokumentation dieser Datei](#)

```

00001
00011
00012 #include <Arduino.h>
00013 #include <WString.h> // Needs for structures
00014 #include "configuration.h"
00015
00022
00023 char CheckSum(String NMEADData) {
00024     char checksum = 0;
00025     // Iterate over the string, XOR each byte with the total sum
00026     for (int c = 0; c < NMEADData.length(); c++) {
00027         checksum = char(checksum ^ NMEADData.charAt(c));
00028     }
00029     // Return the result
00030     return checksum;
00031 }
00032
00033 /*
00034 XDR
00035 Transducer Values
00036     1 2 3 4      n
00037 | | | | | \
00038 * $--XDR,a,x.x,a,c--c, ..... *hh<CR><LF> \
00039
00040 Field Number:
00041 1) Transducer Type
00042 2) Measurement Data
00043 3) Units of measurement
00044 4) Name of transducer

```

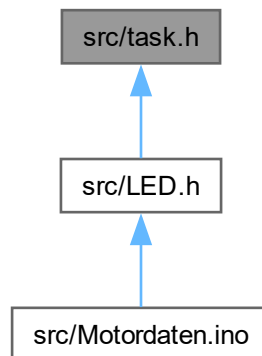
```

00045      x) More of the same
00046      n) Checksum
00047
00048      Example:
00049      Temperatur $IIXDR,C,19.52,C,TempAir*19
00050      Druck      $IIXDR,P,1.02481,B,Barometer*29
00051      Kraegung   $IIXDR,A,0,x.x,ROLL*hh<CR><LF>
00052 */
00053
00059
00060 String sendXDR()
00061 {
00062     String HexChecksum;
00063     String NMEASensor;
00064     String SendSensor;
00065
00066     NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);
00067     //NMEASensorKraeng += ",";
00068     NMEASensor += String(fGaugeDrehzahl);
00069     NMEASensor += ",D,ROLL";
00070
00071     // Build CheckSum
00072     HexChecksum = String(CheckSum(NMEASensor), HEX);
00073     // Build complete NMEA string
00074     SendSensor = "$" + NMEASensor;
00075     SendSensor += "*";
00076     SendSensor += HexChecksum;
00077
00078     Serial.println(SendSensor);
00079
00080     return SendSensor;
00081 }
00082

```

7.34 src/task.h-Dateireferenz

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Makrodefinitionen

- #define taskBegin()
- #define taskEnd()
- #define taskSwitch()
- #define taskPause(interval)
- #define taskWaitFor(condition)
- #define taskStepName(STEPNAME)
- #define taskJumpTo(STEPNAME)

7.34.1 Makro-Dokumentation

7.34.1.1 taskBegin

```
#define taskBegin()
```

Wert:

```
static int mark = 0; static unsigned long __attribute__((unused)) timeStamp = 0; switch(mark){ case 0:
```

Definiert in Zeile 6 der Datei [task.h](#).

7.34.1.2 taskEnd

```
#define taskEnd()
```

Wert:

```
}
```

Definiert in Zeile 7 der Datei [task.h](#).

7.34.1.3 taskSwitch

```
#define taskSwitch()
```

Wert:

```
do { mark = __LINE__; return ; case __LINE__: ; } while (0)
```

Definiert in Zeile 11 der Datei [task.h](#).

7.34.1.4 taskPause

```
#define taskPause(  
    interval)
```

Wert:

```
timeStamp = millis(); while((millis() - timeStamp) < (interval)) taskSwitch()
```

Definiert in Zeile 12 der Datei [task.h](#).

7.34.1.5 taskWaitFor

```
#define taskWaitFor(  
    condition)
```

Wert:

```
while(!(condition)) taskSwitch();
```

Definiert in Zeile 13 der Datei [task.h](#).

7.34.1.6 taskStepName

```
#define taskStepName(  
    STEPNAME)
```

Wert:

```
TASKSTEP_##STEPNAME :
```

Definiert in Zeile 16 der Datei [task.h](#).

7.34.1.7 taskJumpTo

```
#define taskJumpTo(  
    STEPNAME)
```

Wert:

```
goto TASKSTEP_##STEPNAME
```

Definiert in Zeile 17 der Datei [task.h](#).

7.35 task.h

[gehe zur Dokumentation dieser Datei](#)

```
00001 #ifndef _TASK_H_
00002 #define _TASK_H_
00003
00004
00005 // grundlegene Worte um einen Task Bereich einzugrenzen
00006 #define taskBegin() static int mark = 0; static unsigned long __attribute__((unused)) timeStamp = 0;
    switch(mark){ case 0:
00007 #define taskEnd() }
00008
00009
00010 // Task Kontrol Worte, diese werden Taskwechsel einleiten
00011 #define taskSwitch() do { mark = __LINE__; return ; case __LINE__: ; } while (0)
00012 #define taskPause(interval) timeStamp = millis(); while((millis() - timeStamp) < (interval))
    taskSwitch()
00013 #define taskWaitFor(condition) while(!(condition)) taskSwitch();
00014
00015 // Benennen und anspringen von Schrittketten Verzweigungen
00016 #define taskStepName(STEPNAME) TASKSTEP_##STEPNAME :
00017 #define taskJumpTo(STEPNAME) goto TASKSTEP_##STEPNAME
00018
00019 #endif
```

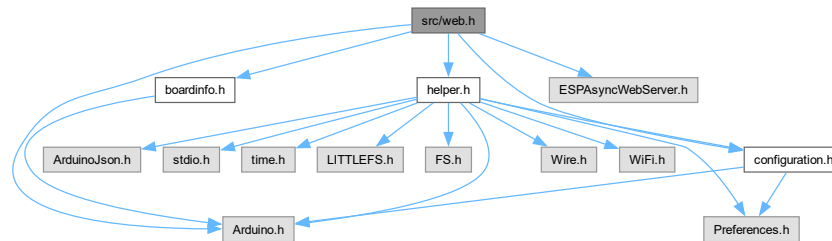
7.36 src/web.h-Dateireferenz

Webseite Variablen lesen und schreiben, Webseiten erstellen.

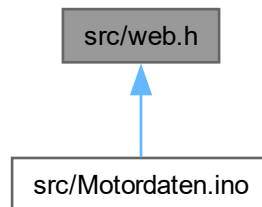
```
#include "helper.h"
#include "configuration.h"
#include "boardinfo.h"
#include <ESPAsyncWebServer.h>
```

```
#include <Arduino.h>
```

Include-Abhängigkeitsdiagramm für web.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- AsyncWebServer [server](#) (80)
- String [processor](#) (const String &var)
- String [replaceVariable](#) (const String &var)
- void [website](#) ()

Variablen

- String [sBoardInfo](#)
- [BoardInfo](#) boardInfo
- bool [IsRebootRequired](#) = false
- String [sCL_Status](#) = [sWifiStatus](#)(WiFi.status())

7.36.1 Ausführliche Beschreibung

Webseite Variablen lesen und schreiben, Webseiten erstellen.

Autor

Gerry Sebb

Version

0.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei [web.h](#).

7.36.2 Dokumentation der Funktionen

7.36.2.1 server()

```
AsyncWebServer server (  
    80 )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.36.2.2 processor()

```
String processor (  
    const String & var)
```

Definiert in Zeile [27](#) der Datei [web.h](#).

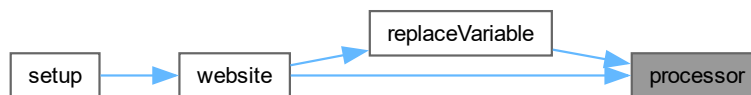
```
00028 {  
00029     if (var == "CONFIGPLACEHOLDER")  
00030     {  
00031         String buttons = "";  
00032         buttons += "<form onSubmit = \"event.preventDefault(); formToJson(this);\">";  
00033         buttons += "<p class=\"CInput\"><label>SSID </label><input type = \"text\" name = \"SSID\"  
value=\"\";  
00034         buttons += tAP_Config.wAP_SSID;  
00035         buttons += "\"/></p>";
```

```

00036     buttons += "<p class=\"CInput\"><label>IP </label><input type = \"text\" name = \"IP\"
value=\"";
00037     buttons += tAP_Config.wAP_IP;
00038     buttons += "</p>";
00039     buttons += "<p class=\"CInput\"><label>Password </label><input type = \"text\" name =
\"Password\" value=\"";
00040     buttons += tAP_Config.wAP_Password;
00041     buttons += "</p>";
00042     buttons += "<p class=\"CInput\"><label>Oil Offset </label><input type = \"text\" name =
\"Temp1Offset\" value=\"";
00043     buttons += tAP_Config.wTemp1_Offset;
00044     buttons += "</p>";
00045     buttons += "<p class=\"CInput\"><label>Mot Offset </label><input type = \"text\" name =
\"Temp2Offset\" value=\"";
00046     buttons += tAP_Config.wTemp2_Offset;
00047     buttons += "</p>";
00048     buttons += "<p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name
= \"Fuellstandmax\" value=\"";
00049     buttons += tAP_Config.wFuellstandmax;
00050     buttons += "</p>";
00051     buttons += "<p class=\"button\"><input type=\"submit\" value=\"Speichern\"></p>";
00052     buttons += "</form>";
00053     return buttons;
00054 }
00055 return String();
00056 }

```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.36.2.3 replaceVariable()

```

String replaceVariable (
    const String & var)

```

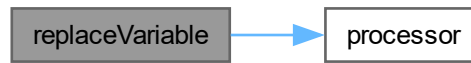
Definiert in Zeile 61 der Datei [web.h](#).

```

00062 {
00063     if (var == "sDrehzahl") return String(fDrehzahl,1);
00064     if (var == "sFuellstand") return String(FuelLevel,1);
00065     if (var == "sFuellstandmax") return String(FuelLevelMax,1);
00066     if (var == "sBordspannung") return String(fBordSpannung,1);
00067     if (var == "sOilTemp1") return String(fOilTemp1,1);
00068     if (var == "sMotTemp2") return String(fMotTemp2,1);
00069     if (var == "sTemp1Offset") return String(fTemp1Offset);
00070     if (var == "sTemp2Offset") return String(fTemp2Offset);
00071     if (var == "sBoardInfo") return sBoardInfo;
00072     if (var == "sFS_USpace") return String(LittleFS.usedBytes());
00073     if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00074     if (var == "sAP_IP") return WiFi.softAPIP().toString();
00075     if (var == "sAP_Clients") return String(sAP_Station);
00076     if (var == "sCL_Addr") return WiFi.localIP().toString();
00077     if (var == "sCL_Status") return String(sCL_Status);
00078     if (var == "sOneWire_Status") return String(sOneWire_Status);
00079     if (var == "sVersion") return Version;
00080     if (var == "sCounter") return String(Counter);
00081     if (var == "CONFIGPLACEHOLDER") return processor(var);
00082     return "NoVariable";
00083 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.36.2.4 website()

```
void website ()
```

Definiert in Zeile 85 der Datei [web.h](#).

```

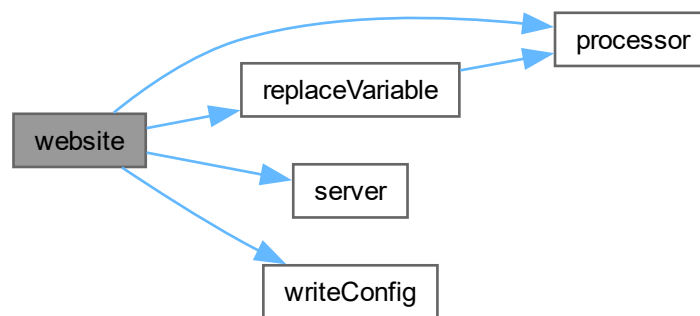
00085     {
00086         server.on("/favicon.ico", HTTP_GET, [] (AsyncWebServerRequest *request){
00087             request->send(LittleFS, "/favicon.ico", "image/x-icon");
00088         });
00089         server.on("/logo80.jpg", HTTP_GET, [] (AsyncWebServerRequest *request){
00090             request->send(LittleFS, "/logo80.jpg", "image/jpeg");
00091         });
00092         server.on("/", HTTP_GET, [] (AsyncWebServerRequest* request) {
00093             request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00094         });
00095         server.on("/system.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00096             request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00097         });
00098         server.on("/settings.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00099             request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00100         });
00101         server.on("/werte.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00102             request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00103         });
00104         server.on("/ueber.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00105             request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00106         });
00107         server.on("/reboot", HTTP_GET, [] (AsyncWebServerRequest * request) {
00108             request->send(LittleFS, "/reboot.html", String(), false, processor);
00109             IsRebootRequired = true;
00110         });
00111         server.on("/gauge.min.js", HTTP_GET, [] (AsyncWebServerRequest* request) {
00112             request->send(LittleFS, "/gauge.min.js");
00113         });
00114         server.on("/style.css", HTTP_GET, [] (AsyncWebServerRequest *request) {
00115             request->send(LittleFS, "/style.css", "text/css");
00116         });
00117         server.on("/settings.html", HTTP_POST, [] (AsyncWebServerRequest *request)
00118         {
00119             int count = request->params();
00120             Serial.printf("Anzahl: %i\n", count);
00121             for (int i = 0; i < count; i++)
  
```

```

00122     {
00123         AsyncWebParameter* p = request->getParam(i);
00124         Serial.print("Pwerte von der Internet - Seite: ");
00125         Serial.print("Param name: ");
00126         Serial.println(p->name());
00127         Serial.print("Param value: ");
00128         Serial.println(p->value());
00129         Serial.println("-----");
00130         // p->value in die config schreiben
00131         writeConfig(p->value());
00132     }
00133     request->send(200, "text/plain", "Daten gespeichert");
00134 };
00135
00136 }

```

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.36.3 Variablen-Dokumentation

7.36.3.1 sBoardInfo

`String sBoardInfo`

Definiert in Zeile 23 der Datei [web.h](#).

7.36.3.2 boardInfo

`BoardInfo boardInfo`

Definiert in Zeile 24 der Datei [web.h](#).

7.36.3.3 IsRebootRequired

```
bool IsRebootRequired = false
```

Definiert in Zeile 25 der Datei [web.h](#).

7.36.3.4 sCL_Status

```
String sCL_Status = sWifiStatus(WiFi.status())
```

Definiert in Zeile 59 der Datei [web.h](#).

7.37 web.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00012
00013 #include "helper.h"
00014 #include "configuration.h"
00015 #include "boardinfo.h"
00016 #include <ESPAsyncWebServer.h>
00017 #include <Arduino.h>
00018
00019 // Set web server port number to 80
00020 AsyncWebServer server(80);
00021
00022 // Info Board for HTML-Output
00023 String sBoardInfo;
00024 BoardInfo boardInfo;
00025 bool IsRebootRequired = false;
00026
00027 String processor(const String& var)
00028 {
00029     if (var == "CONFIGPLACEHOLDER")
00030     {
00031         String buttons = "";
00032         buttons += "<form onSubmit = \"event.preventDefault(); formToJson(this);\">";
00033         buttons += "<p class=\"CInput\"><label>SSID </label><input type = \"text\" name = \"SSID\"";
00034         value="\"";
00035         buttons += tAP_Config.wAP_SSID;
00036         buttons += "\"/></p>";
00037         buttons += "<p class=\"CInput\"><label>IP </label><input type = \"text\" name = \"IP\"";
00038         value="\"";
00039         buttons += tAP_Config.wAP_IP;
00040         buttons += "\"/></p>";
00041         buttons += "<p class=\"CInput\"><label>Password </label><input type = \"text\" name =";
00042         value="\"";
00043         buttons += tAP_Config.wAP_Password;
00044         buttons += "\"/></p>";
00045         buttons += "<p class=\"CInput\"><label>Oil Offset </label><input type = \"text\" name =";
00046         value="\"";
00047         buttons += tAP_Config.wTemp1_Offset;
00048         buttons += "\"/> &deg;C</p>";
00049         buttons += "<p class=\"CInput\"><label>Mot Offset </label><input type = \"text\" name =";
00050         value="\"";
00051         buttons += tAP_Config.wTemp2_Offset;
00052         buttons += "\"/> &deg;C</p>";
00053         buttons += "<p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name =";
00054         value="\"";
00055         buttons += tAP_Config.wFuellstandmax;
00056         buttons += "\"/> l</p>";
00057         buttons += "<p class=\"button\"><input type=\"submit\" value=\"Speichern\"></p>";
00058         buttons += "</form>";
00059         return buttons;
00060     }
00061     return String();
00062 }
00063
00064 //Variables for website
00065 String sCL_Status = sWifiStatus(WiFi.status());
00066
00067 String replaceVariable(const String& var)
```

```

00062 {
00063     if (var == "sDrehzahl") return String(fDrehzahl,1);
00064     if (var == "sFuellstand") return String(FuelLevel,1);
00065     if (var == "sFuellstandmax") return String(FuelLevelMax,1);
00066     if (var == "sBordspannung") return String(fBordSpannung,1);
00067     if (var == "sOilTemp1") return String(fOilTemp1,1);
00068     if (var == "sMotTemp2") return String(fMotTemp2,1);
00069     if (var == "sTemp1Offset") return String(fTemp1Offset);
00070     if (var == "sTemp2Offset") return String(fTemp2Offset);
00071     if (var == "sBoardInfo") return sBoardInfo;
00072     if (var == "sFS_USpace") return String(LittleFS.usedBytes());
00073     if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00074     if (var == "sAP_IP") return WiFi.softAPIP().toString();
00075     if (var == "sAP_Clients") return String(sAP_Station);
00076     if (var == "sCL_Addr") return WiFi.localIP().toString();
00077     if (var == "sCL_Status") return String(sCL_Status);
00078     if (var == "sOneWire_Status") return String(sOneWire_Status);
00079     if (var == "sVersion") return Version;
00080     if (var == "sCounter") return String(Counter);
00081     if (var == "CONFIGPLACEHOLDER") return processor(var);
00082     return "NoVariable";
00083 }
00084
00085 void website() {
00086     server.on("/favicon.ico", HTTP_GET, [] (AsyncWebServerRequest *request) {
00087         request->send(LittleFS, "/favicon.ico", "image/x-icon");
00088     });
00089     server.on("/logo80.jpg", HTTP_GET, [] (AsyncWebServerRequest *request) {
00090         request->send(LittleFS, "/logo80.jpg", "image/jpeg");
00091     });
00092     server.on("/", HTTP_GET, [] (AsyncWebServerRequest* request) {
00093         request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00094     });
00095     server.on("/system.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00096         request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00097     });
00098     server.on("/settings.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00099         request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00100     });
00101     server.on("/werte.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00102         request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00103     });
00104     server.on("/ueber.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00105         request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00106     });
00107     server.on("/reboot", HTTP_GET, [] (AsyncWebServerRequest * request) {
00108         request->send(LittleFS, "/reboot.html", String(), false, processor);
00109         IsRebootRequired = true;
00110     });
00111     server.on("/gauge.min.js", HTTP_GET, [] (AsyncWebServerRequest* request) {
00112         request->send(LittleFS, "/gauge.min.js");
00113     });
00114     server.on("/style.css", HTTP_GET, [] (AsyncWebServerRequest *request) {
00115         request->send(LittleFS, "/style.css", "text/css");
00116     });
00117     server.on("/settings.html", HTTP_POST, [] (AsyncWebServerRequest *request)
00118     {
00119         int count = request->params();
00120         Serial.printf("Anzahl: %i\n", count);
00121         for (int i = 0; i < count; i++)
00122         {
00123             AsyncWebParameter* p = request->getParam(i);
00124             Serial.print("PWerte von der Internet - Seite: ");
00125             Serial.print("Param name: ");
00126             Serial.println(p->name());
00127             Serial.print("Param value: ");
00128             Serial.println(p->value());
00129             Serial.println("-----");
00130             // p->value in die config schreiben
00131             writeConfig(p->value());
00132         }
00133         request->send(200, "text/plain", "Daten gespeichert");
00134     });
00135 }
00136 }
00137

```


Index

- ADC_Calibration_Value1
 - Motordaten.ino, [73](#)
- ADC_Calibration_Value2
 - Motordaten.ino, [74](#)
- ADCPin1
 - Motordaten.ino, [91](#)
- ADCPin2
 - Motordaten.ino, [91](#)
- Altitude
 - tBoatData, [18](#)
- AP_IP
 - configuration.h, [42](#)
- AP_PASSWORD
 - configuration.h, [42](#)
- AP_SSID
 - configuration.h, [42](#)
- Bat1Capacity
 - configuration.h, [45](#)
- Bat2Capacity
 - configuration.h, [45](#)
- BatSoC
 - configuration.h, [45](#)
- BatteryDCSendOffset
 - configuration.h, [38](#)
- BatteryDCStatusSendOffset
 - configuration.h, [38](#)
- baudrate
 - Motordaten.ino, [91](#)
- bClientConnected
 - configuration.h, [43](#)
- bConnect_CL
 - configuration.h, [43](#)
- bl2C_Status
 - configuration.h, [44](#)
- Blue
 - LED.h, [67](#)
- BoardInfo, [13](#)
 - BoardInfo, [13](#)
 - m_chipid, [15](#)
 - m_chipinfo, [15](#)
 - ShowChipID, [14](#)
 - ShowChipIDtoString, [14](#)
 - ShowChipInfo, [14](#)
 - ShowChipTemperature, [14](#)
- boardInfo
 - web.h, [108](#)
- BoardInfo.cpp
 - BUF, [30](#)
 - temprature_sens_read, [30](#)
- BordSpannung
 - configuration.h, [45](#)
- bsz1
 - hourmeter.h, [64](#)
- BUF
 - BoardInfo.cpp, [30](#)
- channel
 - configuration.h, [41](#)
- Checksum
 - NMEA0183Telegram.h, [99](#)
- chipid
 - configuration.h, [41](#)
- CL_IP
 - configuration.h, [42](#)
- CL_PASSWORD
 - configuration.h, [39](#)
- CL_SSID
 - configuration.h, [39](#)
- COG
 - tBoatData, [17](#)
- configuration.h
 - AP_IP, [42](#)
 - AP_PASSWORD, [42](#)
 - AP_SSID, [42](#)
 - Bat1Capacity, [45](#)
 - Bat2Capacity, [45](#)
 - BatSoC, [45](#)
 - BatteryDCSendOffset, [38](#)
 - BatteryDCStatusSendOffset, [38](#)
 - bClientConnected, [43](#)
 - bConnect_CL, [43](#)
 - bl2C_Status, [44](#)
 - BordSpannung, [45](#)
 - channel, [41](#)
 - chipid, [41](#)
 - CL_IP, [42](#)
 - CL_PASSWORD, [39](#)
 - CL_SSID, [39](#)
 - Counter, [45](#)
 - dMWV_WindDirectionT, [47](#)
 - dMWV_WindSpeedM, [47](#)
 - DNS_PORT, [40](#)
 - dVWR_WindAngle, [47](#)
 - dVWR_WindDirectionM, [47](#)
 - dVWR_WindSpeedkn, [47](#)
 - dVWR_WindSpeedms, [47](#)
 - Engine_RPM_Pin, [40](#)
 - EngineOn, [45](#)
 - EngineRPM, [45](#)

- EngineSendOffset, 38
- EngineStatus, 40
- ESP32_CAN_RX_PIN, 37
- ESP32_CAN_TX_PIN, 37
- fbmp_altitude, 43
- fbmp_pressure, 43
- fbmp_temperature, 43
- fBordSpannung, 46
- fDrehzahl, 46
- fGaugeDrehzahl, 46
- fMotTemp2, 46
- fOilTemp1, 46
- fTemp1Offset, 46
- fTemp2Offset, 46
- FuelLevel, 44
- FuelLevelMax, 44
- Gateway, 42
- hide_SSID, 41
- HostName, 39
- i, 41
- I2C_SCL, 39
- I2C_SDA, 39
- id, 41
- iDistance, 44
- iMaxSonar, 44
- IP, 42
- iSTA_on, 43
- max_connection, 42
- MotTemp, 44
- N2K_SOURCE, 38
- NMask, 42
- NodeAddress, 41
- Off, 40
- OilTemp, 44
- On, 40
- ONE_WIRE_BUS, 40
- PAGE_REFRESH, 38
- preferences, 41
- RPM_Calibration_Value, 39
- RPMsSendOffset, 38
- sAP_Station, 43
- SEALEVELPRESSURE_HPA, 39
- SELF_IP, 43
- SERVER_HOST_NAME, 40
- sl2C_Status, 44
- SlowDataUpdatePeriod, 38
- SoCError, 45
- sOneWire_Status, 46
- sOrient, 47
- sSTBB, 47
- TankSendOffset, 38
- tAP_Config, 41
- TCP_PORT, 40
- udpAddress, 48
- udpPort, 48
- Version, 37
- WEB_TITEL, 39
- Counter
 - configuration.h, 45
 - CounterOld
 - hourmeter.h, 65
 - data/index.html, 23
 - data/reboot.html, 25
 - data/settings.html, 26
 - data/system.html, 27
 - data/ueber.html, 27
 - data/werte.html, 27
 - DaysSince1970
 - tBoatData, 17
 - debug_log
 - Motordaten.ino, 74
 - DGPSAge
 - tBoatData, 18
 - DGPSReferenceStationID
 - tBoatData, 20
 - dMWV_WindDirectionT
 - configuration.h, 47
 - dMWV_WindSpeedM
 - configuration.h, 47
 - DNS_PORT
 - configuration.h, 40
 - dVWR_WindAngle
 - configuration.h, 47
 - dVWR_WindDirectionM
 - configuration.h, 47
 - dVWR_WindSpeedkn
 - configuration.h, 47
 - dVWR_WindSpeedms
 - configuration.h, 47
 - Engine_RPM_Pin
 - configuration.h, 40
 - ENABLE_DEBUG_LOG
 - Motordaten.ino, 73
 - EngineHours
 - hourmeter.h, 63
 - EngineOn
 - configuration.h, 45
 - EngineRPM
 - configuration.h, 45
 - EngineSendOffset
 - configuration.h, 38
 - EngineStatus
 - configuration.h, 40
 - ESP32_CAN_RX_PIN
 - configuration.h, 37
 - ESP32_CAN_TX_PIN
 - configuration.h, 37
 - fbmp_altitude
 - configuration.h, 43
 - fbmp_pressure
 - configuration.h, 43
 - fbmp_temperature
 - configuration.h, 43
 - fBordSpannung

- configuration.h, 46
- fDrehzahl
 - configuration.h, 46
- fGaugeDrehzahl
 - configuration.h, 46
- flashLED
 - LED.h, 68
- fMotTemp2
 - configuration.h, 46
- fOilTemp1
 - configuration.h, 46
- freeHeapSpace
 - helper.h, 51
- fTemp1Offset
 - configuration.h, 46
- fTemp2Offset
 - configuration.h, 46
- FuelLevel
 - configuration.h, 44
- FuelLevelMax
 - configuration.h, 44
- Gateway
 - configuration.h, 42
- GeoidalSeparation
 - tBoatData, 18
- GetTemperature
 - Motordaten.ino, 78
- GPSQualityIndicator
 - tBoatData, 19
- GPSTime
 - tBoatData, 18
- Green
 - LED.h, 67
- handleInterrupt
 - Motordaten.ino, 74
- HDOP
 - tBoatData, 18
- helper.h
 - freeHeapSpace, 51
 - I2C_scan, 56
 - listDir, 53
 - readConfig, 54
 - ShowTime, 51
 - sWifiStatus, 57
 - toChar, 57
 - WiFiDiag, 52
 - writeConfig, 55
- hide_SSID
 - configuration.h, 41
- HostName
 - configuration.h, 39
- hourmeter.h
 - bsz1, 64
 - CounterOld, 65
 - EngineHours, 63
 - lastRun, 64
 - laststate1, 65
 - milliRest, 65
 - state1, 65
- i
 - configuration.h, 41
- I2C_scan
 - helper.h, 56
- I2C_SCL
 - configuration.h, 39
- I2C_SDA
 - configuration.h, 39
- id
 - configuration.h, 41
- iDistance
 - configuration.h, 44
- iMaxSonar
 - configuration.h, 44
- InitNextUpdate
 - Motordaten.ino, 80
- IP
 - configuration.h, 42
- IsRebootRequired
 - web.h, 108
- iSTA_on
 - configuration.h, 43
- IsTimeToUpdate
 - Motordaten.ino, 79
- Last_int_time
 - Motordaten.ino, 90
- lastRun
 - hourmeter.h, 64
- laststate1
 - hourmeter.h, 65
- Latitude
 - tBoatData, 18
- LED
 - LED.h, 67
- LED.h
 - Blue, 67
 - flashLED, 68
 - Green, 67
 - LED, 67
 - LEDblink, 68
 - LEDBoard, 67
 - LEDflash, 68
 - LEDInit, 68
 - LEDOff, 69
 - LEDOff_RGB, 70
 - LEDOn, 69
 - Red, 67
- LEDblink
 - LED.h, 68
- LEDBoard
 - LED.h, 67
- LEDflash
 - LED.h, 68
- LEDInit
 - LED.h, 68

- LEDOff
 - LED.h, [69](#)
- LEDOff_RGB
 - LED.h, [70](#)
- LEDOn
 - LED.h, [69](#)
- listDir
 - helper.h, [53](#)
- Longitude
 - tBoatData, [18](#)
- loop
 - Motordaten.ino, [87](#)
- m_chipid
 - BoardInfo, [15](#)
- m_chipinfo
 - BoardInfo, [15](#)
- max_connection
 - configuration.h, [42](#)
- milliRest
 - hourmeter.h, [65](#)
- MKSPIFFSTOOL
 - replace_fs, [11](#)
- MOBActivated
 - tBoatData, [20](#)
- MotorCoolant
 - Motordaten.ino, [90](#)
- MotorData NMEA2000, [1](#)
- Motordaten.ino
 - ADC_Calibration_Value1, [73](#)
 - ADC_Calibration_Value2, [74](#)
 - ADCpin1, [91](#)
 - ADCpin2, [91](#)
 - baudrate, [91](#)
 - debug_log, [74](#)
 - ENABLE_DEBUG_LOG, [73](#)
 - GetTemperature, [78](#)
 - handleInterrupt, [74](#)
 - InitNextUpdate, [80](#)
 - IsTimeToUpdate, [79](#)
 - Last_int_time, [90](#)
 - loop, [87](#)
 - MotorCoolant, [90](#)
 - MotorOil, [91](#)
 - mux, [90](#)
 - oneWire, [74](#), [90](#)
 - PeriodCount, [90](#)
 - PROGMEM, [89](#)
 - ReadRPM, [79](#)
 - ReadVoltage, [87](#)
 - rs_config, [91](#)
 - SendN2kBattery, [82](#)
 - SendN2kDCStatus, [81](#)
 - SendN2kEngineData, [84](#)
 - SendN2kEngineRPM, [86](#)
 - SendN2kTankLevel, [83](#)
 - SetNextUpdate, [81](#)
 - setup, [75](#)
 - StartValue, [90](#)
 - Task1, [91](#)
 - timer, [90](#)
- MotorOil
 - Motordaten.ino, [91](#)
- MotTemp
 - configuration.h, [44](#)
- mux
 - Motordaten.ino, [90](#)
- N2K_SOURCE
 - configuration.h, [38](#)
- NMask
 - configuration.h, [42](#)
- NMEA0183Telegram.h
 - Checksum, [99](#)
 - sendXDR, [99](#)
- NodeAddress
 - configuration.h, [41](#)
- Off
 - configuration.h, [40](#)
- Offset
 - tBoatData, [19](#)
- OilTemp
 - configuration.h, [44](#)
- On
 - configuration.h, [40](#)
- ONE_WIRE_BUS
 - configuration.h, [40](#)
- oneWire
 - Motordaten.ino, [74](#), [90](#)
- PAGE_REFRESH
 - configuration.h, [38](#)
- PeriodCount
 - Motordaten.ino, [90](#)
- preferences
 - configuration.h, [41](#)
- processor
 - web.h, [105](#)
- PROGMEM
 - Motordaten.ino, [89](#)
- readConfig
 - helper.h, [54](#)
- README.md, [28](#)
- ReadRPM
 - Motordaten.ino, [79](#)
- ReadVoltage
 - Motordaten.ino, [87](#)
- Red
 - LED.h, [67](#)
- replace_fs, [11](#)
 - MKSPIFFSTOOL, [11](#)
- replace_fs.py, [28](#)
- replaceVariable
 - web.h, [106](#)
- RPM_Calibration_Value
 - configuration.h, [39](#)

- RPMsSendOffset
 - configuration.h, 38
- rs_config
 - Motordaten.ino, 91
- sAP_Station
 - configuration.h, 43
- SatelliteCount
 - tBoatData, 20
- sBoardInfo
 - web.h, 108
- sCL_Status
 - web.h, 109
- SEALEVELPRESSURE_HPA
 - configuration.h, 39
- SELF_IP
 - configuration.h, 43
- SendN2kBattery
 - Motordaten.ino, 82
- SendN2kDCStatus
 - Motordaten.ino, 81
- SendN2kEngineData
 - Motordaten.ino, 84
- SendN2kEngineRPM
 - Motordaten.ino, 86
- SendN2kTankLevel
 - Motordaten.ino, 83
- sendXDR
 - NMEA0183Telegram.h, 99
- server
 - web.h, 105
- SERVER_HOST_NAME
 - configuration.h, 40
- SetNextUpdate
 - Motordaten.ino, 81
- setup
 - Motordaten.ino, 75
- ShowChipID
 - BoardInfo, 14
- ShowChipIDtoString
 - BoardInfo, 14
- ShowChipInfo
 - BoardInfo, 14
- ShowChipTemperature
 - BoardInfo, 14
- ShowTime
 - helper.h, 51
- sl2C_Status
 - configuration.h, 44
- SlowDataUpdatePeriod
 - configuration.h, 38
- SoCError
 - configuration.h, 45
- SOG
 - tBoatData, 17
- sOneWire_Status
 - configuration.h, 46
- sOrient
 - configuration.h, 47
- src/BoardInfo.cpp, 29, 30
- src/BoardInfo.h, 32, 33
- src/BoatData.h, 33, 34
- src/configuration.h, 34, 48
- src/helper.h, 50, 58
- src/hourmeter.h, 61, 65
- src/LED.h, 66, 70
- src/Motordaten.ino, 71, 92
- src/NMEA0183Telegram.h, 98, 100
- src/task.h, 101, 103
- src/web.h, 103, 109
- sSTBB
 - configuration.h, 47
- StartValue
 - Motordaten.ino, 90
- state1
 - hourmeter.h, 65
- Status
 - tBoatData, 20
- sWifiStatus
 - helper.h, 57
- TankSendOffset
 - configuration.h, 38
- tAP_Config
 - configuration.h, 41
- task.h
 - taskBegin, 102
 - taskEnd, 102
 - taskJumpTo, 103
 - taskPause, 102
 - taskStepName, 102
 - taskSwitch, 102
 - taskWaitFor, 102
- Task1
 - Motordaten.ino, 91
- taskBegin
 - task.h, 102
- taskEnd
 - task.h, 102
- taskJumpTo
 - task.h, 103
- taskPause
 - task.h, 102
- taskStepName
 - task.h, 102
- taskSwitch
 - task.h, 102
- taskWaitFor
 - task.h, 102
- tBoatData, 16
 - Altitude, 18
 - COG, 17
 - DaysSince1970, 17
 - DGPSAge, 18
 - DGPSReferenceStationID, 20
 - GeoidalSeparation, 18
 - GPSQualityIndicator, 19
 - GPSTime, 18

- HDOP, [18](#)
- Latitude, [18](#)
- Longitude, [18](#)
- MOBActivated, [20](#)
- Offset, [19](#)
- SatelliteCount, [20](#)
- SOG, [17](#)
- Status, [20](#)
- tBoatData, [17](#)
- TrueHeading, [17](#)
- Variation, [17](#)
- WaterDepth, [19](#)
- WaterTemperature, [18](#)
- WindAngle, [19](#)
- WindDirectionM, [19](#)
- WindDirectionT, [19](#)
- WindSpeedK, [19](#)
- WindSpeedM, [19](#)
- TCP_PORT
 - configuration.h, [40](#)
- temperature_sens_read
 - BoardInfo.cpp, [30](#)
- timer
 - Motordaten.ino, [90](#)
- toChar
 - helper.h, [57](#)
- TrueHeading
 - tBoatData, [17](#)
- udpAddress
 - configuration.h, [48](#)
- udpPort
 - configuration.h, [48](#)
- Variation
 - tBoatData, [17](#)
- Version
 - configuration.h, [37](#)
- wAP_IP
 - Web_Config, [21](#)
- wAP_Password
 - Web_Config, [21](#)
- wAP_SSID
 - Web_Config, [21](#)
- WaterDepth
 - tBoatData, [19](#)
- WaterTemperature
 - tBoatData, [18](#)
- web.h
 - boardInfo, [108](#)
 - IsRebootRequired, [108](#)
 - processor, [105](#)
 - replaceVariable, [106](#)
 - sBoardInfo, [108](#)
 - sCL_Status, [109](#)
 - server, [105](#)
 - website, [107](#)
- Web_Config, [20](#)
- wAP_IP, [21](#)
- wAP_Password, [21](#)
- wAP_SSID, [21](#)
- wFuellstandmax, [21](#)
- wTemp1_Offset, [21](#)
- wTemp2_Offset, [21](#)
- WEB_TITEL
 - configuration.h, [39](#)
- website
 - web.h, [107](#)
- wFuellstandmax
 - Web_Config, [21](#)
- WiFiDiag
 - helper.h, [52](#)
- WindAngle
 - tBoatData, [19](#)
- WindDirectionM
 - tBoatData, [19](#)
- WindDirectionT
 - tBoatData, [19](#)
- WindSpeedK
 - tBoatData, [19](#)
- WindSpeedM
 - tBoatData, [19](#)
- writeConfig
 - helper.h, [55](#)
- wTemp1_Offset
 - Web_Config, [21](#)
- wTemp2_Offset
 - Web_Config, [21](#)