

## Motordaten

V 2.7

Erzeugt von Doxygen 1.13.2



<b>1 MotorData NMEA2000</b>	<b>1</b>
1.1 Description	1
1.2 Based on the work of	2
1.3 Website	2
1.4 Plotter	2
1.5 Wiring diagram	2
1.6 PCB Layout	2
1.7 Details	2
1.8 Partlist:	3
1.9 Changes	4
<b>2 Verzeichnis der Namensbereiche</b>	<b>5</b>
2.1 Liste aller Namensbereiche	5
<b>3 Klassen-Verzeichnis</b>	<b>7</b>
3.1 Auflistung der Klassen	7
<b>4 Datei-Verzeichnis</b>	<b>9</b>
4.1 Auflistung der Dateien	9
<b>5 Dokumentation der Namensbereiche</b>	<b>11</b>
5.1 replace_fs-Namensbereichsreferenz	11
5.1.1 Variablen-Dokumentation	11
5.1.1.1 MKSPIFFSTOOL	11
<b>6 Klassen-Dokumentation</b>	<b>13</b>
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 wMotor_Offset	21
6.3.2.5 wCoolant_Offset	21
6.3.2.6 wFuellstandmax	21
6.3.2.7 wADC1_Cal	21
6.3.2.8 wADC2_Cal	21
<b>7 Datei-Dokumentation</b>	<b>23</b>
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 . . . . .	28
7.12 werte.html . . . . .	28
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 RPMsSendOffset . . . . .	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 sHeapspace . . . . .	41
7.22.4.7 tAP_Config . . . . .	41
7.22.4.8 channel . . . . .	42
7.22.4.9 hide_SSID . . . . .	42
7.22.4.10 max_connection . . . . .	42
7.22.4.11 IP . . . . .	42
7.22.4.12 Gateway . . . . .	42
7.22.4.13 NMask . . . . .	42
7.22.4.14 AP_SSID . . . . .	42
7.22.4.15 AP_PASSWORD . . . . .	42
7.22.4.16 AP_IP . . . . .	43
7.22.4.17 CL_IP . . . . .	43
7.22.4.18 SELF_IP . . . . .	43
7.22.4.19 sAP_Station . . . . .	43
7.22.4.20 iSTA_on . . . . .	43
7.22.4.21 bConnect_CL . . . . .	43
7.22.4.22 bClientConnected . . . . .	43
7.22.4.23 ADC_Calibration_Value1 . . . . .	44
7.22.4.24 ADC_Calibration_Value2 . . . . .	44
7.22.4.25 fbmp_temperature . . . . .	44
7.22.4.26 fbmp_pressure . . . . .	44
7.22.4.27 fbmp_altitude . . . . .	44
7.22.4.28 sl2C_Status . . . . .	44
7.22.4.29 bl2C_Status . . . . .	44
7.22.4.30 iMaxSonar . . . . .	45
7.22.4.31 iDistance . . . . .	45
7.22.4.32 FuelLevel . . . . .	45

7.22.4.33 FuelLevelMax . . . . .	45
7.22.4.34 CoolantTemp . . . . .	45
7.22.4.35 MotorTemp . . . . .	45
7.22.4.36 EngineRPM . . . . .	45
7.22.4.37 BordSpannung . . . . .	45
7.22.4.38 EngineOn . . . . .	46
7.22.4.39 motorErrorReported . . . . .	46
7.22.4.40 coolantErrorReported . . . . .	46
7.22.4.41 Counter . . . . .	46
7.22.4.42 Bat1Capacity . . . . .	46
7.22.4.43 Bat2Capacity . . . . .	46
7.22.4.44 SoCError . . . . .	46
7.22.4.45 BatSoC . . . . .	46
7.22.4.46 sOneWire_Status . . . . .	47
7.22.4.47 fDrehzahl . . . . .	47
7.22.4.48 fGaugeDrehzahl . . . . .	47
7.22.4.49 fBordSpannung . . . . .	47
7.22.4.50 fCoolantTemp . . . . .	47
7.22.4.51 fMotorTemp . . . . .	47
7.22.4.52 fCoolantOffset . . . . .	47
7.22.4.53 fMotorOffset . . . . .	47
7.22.4.54 sSTBB . . . . .	48
7.22.4.55 sOrient . . . . .	48
7.22.4.56 dMWV_WindDirectionT . . . . .	48
7.22.4.57 dMWV_WindSpeedM . . . . .	48
7.22.4.58 dVWR_WindDirectionM . . . . .	48
7.22.4.59 dVWR_WindAngle . . . . .	48
7.22.4.60 dVWR_WindSpeedkn . . . . .	48
7.22.4.61 dVWR_WindSpeedms . . . . .	48
7.22.4.62 udpAddress . . . . .	49
7.22.4.63 udpPort . . . . .	49
7.23 configuration.h . . . . .	49
7.24 src/helper.h-Dateireferenz . . . . .	51
7.24.1 Ausführliche Beschreibung . . . . .	52
7.24.2 Dokumentation der Funktionen . . . . .	52
7.24.2.1 ShowTime() . . . . .	52
7.24.2.2 freeHeapSpace() . . . . .	53
7.24.2.3 WiFiDiag() . . . . .	53
7.24.2.4 listDir() . . . . .	54
7.24.2.5 readConfig() . . . . .	55
7.24.2.6 writeConfig() [1/2] . . . . .	56
7.24.2.7 writeConfig() [2/2] . . . . .	57

7.24.2.8 I2C_scan()	58
7.24.2.9 sWifiStatus()	59
7.24.2.10 toChar()	59
7.25 helper.h	60
7.26 src/hourmeter.h-Dateireferenz	63
7.26.1 Ausführliche Beschreibung	65
7.26.2 Dokumentation der Funktionen	65
7.26.2.1 EngineHours()	65
7.26.3 Variablen-Dokumentation	66
7.26.3.1 bsz1	66
7.26.3.2 lastRun	66
7.26.3.3 CounterOld	66
7.26.3.4 milliRest	66
7.26.3.5 state1	66
7.26.3.6 laststate1	67
7.27 hourmeter.h	67
7.28 src/LED.h-Dateireferenz	67
7.28.1 Ausführliche Beschreibung	69
7.28.2 Dokumentation der Aufzählungstypen	69
7.28.2.1 LED	69
7.28.3 Dokumentation der Funktionen	70
7.28.3.1 LEDblink()	70
7.28.3.2 LEDflash()	70
7.28.3.3 flashLED()	71
7.28.3.4 LEDInit()	71
7.28.3.5 LEDon()	71
7.28.3.6 LEDoff()	72
7.28.3.7 LEDoff_RGB()	72
7.29 LED.h	72
7.30 src/LEDIndicator.h-Dateireferenz	73
7.30.1 Ausführliche Beschreibung	75
7.30.2 Dokumentation der Funktionen	75
7.30.2.1 LoopIndicator()	75
7.30.3 Variablen-Dokumentation	76
7.30.3.1 ErrorOff	76
7.30.3.2 ErrorOn	76
7.31 LEDIndicator.h	76
7.32 src/Motordaten.ino-Dateireferenz	77
7.32.1 Ausführliche Beschreibung	78
7.32.2 Makro-Dokumentation	79
7.32.2.1 ENABLE_DEBUG_LOG	79
7.32.3 Dokumentation der Funktionen	79



7.32.3.1 oneWire()	79
7.32.3.2 debug_log()	79
7.32.3.3 handleInterrupt()	79
7.32.3.4 setup()	80
7.32.3.5 GetTemperature()	83
7.32.3.6 ReadRPM()	84
7.32.3.7 IsTimeToUpdate()	85
7.32.3.8 InitNextUpdate()	85
7.32.3.9 SetNextUpdate()	86
7.32.3.10 SendN2kDCStatus()	87
7.32.3.11 SendN2kBattery()	88
7.32.3.12 SendN2kTankLevel()	89
7.32.3.13 SendN2kEngineData()	90
7.32.3.14 SendN2kEngineRPM()	91
7.32.3.15 ReadVoltage()	92
7.32.3.16 loop()	93
7.32.4 Variablen-Dokumentation	94
7.32.4.1 PROGMEM	94
7.32.4.2 StartValue	95
7.32.4.3 PeriodCount	95
7.32.4.4 Last_int_time	95
7.32.4.5 timer	95
7.32.4.6 mux	95
7.32.4.7 oneWire	95
7.32.4.8 MotorCoolant	96
7.32.4.9 MotorOil	96
7.32.4.10 ADCpin2	96
7.32.4.11 ADCpin1	96
7.32.4.12 Task1	96
7.32.4.13 baudrate	96
7.32.4.14 rs_config	97
7.33 Motordaten.ino	97
7.34 src/NMEA0183Telegram.h-Dateireferenz	103
7.34.1 Ausführliche Beschreibung	104
7.34.2 Dokumentation der Funktionen	104
7.34.2.1 CheckSum()	104
7.34.2.2 sendXDR()	105
7.34.2.3 sendRPM()	105
7.35 NMEA0183Telegram.h	106
7.36 src/task.h-Dateireferenz	108
7.36.1 Makro-Dokumentation	108
7.36.1.1 taskBegin	108

---

7.36.1.2 taskEnd	109
7.36.1.3 taskSwitch	109
7.36.1.4 taskPause	109
7.36.1.5 taskWaitFor	109
7.36.1.6 taskStepName	109
7.36.1.7 taskJumpTo	110
7.37 task.h	110
7.38 src/web.h-Dateireferenz	110
7.38.1 Ausführliche Beschreibung	111
7.38.2 Dokumentation der Funktionen	112
7.38.2.1 server()	112
7.38.2.2 processor()	112
7.38.2.3 replaceVariable()	113
7.38.2.4 website()	114
7.38.3 Variablen-Dokumentation	115
7.38.3.1 webSocket	115
7.38.3.2 sBoardInfo	115
7.38.3.3 boardInfo	115
7.38.3.4 IsRebootRequired	116
7.38.3.5 sCL_Status	116
7.39 web.h	116
<b>Index</b>	<b>119</b>

# 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 temperatur 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

## 1.7 Details

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.

It's also possible with Unisensor case.

- UNI sensor [Link](#)

Setup: Open Browser, go to Settings an set your max. Tanklevel, ADC1 Calibration and ADC2 Calibration. For ADC1 mount 90 Ohm Resistor in the input and set calibration value ca. 170 and control on the Plotter "Fuel" = 50% from max. Adjust. For ADC2 measuring voltage with multimeter and set calibration value ca. 17.0 and control the Plotter "Batterie" field. Adjust.

## 1.8 Partlist:

- PCB by Aisler [Link](#)

Assembly: [MD N2k\\_\\_Assembly.pdf](#)

- 1 C1 10 $\mu$  CP\_EIA-7343-15\_Kemet-W\_Pad2.25x2.55mm\_HandSolder 1
- 2 C2 22 $\mu$  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](#)

## 1.9 Changes

- Version 2.7 better error display for website and [LED](#)
- Version 2.6 Add value for ADC calibration to setting.html
- Version 2.5 Error handling OneWire-Temperatur (set sensor output to -5 °C) and change PIN to GPIO14
- 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:

<a href="#">replace_fs</a> . . . . .	11
--------------------------------------	----





## Kapitel 3

# Klassen-Verzeichnis

### 3.1 Auflistung der Klassen

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

<a href="#">BoardInfo</a>	.....	13
<a href="#">tBoatData</a>	.....	16
<a href="#">Web_Config</a>	.....	20



# Kapitel 4

## Datei-Verzeichnis

### 4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

<a href="#">replace_fs.py</a>	28
<a href="#">data/index.html</a>	23
<a href="#">data/reboot.html</a>	25
<a href="#">data/settings.html</a>	26
<a href="#">data/system.html</a>	27
<a href="#">data/ueber.html</a>	27
<a href="#">data/werte.html</a>	28
<a href="#">src/BoardInfo.cpp</a>	
Boardinfo	29
<a href="#">src/BoardInfo.h</a>	
Hardwareinfo from ESP Board	32
<a href="#">src/BoatData.h</a>	33
<a href="#">src/configuration.h</a>	
Konfiguration für GPIO und Variable	34
<a href="#">src/helper.h</a>	
Hilfsfunktionen	51
<a href="#">src/hourmeter.h</a>	
Betriebsstundenzähler	63
<a href="#">src/LED.h</a>	
LED Ansteuerung	67
<a href="#">src/LEDindicator.h</a>	
LED Betriebsanzeige	73
<a href="#">src/Motordaten.ino</a>	
Motordaten NMEA2000	77
<a href="#">src/NMEA0183Telegram.h</a>	
NMEA0183 Telegramme senden	103
<a href="#">src/task.h</a>	108
<a href="#">src/web.h</a>	
Webseite Variablen lesen und schreiben, Webseiten erstellen	110



# 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 [wMotor\\_Offset](#) [6]
- char [wCoolant\\_Offset](#) [6]
- char [wFuellstandmax](#) [6]
- char [wADC1\\_Cal](#) [6]
- char [wADC2\\_Cal](#) [6]

### 6.3.1 Ausführliche Beschreibung

Definiert in Zeile 47 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 49 der Datei [configuration.h](#).

#### 6.3.2.2 wAP\_SSID

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

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

#### 6.3.2.3 wAP\_Password

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

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

#### 6.3.2.4 wMotor\_Offset

```
char Web_Config::wMotor_Offset[6]
```

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

#### 6.3.2.5 wCoolant\_Offset

```
char Web_Config::wCoolant_Offset[6]
```

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

#### 6.3.2.6 wFuelstandmax

```
char Web_Config::wFuelstandmax[6]
```

Definiert in Zeile 54 der Datei [configuration.h](#).

#### 6.3.2.7 wADC1\_Cal

```
char Web_Config::wADC1_Cal[6]
```

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

#### 6.3.2.8 wADC2\_Cal

```
char Web_Config::wADC2_Cal[6]
```

Definiert in Zeile 56 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='%sMotorTemp% &deg;C'
00074         data-value='%sMotorTemp%'
00075     ></canvas>
00076     <canvas data-type="radial-gauge"
00077         data-width="200"
00078         data-height="200"
00079         data-units="%deg;C"
00080         data-title="K&uuml;hlwasser 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='%sCoolantTemp% &deg;C'
00105         data-value='%sCoolantTemp%'
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"> K&uuml;lhlwassertemperatur: %sCoolantTemp% &deg;C<br>
00012     Offset: %sCoolantOffset% &deg;C<br>
00013     Motortemperatur: %sMotorTemp% &deg;C<br>
00014     Offset: %sMotorOffset% &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 CoolantOffset = form.CoolantOffset.value;
00023       var MotorOffset = form.MotorOffset.value;
00024       var Fuellstandmax = form.Fuellstandmax.value;
00025       var ADC1_Cal = form.ADC1_Cal.value;
00026       var ADC2_Cal = form.ADC2_Cal.value;
00027
00028       var jsonFormInfo = JSON.stringify({
00029         SSID: SSID,
00030         IP: IP,
00031         Password: Password,
00032         CoolantOffset: CoolantOffset,
00033         MotorOffset: MotorOffset,
00034         Fuellstandmax: Fuellstandmax,
00035         ADC1_Cal: ADC1_Cal,
00036         ADC2_Cal: ADC2_Cal
00037       });
00038
00039       xhr.open("POST", "/settings.html?save=" + jsonFormInfo, true);
00040       /* window.alert("Json function send end"); */
00041       xhr.send();
00042       window.alert("Gespeichert!");
00043     }
00044   </script>
00045
00046   <p class="label">Nach &Auml;nderungen neu starten!</p>
00047
00048   <button class="button" onclick="reboot_handler()">Neustart</button>
00049
00050   <p id="status"></p>
00051   <script>
00052     function reboot_handler()
00053     {
00054       document.getElementById("status").innerHTML = "Starte Reboot ...";
00055       var xhr = new XMLHttpRequest();
00056       xhr.open("GET", "/reboot", true);
00057       xhr.send();
00058       setTimeout(function(){ window.open("/reboot","_self"); }, 500);
00059     }
00060   </script>
00061
00062
00063   <ul class="bottomnav">
00064     <li><a href="/">Home</a></li>
00065     <li><a href="werte.html">Werte</a></li>
00066     <li><a class="active" href="settings.html">Settings</a></li>
00067     <li><a href="system.html">System</a></li>
00068     <li class="right"><a href="ueber.html">About</a></li>

```

```
00069     </ul>
00070 </body>
00071 </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% Sensoren 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     <p class="label">Free Heapspace: %sHeapspace%</p>
00017     <br />
00018     <br />
00019     <ul class="bottomnav">
00020         <li><a href="/">Home</a></li>
00021         <li><a href="werte.html">Werte</a></li>
00022         <li><a href="settings.html">Setting</a></li>
00023         <li><a class="active" href="system.html">System</a></li>
00024         <li class="right"><a href="ueber.html">About</a></li>
00025     </ul>
00026 </BODY>
00027 </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>DC Status</p>
00014     <p class="label">Bordspannung: %sBordspannung% V</p>
00015     <p>Maschine</p>
00016     <p id="coolwater" class="label">K&uuml;hlwasser Temperatur: %sCoolantTemp% &deg;C</br>
00017         Offset: %sCoolantOffset% &deg;C</br>
00018         Fehler: %sCoolantError%
00019     </p>
00020     <p class="label">Motor Temperatur: %sMotorTemp% &deg;C</br>
00021         Offset: %sMotorOffset% &deg;C</br>
00022         Fehler: %sMotorError%
00023     </p>
00024     <p class="label">Motor Drehzahl: %sDrehzahl% U &frasl; min</p>
00025     <p class="label">Maschinenstunden: %sCounter% h</p>
00026     <p>Tank</p>
00027     <p class="label">Tank F&uuml;llstand: %sFuellstand% &#37;</br>max. F&uuml;llstand:
00028     %sFuellstandmax% l</p>
00028     <p>ADC Kalibrierung</p>
00029     <p class="label">ADC1: %sADC1_Cal%</br>ADC2: %sADC2_Cal%</p>
00030
00031     <ul class="bottomnav">
00032         <li><a href="/">Home</a></li>
00033         <li><a class="active" href="/">Werte</a></li>
00034         <li><a href="settings.html">Setting</a></li>
00035         <li><a href="system.html">System</a></li>
00036         <li class="right"><a href="ueber.html">About</a></li>
00037     </ul>
00038 </body>
00039 </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")

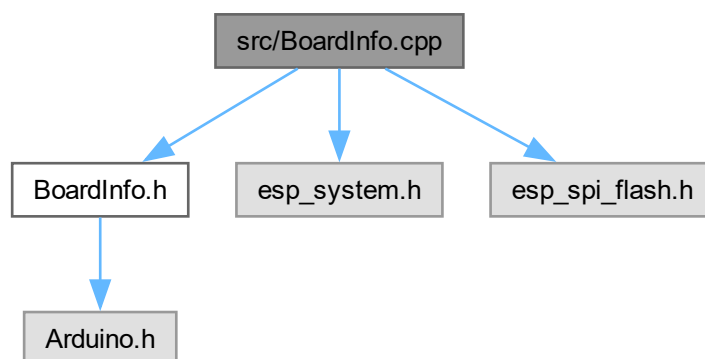
```

## 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 }

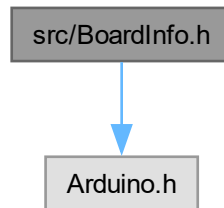
```

## 7.18 src/BoardInfo.h-Dateireferenz

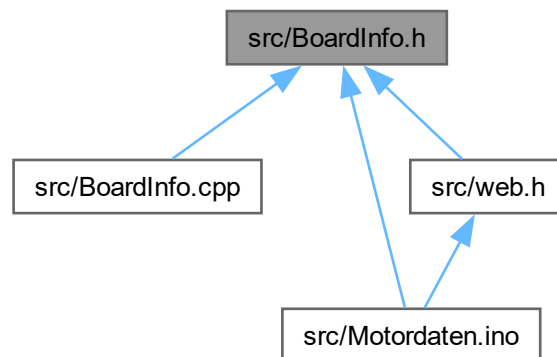
Hardwareinfo from ESP Board.

```
#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

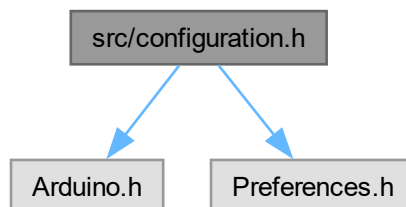
Konfiguration für GPIO und Variable.

```

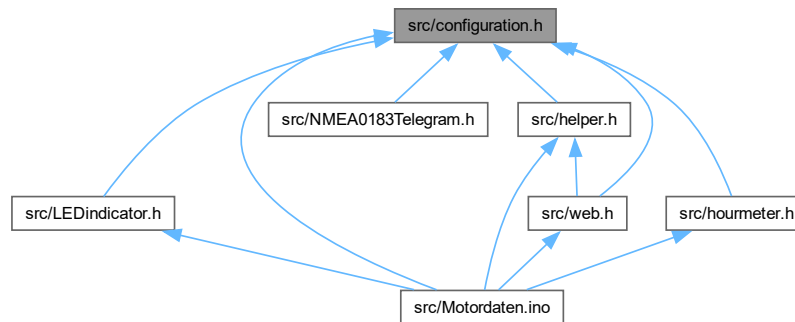
#include <Arduino.h>
#include <Preferences.h>

```

Include-Abhängigkeitsdiagramm für configuration.h:



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



## Klassen

- struct [Web\\_Config](#)

## Makrodefinitionen

- #define [Version](#) "V2.7 vom 20.03.2025"
- #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 [RPMsSendOffset](#) 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](#) 14
- #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
- String `sHeapspace` = ""
- `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
- double `ADC_Calibration_Value1` = 170.0
- double `ADC_Calibration_Value2` = 19.0
- float `fbmp_temperature` = 0
- float `fbmp_pressure` = 0
- float `fbmp_altitude` = 0
- String `sl2C_Status` = ""
- bool `bl2C_Status` = 0
- const int `iMaxSonar` = 35
- int `iDistance` = 0
- float `FuelLevel` = 0
- float `FuelLevelMax` = 30
- float `CoolantTemp` = 0
- float `MotorTemp` = 0
- float `EngineRPM` = 0
- float `BordSpannung` = 0
- bool `EngineOn` = false
- String `motorErrorReported` = "Aus"
- String `coolantErrorReported` = "Aus"
- 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 `fCoolantTemp` = 0
- float `fMotorTemp` = 0
- float `fCoolantOffset` = 0
- float `fMotorOffset` = 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 "V2.7 vom 20.03.2025"
```

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 61 der Datei [configuration.h](#).

### 7.22.2.14 CL\_SSID

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

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

### 7.22.2.15 CL\_PASSWORD

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

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

### 7.22.2.16 I2C\_SDA

```
#define I2C_SDA 21
```

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

### 7.22.2.17 I2C\_SCL

```
#define I2C_SCL 22
```

Definiert in Zeile 90 der Datei [configuration.h](#).

### 7.22.2.18 SEALEVELPRESSURE\_HPA

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

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

#### 7.22.2.19 RPM\_Calibration\_Value

```
#define RPM_Calibration_Value 4.0
```

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

#### 7.22.2.20 Eengine\_RPM\_Pin

```
#define Eengine_RPM_Pin 19
```

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

#### 7.22.2.21 ONE\_WIRE\_BUS

```
#define ONE_WIRE_BUS 14
```

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

#### 7.22.2.22 SERVER\_HOST\_NAME

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

Definiert in Zeile 147 der Datei [configuration.h](#).

#### 7.22.2.23 TCP\_PORT

```
#define TCP_PORT 6666
```

Definiert in Zeile 148 der Datei [configuration.h](#).

#### 7.22.2.24 DNS\_PORT

```
#define DNS_PORT 53
```

Definiert in Zeile 149 der Datei [configuration.h](#).

### 7.22.3 Dokumentation der Aufzählungstypen

#### 7.22.3.1 EngineStatus

```
enum EngineStatus
```

Aufzählungswerte

Off	
On	

Definiert in Zeile 113 der Datei [configuration.h](#).

```
00113 { 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 sHeapspace

```
String sHeapspace = ""
```

Definiert in Zeile 44 der Datei [configuration.h](#).

### 7.22.4.7 tAP\_Config

```
Web_Config tAP_Config
```

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

#### 7.22.4.8 channel

```
const int channel = 10
```

Definiert in Zeile [62](#) der Datei [configuration.h](#).

#### 7.22.4.9 hide\_SSID

```
const bool hide_SSID = false
```

Definiert in Zeile [63](#) der Datei [configuration.h](#).

#### 7.22.4.10 max\_connection

```
const int max_connection = 2
```

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

#### 7.22.4.11 IP

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

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

#### 7.22.4.12 Gateway

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

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

#### 7.22.4.13 NMask

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

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

#### 7.22.4.14 AP\_SSID

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

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

#### 7.22.4.15 AP\_PASSWORD

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

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

**7.22.4.16 AP\_IP**

```
IPAddress AP_IP
```

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

**7.22.4.17 CL\_IP**

```
IPAddress CL_IP
```

Definiert in Zeile [73](#) der Datei [configuration.h](#).

**7.22.4.18 SELF\_IP**

```
IPAddress SELF_IP
```

Definiert in Zeile [74](#) der Datei [configuration.h](#).

**7.22.4.19 sAP\_Station**

```
String sAP_Station = ""
```

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

**7.22.4.20 iSTA\_on**

```
int iSTA_on = 0
```

Definiert in Zeile [80](#) der Datei [configuration.h](#).

**7.22.4.21 bConnect\_CL**

```
int bConnect_CL = 0
```

Definiert in Zeile [81](#) der Datei [configuration.h](#).

**7.22.4.22 bClientConnected**

```
bool bClientConnected = 0
```

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

#### 7.22.4.23 ADC\_Calibration\_Value1

```
double ADC_Calibration_Value1 = 170.0
```

For resistor measure 5 Volt and 180 Ohm equals 100% plus 1K resistor. Old Value 250.0

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

#### 7.22.4.24 ADC\_Calibration\_Value2

```
double 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 86 der Datei [configuration.h](#).

#### 7.22.4.25 fbmp\_temperature

```
float fbmp_temperature = 0
```

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

#### 7.22.4.26 fbmp\_pressure

```
float fbmp_pressure = 0
```

Definiert in Zeile 93 der Datei [configuration.h](#).

#### 7.22.4.27 fbmp\_altitude

```
float fbmp_altitude = 0
```

Definiert in Zeile 94 der Datei [configuration.h](#).

#### 7.22.4.28 sI2C\_Status

```
String sI2C_Status = ""
```

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

#### 7.22.4.29 bI2C\_Status

```
bool bI2C_Status = 0
```

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

**7.22.4.30 iMaxSonar**

```
const int iMaxSonar = 35
```

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

**7.22.4.31 iDistance**

```
int iDistance = 0
```

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

**7.22.4.32 FuelLevel**

```
float FuelLevel = 0
```

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

**7.22.4.33 FuelLevelMax**

```
float FuelLevelMax = 30
```

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

**7.22.4.34 CoolantTemp**

```
float CoolantTemp = 0
```

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

**7.22.4.35 MotorTemp**

```
float MotorTemp = 0
```

Definiert in Zeile 106 der Datei [configuration.h](#).

**7.22.4.36 EngineRPM**

```
float EngineRPM = 0
```

Definiert in Zeile 107 der Datei [configuration.h](#).

**7.22.4.37 BordSpannung**

```
float BordSpannung = 0
```

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

#### 7.22.4.38 EngineOn

```
bool EngineOn = false
```

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

#### 7.22.4.39 motorErrorReported

```
String motorErrorReported = "Aus"
```

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

#### 7.22.4.40 coolantErrorReported

```
String coolantErrorReported = "Aus"
```

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

#### 7.22.4.41 Counter

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

Definiert in Zeile 112 der Datei [configuration.h](#).

#### 7.22.4.42 Bat1Capacity

```
int Bat1Capacity = 55
```

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

#### 7.22.4.43 Bat2Capacity

```
int Bat2Capacity = 90
```

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

#### 7.22.4.44 SoCError

```
int SoCError = 0
```

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

#### 7.22.4.45 BatSoC

```
float BatSoC = 0
```

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



#### 7.22.4.46 sOneWire\_Status

```
String sOneWire_Status = ""
```

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

#### 7.22.4.47 fDrehzahl

```
float fDrehzahl = 0
```

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

#### 7.22.4.48 fGaugeDrehzahl

```
float fGaugeDrehzahl = 0
```

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

#### 7.22.4.49 fBordSpannung

```
float fBordSpannung = 0
```

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

#### 7.22.4.50 fCoolantTemp

```
float fCoolantTemp = 0
```

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

#### 7.22.4.51 fMotorTemp

```
float fMotorTemp = 0
```

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

#### 7.22.4.52 fCoolantOffset

```
float fCoolantOffset = 0
```

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

#### 7.22.4.53 fMotorOffset

```
float fMotorOffset = 0
```

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

#### 7.22.4.54 sSTBB

```
String sSTBB = ""
```

Definiert in Zeile 135 der Datei [configuration.h](#).

#### 7.22.4.55 sOrient

```
String sOrient = ""
```

Definiert in Zeile 136 der Datei [configuration.h](#).

#### 7.22.4.56 dMWV\_WindDirectionT

```
double dMWV_WindDirectionT = 0
```

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

#### 7.22.4.57 dMWV\_WindSpeedM

```
double dMWV_WindSpeedM = 0
```

Definiert in Zeile 140 der Datei [configuration.h](#).

#### 7.22.4.58 dVWR\_WindDirectionM

```
double dVWR_WindDirectionM = 0
```

Definiert in Zeile 141 der Datei [configuration.h](#).

#### 7.22.4.59 dVWR\_WindAngle

```
double dVWR_WindAngle = 0
```

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

#### 7.22.4.60 dVWR\_WindSpeedkn

```
double dVWR_WindSpeedkn = 0
```

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

#### 7.22.4.61 dVWR\_WindSpeedms

```
double dVWR_WindSpeedms = 0
```

Definiert in Zeile 144 der Datei [configuration.h](#).

## 7.22.4.62 udpAddress

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

Definiert in Zeile 152 der Datei [configuration.h](#).

## 7.22.4.63 udpPort

```
const int udpPort = 4444
```

Definiert in Zeile 153 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 "V2.7 vom 20.03.2025" // 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 String sHeapSpace = "";
00045
00046 //Configuration mit Webinterface
00047 struct Web_Config
00048 {
00049     char wAP_IP[20];
00050     char wAP_SSID[64];
00051     char wAP_Password[12];
00052     char wMotor_Offset[6];
00053     char wCoolant_Offset[6];
00054     char wFuellstandmax[6];
00055     char wADC1_Cal[6];
00056     char wADC2_Cal[6];
00057 };
00058 Web_Config tAP_Config;
00059
00060 //Configuration AP
00061 #define HostName "Motordaten"
00062 const int channel = 10; // WiFi Channel number between 1 and 13
00063 const bool hide_SSID = false; // To disable SSID broadcast -> SSID will not appear
// in a basic WiFi scan
00064 const int max_connection = 2; // Maximum simultaneous connected clients on the AP
00065
00066 // Variables for WIFI-AP
00067 IPAddress IP = IPAddress(192, 168, 15, 30);
00068 IPAddress Gateway = IPAddress(192, 168, 15, 30);
00069 IPAddress NMask = IPAddress(255, 255, 255, 0);
00070 const char* AP_SSID = "Motordaten";
```

```

00071 const char* AP_PASSWORD = "12345678";
00072 IPAddress AP_IP;
00073 IPAddress CL_IP;
00074 IPAddress SELF_IP;
00075 String sAP_Station = "";
00076
00077 //Configuration Client (Network Data Windsensor)
00078 #define CL_SSID "NoWa" //Windmesser
00079 #define CL_PASSWORD "12345678"
00080 int iSTA_on = 0; // Status STA-Mode
00081 int bConnect_CL = 0;
00082 bool bClientConnected = 0;
00083
00084 // Calibration data variable definition for ADC1 and ADC2 Input
00085 double ADC_Calibration_Value1 = 170.0;
00086 double ADC_Calibration_Value2 = 19.0;
00087
00088 //Confuration Sensors I2C
00089 #define I2C_SDA 21 //Standard 21
00090 #define I2C_SCL 22 //Standard 22
00091 #define SEALEVELPRESSURE_HPA (1013.25) //1013.25
00092 float fbmp_temperature = 0;
00093 float fbmp_pressure = 0;
00094 float fbmp_altitude = 0;
00095 String sI2C_Status = "";
00096 bool bI2C_Status = 0;
00097
00098 // Global Data Sonar
00099 const int iMaxSonar = 35; //Analoginput
00100 int iDistance = 0;
00101
00102 // Global Data Motordata Sensor
00103 float FuelLevel = 0;
00104 float FuelLevelMax = 30;
00105 float CoolantTemp = 0;
00106 float MotorTemp = 0;
00107 float EngineRPM = 0;
00108 float BordSpannung = 0;
00109 bool EngineOn = false;
00110 String motorErrorReported = "Aus";
00111 String coolantErrorReported = "Aus";
00112 static unsigned long Counter; // Enginehours
00113 enum EngineStatus { Off = 0, On = 1, };
00114 #define RPM_Calibration_Value 4.0 // Translates Generator RPM to Engine RPM
00115 #define Eingine_RPM_Pin 19 // Engine RPM is measured as interrupt on GPIO 23
00116
00117 // Global Data Battery
00118 int Bat1Capacity = 55; // Starterbatterie
00119 int Bat2Capacity = 90; // Versorgerbatterie
00120 int SoCError = 0;
00121 float BatSoC = 0;
00122
00123 // Data wire for teperature (Dallas DS18B20)
00124 #define ONE_WIRE_BUS 14 // Data wire for teperature (Dallas DS18B20) is plugged into GPIO 13
00125 String sOneWire_Status = "";
00126
00127 // Variables Website
00128 float fDrehzahl = 0;
00129 float fGaugeDrehzahl = 0;
00130 float fBordSpannung = 0;
00131 float fCoolantTemp = 0;
00132 float fMotorTemp = 0;
00133 float fCoolantOffset = 0;
00134 float fMotorOffset = 0;
00135 String sSTBB = "";
00136 String sOrient = "";
00137
00138 //Definiton NMEA0183 MWV
00139 double dMWV_WindDirectionT = 0;
00140 double dMWV_WindSpeedM = 0;
00141 double dVWR_WindDirectionM = 0;
00142 double dVWR_WindAngle = 0;
00143 double dVWR_WindSpeedkn = 0;
00144 double dVWR_WindSpeedms = 0;
00145
00146 //Configuration NMEA0183
00147 #define SERVER_HOST_NAME "192.168.30.15" // "192.168.76.34"
00148 #define TCP_PORT 6666 //6666
00149 #define DNS_PORT 53
00150
00151 //Variable NMEA 0183 Stream
00152 const char *udpAddress = "192.168.30.255"; // Set network address for broadcast
00153 const int udpPort = 4444; // UDP port
00154
00155 #endif

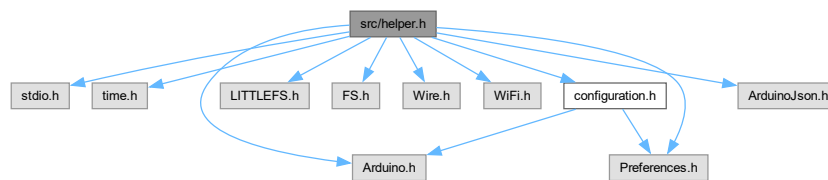
```

## 7.24 src/helper.h-Dateireferenz

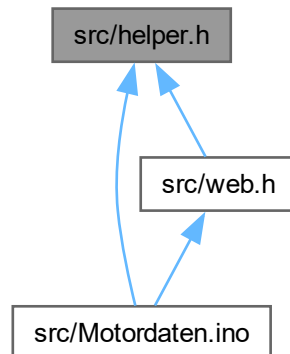
Hilfsfunktionen.

```
#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.*
- bool `writeConfig` (const String &name, const String &value)  
*Webseiten Eingabe in Json-Datei schreiben.*
- void `I2C_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             sHeapSpace = ESP.getFreeHeap();
00042             Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00043         }
00044     }
```

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 47 der Datei [helper.h](#).

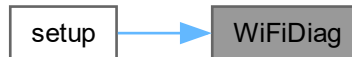
```
00047     {
00048         Serial.println("\nWifi-Diag:");
00049         AP_IP = WiFi.softAPIP();
00050         CL_IP = WiFi.localIP();
00051         Serial.print("AP IP address: ");
00052         Serial.println(AP_IP.toString());
00053         Serial.print("Client IP address: ");
00054         Serial.println(CL_IP.toString());
00055         WiFi.printDiag(Serial);
00056         Serial.print("\nScan AP's ");
00057         {
00058             // WiFi.scanNetworks will return the number of networks found
00059             int n = WiFi.scanNetworks();
00060             Serial.println("scan done");
00061             if (n == 0) {
00062                 Serial.println("no networks found");
00063             } else {
00064                 Serial.print(n);
00065                 Serial.println(" networks found");
00066                 for (int i = 0; i < n; ++i)
00067                 {
00068                     // Print SSID and RSSI for each network found
00069                     Serial.print(i + 1);
00070                     Serial.print(": ");
00071                     Serial.print(WiFi.SSID(i));
00072                     Serial.print(" (");
00073                     Serial.print(WiFi.RSSI(i));
00074                     Serial.print(")");
00075                     Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? " " : "*");
00076                     delay(10);
00077                 }
00078             }
00079         }
00080     }
```

```

00078     }
00079   }
00080 }

```

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 92 der Datei [helper.h](#).

```

00092                                     {
00093     Serial.printf("Listing directory: %s\r\n", dirname);
00094
00095     File root = fs.open(dirname);
00096     if(!root){
00097         Serial.println("- failed to open directory");
00098         return;
00099     }
00100     if(!root.isDirectory()){
00101         Serial.println(" - not a directory");
00102         return;
00103     }
00104
00105     File file = root.openNextFile();
00106     while(file){
00107         if(file.isDirectory()){
00108             Serial.print("  DIR : ");
00109             Serial.println(file.name());
00110             if(levels){
00111                 listDir(fs, file.path(), levels -1);
00112             }
00113         } else {
00114             Serial.print("  FILE: ");
00115             Serial.print(file.name());
00116             Serial.print("\tSIZE: ");
00117             Serial.println(file.size());
00118         }
00119         file = root.openNextFile();
00120     }
00121 }

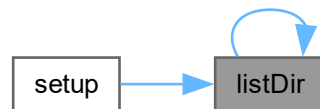
```



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 129 der Datei [helper.h](#).

```

00129     {
00130     JsonDocument testDocument;
00131     File configFile = LittleFS.open(filename);
00132     if (configFile)
00133     {
00134         Serial.println("opened config file");
00135         DeserializationError error = deserializeJson(testDocument, configFile);
00136
00137         // Test if parsing succeeds.
00138         if (error)
00139         {
00140             Serial.print(F("deserializeJson() failed: "));
00141             Serial.println(error.f_str());
00142             return;
00143         }
00144
00145         Serial.println("deserializeJson ok");
00146         {
00147             Serial.println("Lese Daten aus Config - Datei");
00148             strcpy(tAP_Config.wAP_SSID, testDocument["SSID"] | "Motordaten");
  
```

```

00149         strcpy(tAP_Config.wAP_IP, testDocument["IP"] | "192.168.15.30");
00150         strcpy(tAP_Config.wAP_Password, testDocument["Password"] | "12345678");
00151         strcpy(tAP_Config.wMotor_Offset, testDocument["MotorOffset"] | "0.0");
00152         strcpy(tAP_Config.wCoolant_Offset, testDocument["CoolantOffset"] | "0.0");
00153         strcpy(tAP_Config.wFuelstandmax, testDocument["Fuelstandmax"] | "0.0");
00154         strcpy(tAP_Config.wADC1_Cal, testDocument["ADC1_Cal"] | "0.0");
00155         strcpy(tAP_Config.wADC2_Cal, testDocument["ADC2_Cal"] | "0.0");
00156     }
00157     configFile.close();
00158     Serial.println("Config - Datei geschlossen");
00159 }
00160
00161 else
00162 {
00163     Serial.println("failed to load json config");
00164 }
00165 }

```

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



#### 7.24.2.6 writeConfig() [1/2]

```

bool writeConfig (
    String json)

```

Webseiten Eingabe in Json-Datei schreiben.

##### Parameter

json	
------	--

##### Rückgabe

true

false

Definiert in Zeile 175 der Datei [helper.h](#).

```

00176 {
00177     Serial.println(json);
00178
00179     Serial.println("neue Konfiguration speichern");
00180
00181     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00182     if (configFile)
00183     {
00184         Serial.println("Config - Datei öffnen");
00185         File configFile = LittleFS.open("/config.json", FILE_WRITE);
00186         if (configFile)
00187         {
00188             Serial.println("Config - Datei zum Schreiben geöffnet");
00189             JsonDocument testDocument;
00190             Serial.println("JSON - Daten übergeben");
00191             DeserializationError error = deserializeJson(testDocument, json);
00192             // Test if parsing succeeds.

```

```

00193         if (error)
00194         {
00195             Serial.print(F("deserializeJson() failed: "));
00196             Serial.println(error.f_str());
00197             // bei Memory - Fehler den <Wert> in StaticJsonDocument<200> testDocument; erhöhen
00198             return false;
00199         }
00200         Serial.println("Konfiguration schreiben...");
00201         serializeJson(testDocument, configFile);
00202         Serial.println("Konfiguration geschrieben...");
00203
00204         // neue Config in Serial ausgeben zur Kontrolle
00205         serializeJsonPretty(testDocument, Serial);
00206
00207         Serial.println("Config - Datei geschlossen");
00208         configFile.close();
00209     }
00210 }
00211 return true;
00212 }

```

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



### 7.24.2.7 writeConfig() [2/2]

```

bool writeConfig (
    const String & name,
    const String & value)

```

Webseiten Eingabe in Json-Datei schreiben.

#### Parameter

<i>name</i>	
<i>value</i>	

#### Rückgabe

true

false

Definiert in Zeile 223 der Datei [helper.h](#).

```

00224 {
00225     Serial.println("neue Konfiguration speichern");
00226
00227     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00228     if (configFile)
00229     {
00230         Serial.println("Config - Datei öffnen");
00231         JsonDocument testDocument;
00232         DeserializationError error = deserializeJson(testDocument, configFile);
00233         if (error)
00234         {

```

```

00235         Serial.print(F("deserializeJson() failed: "));
00236         Serial.println(error.f_str());
00237         return false;
00238     }
00239
00240     // Update the configuration with the new value
00241     testDocument[name] = value;
00242
00243     // Write the updated configuration back to the file
00244     configFile.close();
00245     configFile = LittleFS.open("/config.json", FILE_WRITE);
00246     if (!configFile)
00247     {
00248         Serial.println("failed to open config file for writing");
00249         return false;
00250     }
00251
00252     serializeJson(testDocument, configFile);
00253     Serial.println("Konfiguration geschrieben...");
00254
00255     // neue Config in Serial ausgeben zur Kontrolle
00256     serializeJsonPretty(testDocument, Serial);
00257
00258     Serial.println("Config - Datei geschlossen");
00259     configFile.close();
00260 }
00261 else
00262 {
00263     Serial.println("failed to open config file");
00264     return false;
00265 }
00266 return true;
00267 }

```

### 7.24.2.8 I2C\_scan()

```

void I2C_scan (
    void )

```

I2C Bus auslesen, alle Geräte mit Adresse ausgegeben

Definiert in Zeile 272 der Datei [helper.h](#).

```

00272     {
00273         byte error, address;
00274         int nDevices;
00275         Serial.println("Scanning...");
00276         nDevices = 0;
00277         for(address = 1; address < 127; address++)
00278         {
00279             Wire.beginTransmission(address);
00280             error = Wire.endTransmission();
00281             if (error == 0)
00282             {
00283                 Serial.print("I2C device found at address 0x");
00284                 if (address < 16)
00285                 {
00286                     Serial.print("0");
00287                 }
00288                 Serial.println(address, HEX);
00289                 nDevices++;
00290             }
00291             else if (error == 4)
00292             {
00293                 Serial.print("Unknow error at address 0x");
00294                 if (address < 16)
00295                 {
00296                     Serial.print("0");
00297                 }
00298                 Serial.println(address, HEX);
00299                 nDevices++;
00300             }
00301             else if (error == 4) {
00302                 Serial.print("Unknow error at address 0x");
00303                 if (address < 16) {
00304                     Serial.print("0");
00305                 }
00306                 Serial.println(address, HEX);
00307             }
00308         }
00309         if (nDevices == 0) {

```

```

00310     Serial.println("No I2C devices found\n");
00311   }
00312   else {
00313     Serial.println("done\n");
00314   }
00315 }

```

### 7.24.2.9 sWifiStatus()

```

String sWifiStatus (
    int Status)

```

WIFI Status lesen.

Parameter

<i>Status</i>	
---------------	--

Rückgabe

String

Definiert in Zeile [324](#) der Datei [helper.h](#).

```

00325 {
00326   switch(Status){
00327     case WL_IDLE_STATUS: return "Warten";
00328     case WL_NO_SSID_AVAIL: return "Keine SSID vorhanden";
00329     case WL_SCAN_COMPLETED: return "Scan komplett";
00330     case WL_CONNECTED: return "Verbunden";
00331     case WL_CONNECT_FAILED: return "Verbindung fehlerhaft";
00332     case WL_CONNECTION_LOST: return "Verbindung verloren";
00333     case WL_DISCONNECTED: return "Nicht verbunden";
00334     default: return "unbekannt";
00335   }
00336 }

```

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



### 7.24.2.10 toChar()

```

char * toChar (
    String command)

```

Convert string to char.

## Parameter

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

## Rückgabe

char\*

Definiert in Zeile 345 der Datei [helper.h](#).

```
00345 {
00346     if(command.length() != 0) {
00347         char *p = const_cast<char*>(command.c_str());
00348         return p;
00349     }
00350     else{
00351         return 0;
00352     }
00353 }
```

## 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         sHeapSpace = ESP.getFreeHeap();
00042         Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00043     }
00044 }
00045
00047 void WiFiDiag(void) {
00048     Serial.println("\nWifi-Diag:");
00049     AP_IP = WiFi.softAPIP();
00050     CL_IP = WiFi.localIP();
00051     Serial.print("AP IP address: ");
00052     Serial.println(AP_IP.toString());
00053     Serial.print("Client IP address: ");
00054     Serial.println(CL_IP.toString());
00055     WiFi.printDiag(Serial);
00056     Serial.print("\nScan AP's ");
00057     {
00058         // WiFi.scanNetworks will return the number of networks found
00059         int n = WiFi.scanNetworks();
00060         Serial.println("scan done");
00061         if (n == 0) {
00062             Serial.println("no networks found");
00063         } else {
```

```

00064     Serial.print(n);
00065     Serial.println(" networks found");
00066     for (int i = 0; i < n; ++i)
00067     {
00068         // Print SSID and RSSI for each network found
00069         Serial.print(i + 1);
00070         Serial.print(": ");
00071         Serial.print(WiFi.SSID(i));
00072         Serial.print(" ");
00073         Serial.print(WiFi.RSSI(i));
00074         Serial.print(" ");
00075         Serial.println(WiFi.encryptionType(i) == WIFI_AUTH_OPEN ? " ":"*");
00076         delay(10);
00077     }
00078 }
00079 }
00080 }
00081
00082 /***** Filesystem *****/
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.wMotor_Offset, testDocument["MotorOffset"] | "0.0");
00151             strcpy(tAP_Config.wCoolant_Offset, testDocument["CoolantOffset"] | "0.0");
00152             strcpy(tAP_Config.wFuellstandmax, testDocument["Fuellstandmax"] | "0.0");
00153             strcpy(tAP_Config.wADC1_Cal, testDocument["ADC1_Cal"] | "0.0");
00154             strcpy(tAP_Config.wADC2_Cal, testDocument["ADC2_Cal"] | "0.0");
00155         }
00156         configFile.close();
00157         Serial.println("Config - Datei geschlossen");
00158     }
00159 }
00160
00161 else
00162 {

```

```

00163         Serial.println("failed to load json config");
00164     }
00165 }
00166
00174
00175 bool writeConfig(String json)
00176 {
00177     Serial.println(json);
00178
00179     Serial.println("neue Konfiguration speichern");
00180
00181     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00182     if (configFile)
00183     {
00184         Serial.println("Config - Datei öffnen");
00185         File configFile = LittleFS.open("/config.json", FILE_WRITE);
00186         if (configFile)
00187         {
00188             Serial.println("Config - Datei zum Schreiben geöffnet");
00189             JsonDocument testDocument;
00190             Serial.println("JSON - Daten übergeben");
00191             DeserializationError error = deserializeJson(testDocument, json);
00192             // Test if parsing succeeds.
00193             if (error)
00194             {
00195                 Serial.print(F("deserializeJson() failed: "));
00196                 Serial.println(error.f_str());
00197                 // bei Memory - Fehler den <Wert> in StaticJsonDocument<200> testDocument; erhöhen
00198                 return false;
00199             }
00200             Serial.println("Konfiguration schreiben...");
00201             serializeJson(testDocument, configFile);
00202             Serial.println("Konfiguration geschrieben...");
00203
00204             // neue Config in Serial ausgeben zur Kontrolle
00205             serializeJsonPretty(testDocument, Serial);
00206
00207             Serial.println("Config - Datei geschlossen");
00208             configFile.close();
00209         }
00210     }
00211     return true;
00212 }
00213
00222
00223 bool writeConfig(const String& name, const String& value)
00224 {
00225     Serial.println("neue Konfiguration speichern");
00226
00227     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00228     if (configFile)
00229     {
00230         Serial.println("Config - Datei öffnen");
00231         JsonDocument testDocument;
00232         DeserializationError error = deserializeJson(testDocument, configFile);
00233         if (error)
00234         {
00235             Serial.print(F("deserializeJson() failed: "));
00236             Serial.println(error.f_str());
00237             return false;
00238         }
00239
00240         // Update the configuration with the new value
00241         testDocument[name] = value;
00242
00243         // Write the updated configuration back to the file
00244         configFile.close();
00245         configFile = LittleFS.open("/config.json", FILE_WRITE);
00246         if (!configFile)
00247         {
00248             Serial.println("failed to open config file for writing");
00249             return false;
00250         }
00251
00252         serializeJson(testDocument, configFile);
00253         Serial.println("Konfiguration geschrieben...");
00254
00255         // neue Config in Serial ausgeben zur Kontrolle
00256         serializeJsonPretty(testDocument, Serial);
00257
00258         Serial.println("Config - Datei geschlossen");
00259         configFile.close();
00260     }
00261     else
00262     {
00263         Serial.println("failed to open config file");
00264         return false;

```



```

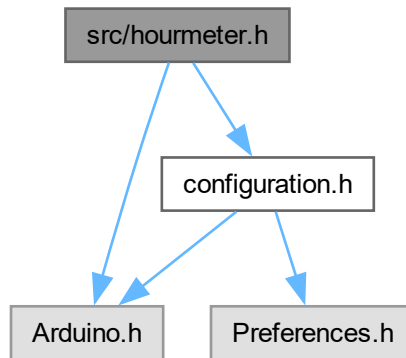
00265     }
00266     return true;
00267 }
00268
00269 /***** I2C Bus *****/
00271
00272 void I2C_scan(void){
00273     byte error, address;
00274     int nDevices;
00275     Serial.println("Scanning...");
00276     nDevices = 0;
00277     for(address = 1; address < 127; address++ )
00278     {
00279         Wire.beginTransmission(address);
00280         error = Wire.endTransmission();
00281         if (error == 0)
00282         {
00283             Serial.print("I2C device found at address 0x");
00284             if (address<16)
00285             {
00286                 Serial.print("0");
00287             }
00288             Serial.println(address,HEX);
00289             nDevices++;
00290         }
00291         else if (error==4)
00292         {
00293             Serial.print("Unknow error at address 0x");
00294             if (address<16)
00295             {
00296                 Serial.print("0");
00297             }
00298             Serial.println(address,HEX);
00299             nDevices++;
00300         }
00301         else if (error==4) {
00302             Serial.print("Unknow error at address 0x");
00303             if (address<16) {
00304                 Serial.print("0");
00305             }
00306             Serial.println(address,HEX);
00307         }
00308     }
00309     if (nDevices == 0) {
00310         Serial.println("No I2C devices found\n");
00311     }
00312     else {
00313         Serial.println("done\n");
00314     }
00315 }
00316
00323
00324 String sWifiStatus(int Status)
00325 {
00326     switch(Status){
00327         case WL_IDLE_STATUS:return "Warten";
00328         case WL_NO_SSID_AVAIL:return "Keine SSID vorhanden";
00329         case WL_SCAN_COMPLETED:return "Scan komplett";
00330         case WL_CONNECTED:return "Verbunden";
00331         case WL_CONNECT_FAILED:return "Verbindung fehlerhaft";
00332         case WL_CONNECTION_LOST:return "Verbindung verloren";
00333         case WL_DISCONNECTED:return "Nicht verbunden";
00334         default:return "unbekannt";
00335     }
00336 }
00337
00344
00345 char* toChar(String command){
00346     if(command.length() !=0){
00347         char *p = const_cast<char*>(command.c_str());
00348         return p;
00349     }
00350     else{
00351         return 0;
00352     }
00353 }
00354
00355
00356 #endif

```

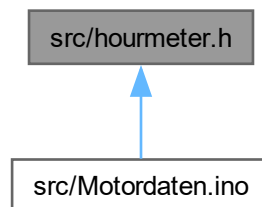
## 7.26 src/hourmeter.h-Dateireferenz

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)  
*Betriebsstundenzähler Berechnet Betriebsstunden, 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)
```

Betriebstundenzähler Berechnet Betriebstunden, wenn Anlage eingeschaltet ist.

### Parameter

<i>CountOn</i>	
----------------	--

### Rückgabe

unsigned long

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

```
00029 {
00030     unsigned long now = millis();
00031     milliRest += now - lastRun;
00032     if (CountOn == 1) {
00033         while (milliRest >= 1000) {
00034             Counter++;
00035             milliRest -= 1000;
00036         }
00037     } else {
00038         milliRest = 0;
00039     }
00040     lastRun = now;
00041
00042     statel = CountOn;
00043     if (laststatel == HIGH && statel == LOW) { // speichern bei Flanke negativ
00044         bsz1.begin("bsz", false); // NVS nutzen, BSZ erstellen, lesen und schreiben (false)
```

```
00045         CounterOld = bsz1.getUInt("Start", 0); // Speicher auslesen
00046         Counter = CounterOld + Counter; // Laufzeit alt + aktuell
00047         bsz1.putUInt("Start", Counter); // Speicher schreiben
00048         bsz1.end(); // Preferences beenden
00049     }
00050     laststatel = statel; // Aktualisiere laststatel
00051     return Counter;
00052 }
```

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 statel = 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](#)

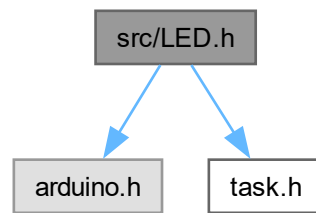
```
00001 #ifndef _HOURMETER_H_
00002 #define _HOURMETER_H_
00003
00014
00015 #include <Arduino.h>
00016 #include "configuration.h"
00017
00018 Preferences bsz1;
00019
00020 static unsigned long lastRun, CounterOld, milliRest;
00021 int statel = LOW, laststate1 = LOW;
00022
00029 unsigned long EngineHours(bool CountOn = 0) {
00030     unsigned long now = millis();
00031     milliRest += now - lastRun;
00032     if (CountOn == 1) {
00033         while (milliRest >= 1000) {
00034             Counter++;
00035             milliRest -= 1000;
00036         }
00037     } else {
00038         milliRest = 0;
00039     }
00040     lastRun = now;
00041
00042     statel = CountOn;
00043     if (laststate1 == HIGH && statel == LOW) { // speichern bei Flanke negativ
00044         bsz1.begin("bsz", false); // NVS nutzen, BSZ erstellen, lesen und schreiben (false)
00045         CounterOld = bsz1.getUInt("Start", 0); // Speicher auslesen
00046         Counter = CounterOld + Counter; // Laufzeit alt + aktuell
00047         bsz1.putUInt("Start", Counter); // Speicher schreiben
00048         bsz1.end(); // Preferences beenden
00049     }
00050     laststate1 = statel; // Aktualisiere laststate1
00051     return Counter;
00052 }
00053
00054 #endif
```

## 7.28 src/LED.h-Dateireferenz

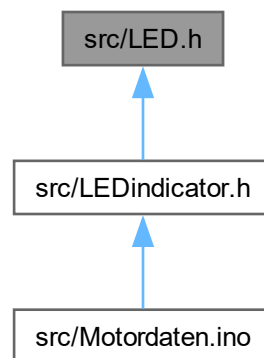
[LED](#) Ansteuerung.

```
#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=`LEDBoard`)
- void `LEDflash` (int PIN=`LEDBoard`)
- void `flashLED` (int PIN=`LEDBoard`)
- void `LEDInit` ()

*Start Initialisierung LEDtest.*

- void `LEDOn` (int PIN=`LEDBoard`)
- void `LEDOff` (int PIN=`LEDBoard`)
- 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

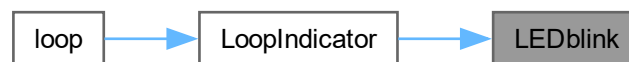
### 7.28.3.1 LEDblink()

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

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

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

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



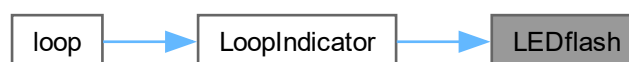
### 7.28.3.2 LEDflash()

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

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

```
00037                                     {  
00038     taskBegin();  
00039     while (1) { // blockiert dank der TaskPause nicht  
00040         digitalWrite(PIN, HIGH); // LED ein  
00041         delay(5);  
00042         //taskPause(2); // gibt Rechenzeit ab  
00043         digitalWrite(PIN, LOW); // LED aus  
00044         taskPause(3000); // gibt Rechenzeit ab  
00045     }  
00046     taskEnd();  
00047 }
```

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





### 7.28.3.3 flashLED()

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

Definiert in Zeile 49 der Datei LED.h.

```
00049 {  
00050     if (millis() % 1000 > 500) {  
00051         digitalWrite(PIN, HIGH);  
00052     } else {  
00053         digitalWrite(PIN, LOW);  
00054     }  
00055 }
```

### 7.28.3.4 LEDInit()

```
void LEDInit ()
```

Start Initialisierung LEDtest.

Definiert in Zeile 61 der Datei LED.h.

```
00061 {  
00062     pinMode(LED::Red, OUTPUT);  
00063     pinMode(LED::Blue, OUTPUT);  
00064     pinMode(LED::Green, OUTPUT);  
00065     digitalWrite(LED::Red, HIGH);  
00066     delay(250);  
00067     digitalWrite(LED::Red, LOW);  
00068     digitalWrite(LED::Blue, HIGH);  
00069     delay(250);  
00070     digitalWrite(LED::Blue, LOW);  
00071     digitalWrite(LED::Green, HIGH);  
00072     delay(250);  
00073     digitalWrite(LED::Green, LOW);  
00074 }
```

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



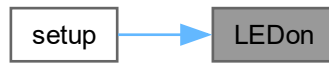
### 7.28.3.5 LEDon()

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

Definiert in Zeile 76 der Datei LED.h.

```
00076 {  
00077     digitalWrite(PIN, HIGH);  
00078 }
```

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



### 7.28.3.6 LEDoff()

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

Definiert in Zeile 80 der Datei [LED.h](#).

```
00080 {  
00081     digitalWrite(PIN, LOW);  
00082 }
```

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



### 7.28.3.7 LEDoff\_RGB()

```
void LEDoff_RGB ()
```

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

```
00084 {  
00085     digitalWrite(LED::Blue, LOW);  
00086     digitalWrite(LED::Green, LOW);  
00087     digitalWrite(LED::Red, LOW);  
00088 }
```

## 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 = LEDBoard) {
00027     taskBegin();
00028     while (1) { // blockiert dank der TaskPause nicht
00029         digitalWrite(PIN, HIGH); // LED ein
00030         taskPause(250); // gibt Rechenzeit ab
00031         digitalWrite(PIN, LOW); // LED aus
00032         taskPause(1000); // gibt Rechenzeit ab
00033     }
00034     taskEnd();
00035 }
00036
00037 void LEDflash(int PIN = LEDBoard) {
00038     taskBegin();
00039     while (1) { // blockiert dank der TaskPause nicht
00040         digitalWrite(PIN, HIGH); // LED ein
00041         delay(5);
00042         //taskPause(2); // gibt Rechenzeit ab
00043         digitalWrite(PIN, LOW); // LED aus
00044         taskPause(3000); // gibt Rechenzeit ab
00045     }
00046     taskEnd();
00047 }
00048
00049 void flashLED(int PIN = LEDBoard) {
00050     if (millis() % 1000 > 500) {
00051         digitalWrite(PIN, HIGH);
00052     } else {
00053         digitalWrite(PIN, LOW);
00054     }
00055 }
00056
00061 void LEDInit() {
00062     pinMode(LED::Red, OUTPUT);
00063     pinMode(LED::Blue, OUTPUT);
00064     pinMode(LED::Green, OUTPUT);
00065     digitalWrite(LED::Red, HIGH);
00066     delay(250);
00067     digitalWrite(LED::Red, LOW);
00068     digitalWrite(LED::Blue, HIGH);
00069     delay(250);
00070     digitalWrite(LED::Blue, LOW);
00071     digitalWrite(LED::Green, HIGH);
00072     delay(250);
00073     digitalWrite(LED::Green, LOW);
00074 }
00075
00076 void LEDon(int PIN = LEDBoard) {
00077     digitalWrite(PIN, HIGH);
00078 }
00079
00080 void LEDoff(int PIN = LEDBoard) {
00081     digitalWrite(PIN, LOW);
00082 }
00083
00084 void LEDoff_RGB() {
00085     digitalWrite(LED::Blue, LOW);
00086     digitalWrite(LED::Green, LOW);
00087     digitalWrite(LED::Red, LOW);
00088 }
00089

```

## 7.30 src/LEDIndicator.h-Dateireferenz

LED Betriebsanzeige.

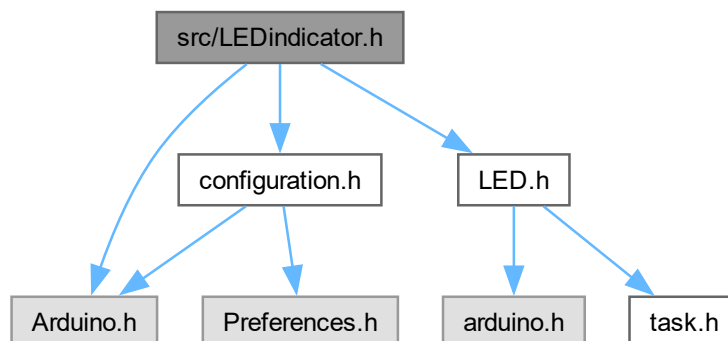
```

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

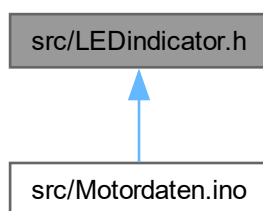
```

```
#include "LED.h"
```

Include-Abhängigkeitsdiagramm für LEDIndicator.h:



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



## Funktionen

- void `LoopIndicator()`

## Variablen

- bool `ErrorOff` = false  
*Sensor failure switch `LED` green/red.*
- bool `ErrorOn` = false

## 7.30.1 Ausführliche Beschreibung

LED Betriebsanzeige.

Autor

Gerry Sebb

Version

1.0

Datum

2025-02-23

Copyright

Copyright (c) 2025

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

## 7.30.2 Dokumentation der Funktionen

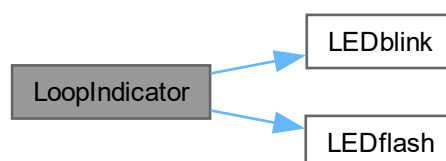
### 7.30.2.1 LoopIndicator()

void LoopIndicator ()

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

```
00027     {
00028     // Reset Error flags
00029     ErrorOff = false;
00030     ErrorOn = false;
00031
00032     if (MotorTemp != -5.0 && CoolantTemp != -5.0) {
00033         ErrorOff = true;
00034     }
00035     if (MotorTemp == -5.0 || CoolantTemp == -5.0) {
00036         ErrorOn = true;
00037     }
00038     if (ErrorOff == true ) {
00039         LEDflash(LED(Green)); // flash for loop run without temp-failure
00040     }
00041     if (ErrorOn == true) {
00042         LEDblink(LED(Red));
00043     }
00044 }
```

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



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



### 7.30.3 Variablen-Dokumentation

#### 7.30.3.1 ErrorOff

```
bool ErrorOff = false
```

Sensor failure switch [LED](#) green/red.

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

#### 7.30.3.2 ErrorOn

```
bool ErrorOn = false
```

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

## 7.31 LEDindicator.h

[gehe zur Dokumentation dieser Datei](#)

```
00001
00011
00012 #ifndef _LEDINDICATOR_
00013 #define _LEDINDICATOR_
00014
00015 # include <Arduino.h>
00016 # include "configuration.h"
00017 # include "LED.h"
00018
00019
00024 bool ErrorOff = false;
00025 bool ErrorOn = false;
00026
00027 void LoopIndicator(){
00028     // Reset Error flags
00029     ErrorOff = false;
00030     ErrorOn = false;
00031
00032     if (MotorTemp != -5.0 && CoolantTemp != -5.0){
00033         ErrorOff = true;
00034     }
00035     if (MotorTemp == -5.0 || CoolantTemp == -5.0){
00036         ErrorOn = true;
00037     }
00038     if (ErrorOff == true){
00039         LEDflash(LED(Green)); // flash for loop run without temp-failure
00040     }
00041     if (ErrorOn == true){
00042         LEDblink(LED(Red));
00043     }
00044 }
00045
00046 #endif
```



*Send PGN127508.*

- void [SendN2kTankLevel](#) (double level, double capacity)

*Send PGN 127505.*

- void [SendN2kEngineData](#) (double Oiltemp, double Coolanttemp, 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

## 7.32.1 Ausführliche Beschreibung

Motordaten NMEA2000.

### Autor

Gerry Sebb

### Version

2.7

### Datum

2025-03-20

### Copyright

Copyright (c) 2025

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



## 7.32.2 Makro-Dokumentation

### 7.32.2.1 ENABLE\_DEBUG\_LOG

```
#define ENABLE_DEBUG_LOG 0
```

Definiert in Zeile 44 der Datei [Motordaten.ino](#).

## 7.32.3 Dokumentation der Funktionen

### 7.32.3.1 oneWire()

```
OneWire oneWire (
    ONE_WIRE_BUS )
```

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

### 7.32.3.2 debug\_log()

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

Definiert in Zeile 88 der Datei [Motordaten.ino](#).

```
00088 {
00089 #if ENABLE_DEBUG_LOG == 1
00090     Serial.println(str);
00091 #endif
00092 }
```

### 7.32.3.3 handleInterrupt()

```
void IRAM_ATTR handleInterrupt ()
```

RPM Event Interrupt Enters on falling edge.

Rückgabe

\* void

Definiert in Zeile 100 der Datei [Motordaten.ino](#).

```
00101 {
00102     portENTER_CRITICAL_ISR(&mux);
00103     uint64_t TempVal = timerRead(timer);           // value of timer at interrupt
00104     PeriodCount = TempVal - StartValue;           // period count between rising edges in 0.000001 of a
        second
00105     StartValue = TempVal;                         // puts latest reading as start for next calculation
00106     portEXIT_CRITICAL_ISR(&mux);
00107     Last_int_time = millis();
00108 }
```

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



### 7.32.3.4 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 111 der Datei [Motordaten.ino](#).

```
00111         {
00112
00113     // Init USB serial port
00114     Serial.begin(115200);
00115
00116     Serial.printf("Motordaten setup %s start\n", Version);
00117
00122     if (!LittleFS.begin(true)) {
00123         Serial.println("An Error has occurred while mounting LittleFS");
00124         return;
00125     }
00126     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00127
00128     File root = LittleFS.open("/");
00129     listDir(LittleFS, "/", 3);
00130
00135     readConfig("/config.json");
00136     IP = inet_addr(tAP_Config.wAP_IP);
00137     AP_SSID = tAP_Config.wAP_SSID;
00138     AP_PASSWORD = tAP_Config.wAP_Password;
00139     fMotorOffset = atof(tAP_Config.wMotor_Offset);
00140     fCoolantOffset = atof(tAP_Config.wCoolant_Offset);
00141     FuelLevelMax = atof(tAP_Config.wFuellstandmax);
00142     ADC_Calibration_Value1 = atof(tAP_Config.wADC1_Cal);
00143     ADC_Calibration_Value2 = atof(tAP_Config.wADC2_Cal);
00144     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + ", Passwort: " + AP_PASSWORD + ", MotorTOffset: " + fMotorOffset + ", CoolantTOffset: " + fCoolantOffset + " read from file");
00145
00146     // LED
00147     LEDInit();
00148
00149     // Boardinfo
00154     sBoardInfo = boardInfo.ShowChipIDtoString();
00155
00156     //Wifi
00157     WiFi.mode(WIFI_AP_STA);
00158     WiFi.softAPdisconnect();
00159     if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00160         WiFi.softAPConfig(IP, Gateway, NMask);
00161         Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00162         Serial.println("\nSet IP " + IP.toString() + ", Gateway: " + Gateway.toString() + ", NetMask: " + NMask.toString() + " ready");
00163         LEDon(LED.Green);
00164         delay(1000);
00165         LEDOff(LED.Green);
00166     } else {
00167         Serial.println("Starting AP failed.");
00168         LEDon(LED.Red);
00169         delay(1000);
00170         ESP.restart();
00171     }
00172
00173     WiFi.setHostname(HostName);
00174     Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00175
00176     delay(1000);
00177     WiFiDiag();
```

```

00178
00179     if (!MDNS.begin(AP_SSID)) {
00180         Serial.println("Error setting up MDNS responder!");
00181         while (1) {
00182             delay(1000);
00183         }
00184     }
00185     Serial.println("mDNS responder started\n");
00186
00187 // Start TCP (HTTP) server
00188     server.begin();
00189     Serial.println("TCP server started\n");
00190
00191 // Add service to MDNS-SD
00192     MDNS.addService("http", "tcp", 80);
00193     MDNS.addService("ws", "tcp", 81);
00194
00195 // Webconfig laden
00196     website();
00197
00202     pinMode(Engine_RPM_Pin, INPUT_PULLUP); // sets pin high
00203     attachInterrupt(digitalPinToInterrupt(Engine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
to interrupt on Falling Edge
00204     timer = timerBegin(0, 80, true); // this returns a
pointer to the hw_timer_t global variable
00205     // 0 = first timer
00206     // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00207     // true - counts up
00208     timerStart(timer); // starts the timer
00209
00214     sensors.begin();
00215     oneWire.reset();
00216     Serial.print("OneWire: Found ");
00217     Serial.print(sensors.getDeviceCount(), DEC);
00218     Serial.println(" devices.");
00219     Serial.print("Parasite power is: ");
00220     if (sensors.isParasitePowerMode()) Serial.println("ON");
00221     else Serial.println("OFF");
00222     sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00223
00224     byte i;
00225     byte present = 0;
00226     byte data[12];
00227     byte addr[8];
00228
00229     Serial.print("Looking for 1-Wire devices...\n\r");
00230     while(oneWire.search(addr)) {
00231         Serial.print("\n\rFound '1-Wire' device with address:\n\r");
00232         for( i = 0; i < 8; i++) {
00233             Serial.print("0x");
00234             if (addr[i] < 16) {
00235                 Serial.print('0');
00236             }
00237             Serial.print(addr[i], HEX);
00238             if (i < 7) {
00239                 Serial.print(", ");
00240             }
00241         }
00242         if (OneWire::crc8( addr, 7) != addr[7]) {
00243             Serial.print("CRC is not valid!\n");
00244             return;
00245         }
00246     }
00247     Serial.print("\n\rNo more sensors!\n\r");
00248     oneWire.reset_search();
00249     delay(250);
00250
00251 // search for devices on the bus and assign based on an index
00252     if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00253     if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00254
00255
00256
00257 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
or Mega
00258     NMEA2000.SetN2kCANMsgBufSize(8);
00259     NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00260     NMEA2000.SetN2kCANSendFrameBufSize(250);
00261
00262     esp_efuse_mac_get_default(chipid);
00263     for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00264
00269     NMEA2000.SetProductInformation("MD01.2501", // Manufacturer's Model serial code
00270                                     100, // Manufacturer's product code
00271                                     "MD Sensor Module", // Manufacturer's Model ID
00272                                     "2.5.1.0 (2025-02-20)", // Manufacturer's Software version code
00273                                     "2.0.0.0 (2024-11-30)" // Manufacturer's Model version

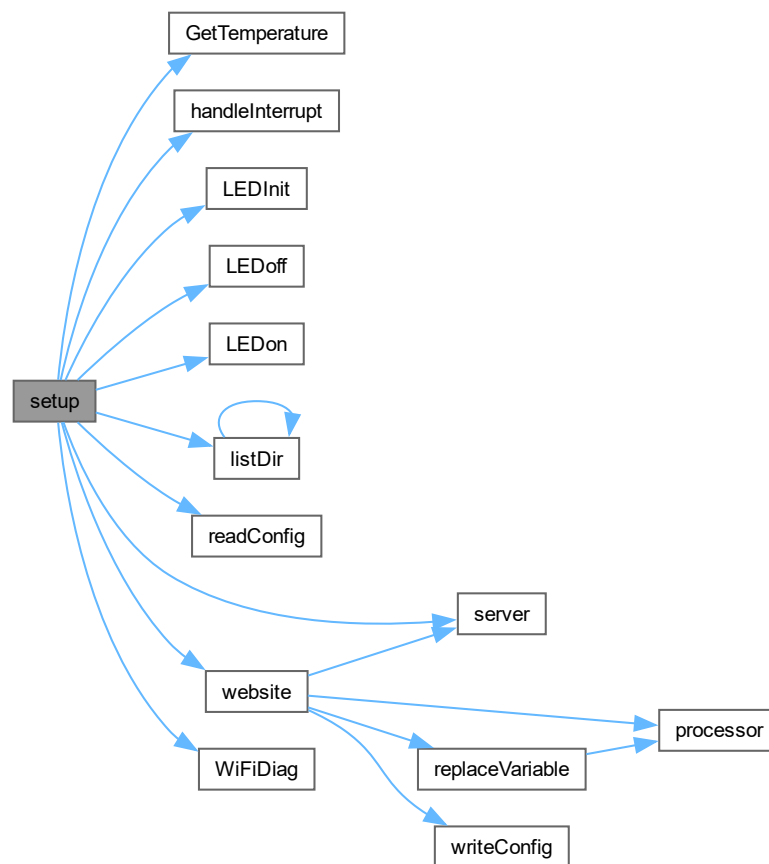
```

```

00274                                     );
00275 // Set device information
00276 NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00277                               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
00278                               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
00279                               2046 // Just choosen free from code list on
                                http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00280                                     );
00281
00282 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
    below
00283
00284 NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
    Actisense format.
00285
00286 preferences.begin("nvs", false); // Open nonvolatile storage (nvs)
00287 NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
    33
00288 preferences.end();
00289 Serial.printf("NodeAddress=%d\n", NodeAddress);
00290
00291 NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00292 NMEA2000.ExtendTransmitMessages(TransmitMessages);
00293 NMEA2000.Open();
00294
00295 xTaskCreatePinnedToCore(
00296     GetTemperature, /* Function to implement the task */
00297     "Task1", /* Name of the task */
00298     10000, /* Stack size in words */
00299     NULL, /* Task input parameter */
00300     0, /* Priority of the task */
00301     &Task1, /* Task handle. */
00302     0); /* Core where the task should run */
00303
00304 delay(200);
00305
00310 ArduinoOTA
00311     .onStart([]() {
00312         String type;
00313         if (ArduinoOTA.getCommand() == U_FLASH)
00314             type = "sketch";
00315         else // U_SPIFFS
00316             type = "filesystem";
00317
00318         // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00319         Serial.println("Start updating " + type);
00320     })
00321     .onEnd([]() {
00322         Serial.println("\nEnd");
00323     })
00324     .onProgress([](unsigned int progress, unsigned int total) {
00325         Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00326     })
00327     .onError([](ota_error_t error) {
00328         Serial.printf("Error[%u]: ", error);
00329         if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00330         else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00331         else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00332         else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00333         else if (error == OTA_END_ERROR) Serial.println("End Failed");
00334     });
00335
00336 ArduinoOTA.begin();
00337
00338 printf("Setup end\n");
00339 }

```

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



### 7.32.3.5 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 With error on Sensor set output to -5°C.

#### Parameter

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

Definiert in Zeile 348 der Datei [Motordaten.ino](#).

```

00348                                     {
00349     float tmp0 = 0;
00350     float tmp1 = 0;
00351     for (;;) {
00352         sensors.requestTemperatures();           // Send the command to get temperatures
00353         vTaskDelay(100);
00354         tmp0 = sensors.getTempC(MotorOil);
00355         if (tmp0 == DEVICE_DISCONNECTED_C) {
```

```

00356         if (motorErrorReported == "Aus") {                                // Nur einmal melden
00357             Serial.print("Error read Motor Temp\n");
00358             motorErrorReported = "Ein";
00359             MotorTemp = -5.0;
00360         } else {
00361             MotorTemp = tmp0 + fMotorOffset;
00362             motorErrorReported = "Aus";                                    // Fehler wurde behoben
00363         }
00364         vTaskDelay(100);
00365         tmp1 = sensors.getTempC(MotorCoolant);
00366         if (tmp1 == DEVICE_DISCONNECTED_C) {
00367             if (coolantErrorReported == "Aus") {                            // Nur einmal melden
00368                 Serial.print("Error read Coolant Temp\n");
00369                 coolantErrorReported = "Ein";
00370                 CoolantTemp = -5.0;
00371             } else {
00372                 CoolantTemp = tmp1 + fCoolantOffset;
00373                 coolantErrorReported = "Aus";                                // Fehler wurde behoben
00374             }
00375             vTaskDelay(100);
00376         }
00377     }

```

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



### 7.32.3.6 ReadRPM()

```
double ReadRPM ()
```

Calculate engine RPM from number of interrupts per time.

**Rückgabe**

double

Definiert in Zeile 384 der Datei [Motordaten.ino](#).

```

00384     {
00385     double RPM = 0;
00386
00387     portENTER_CRITICAL(&mux);
00388     if (PeriodCount != 0) {                                                // 0 means no signals measured
00389         RPM = 1000000.00 / PeriodCount;                                    // PeriodCount in 0.000001 of a second
00390     }
00391     portEXIT_CRITICAL(&mux);
00392     if (millis() > Last_int_time + 200) RPM = 0;                          // No signals RPM=0;
00393     return (RPM);
00394 }

```

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



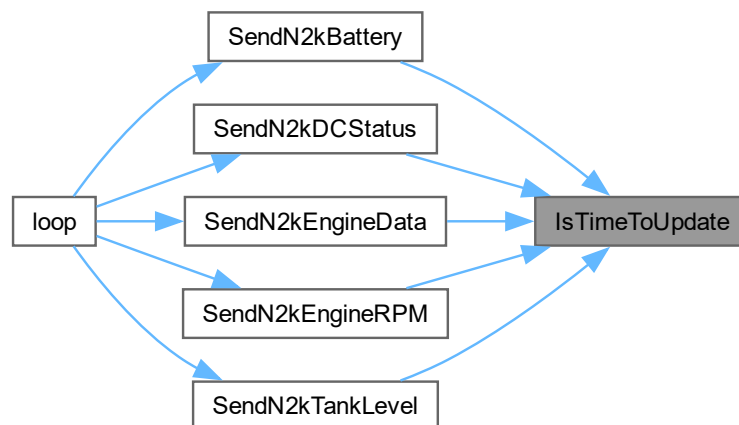
### 7.32.3.7 IsTimeToUpdate()

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

Definiert in Zeile 397 der Datei [Motordaten.ino](#).

```
00397                                     {
00398     return (NextUpdate < millis());
00399 }
```

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



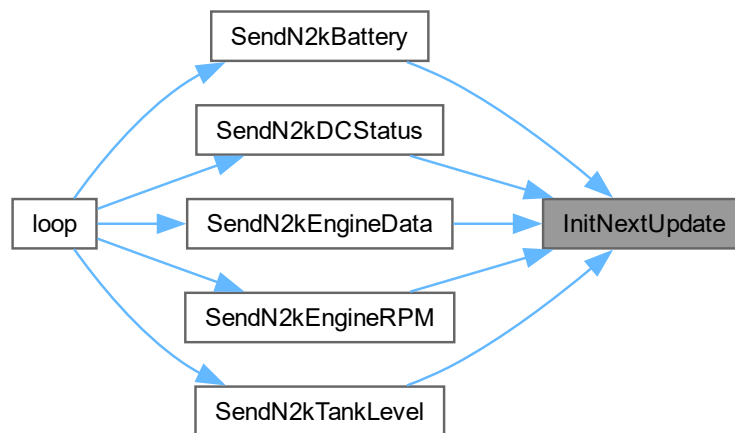
### 7.32.3.8 InitNextUpdate()

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

Definiert in Zeile 400 der Datei [Motordaten.ino](#).

```
00400                                     {
00401     return millis() + Period + Offset;
00402 }
```

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



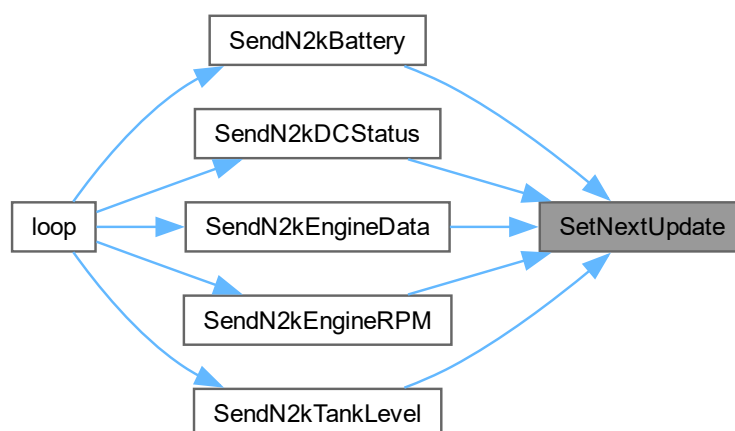
### 7.32.3.9 SetNextUpdate()

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

Definiert in Zeile 404 der Datei [Motordaten.ino](#).

```
00404 {
00405   while ( NextUpdate < millis() ) NextUpdate += Period;
00406 }
```

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





### 7.32.3.10 SendN2kDCStatus()

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

Send PGN127506.

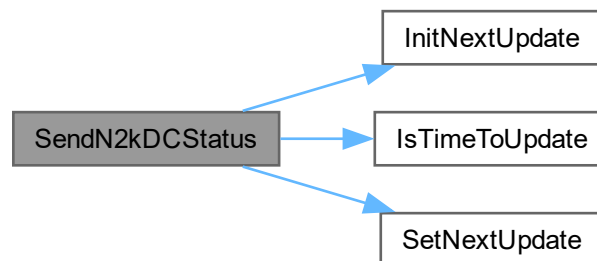
Parameter

<i>BatteryVoltage</i>	
<i>SoC</i>	
<i>BatCapacity</i>	

Definiert in Zeile 416 der Datei [Motordaten.ino](#).

```
00416
00417     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
00418     BatteryDCStatusSendOffset);
00418     tN2kMsg N2kMsg;
00419
00420     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00421         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00422
00423         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00424         Serial.printf("SoC          : %3.1f %\n", SoC);
00425         Serial.printf("Capacity    : %3.1f Ah\n", BatCapacity);
00426         // SetN2kDCStatus(N2kMsg, 1, 1, N2kDct_Battery, 56, 92, 38500, 0.012, AhToCoulomb(420));
00427         SetN2kDCStatus(N2kMsg, 1, 2, N2kDct_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
00428         AhToCoulomb(55));
00428         NMEA2000.SendMessage(N2kMsg);
00429     }
00430 }
```

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



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



### 7.32.3.11 SendN2kBattery()

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

Send PGN127508.

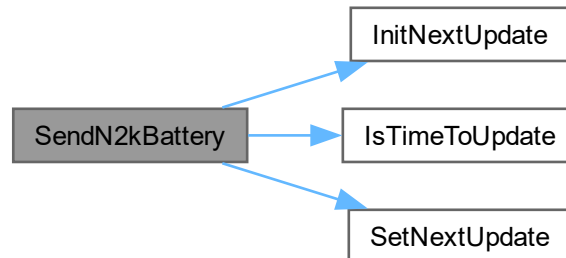
Parameter

<i>BatteryVoltage</i>	
-----------------------	--

Definiert in Zeile [437](#) der Datei [Motordaten.ino](#).

```
00437 {
00438     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00439     tN2kMsg N2kMsg;
00440
00441     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00442         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00443
00444         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00445
00446         SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00447         NMEA2000.SendMessage(N2kMsg);
00448     }
00449 }
```

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



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



## 7.32.3.12 SendN2kTankLevel()

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

Send PGN 127505.

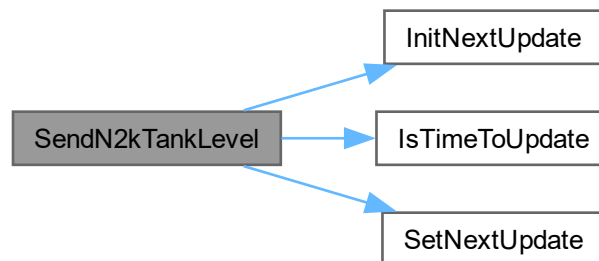
## Parameter

<i>level</i>	
<i>capacity</i>	

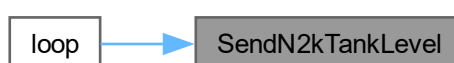
Definiert in Zeile 457 der Datei [Motordaten.ino](#).

```
00457 {
00458     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00459     tN2kMsg N2kMsg;
00460
00461     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00462         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00463
00464         Serial.printf("Fuel Level   : %3.1f %%\n", level);
00465         Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00466
00467         SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00468         NMEA2000.SendMessage(N2kMsg);
00469     }
00470 }
```

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



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



### 7.32.3.13 SendN2kEngineData()

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

Send PGN 127489.

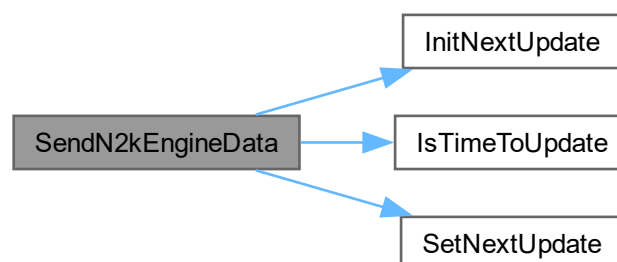
Parameter

<i>Oiltemp</i>	
<i>Coolanttemp</i>	
<i>rpm</i>	
<i>hours</i>	
<i>voltage</i>	

Definiert in Zeile 481 der Datei [Motordaten.ino](#).

```
00481
00482 static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00483 tN2kMsg N2kMsg;
00484 tN2kEngineDiscreteStatus1 Status1;
00485 tN2kEngineDiscreteStatus2 Status2;
00486 Status1.Bits.OverTemperature = Oiltemp > 90; // Alarm Motor over temp
00487 Status1.Bits.LowCoolantLevel = Coolanttemp > 90; // Alarm low cooling
00488 Status1.Bits.LowSystemVoltage = voltage < 11;
00489 Status2.Bits.EngineShuttingDown = rpm < 100; // Alarm Motor off
00490 EngineOn = !Status2.Bits.EngineShuttingDown;
00491
00492 if ( IsTimeToUpdate(SlowDataUpdated) ) {
00493     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00494
00495     Serial.printf("Oil Temp      : %3.1f °C \n", Oiltemp);
00496     Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
00497     Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00498     Serial.printf("Overtemp Oil: %s \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00499     Serial.printf("Overtemp Mot: %s \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00500     Serial.printf("Engine Off  : %s \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00501
00502     // SetN2kTemperatureExt (N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
    // PGN130312, uncomment the PGN to be used
00503
00504     SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Coolanttemp),
        N2kDoubleNA, N2kDoubleNA, hours, N2kDoubleNA, N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00505
00506     NMEA2000.SendMessage(N2kMsg);
00507 }
00508 }
```

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



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



### 7.32.3.14 SendN2kEngineRPM()

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

Send PGN 127488.

Parameter

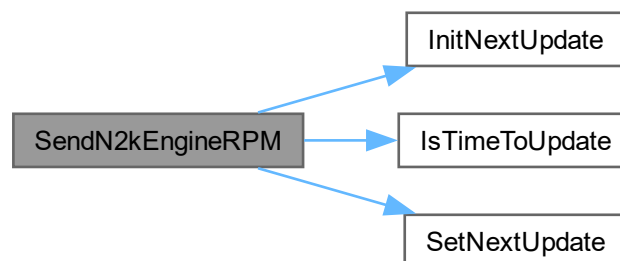
<i>RPM</i>	
------------	--

Definiert in Zeile 515 der Datei [Motordaten.ino](#).

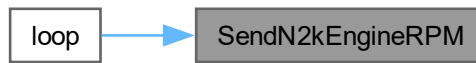
```

00515     {
00516     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMsSendOffset);
00517     tN2kMsg N2kMsg;
00518
00519     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00520         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00521
00522         Serial.printf("Engine RPM : %4.0f RPM \n", RPM);
00523
00524         SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00525
00526         NMEA2000.SendMsg(N2kMsg);
00527     }
00528 }
```

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



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



### 7.32.3.15 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 536 der Datei [Motordaten.ino](#).

```
00536     {  
00537     double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in  
        range 0 to 4095  
00538     if (reading < 1 || reading > 4095) return 0;  
00539     // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +  
        0.000854596860691 * reading + 0.065440348345433;  
00540     return (-0.000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -  
        0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;  
00541 } // Added an improved polynomial, use either, comment out as required
```

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



## 7.32.3.16 loop()

```
void loop ()
```

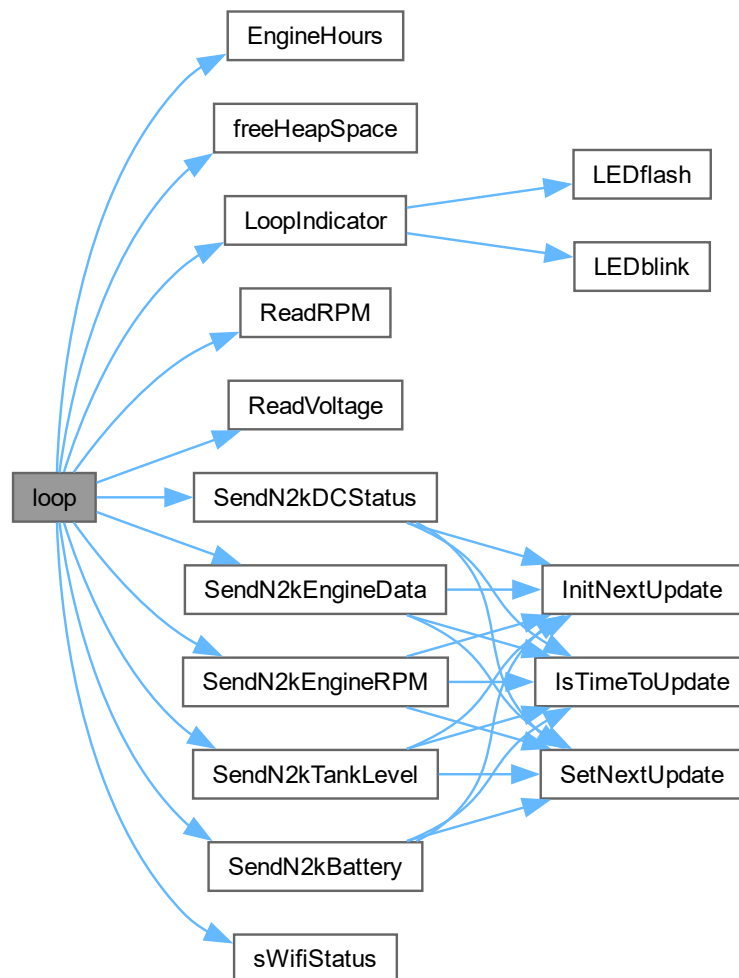
Actual Website Data

Construct a new if object Reboot from Website

Definiert in Zeile 544 der Datei [Motordaten.ino](#).

```
00544     {
00545
00546         LoopIndicator();
00547
00548         BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
// This implements a low pass filter to eliminate spike for ADC readings
00549
00550         FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
This implements a low pass filter to eliminate spike for ADC readings
00551
00552         EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
pass filter to eliminate spike for RPM measurements
00553
00554         BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
Zustand über Spannung
00555         // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00556
00557         EngineHours(EngineOn);
00558
00559         SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00560         SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00561         SendN2kEngineRPM(EngineRPM);
00562         SendN2kBattery(BordSpannung);
00563         SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00564
00565         NMEA2000.ParseMessages();
00566         int SourceAddress = NMEA2000.GetN2kSource();
00567         if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00568             NodeAddress = SourceAddress; // Set new Node Address (to save only once)
00569             preferences.begin("nvs", false);
00570             preferences.putInt("LastNodeAddress", SourceAddress);
00571             preferences.end();
00572             Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00573         }
00574
00575         // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00576         if ( Serial.available() ) {
00577             Serial.read();
00578         }
00579
00580
00581         // OTA
00582         ArduinoOTA.handle();
00583
00584         webSocket.loop();
00585         fCoolantTemp = CoolantTemp;
00586         fMotorTemp = MotorTemp;
00587         fBordSpannung = BordSpannung;
00588         fDrehzahl = EngineRPM;
00589         sCL_Status = sWifiStatus(WiFi.status());
00590         sAP_Station = WiFi.softAPgetStationNum();
00591         freeHeapSpace();
00592
00593
00601         if (IsRebootRequired) {
00602             Serial.println("Rebooting ESP32: ");
00603             delay(1000); // give time for reboot page to load
00604             ESP.restart();
00605         }
00606
00607
00608     }
```

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



## 7.32.4 Variablen-Dokumentation

### 7.32.4.1 PROGMEM

```
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 50 der Datei [Motordaten.ino](#).



```

00050                                     {127488L, // Engine Rapid / RPM
00051                                     127489L, // Engine parameters dynamic
00052                                     127505L, // Fluid Level
00053                                     127506L, // Battery
00054                                     127508L, // Battery Status
00055                                     0
00056                                     };

```

#### 7.32.4.2 StartValue

```
volatile uint64_t StartValue = 0
```

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

Definiert in Zeile 63 der Datei [Motordaten.ino](#).

#### 7.32.4.3 PeriodCount

```
volatile uint64_t PeriodCount = 0
```

period in counts of 0.000001 of a second

Definiert in Zeile 64 der Datei [Motordaten.ino](#).

#### 7.32.4.4 Last\_int\_time

```
unsigned long Last_int_time = 0
```

Definiert in Zeile 65 der Datei [Motordaten.ino](#).

#### 7.32.4.5 timer

```
hw_timer_t* timer = NULL
```

pointer to a variable of type hw\_timer\_t

Definiert in Zeile 66 der Datei [Motordaten.ino](#).

#### 7.32.4.6 mux

```
portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED
```

synchs between maon cose and interrupt?

Definiert in Zeile 67 der Datei [Motordaten.ino](#).

#### 7.32.4.7 oneWire

```
DallasTemperature sensors& oneWire
```

Pass our oneWire reference to Dallas Temperature.

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

#### 7.32.4.8 MotorCoolant

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

DeviceAddress Coolant

Definiert in Zeile 75 der Datei [Motordaten.ino](#).

```
00075 { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
```

#### 7.32.4.9 MotorOil

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

DeviceAddress Engine Oil

Definiert in Zeile 76 der Datei [Motordaten.ino](#).

```
00076 { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
```

#### 7.32.4.10 ADCpin2

```
const int ADCpin2 = 35
```

Voltage measure is connected GPIO 35 (Analog ADC1\_CH7)

Definiert in Zeile 78 der Datei [Motordaten.ino](#).

#### 7.32.4.11 ADCpin1

```
const int ADCpin1 = 34
```

Tank fluid level measure is connected GPIO 34 (Analog ADC1\_CH6)

Definiert in Zeile 79 der Datei [Motordaten.ino](#).

#### 7.32.4.12 Task1

```
TaskHandle_t Task1
```

Task handle for OneWire read (Core 0 on ESP32)

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

#### 7.32.4.13 baudrate

```
const int baudrate = 38400
```

Serial port 2 config (GPIO 16)

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

## 7.32.4.14 rs\_config

```
const int rs_config = SERIAL_8N1
```

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

## 7.33 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 #include <Arduino.h>
00017 #include "configuration.h"
00018 #include <Preferences.h>
00019 #include <ArduinoOTA.h>
00020 #include <OneWire.h>
00021 #include <DallasTemperature.h>
00022 #include <ESP_WiFi.h>
00023 #include <ESPAsyncWebServer.h>
00024 #include <NMEA2000_CAN.h> // This will automatically choose right CAN library and create suitable
00025 NMEA2000 object
00026 #include <N2kMessages.h>
00027 #include <ESPMDNS.h>
00028 #include <arpa/inet.h>
00029 #include "BoardInfo.h"
00030 #include "helper.h"
00031 #include "web.h"
00032 #include "hourmeter.h"
00033 #include "LEDindicator.h"
00034
00035 #define ENABLE_DEBUG_LOG 0 // Debug log
00036
00037
00038 const unsigned long TransmitMessages[] PROGMEM = {127488L, // Engine Rapid / RPM
00039 127489L, // Engine parameters dynamic
00040 127505L, // Fluid Level
00041 127506L, // Battery
00042 127508L, // Battery Status
00043 0
00044 };
00045
00046
00047 volatile uint64_t StartValue = 0;
00048 volatile uint64_t PeriodCount = 0;
00049 unsigned long Last_int_time = 0;
00050 hw_timer_t * timer = NULL;
00051 portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
00052
00053 OneWire oneWire(ONE_WIRE_BUS);
00054 DallasTemperature sensors(&oneWire);
00055 // DeviceAddress MotorThermometer; //**< arrays to hold device addresses
00056 uint8_t MotorCoolant[8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
00057 uint8_t MotorOil[8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
00058
00059 const int ADCpin2 = 35;
00060 const int ADCpin1 = 34;
00061
00062 TaskHandle_t Task1;
00063
00064 const int baudrate = 38400;
00065 const int rs_config = SERIAL_8N1;
00066
00067 void debug_log(char* str) {
00068 #if ENABLE_DEBUG_LOG == 1
```

```

00090 Serial.println(str);
00091 #endif
00092 }
00093
00099 //=====
00100 void IRAM_ATTR handleInterrupt()
00101 {
00102     portENTER_CRITICAL_ISR(&mux);
00103     uint64_t TempVal = timerRead(timer); // value of timer at interrupt
00104     PeriodCount = TempVal - StartValue; // period count between rising edges in 0.000001 of a
    second
00105     StartValue = TempVal; // puts latest reading as start for next calculation
00106     portEXIT_CRITICAL_ISR(&mux);
00107     Last_int_time = millis();
00108 }
00109
00110 /***** Setup
*****
00111 void setup() {
00112
00113     // Init USB serial port
00114     Serial.begin(115200);
00115
00116     Serial.printf("Motordaten setup %s start\n", Version);
00117
00122     if (!LittleFS.begin(true)) {
00123         Serial.println("An Error has occurred while mounting LittleFS");
00124         return;
00125     }
00126     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00127
00128     File root = LittleFS.open("/");
00129     listDir(LittleFS, "/", 3);
00130
00135     readConfig("/config.json");
00136     IP = inet_addr(tAP_Config.wAP_IP);
00137     AP_SSID = tAP_Config.wAP_SSID;
00138     AP_PASSWORD = tAP_Config.wAP_Password;
00139     fMotorOffset = atof(tAP_Config.wMotor_Offset);
00140     fCoolantOffset = atof(tAP_Config.wCoolant_Offset);
00141     FuelLevelMax = atof(tAP_Config.wFuelstandmax);
00142     ADC_Calibration_Value1 = atof(tAP_Config.wADC1_Cal);
00143     ADC_Calibration_Value2 = atof(tAP_Config.wADC2_Cal);
00144     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + ", Passwort: "
+ AP_PASSWORD + ", MotorTOffset: " + fMotorOffset + ", CoolantTOffset: " + fCoolantOffset + " read
from file");
00145
00146     // LED
00147     LEDInit();
00148
00149     // Boardinfo
00154     sBoardInfo = boardInfo.ShowChipIDtoString();
00155
00156     //Wifi
00157     WiFi.mode(WIFI_AP_STA);
00158     WiFi.softAPdisconnect();
00159     if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00160         WiFi.softAPConfig(IP, Gateway, NMask);
00161         Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00162         Serial.println("\nSet IP " + IP.toString() + ", Gateway: " + Gateway.toString() + ", NetMask: " +
NMask.toString() + " ready");
00163         LEDon(LED(Green));
00164         delay(1000);
00165         LEDoff(LED(Green));
00166     } else {
00167         Serial.println("Starting AP failed.");
00168         LEDon(LED(Red));
00169         delay(1000);
00170         ESP.restart();
00171     }
00172
00173     WiFi.setHostname(HostName);
00174     Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00175
00176     delay(1000);
00177     WiFiDiag();
00178
00179     if (!MDNS.begin(AP_SSID)) {
00180         Serial.println("Error setting up MDNS responder!");
00181         while (1) {
00182             delay(1000);
00183         }
00184     }
00185     Serial.println("mDNS responder started\n");
00186
00187     // Start TCP (HTTP) server
00188     server.begin();

```

```

00189     Serial.println("TCP server started\n");
00190
00191 // Add service to MDNS-SD
00192     MDNS.addService("http", "tcp", 80);
00193     MDNS.addService("ws", "tcp", 81);
00194
00195 // Webconfig laden
00196     website();
00197
00202     pinMode(Eingine_RPM_Pin, INPUT_PULLUP); // sets pin high
00203     attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
to interrupt on Falling Edge
00204     timer = timerBegin(0, 80, true); // this returns a
pointer to the hw_timer_t global variable
00205     // 0 = first timer
00206     // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00207     // true - counts up
00208     timerStart(timer); // starts the timer
00209
00214     sensors.begin();
00215     oneWire.reset();
00216     Serial.print("OneWire: Found ");
00217     Serial.print(sensors.getDeviceCount(), DEC);
00218     Serial.println(" devices.");
00219     Serial.print("Parasite power is: ");
00220     if (sensors.isParasitePowerMode()) Serial.println("ON");
00221     else Serial.println("OFF");
00222     sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00223
00224     byte i;
00225     byte present = 0;
00226     byte data[12];
00227     byte addr[8];
00228
00229     Serial.print("Looking for 1-Wire devices...\n\r");
00230     while(oneWire.search(addr)) {
00231         Serial.print("\n\rFound '1-Wire' device with address:\n\r");
00232         for( i = 0; i < 8; i++) {
00233             Serial.print("0x");
00234             if (addr[i] < 16) {
00235                 Serial.print('0');
00236             }
00237             Serial.print(addr[i], HEX);
00238             if (i < 7) {
00239                 Serial.print(", ");
00240             }
00241         }
00242         if (OneWire::crc8( addr, 7) != addr[7]) {
00243             Serial.print("CRC is not valid!\n");
00244             return;
00245         }
00246     }
00247     Serial.print("\n\rNo more sensors!\n\r");
00248     oneWire.reset_search();
00249     delay(250);
00250
00251 // search for devices on the bus and assign based on an index
00252     if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00253     if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00254
00255
00256
00257 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
or Mega
00258     NMEA2000.SetN2kCANMsgBufSize(8);
00259     NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00260     NMEA2000.SetN2kCANSendFrameBufSize(250);
00261
00262     esp_efuse_mac_get_default(chipid);
00263     for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00264
00269     NMEA2000.SetProductInformation("MD01.2501", // Manufacturer's Model serial code
100, // Manufacturer's product code
"MD Sensor Module", // Manufacturer's Model ID
"2.5.1.0 (2025-02-20)", // Manufacturer's Software version code
"2.0.0.0 (2024-11-30)" // Manufacturer's Model version
);
00275 // Set device information
00276     NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
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
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
2046 // Just choosen free from code list on
http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
);
00280
00281

```

```

00282 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
00283 below
00284 NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
00285 Actisense format.
00286 preferences.begin("nvs", false); // Open nonvolatile storage (nvs)
00287 NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
00288 33
00288 preferences.end();
00289 Serial.printf("NodeAddress=%d\n", NodeAddress);
00290
00291 NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00292 NMEA2000.ExtendTransmitMessages(TransmitMessages);
00293 NMEA2000.Open();
00294
00295 xTaskCreatePinnedToCore(
00296     GetTemperature, /* Function to implement the task */
00297     "Task1", /* Name of the task */
00298     10000, /* Stack size in words */
00299     NULL, /* Task input parameter */
00300     0, /* Priority of the task */
00301     &Task1, /* Task handle. */
00302     0); /* Core where the task should run */
00303
00304 delay(200);
00305
00310 ArduinoOTA
00311 .onStart([]() {
00312     String type;
00313     if (ArduinoOTA.getCommand() == U_FLASH)
00314         type = "sketch";
00315     else // U_SPIFFS
00316         type = "filesystem";
00317
00318     // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00319     Serial.println("Start updating " + type);
00320 })
00321 .onEnd([]() {
00322     Serial.println("\nEnd");
00323 })
00324 .onProgress([](unsigned int progress, unsigned int total) {
00325     Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00326 })
00327 .onError([](ota_error_t error) {
00328     Serial.printf("Error[%u]: ", error);
00329     if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00330     else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00331     else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00332     else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00333     else if (error == OTA_END_ERROR) Serial.println("End Failed");
00334 });
00335
00336 ArduinoOTA.begin();
00337
00338 printf("Setup end\n");
00339 }
00340
00347
00348 void GetTemperature(void * parameter) {
00349     float tmp0 = 0;
00350     float tmp1 = 0;
00351     for (;;) {
00352         sensors.requestTemperatures(); // Send the command to get temperatures
00353         vTaskDelay(100);
00354         tmp0 = sensors.getTempC(MotorOil);
00355         if (tmp0 == DEVICE_DISCONNECTED_C) {
00356             if (motorErrorReported == "Aus") { // Nur einmal melden
00357                 Serial.print("Error read Motor Temp\n");
00358                 motorErrorReported = "Ein";
00359                 MotorTemp = -5.0;
00360             } else {
00361                 MotorTemp = tmp0 + fMotorOffset;
00362                 motorErrorReported = "Aus"; // Fehler wurde behoben
00363             }
00364         }
00365         vTaskDelay(100);
00366         tmp1 = sensors.getTempC(MotorCoolant);
00367         if (tmp1 == DEVICE_DISCONNECTED_C) {
00368             if (coolantErrorReported == "Aus") { // Nur einmal melden
00369                 Serial.print("Error read Coolant Temp\n");
00370                 coolantErrorReported = "Ein";
00371                 CoolantTemp = -5.0;
00372             } else {
00373                 CoolantTemp = tmp1 + fCoolantOffset;
00374                 coolantErrorReported = "Aus"; // Fehler wurde behoben
00375             }
00376         }
00377         vTaskDelay(100);
00378     }
00379 }

```

```

00376     }
00377 }
00378
00384 double ReadRPM() {
00385     double RPM = 0;
00386
00387     portENTER_CRITICAL(&mux);
00388     if (PeriodCount != 0) {
00389         RPM = 1000000.00 / PeriodCount;
00390     }
00391     portEXIT_CRITICAL(&mux);
00392     if (millis() > Last_int_time + 200) RPM = 0;
00393     return (RPM);
00394 }
00395
00396
00397 bool IsTimeToUpdate(unsigned long NextUpdate) {
00398     return (NextUpdate < millis());
00399 }
00400 unsigned long InitNextUpdate(unsigned long Period, unsigned long Offset = 0) {
00401     return millis() + Period + Offset;
00402 }
00403
00404 void SetNextUpdate(unsigned long &NextUpdate, unsigned long Period) {
00405     while ( NextUpdate < millis() ) NextUpdate += Period;
00406 }
00407
00408 /***** n2k Datenfunktionen *****/
00416 void SendN2kDCStatus(double BatteryVoltage, double SoC, double BatCapacity) {
00417     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
00418         BatteryDCStatusSendOffset);
00419     tN2kMsg N2kMsg;
00420     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00421         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00422
00423         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00424         Serial.printf("SoC          : %3.1f %\n", SoC);
00425         Serial.printf("Capacity    : %3.1f Ah\n", BatCapacity);
00426         // SetN2kDCStatus(N2kMsg,1,1,N2kDCT_Battery,56,92,38500,0.012, AhToCoulomb(420));
00427         SetN2kDCStatus(N2kMsg, 1, 2, N2kDCT_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
00428             AhToCoulomb(55));
00429         NMEA2000.SendMsg(N2kMsg);
00430     }
00431
00437 void SendN2kBattery(double BatteryVoltage) {
00438     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00439     tN2kMsg N2kMsg;
00440
00441     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00442         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00443
00444         Serial.printf("Voltage      : %3.1f V\n", BatteryVoltage);
00445
00446         SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00447         NMEA2000.SendMsg(N2kMsg);
00448     }
00449 }
00450
00457 void SendN2kTankLevel(double level, double capacity) {
00458     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00459     tN2kMsg N2kMsg;
00460
00461     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00462         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00463
00464         Serial.printf("Fuel Level   : %3.1f %\n", level);
00465         Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00466
00467         SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00468         NMEA2000.SendMsg(N2kMsg);
00469     }
00470 }
00471
00481 void SendN2kEngineData(double Oiltemp, double Coolanttemp, double rpm, double hours, double voltage) {
00482     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00483     tN2kMsg N2kMsg;
00484     tN2kEngineDiscreteStatus1 Status1;
00485     tN2kEngineDiscreteStatus2 Status2;
00486     Status1.Bits.OverTemperature = Oiltemp > 90;
00487     Status1.Bits.LowCoolantLevel = Coolanttemp > 90;
00488     Status1.Bits.LowSystemVoltage = voltage < 11;
00489     Status2.Bits.EngineShuttingDown = rpm < 100;
00490     EngineOn = !Status2.Bits.EngineShuttingDown;
00491
00492     if ( IsTimeToUpdate(SlowDataUpdated) ) {

```

```

00493     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00494
00495     Serial.printf("Oil Temp      : %3.1f °C \n", Oiltemp);
00496     Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
00497     Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00498     Serial.printf("Overtemp Oil: %s \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00499     Serial.printf("Overtemp Mot: %s \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00500     Serial.printf("Engine Off  : %s \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00501
00502     // SetN2kTemperatureExt(N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
    // PGN130312, uncomment the PGN to be used
00503
00504     SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Coolanttemp),
    N2kDoubleNA, N2kDoubleNA, hours, N2kDoubleNA, N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00505
00506     NMEA2000.SendMsg(N2kMsg);
00507 }
00508 }
00509
00515 void SendN2kEngineRPM(double RPM) {
00516     static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMsSendOffset);
00517     tN2kMsg N2kMsg;
00518
00519     if ( IsTimeToUpdate(SlowDataUpdated) ) {
00520         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00521
00522         Serial.printf("Engine RPM   : %4.0f RPM \n", RPM);
00523
00524         SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00525
00526         NMEA2000.SendMsg(N2kMsg);
00527     }
00528 }
00529
00536 double ReadVoltage(byte pin) {
00537     double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in
    range 0 to 4095
00538     if (reading < 1 || reading > 4095) return 0;
00539     // return -0.0000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +
    0.000854596860691 * reading + 0.065440348345433;
00540     return (-0.0000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -
    0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;
00541 } // Added an improved polynomial, use either, comment out as required
00542
00543 /***** Loop *****/
00544 void loop() {
00545
00546     LoopIndicator();
00547
00548     BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
    // This implements a low pass filter to eliminate spike for ADC readings
00549
00550     FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
    This implements a low pass filter to eliminate spike for ADC readings
00551
00552     EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
    pass filter to eliminate spike for RPM measurements
00553
00554     BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
    Zustand über Spannung
00555     // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00556
00557     EngineHours(EngineOn);
00558
00559     SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00560     SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00561     SendN2kEngineRPM(EngineRPM);
00562     SendN2kBattery(BordSpannung);
00563     SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00564
00565     NMEA2000.ParseMessages();
00566     int SourceAddress = NMEA2000.GetN2kSource();
00567     if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00568         NodeAddress = SourceAddress; // Set new Node Address (to save only once)
00569         preferences.begin("nvs", false);
00570         preferences.putInt("LastNodeAddress", SourceAddress);
00571         preferences.end();
00572         Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00573     }
00574
00575     // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00576     if ( Serial.available() ) {
00577         Serial.read();
00578     }
00579
00580
00581 // OTA

```



```

00582     ArduinoOTA.handle();
00583
00588     websocket.loop();
00589     fCoolantTemp = CoolantTemp;
00590     fMotorTemp = MotorTemp;
00591     fBordSpannung = BordSpannung;
00592     fDrehzahl = EngineRPM;
00593     sCL_Status = sWiFiStatus(WiFi.status());
00594     sAP_Station = WiFi.softAPgetStationNum();
00595     freeHeapSpace();
00596
00601     if (IsRebootRequired) {
00602         Serial.println("Rebooting ESP32: ");
00603         delay(1000); // give time for reboot page to load
00604         ESP.restart();
00605     }
00606
00607
00608 }

```

## 7.34 src/NMEA0183Telegram.h-Dateireferenz

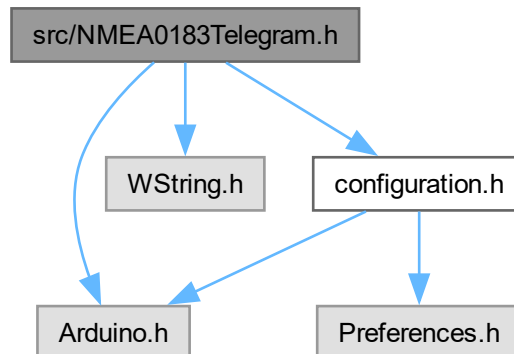
NMEA0183 Telegramme senden.

```

#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 XDR Sensor data.*
- String `sendRPM` ()  
*Send NMEA0183 Send RPM Sensor data.*

### 7.34.1 Ausführliche Beschreibung

NMEA0183 Telegramme senden.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

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

### 7.34.2 Dokumentation der Funktionen

#### 7.34.2.1 CheckSum()

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

Checksum calculation for NMEA.

Parameter

NMEADData	
-----------	--

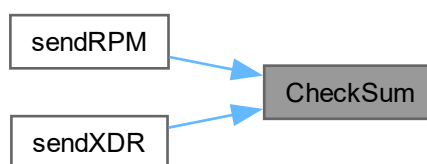
Rückgabe

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:



### 7.34.2.2 sendXDR()

```
String sendXDR ()
```

Send NMEA0183 Send XDR Sensor data.

Rückgabe

String

Definiert in Zeile 76 der Datei [NMEA0183Telegram.h](#).

```
00077 {  
00078     String HexChecksum;  
00079     String NMEASensor;  
00080     String SendSensor;  
00081  
00082     NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);  
00083     //NMEASensorKraeng += ",";  
00084     NMEASensor += String(fGaugeDrehzahl);  
00085     NMEASensor += ",D,ROLL";  
00086  
00087     // Build CheckSum  
00088     HexChecksum = String(CheckSum(NMEASensor), HEX);  
00089     // Build complete NMEA string  
00090     SendSensor = "$" + NMEASensor;  
00091     SendSensor += "*";  
00092     SendSensor += HexChecksum;  
00093  
00094     Serial.println(SendSensor);  
00095  
00096     return SendSensor;  
00097 }
```

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



### 7.34.2.3 sendRPM()

```
String sendRPM ()
```

Send NMEA0183 Send RPM Sensor data.

## Rückgabe

String

Definiert in Zeile 105 der Datei [NMEA0183Telegram.h](#).

```

00106 {
00107     String HexChecksum;
00108     String NMEASensor;
00109     String SendSensor;
00110
00111     NMEASensor = "IIRPM,E,1,"; //NMEASensor = "IIXDR,E,1," + String(SensorID);
00112     NMEASensor += String(fGaugeDrehzahl);
00113     NMEASensor += ",15,A";
00114
00115     // Build CheckSum
00116     HexChecksum = String(CheckSum(NMEASensor), HEX);
00117     // Build complete NMEA string
00118     SendSensor = "$" + NMEASensor;
00119     SendSensor += "*";
00120     SendSensor += HexChecksum;
00121
00122     Serial.println(SendSensor);
00123
00124     return SendSensor;
00125 }

```

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



## 7.35 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

```

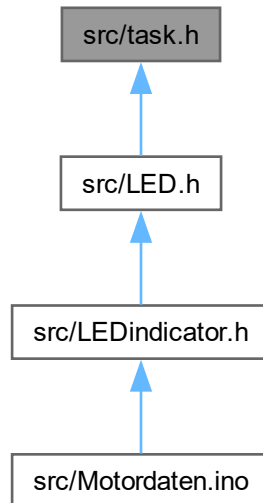
```

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
00054     RPM - Revolutions
00055
00056         1 2 3   4   5 6
00057         | | |   |   | |
00058     $--RPM,a,x,x.x,x.x,A*hh<CR><LF>
00059
00060     Field Number:
00061     1) Sourse, S = Shaft, E = Engine
00062     2) Engine or shaft number
00063     3) Speed, Revolutions per minute
00064     4) Propeller pitch, % of maximum, "-" means astern
00065     5) Status, A means data is valid
00066     6) Checksum
00067
00068 */
00069
00075
00076 String sendXDR()
00077 {
00078     String HexChecksum;
00079     String NMEASensor;
00080     String SendSensor;
00081
00082     NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);
00083     //NMEASensorKraeng += ",";
00084     NMEASensor += String(fGaugeDrehzahl);
00085     NMEASensor += ",D,ROLL";
00086
00087     // Build CheckSum
00088     HexChecksum = String(CheckSum(NMEASensor), HEX);
00089     // Build complete NMEA string
00090     SendSensor = "$" + NMEASensor;
00091     SendSensor += "*";
00092     SendSensor += HexChecksum;
00093
00094     Serial.println(SendSensor);
00095
00096     return SendSensor;
00097 }
00098
00104
00105 String sendRPM()
00106 {
00107     String HexChecksum;
00108     String NMEASensor;
00109     String SendSensor;
00110
00111     NMEASensor = "IIRPM,E,1,"; //NMEASensor = "IIXDR,E,1," + String(SensorID);
00112     NMEASensor += String(fGaugeDrehzahl);
00113     NMEASensor += ",15,A";
00114
00115     // Build CheckSum
00116     HexChecksum = String(CheckSum(NMEASensor), HEX);
00117     // Build complete NMEA string
00118     SendSensor = "$" + NMEASensor;
00119     SendSensor += "*";
00120     SendSensor += HexChecksum;
00121
00122     Serial.println(SendSensor);
00123
00124     return SendSensor;
00125 }

```

## 7.36 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)`

### 7.36.1 Makro-Dokumentation

#### 7.36.1.1 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`.

### 7.36.1.2 taskEnd

```
#define taskEnd()
```

**Wert:**

```
}
```

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

### 7.36.1.3 taskSwitch

```
#define taskSwitch()
```

**Wert:**

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

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

### 7.36.1.4 taskPause

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

**Wert:**

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

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

### 7.36.1.5 taskWaitFor

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

**Wert:**

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

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

### 7.36.1.6 taskStepName

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

**Wert:**

```
TASKSTEP_##STEPNAME :
```

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

### 7.36.1.7 taskJumpTo

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

#### Wert:

```
goto TASKSTEP_##STEPNAME
```

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

## 7.37 task.h

[gehe zur Dokumentation dieser Datei](#)

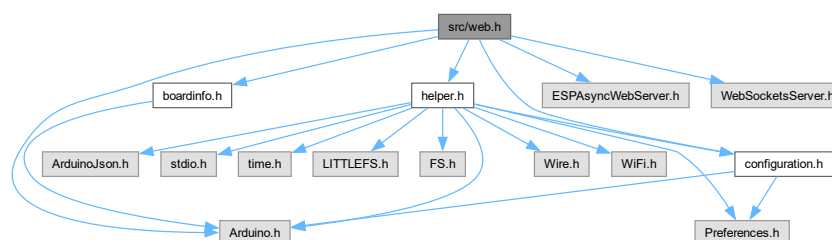
```
00001 #ifndef __TASK_H_  
00002 #define __TASK_H_  
00003  
00004  
00005 // grundlegene Worte um einen Task Bereich einzugrenzen  
00006 #define taskBegin() static int mark = 0; static unsigned long __attribute__((unused)) timeStamp = 0;  
    switch(mark){ case 0:  
00007 #define taskEnd() }  
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
```

## 7.38 src/web.h-Dateireferenz

Webseite Variablen lesen und schreiben, Webseiten erstellen.

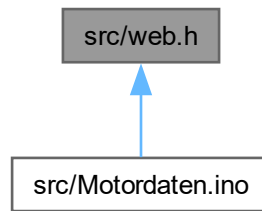
```
#include "helper.h"  
#include "configuration.h"  
#include "boardinfo.h"  
#include <ESPAsyncWebServer.h>  
#include <WebSocketsServer.h>  
#include <Arduino.h>
```

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

- WebSocketsServer [webSocket](#) = WebSocketsServer(81)
- String [sBoardInfo](#)
- [BoardInfo](#) [boardInfo](#)
- bool [IsRebootRequired](#) = false
- String [sCL\\_Status](#) = [sWifiStatus](#)(WiFi.status())

## 7.38.1 Ausführliche Beschreibung

Webseite Variablen lesen und schreiben, Webseiten erstellen.

### Autor

Gerry Sebb

### Version

0.1

### Datum

2025-01-06

### Copyright

Copyright (c) 2025

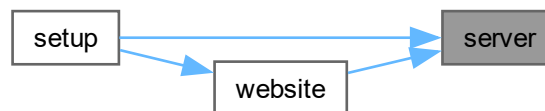
Definiert in Datei [web.h](#).

## 7.38.2 Dokumentation der Funktionen

### 7.38.2.1 server()

```
AsyncWebServer server (
    80 )
```

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



### 7.38.2.2 processor()

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

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

```

00030 {
00031     if (var == "CONFIGPLACEHOLDER")
00032     {
00033         String buttons = "";
00034         buttons += "<form onSubmit = \"event.preventDefault(); formToJson(this);\">";
00035         buttons += "<p class=\"CInput\"><label>SSID </label><input type = \"text\" name = \"SSID\" \"
value=\"\";
00036         buttons += tAP_Config.wAP_SSID;
00037         buttons += "\"/></p>";
00038         buttons += "<p class=\"CInput\"><label>IP </label><input type = \"text\" name = \"IP\" \"
value=\"\";
00039         buttons += tAP_Config.wAP_IP;
00040         buttons += "\"/></p>";
00041         buttons += "<p class=\"CInput\"><label>Password </label><input type = \"text\" name = \"
Password\" value=\"\";
00042         buttons += tAP_Config.wAP_Password;
00043         buttons += "\"/></p>";
00044         buttons += "<p class=\"CInput\"><label>Oil Offset </label><input type = \"text\" name = \"
MotorOffset\" value=\"\";
00045         buttons += tAP_Config.wMotor_Offset;
00046         buttons += "\"/> &deg;C</p>";
00047         buttons += "<p class=\"CInput\"><label>K&uuml;hlwasser Offset </label><input type = \"text\" \"
name = \"CoolantOffset\" value=\"\";
00048         buttons += tAP_Config.wCoolant_Offset;
00049         buttons += "\"/> &deg;C</p>";
00050         buttons += "<p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name = \"
Fuellstandmax\" value=\"\";
00051         buttons += tAP_Config.wFuellstandmax;
00052         buttons += "\"/> l</p>";
00053         buttons += "<p class=\"CInput\"><label>ADC1 Kalibrierung </label><input type = \"text\" name = \"
ADC1_Cal\" value=\"\";
00054         buttons += tAP_Config.wADC1_Cal;
00055         buttons += "\"/></p>";
00056         buttons += "<p class=\"CInput\"><label>ADC2 Kalibrierung </label><input type = \"text\" name = \"
ADC2_Cal\" value=\"\";
00057         buttons += tAP_Config.wADC2_Cal;
00058         buttons += "\"/></p>";
00059         buttons += "<p class=\"button\"><input type=\"submit\" value=\"Speichern\"></p>";
00060         buttons += "</form>";
00061         return buttons;
00062     }
```

```
00063     return String();
00064 }
```

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



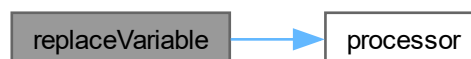
### 7.38.2.3 replaceVariable()

```
String replaceVariable (
    const String & var)
```

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

```
00070 {
00071     if (var == "sDrehzahl") return String(fDrehzahl, 1);
00072     if (var == "sFuellstand") return String(FuellLevel, 1);
00073     if (var == "sFuellstandmax") return String(FuellLevelMax, 1);
00074     if (var == "sBordspannung") return String(fBordSpannung, 1);
00075     if (var == "sCoolantTemp") return String(fCoolantTemp, 1);
00076     if (var == "sMotorTemp") return String(fMotorTemp, 1);
00077     if (var == "sCoolantOffset") return String(fCoolantOffset);
00078     if (var == "sMotorOffset") return String(fMotorOffset);
00079     if (var == "sMotorError") return String(motorErrorReported);
00080     if (var == "sCoolantError") return String(coolantErrorReported);
00081     if (var == "sBoardInfo") return sBoardInfo;
00082     if (var == "sADC1_Cal") return String(ADC_Calibration_Value1);
00083     if (var == "sADC2_Cal") return String(ADC_Calibration_Value2);
00084     if (var == "sHeapspace") return sHeapspace;
00085     if (var == "sFS_USpace") return String(LittleFS.usedBytes());
00086     if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00087     if (var == "sAP_IP") return WiFi.softAPIP().toString();
00088     if (var == "sAP_Clients") return String(sAP_Station);
00089     if (var == "sCL_Addr") return WiFi.localIP().toString();
00090     if (var == "sCL_Status") return String(sCL_Status);
00091     if (var == "sOneWire_Status") return String(sOneWire_Status);
00092     if (var == "sVersion") return Version;
00093     if (var == "sCounter") return String(Counter);
00094     if (var == "CONFIGPLACEHOLDER") return processor(var);
00095     return "NoVariable";
00096 }
```

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



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



### 7.38.2.4 website()

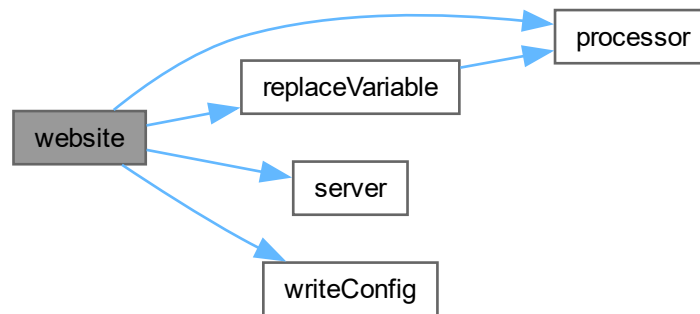
```
void website ()
```

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

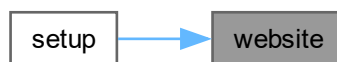
```

00098     {
00099     server.on("/favicon.ico", HTTP_GET, [] (AsyncWebServerRequest *request){
00100         request->send(LittleFS, "/favicon.ico", "image/x-icon");
00101     });
00102     server.on("/logo80.jpg", HTTP_GET, [] (AsyncWebServerRequest *request){
00103         request->send(LittleFS, "/logo80.jpg", "image/jpeg");
00104     });
00105     server.on("/", HTTP_GET, [] (AsyncWebServerRequest* request) {
00106         request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00107     });
00108     server.on("/system.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00109         request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00110     });
00111     server.on("/settings.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00112         request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00113     });
00114     server.on("/werte.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00115         request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00116     });
00117     server.on("/ueber.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00118         request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00119     });
00120     server.on("/reboot", HTTP_GET, [] (AsyncWebServerRequest * request) {
00121         request->send(LittleFS, "/reboot.html", String(), false, processor);
00122         IsRebootRequired = true;
00123     });
00124     server.on("/gauge.min.js", HTTP_GET, [] (AsyncWebServerRequest* request) {
00125         request->send(LittleFS, "/gauge.min.js");
00126     });
00127     server.on("/style.css", HTTP_GET, [] (AsyncWebServerRequest *request) {
00128         request->send(LittleFS, "/style.css", "text/css");
00129     });
00130     server.on("/settings.html", HTTP_POST, [] (AsyncWebServerRequest *request)
00131     {
00132         int count = request->params();
00133         Serial.printf("Anzahl: %i\n", count);
00134         for (int i = 0; i < count; i++)
00135         {
00136             AsyncWebParameter* p = request->getParam(i);
00137             Serial.print("PWerte von der Internet - Seite: ");
00138             Serial.print("Param name: ");
00139             Serial.println(p->name());
00140             Serial.print("Param value: ");
00141             Serial.println(p->value());
00142             Serial.println("-----");
00143             // p->value in die config schreiben
00144             writeConfig(p->name(), p->value());
00145         }
00146         request->send(200, "text/plain", "Daten gespeichert");
00147     });
00148 }
  
```

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



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



## 7.38.3 Variablen-Dokumentation

### 7.38.3.1 webSocket

```
WebSocketsServer webSocket = WebSocketsServer(81)
```

Definiert in Zeile 22 der Datei [web.h](#).

### 7.38.3.2 sBoardInfo

```
String sBoardInfo
```

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

### 7.38.3.3 boardInfo

```
BoardInfo boardInfo
```

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

### 7.38.3.4 IsRebootRequired

```
bool IsRebootRequired = false
```

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

### 7.38.3.5 sCL\_Status

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

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

## 7.39 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 <WebSocketsServer.h>
00018 #include <Arduino.h>
00019
00020 // Set web server port number to 80
00021 AsyncWebServer server(80);
00022 WebSocketsServer websocket = WebSocketsServer(81); // WebSocket server on port 81
00023
00024 // Info Board for HTML-Output
00025 String sBoardInfo;
00026 BoardInfo boardInfo;
00027 bool IsRebootRequired = false;
00028
00029 String processor(const String& var)
00030 {
00031     if (var == "CONFIGPLACEHOLDER")
00032     {
00033         String buttons = "";
00034         buttons += "<form onSubmit = \"event.preventDefault(); formToJson(this);\">";
00035         buttons += "<p class=\"CInput\"><label>SSID </label><input type = \"text\" name = \"SSID\" \"
value=\"";
00036         buttons += tAP_Config.wAP_SSID;
00037         buttons += "\"/></p>";
00038         buttons += "<p class=\"CInput\"><label>IP </label><input type = \"text\" name = \"IP\" \"
value=\"";
00039         buttons += tAP_Config.wAP_IP;
00040         buttons += "\"/></p>";
00041         buttons += "<p class=\"CInput\"><label>Password </label><input type = \"text\" name = \"
Password\" value=\"";
00042         buttons += tAP_Config.wAP_Password;
00043         buttons += "\"/></p>";
00044         buttons += "<p class=\"CInput\"><label>Oil Offset </label><input type = \"text\" name = \"
MotorOffset\" value=\"";
00045         buttons += tAP_Config.wMotor_Offset;
00046         buttons += "\"/> &deg;C</p>";
00047         buttons += "<p class=\"CInput\"><label>K&uuml;hlwasser Offset </label><input type = \"text\" \"
name = \"CoolantOffset\" value=\"";
00048         buttons += tAP_Config.wCoolant_Offset;
00049         buttons += "\"/> &deg;C</p>";
00050         buttons += "<p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name = \"
Fuellstandmax\" value=\"";
00051         buttons += tAP_Config.wFuellstandmax;
00052         buttons += "\"/> l</p>";
00053         buttons += "<p class=\"CInput\"><label>ADC1 Kalibrierung </label><input type = \"text\" name = \"
ADC1_Cal\" value=\"";
00054         buttons += tAP_Config.wADC1_Cal;
00055         buttons += "\"/></p>";
00056         buttons += "<p class=\"CInput\"><label>ADC2 Kalibrierung </label><input type = \"text\" name = \"
ADC2_Cal\" value=\"";
00057         buttons += tAP_Config.wADC2_Cal;
00058         buttons += "\"/></p>";
00059         buttons += "<p class=\"button\"><input type=\"submit\" value=\"Speichern\"></p>";
```

```

00060         buttons += "</form>";
00061         return buttons;
00062     }
00063     return String();
00064 }
00065
00066 //Variables for website
00067 String sCL_Status = sWifiStatus(WiFi.status());
00068
00069 String replaceVariable(const String& var)
00070 {
00071     if (var == "sDrehzahl") return String(fDrehzahl, 1);
00072     if (var == "sFuellstand") return String(FuellLevel, 1);
00073     if (var == "sFuellstandmax") return String(FuellLevelMax, 1);
00074     if (var == "sBordspannung") return String(fBordSpannung, 1);
00075     if (var == "sCoolantTemp") return String(fCoolantTemp, 1);
00076     if (var == "sMotorTemp") return String(fMotorTemp, 1);
00077     if (var == "sCoolantOffset") return String(fCoolantOffset);
00078     if (var == "sMotorOffset") return String(fMotorOffset);
00079     if (var == "sMotorError") return String(motorErrorReported);
00080     if (var == "sCoolantError") return String(coolantErrorReported);
00081     if (var == "sBoardInfo") return sBoardInfo;
00082     if (var == "sADC1_Cal") return String(ADC_Calibration_Value1);
00083     if (var == "sADC2_Cal") return String(ADC_Calibration_Value2);
00084     if (var == "sHeapspace") return sHeapspace;
00085     if (var == "sFS_USpace") return String(LittleFS.usedBytes());
00086     if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00087     if (var == "sAP_IP") return WiFi.softAPIP().toString();
00088     if (var == "sAP_Clients") return String(sAP_Station);
00089     if (var == "sCL_Addr") return WiFi.localIP().toString();
00090     if (var == "sCL_Status") return String(sCL_Status);
00091     if (var == "sOneWire_Status") return String(sOneWire_Status);
00092     if (var == "sVersion") return Version;
00093     if (var == "sCounter") return String(Counter);
00094     if (var == "CONFIGPLACEHOLDER") return processor(var);
00095     return "NoVariable";
00096 }
00097
00098 void website() {
00099     server.on("/favicon.ico", HTTP_GET, [] (AsyncWebServerRequest *request){
00100         request->send(LittleFS, "/favicon.ico", "image/x-icon");
00101     });
00102     server.on("/logo80.jpg", HTTP_GET, [] (AsyncWebServerRequest *request){
00103         request->send(LittleFS, "/logo80.jpg", "image/jpeg");
00104     });
00105     server.on("/", HTTP_GET, [] (AsyncWebServerRequest* request) {
00106         request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00107     });
00108     server.on("/system.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00109         request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00110     });
00111     server.on("/settings.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00112         request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00113     });
00114     server.on("/werte.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00115         request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00116     });
00117     server.on("/ueber.html", HTTP_GET, [] (AsyncWebServerRequest* request) {
00118         request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00119     });
00120     server.on("/reboot", HTTP_GET, [] (AsyncWebServerRequest * request) {
00121         request->send(LittleFS, "/reboot.html", String(), false, processor);
00122         IsRebootRequired = true;
00123     });
00124     server.on("/gauge.min.js", HTTP_GET, [] (AsyncWebServerRequest* request) {
00125         request->send(LittleFS, "/gauge.min.js");
00126     });
00127     server.on("/style.css", HTTP_GET, [] (AsyncWebServerRequest *request) {
00128         request->send(LittleFS, "/style.css", "text/css");
00129     });
00130     server.on("/settings.html", HTTP_POST, [] (AsyncWebServerRequest *request)
00131     {
00132         int count = request->params();
00133         Serial.printf("Anzahl: %i\n", count);
00134         for (int i = 0; i < count; i++)
00135         {
00136             AsyncWebParameter* p = request->getParam(i);
00137             Serial.print("PWerte von der Internet - Seite: ");
00138             Serial.print("Param name: ");
00139             Serial.println(p->name());
00140             Serial.print("Param value: ");
00141             Serial.println(p->value());
00142             Serial.println("-----");
00143             // p->value in die config schreiben
00144             writeConfig(p->name(), p->value());
00145         }
00146         request->send(200, "text/plain", "Daten gespeichert");

```

```
00147     });  
00148 }  
00149
```



# Index

ADC\_Calibration\_Value1  
    configuration.h, [43](#)  
ADC\_Calibration\_Value2  
    configuration.h, [44](#)  
ADCPin1  
    Motordaten.ino, [96](#)  
ADCPin2  
    Motordaten.ino, [96](#)  
Altitude  
    tBoatData, [18](#)  
AP\_IP  
    configuration.h, [42](#)  
AP\_PASSWORD  
    configuration.h, [42](#)  
AP\_SSID  
    configuration.h, [42](#)  
  
Bat1Capacity  
    configuration.h, [46](#)  
Bat2Capacity  
    configuration.h, [46](#)  
BatSoC  
    configuration.h, [46](#)  
BatteryDCSendOffset  
    configuration.h, [38](#)  
BatteryDCStatusSendOffset  
    configuration.h, [38](#)  
baudrate  
    Motordaten.ino, [96](#)  
bClientConnected  
    configuration.h, [43](#)  
bConnect\_CL  
    configuration.h, [43](#)  
bl2C\_Status  
    configuration.h, [44](#)  
Blue  
    LED.h, [69](#)  
BoardInfo, [13](#)  
    BoardInfo, [13](#)  
    m\_chipid, [15](#)  
    m\_chipinfo, [15](#)  
    ShowChipID, [14](#)  
    ShowChipIDtoString, [14](#)  
    ShowChipInfo, [14](#)  
    ShowChipTemperature, [14](#)  
boardInfo  
    web.h, [115](#)  
BoardInfo.cpp  
    BUF, [30](#)  
    temprature\_sens\_read, [30](#)  
  
BordSpannung  
    configuration.h, [45](#)  
bsz1  
    hourmeter.h, [66](#)  
BUF  
    BoardInfo.cpp, [30](#)  
  
channel  
    configuration.h, [41](#)  
Checksum  
    NMEA0183Telegram.h, [104](#)  
chipid  
    configuration.h, [41](#)  
CL\_IP  
    configuration.h, [43](#)  
CL\_PASSWORD  
    configuration.h, [39](#)  
CL\_SSID  
    configuration.h, [39](#)  
COG  
    tBoatData, [17](#)  
configuration.h  
    ADC\_Calibration\_Value1, [43](#)  
    ADC\_Calibration\_Value2, [44](#)  
    AP\_IP, [42](#)  
    AP\_PASSWORD, [42](#)  
    AP\_SSID, [42](#)  
    Bat1Capacity, [46](#)  
    Bat2Capacity, [46](#)  
    BatSoC, [46](#)  
    BatteryDCSendOffset, [38](#)  
    BatteryDCStatusSendOffset, [38](#)  
    bClientConnected, [43](#)  
    bConnect\_CL, [43](#)  
    bl2C\_Status, [44](#)  
    BordSpannung, [45](#)  
    channel, [41](#)  
    chipid, [41](#)  
    CL\_IP, [43](#)  
    CL\_PASSWORD, [39](#)  
    CL\_SSID, [39](#)  
    coolantErrorReported, [46](#)  
    CoolantTemp, [45](#)  
    Counter, [46](#)  
    dMWV\_WindDirectionT, [48](#)  
    dMWV\_WindSpeedM, [48](#)  
    DNS\_PORT, [40](#)  
    dVWR\_WindAngle, [48](#)  
    dVWR\_WindDirectionM, [48](#)  
    dVWR\_WindSpeedkn, [48](#)

dVWR\_WindSpeedms, 48  
 Engine\_RPM\_Pin, 40  
 EngineOn, 45  
 EngineRPM, 45  
 EngineSendOffset, 38  
 EngineStatus, 40  
 ESP32\_CAN\_RX\_PIN, 37  
 ESP32\_CAN\_TX\_PIN, 37  
 fbmp\_altitude, 44  
 fbmp\_pressure, 44  
 fbmp\_temperature, 44  
 fBordSpannung, 47  
 fCoolantOffset, 47  
 fCoolantTemp, 47  
 fDrehzahl, 47  
 fGaugeDrehzahl, 47  
 fMotorOffset, 47  
 fMotorTemp, 47  
 FuelLevel, 45  
 FuelLevelMax, 45  
 Gateway, 42  
 hide\_SSID, 42  
 HostName, 39  
 i, 41  
 I2C\_SCL, 39  
 I2C\_SDA, 39  
 id, 41  
 iDistance, 45  
 iMaxSonar, 44  
 IP, 42  
 iSTA\_on, 43  
 max\_connection, 42  
 motorErrorReported, 46  
 MotorTemp, 45  
 N2K\_SOURCE, 38  
 NMask, 42  
 NodeAddress, 41  
 Off, 40  
 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  
 sHeapSpace, 41  
 sl2C\_Status, 44  
 SlowDataUpdatePeriod, 38  
 SoCError, 46  
 sOneWire\_Status, 46  
 sOrient, 48  
 sSTBB, 47  
 TankSendOffset, 38  
 tAP\_Config, 41  
 TCP\_PORT, 40  
 udpAddress, 48  
 udpPort, 49  
 Version, 37  
 WEB\_TITEL, 39  
 coolantErrorReported  
   configuration.h, 46  
 CoolantTemp  
   configuration.h, 45  
 Counter  
   configuration.h, 46  
 CounterOld  
   hourmeter.h, 66  
 data/index.html, 23  
 data/reboot.html, 25  
 data/settings.html, 26  
 data/system.html, 27  
 data/ueber.html, 27  
 data/werte.html, 28  
 DaysSince1970  
   tBoatData, 17  
 debug\_log  
   Motordaten.ino, 79  
 DGPSAge  
   tBoatData, 18  
 DGPSReferenceStationID  
   tBoatData, 20  
 dMWV\_WindDirectionT  
   configuration.h, 48  
 dMWV\_WindSpeedM  
   configuration.h, 48  
 DNS\_PORT  
   configuration.h, 40  
 dVWR\_WindAngle  
   configuration.h, 48  
 dVWR\_WindDirectionM  
   configuration.h, 48  
 dVWR\_WindSpeedkn  
   configuration.h, 48  
 dVWR\_WindSpeedms  
   configuration.h, 48  
 Engine\_RPM\_Pin  
   configuration.h, 40  
 ENABLE\_DEBUG\_LOG  
   Motordaten.ino, 79  
 EngineHours  
   hourmeter.h, 65  
 EngineOn  
   configuration.h, 45  
 EngineRPM  
   configuration.h, 45  
 EngineSendOffset  
   configuration.h, 38  
 EngineStatus  
   configuration.h, 40  
 ErrorOff  
   LEDIndicator.h, 76  
 ErrorOn

- LEDIndicator.h, 76
- ESP32\_CAN\_RX\_PIN
  - configuration.h, 37
- ESP32\_CAN\_TX\_PIN
  - configuration.h, 37
- fbmp\_altitude
  - configuration.h, 44
- fbmp\_pressure
  - configuration.h, 44
- fbmp\_temperature
  - configuration.h, 44
- fBordSpannung
  - configuration.h, 47
- fCoolantOffset
  - configuration.h, 47
- fCoolantTemp
  - configuration.h, 47
- fDrehzahl
  - configuration.h, 47
- fGaugeDrehzahl
  - configuration.h, 47
- flashLED
  - LED.h, 70
- fMotorOffset
  - configuration.h, 47
- fMotorTemp
  - configuration.h, 47
- freeHeapSpace
  - helper.h, 52
- FuelLevel
  - configuration.h, 45
- FuelLevelMax
  - configuration.h, 45
- Gateway
  - configuration.h, 42
- GeoidalSeparation
  - tBoatData, 18
- GetTemperature
  - Motordaten.ino, 83
- GPSQualityIndicator
  - tBoatData, 19
- GPSTime
  - tBoatData, 18
- Green
  - LED.h, 69
- handleInterrupt
  - Motordaten.ino, 79
- HDOP
  - tBoatData, 18
- helper.h
  - freeHeapSpace, 52
  - I2C\_scan, 58
  - listDir, 54
  - readConfig, 55
  - ShowTime, 52
  - sWifiStatus, 59
  - toChar, 59
  - WiFiDiag, 53
  - writeConfig, 56, 57
- hide\_SSID
  - configuration.h, 42
- HostName
  - configuration.h, 39
- hourmeter.h
  - bsz1, 66
  - CounterOld, 66
  - EngineHours, 65
  - lastRun, 66
  - laststate1, 66
  - milliRest, 66
  - state1, 66
- i
  - configuration.h, 41
- I2C\_scan
  - helper.h, 58
- I2C\_SCL
  - configuration.h, 39
- I2C\_SDA
  - configuration.h, 39
- id
  - configuration.h, 41
- iDistance
  - configuration.h, 45
- iMaxSonar
  - configuration.h, 44
- InitNextUpdate
  - Motordaten.ino, 85
- IP
  - configuration.h, 42
- IsRebootRequired
  - web.h, 115
- iSTA\_on
  - configuration.h, 43
- IsTimeToUpdate
  - Motordaten.ino, 84
- Last\_int\_time
  - Motordaten.ino, 95
- lastRun
  - hourmeter.h, 66
- laststate1
  - hourmeter.h, 66
- Latitude
  - tBoatData, 18
- LED
  - LED.h, 69
- LED.h
  - Blue, 69
  - flashLED, 70
  - Green, 69
  - LED, 69
  - LEDblink, 70
  - LEDBoard, 69
  - LEDflash, 70

- LEDInit, [71](#)
- LEDOff, [72](#)
- LEDOff\_RGB, [72](#)
- LEDOn, [71](#)
- Red, [69](#)
- LEDblink
  - LED.h, [70](#)
- LEDBoard
  - LED.h, [69](#)
- LEDflash
  - LED.h, [70](#)
- LEDindicator.h
  - ErrorOff, [76](#)
  - ErrorOn, [76](#)
  - LoopIndicator, [75](#)
- LEDInit
  - LED.h, [71](#)
- LEDOff
  - LED.h, [72](#)
- LEDOff\_RGB
  - LED.h, [72](#)
- LEDOn
  - LED.h, [71](#)
- listDir
  - helper.h, [54](#)
- Longitude
  - tBoatData, [18](#)
- loop
  - Motordaten.ino, [92](#)
- LoopIndicator
  - LEDindicator.h, [75](#)
- m\_chipid
  - BoardInfo, [15](#)
- m\_chipinfo
  - BoardInfo, [15](#)
- max\_connection
  - configuration.h, [42](#)
- milliRest
  - hourmeter.h, [66](#)
- MKSPIFFSTOOL
  - replace\_fs, [11](#)
- MOBActivated
  - tBoatData, [20](#)
- MotorCoolant
  - Motordaten.ino, [95](#)
- MotorData NMEA2000, [1](#)
- Motordaten.ino
  - ADCpin1, [96](#)
  - ADCpin2, [96](#)
  - baudrate, [96](#)
  - debug\_log, [79](#)
  - ENABLE\_DEBUG\_LOG, [79](#)
  - GetTemperature, [83](#)
  - handleInterrupt, [79](#)
  - InitNextUpdate, [85](#)
  - IsTimeToUpdate, [84](#)
  - Last\_int\_time, [95](#)
  - loop, [92](#)
  - MotorCoolant, [95](#)
  - MotorOil, [96](#)
  - mux, [95](#)
  - oneWire, [79](#), [95](#)
  - PeriodCount, [95](#)
  - PROGMEM, [94](#)
  - ReadRPM, [84](#)
  - ReadVoltage, [92](#)
  - rs\_config, [96](#)
  - SendN2kBattery, [87](#)
  - SendN2kDCStatus, [86](#)
  - SendN2kEngineData, [89](#)
  - SendN2kEngineRPM, [91](#)
  - SendN2kTankLevel, [88](#)
  - SetNextUpdate, [86](#)
  - setup, [79](#)
  - StartValue, [95](#)
  - Task1, [96](#)
  - timer, [95](#)
- motorErrorReported
  - configuration.h, [46](#)
- MotorOil
  - Motordaten.ino, [96](#)
- MotorTemp
  - configuration.h, [45](#)
- mux
  - Motordaten.ino, [95](#)
- N2K\_SOURCE
  - configuration.h, [38](#)
- NMask
  - configuration.h, [42](#)
- NMEA0183Telegram.h
  - Checksum, [104](#)
  - sendRPM, [105](#)
  - sendXDR, [104](#)
- NodeAddress
  - configuration.h, [41](#)
- Off
  - configuration.h, [40](#)
- Offset
  - tBoatData, [19](#)
- On
  - configuration.h, [40](#)
- ONE\_WIRE\_BUS
  - configuration.h, [40](#)
- oneWire
  - Motordaten.ino, [79](#), [95](#)
- PAGE\_REFRESH
  - configuration.h, [38](#)
- PeriodCount
  - Motordaten.ino, [95](#)
- preferences
  - configuration.h, [41](#)
- processor
  - web.h, [112](#)
- PROGMEM

- Motordaten.ino, [94](#)
- readConfig
  - helper.h, [55](#)
- README.md, [28](#)
- ReadRPM
  - Motordaten.ino, [84](#)
- ReadVoltage
  - Motordaten.ino, [92](#)
- Red
  - LED.h, [69](#)
- replace\_fs, [11](#)
  - MKSPIFFSTOOL, [11](#)
- replace\_fs.py, [28](#)
- replaceVariable
  - web.h, [113](#)
- RPM\_Calibration\_Value
  - configuration.h, [39](#)
- RPMsSendOffset
  - configuration.h, [38](#)
- rs\_config
  - Motordaten.ino, [96](#)
- sAP\_Station
  - configuration.h, [43](#)
- SatelliteCount
  - tBoatData, [20](#)
- sBoardInfo
  - web.h, [115](#)
- sCL\_Status
  - web.h, [116](#)
- SEALEVELPRESSURE\_HPA
  - configuration.h, [39](#)
- SELF\_IP
  - configuration.h, [43](#)
- SendN2kBattery
  - Motordaten.ino, [87](#)
- SendN2kDCStatus
  - Motordaten.ino, [86](#)
- SendN2kEngineData
  - Motordaten.ino, [89](#)
- SendN2kEngineRPM
  - Motordaten.ino, [91](#)
- SendN2kTankLevel
  - Motordaten.ino, [88](#)
- sendRPM
  - NMEA0183Telegram.h, [105](#)
- sendXDR
  - NMEA0183Telegram.h, [104](#)
- server
  - web.h, [112](#)
- SERVER\_HOST\_NAME
  - configuration.h, [40](#)
- SetNextUpdate
  - Motordaten.ino, [86](#)
- setup
  - Motordaten.ino, [79](#)
- sHeapSpace
  - configuration.h, [41](#)
- ShowChipID
  - BoardInfo, [14](#)
- ShowChipIDtoString
  - BoardInfo, [14](#)
- ShowChipInfo
  - BoardInfo, [14](#)
- ShowChipTemperature
  - BoardInfo, [14](#)
- ShowTime
  - helper.h, [52](#)
- sl2C\_Status
  - configuration.h, [44](#)
- SlowDataUpdatePeriod
  - configuration.h, [38](#)
- SoCError
  - configuration.h, [46](#)
- SOG
  - tBoatData, [17](#)
- sOneWire\_Status
  - configuration.h, [46](#)
- sOrient
  - configuration.h, [48](#)
- src/BoardInfo.cpp, [29](#), [30](#)
- src/BoardInfo.h, [32](#), [33](#)
- src/BoatData.h, [33](#), [34](#)
- src/configuration.h, [34](#), [49](#)
- src/helper.h, [51](#), [60](#)
- src/hourmeter.h, [63](#), [67](#)
- src/LED.h, [67](#), [72](#)
- src/LEDIndicator.h, [73](#), [76](#)
- src/Motordaten.ino, [77](#), [97](#)
- src/NMEA0183Telegram.h, [103](#), [106](#)
- src/task.h, [108](#), [110](#)
- src/web.h, [110](#), [116](#)
- sSTBB
  - configuration.h, [47](#)
- StartValue
  - Motordaten.ino, [95](#)
- state1
  - hourmeter.h, [66](#)
- Status
  - tBoatData, [20](#)
- sWifiStatus
  - helper.h, [59](#)
- TankSendOffset
  - configuration.h, [38](#)
- tAP\_Config
  - configuration.h, [41](#)
- task.h
  - taskBegin, [108](#)
  - taskEnd, [108](#)
  - taskJumpTo, [109](#)
  - taskPause, [109](#)
  - taskStepName, [109](#)
  - taskSwitch, [109](#)
  - taskWaitFor, [109](#)
- Task1
  - Motordaten.ino, [96](#)

- taskBegin
  - task.h, 108
- taskEnd
  - task.h, 108
- taskJumpTo
  - task.h, 109
- taskPause
  - task.h, 109
- taskStepName
  - task.h, 109
- taskSwitch
  - task.h, 109
- taskWaitFor
  - task.h, 109
- 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, 95
- toChar
  - helper.h, 59
- TrueHeading
  - tBoatData, 17
- udpAddress
  - configuration.h, 48
- udpPort
  - configuration.h, 49
- Variation
  - tBoatData, 17
- Version
  - configuration.h, 37
- wADC1\_Cal
  - Web\_Config, 21
- wADC2\_Cal
  - Web\_Config, 21
- wAP\_IP
  - Web\_Config, 21
- wAP\_Password
  - Web\_Config, 21
- wAP\_SSID
  - Web\_Config, 21
- WaterDepth
  - tBoatData, 19
- WaterTemperature
  - tBoatData, 18
- wCoolant\_Offset
  - Web\_Config, 21
- web.h
  - boardInfo, 115
  - IsRebootRequired, 115
  - processor, 112
  - replaceVariable, 113
  - sBoardInfo, 115
  - sCL\_Status, 116
  - server, 112
  - website, 114
  - webSocket, 115
- Web\_Config, 20
  - wADC1\_Cal, 21
  - wADC2\_Cal, 21
  - wAP\_IP, 21
  - wAP\_Password, 21
  - wAP\_SSID, 21
  - wCoolant\_Offset, 21
  - wFuelstandmax, 21
  - wMotor\_Offset, 21
- WEB\_TITEL
  - configuration.h, 39
- website
  - web.h, 114
- webSocket
  - web.h, 115
- wFuelstandmax
  - Web\_Config, 21
- WiFiDiag
  - helper.h, 53
- WindAngle
  - tBoatData, 19
- WindDirectionM
  - tBoatData, 19
- WindDirectionT
  - tBoatData, 19
- WindSpeedK
  - tBoatData, 19
- WindSpeedM
  - tBoatData, 19
- wMotor\_Offset

---

Web\_Config, [21](#)  
writeConfig  
  helper.h, [56](#), [57](#)