Motordaten V 2.8.1

Erzeugt von Doxygen 1.13.2

1 MotorData NMEA2000	1
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
2 Verzeichnis der Namensbereiche	5
2.1 Liste aller Namensbereiche	5
3 Klassen-Verzeichnis	7
3.1 Auflistung der Klassen	7
4 Datei-Verzeichnis	9
4.1 Auflistung der Dateien	9
5 Dokumentation der Namensbereiche	11
5.1 replace_fs-Namensbereichsreferenz	. 11
5.1.1 Variablen-Dokumentation	11
5.1.1.1 MKSPIFFSTOOL	11
6 Klassen-Dokumentation	13
6.1 BoardInfo Klassenreferenz	13
6.1.1 Ausführliche Beschreibung	13
6.1.2 Beschreibung der Konstruktoren und Destruktoren	13
6.1.2.1 BoardInfo()	13
6.1.3 Dokumentation der Elementfunktionen	14
6.1.3.1 ShowChipID()	14
6.1.3.2 ShowChipInfo()	14
6.1.3.3 ShowChipTemperature()	14
6.1.3.4 ShowChipIDtoString()	. 15
6.1.4 Dokumentation der Datenelemente	15
6.1.4.1 m_chipid	15
6.1.4.2 m_chipinfo	. 15
6.2 tBoatData Strukturreferenz	16
6.2.1 Ausführliche Beschreibung	16
6.2.2 Beschreibung der Konstruktoren und Destruktoren	17
6.2.2.1 tBoatData()	17
6.2.3 Dokumentation der Datenelemente	17
6.2.3.1 DaysSince1970	17

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

7.6 settings.html	6
7.7 data/system.html-Dateireferenz	7
7.8 system.html	7
7.9 data/ueber.html-Dateireferenz	7
7.10 ueber.html	7
7.11 data/werte.html-Dateireferenz	8
7.12 werte.html	8
7.13 README.md-Dateireferenz	8
7.14 replace_fs.py-Dateireferenz	8
7.15 replace_fs.py	8
7.16 src/BoardInfo.cpp-Dateireferenz	9
7.16.1 Ausführliche Beschreibung	9
7.16.2 Makro-Dokumentation	0
7.16.2.1 BUF	0
7.16.3 Dokumentation der Funktionen	0
7.16.3.1 temprature_sens_read()	0
7.17 BoardInfo.cpp	0
7.18 src/BoardInfo.h-Dateireferenz	2
7.18.1 Ausführliche Beschreibung	2
7.19 BoardInfo.h	3
7.20 src/BoatData.h-Dateireferenz	3
7.21 BoatData.h	4
7.22 src/configuration.h-Dateireferenz	4
7.22.1 Ausführliche Beschreibung	7
7.22.2 Makro-Dokumentation	7
7.22.2.1 VersionSoftware	7
7.22.2.2 VersionHardware	7
7.22.2.3 ESP32_CAN_TX_PIN	8
7.22.2.4 ESP32_CAN_RX_PIN	8
7.22.2.5 N2K_SOURCE	8
7.22.2.6 EngineSendOffset	8
7.22.2.7 TankSendOffset	8
7.22.2.8 RPMSendOffset	8
7.22.2.9 BatteryDCSendOffset	8
7.22.2.10 BatteryDCStatusSendOffset	9
7.22.2.11 SlowDataUpdatePeriod	9
7.22.2.12 PAGE_REFRESH	9
7.22.2.13 WEB_TITEL	9
7.22.2.14 HostName	9
7.22.2.15 CL_SSID	9
7.22.2.16 CL_PASSWORD	9
7.22.2.17 I2C_SDA	9

7.22.2.18 I2C_SCL	40
7.22.2.19 SEALEVELPRESSURE_HPA	40
7.22.2.20 RPM_Calibration_Value	40
7.22.2.21 Eingine_RPM_Pin	40
7.22.2.22 ONE_WIRE_BUS	40
7.22.2.23 SERVER_HOST_NAME	40
7.22.2.24 TCP_PORT	40
7.22.2.25 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	42
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	43
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 sI2C_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()	56

7.24.2.7 I2C_scan()	57
7.24.2.8 sWifiStatus()	58
7.24.2.9 toChar()	58
7.25 helper.h	59
7.26 src/hourmeter.h-Dateireferenz	62
7.26.1 Ausführliche Beschreibung	63
7.26.2 Dokumentation der Funktionen	64
7.26.2.1 EngineHours()	64
7.26.3 Variablen-Dokumentation	65
7.26.3.1 bsz1	65
7.26.3.2 lastRun	65
7.26.3.3 CounterOld	66
7.26.3.4 milliRest	66
7.26.3.5 state1	66
7.26.3.6 laststate1	66
7.27 hourmeter.h	66
7.28 src/LED.h-Dateireferenz	67
7.28.1 Ausführliche Beschreibung	68
7.28.2 Dokumentation der Aufzählungstypen	68
7.28.2.1 LED	68
7.28.3 Dokumentation der Funktionen	69
7.28.3.1 LEDblink()	69
7.28.3.2 LEDflash()	69
7.28.3.3 flashLED()	70
7.28.3.4 LEDInit()	70
7.28.3.5 LEDon()	70
7.28.3.6 LEDoff()	71
7.28.3.7 LEDoff_RGB()	71
7.29 LED.h	71
7.30 src/LEDindicator.h-Dateireferenz	72
7.30.1 Ausführliche Beschreibung	74
7.30.2 Dokumentation der Funktionen	74
7.30.2.1 LoopIndicator()	74
7.30.3 Variablen-Dokumentation	75
7.30.3.1 ErrorOff	75
7.30.3.2 ErrorOn	75
7.31 LEDindicator.h	75
7.32 src/Motordaten.ino-Dateireferenz	76
7.32.1 Ausführliche Beschreibung	77
7.32.2 Makro-Dokumentation	78
7.32.2.1 ENABLE_DEBUG_LOG	78
7.32.3 Dokumentation der Funktionen	78

7.32.3.1 oneWire()	. 78
7.32.3.2 debug_log()	. 78
7.32.3.3 handleInterrupt()	. 78
7.32.3.4 setup()	. 79
7.32.3.5 GetTemperature()	. 82
7.32.3.6 ReadRPM()	. 83
7.32.3.7 IsTimeToUpdate()	. 84
7.32.3.8 InitNextUpdate()	. 84
7.32.3.9 SetNextUpdate()	. 85
7.32.3.10 SendN2kDCStatus()	. 86
7.32.3.11 SendN2kBattery()	. 87
7.32.3.12 SendN2kTankLevel()	. 88
7.32.3.13 SendN2kEngineData()	. 89
7.32.3.14 SendN2kEngineRPM()	. 90
7.32.3.15 ReadVoltage()	. 91
7.32.3.16 loop()	. 92
7.32.4 Variablen-Dokumentation	. 93
7.32.4.1 PROGMEM	. 93
7.32.4.2 StartValue	. 94
7.32.4.3 PeriodCount	. 94
7.32.4.4 Last_int_time	. 94
7.32.4.5 timer	. 94
7.32.4.6 mux	. 94
7.32.4.7 oneWire	. 94
7.32.4.8 MotorCoolant	. 95
7.32.4.9 MotorOil	. 95
7.32.4.10 ADCpin2	. 95
7.32.4.11 ADCpin1	. 95
7.32.4.12 Task1	. 95
7.32.4.13 baudrate	. 95
7.32.4.14 rs_config	. 96
7.33 Motordaten.ino	. 96
7.34 src/NMEA0183Telegram.h-Dateireferenz	. 102
7.34.1 Ausführliche Beschreibung	. 103
7.34.2 Dokumentation der Funktionen	. 103
7.34.2.1 CheckSum()	. 103
7.34.2.2 sendXDR()	. 104
7.34.2.3 sendRPM()	. 105
7.35 NMEA0183Telegram.h	. 106
7.36 src/task.h-Dateireferenz	. 107
7.36.1 Makro-Dokumentation	. 108
7.36.1.1 taskBegin	. 108

Index	119
7.39 web.h	. 115
7.38.3.5 sCL_Status	115
7.38.3.4 IsRebootRequired	115
7.38.3.3 boardInfo	115
7.38.3.2 sBoardInfo	115
7.38.3.1 webSocket	114
7.38.3 Variablen-Dokumentation	114
7.38.2.4 website()	113
7.38.2.3 replaceVariable()	112
7.38.2.2 processor()	111
7.38.2.1 server()	111
7.38.2 Dokumentation der Funktionen	111
7.38.1 Ausführliche Beschreibung	110
7.38 src/web.h-Dateireferenz	109
7.37 task.h	109
7.36.1.7 taskJumpTo	109
7.36.1.6 taskStepName	109
7.36.1.5 taskWaitFor	108
7.36.1.4 taskPause	108
7.36.1.3 taskSwitch	108
7.36.1.2 taskEnd	108

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

2 MotorData NMEA2000

1.2 Based on the work of

 ${\tt NMEA2000-Data-Sender} \ \hbox{@AK-Homberger}$

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

For my Engine Yamaha F9.9 without Generator (only Charging coil) is R7 1,1 kOhm.

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:

1.8 Partlist:

• PCB by Aisler Link

Assembly: MD N2k__Assembly.pdf

- 1 C1 10µ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 2 C2 22µ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 3 R1 100k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 4 R2 27k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 5 R3 300R R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 6 R4 10k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 7 R5 1k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 8 R6 4k7 R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 9 R7 2k R Axial DIN0204 L3.6mm D1.6mm P7.62mm Horizontal 1 or 1k1 (Setup for Yamaha F9.9)
- 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

4 MotorData NMEA2000

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

Verzeichnis der Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:														
replace fs	1													

Klassen-Verzeichnis

3.1 Auflistung der Klassen

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

BoardInfo									 														1	13
$t\\Boat\\Data$									 														1	16
Web_Conf	ig								 														2	2(

8 Klassen-Verzeichnis

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

replace_fs.py
data/index.html
data/reboot.html
data/settings.html
data/system.html
data/ueber.html
data/werte.html
src/BoardInfo.cpp
Boardinfo
src/BoardInfo.h
Hardwareinfo from ESP Board
src/BoatData.h
src/configuration.h
Konfiguration für GPIO und Variable
src/helper.h
Hilfsfunktionen
src/hourmeter.h
Betriebstundenzähler
src/LED.h
LED Ansteuerung
src/LEDindicator.h
LED Betriebsanzeige
src/Motordaten.ino
Motordaten NMEA2000
src/NMEA0183Telegram.h
NMEA0183 Telegrame senden
src/task.h
src/web.h
Webseite Variablen lesen und schreiben, Webseiten erstellen

10 Datei-Verzeichnis

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.

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.

6.1.3 Dokumentation der Elementfunktionen

6.1.3.1 ShowChipID()

```
void BoardInfo::ShowChipID ()
```

Definiert in Zeile 47 der Datei BoardInfo.cpp.

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

6.1.3.2 ShowChipInfo()

```
void BoardInfo::ShowChipInfo ()
```

Definiert in Zeile 100 der Datei BoardInfo.cpp.

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

6.1.3.3 ShowChipTemperature()

```
void BoardInfo::ShowChipTemperature ()
```

Definiert in Zeile 119 der Datei BoardInfo.cpp.

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

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

BoardInfo::ShowChipTemperature temprature_sens_read

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];
               sprintf(string1, "ESP32 Chip ID = %04X%08X<br>",(uint16_t)(m_chipid»32),(uint32_t)m_chipid);
00067
               msg = (char*)string1;
msg += "<br/>;
00068
00069
               sprintf(string1, "%d CPU - Kerne<br>WLAN: %s<br>Bluetooth: %s%s",
00070
00071
                   m_chipinfo.cores,
00072
                    (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
                    (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
(m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00073
00074
00075
               msg += (char*)string1;
msg += "<br/>;
00076
00077
               sprintf(string1, "Silicon revision: %d", m_chipinfo.revision);
               msg += (char*)string1;
msg += "<br/>;
00078
00079
               sprintf(string1, "%s Speicher %dMB", (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ?
08000
      "embedded" : "external",
00081
                                                          spi flash get chip size() / (1024 * 1024));
00082
              msg += (char*)string1;
msg += "<br>";
00084
00085
               sprintf(string1, "Freier Speicher: %d bytes", ESP.getFreeHeap());
00086
               msq += (char*)string1;
               msg += "<br>";
00087
00088
               sprintf(string1, "Min freier Speicher: %d bytes", esp_get_minimum_free_heap_size());
               msg += (char*)string1;
msg += "<br/>br>";
00089
00090
00091
00092
           else
00093
00094
               // Fehler beim Lesen der ID....
               msg = "ESP32 Chip ID konnte nicht ausgelesen werden";
00096
00097
           return msg;
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.

```
00019
           TrueHeading=0;
          SOG=0;
00021
          COG=0;
           Variation=7.0;
00022
          GPSTime=0;
          Latitude = 0;
Longitude = 0;
00024
00025
          Altitude=0;
HDOP=100000;
00026
00027
00028
          DGPSAge=100000;
          WaterTemperature = 0;
00029
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 48 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 50 der Datei configuration.h.

6.3.2.2 wAP_SSID

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

Definiert in Zeile 51 der Datei configuration.h.

6.3.2.3 wAP_Password

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

Definiert in Zeile 52 der Datei configuration.h.

6.3.2.4 wMotor_Offset

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

Definiert in Zeile 53 der Datei configuration.h.

6.3.2.5 wCoolant_Offset

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

Definiert in Zeile 54 der Datei configuration.h.

6.3.2.6 wFuellstandmax

```
char Web_Config::wFuellstandmax[6]
```

Definiert in Zeile 55 der Datei configuration.h.

6.3.2.7 wADC1_Cal

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

Definiert in Zeile 56 der Datei configuration.h.

6.3.2.8 wADC2_Cal

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

Definiert in Zeile 57 der Datei configuration.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

· src/configuration.h

Datei-Dokumentation

7.1 data/index.html-Dateireferenz

7.2 index.html

gehe zur Dokumentation dieser Datei

```
00001 <!DOCTYPE html>
00002 <html>
00003 <head>
            <title>Motordaten</title>
            <meta name="viewport" content="width=device-width, initial-scale=1">
00006
            <link rel="shortcut icon" type="image/x-icon" href="favicon.ico";</pre>

00007
80000
00009
00010
00011 </head>
00012 <body>
00013
            <canvas data-type="radial-gauge"</pre>
                     data-width="200"
data-height="200"
00014
00015
                     data-units="U ⁄ min" data-title="Drehzahl"
00016
00017
00018
                     data-min-value="0"
00019
                     data-start-angle="70"
                     data-ticks-angle="220"
00020
00021
                     data-value-box="true"
                     data-max-value="5000"
00022
00023
                     data-major-ticks="0,1000,2000,3000,4000,5000"
00024
                     data-minor-ticks="5"
00025
                     data-stroke-ticks="true"
                data-highlights='[
{"from": 0, "to": 800, "color": "rgba(255, 165, 0, .75)"},
{"from": 800, "to": 3000, "color": "rgba(0, 255, 0, .75)"},
{"from": 3000, "to": 5000, "color": "rgba(255, 50, 50, .75)"}
00026
00027
00028
00029
00031
                     \verb|data-color-plate="#fff"|
00032
                     data-border-shadow-width="0"
00033
                     data-borders="false"
                     data-needle-type="arrow"
00034
00035
                     data-needle-width="4"
00036
                     data-needle-circle-size="7"
00037
                     data-needle-circle-outer="true"
00038
                     data-needle-circle-inner="false"
                     data-animation-duration="1500"
00039
                     data-animation-rule="linear"
00040
                     data-value-text='%sDrehzahl% U ⁄ min'
00041
00042
                     data-value='%sDrehzahl%
00043
           ></canvas>
00044
           00045
00046
                     data-height="200"
00047
00048
                     data-units="°C"
                     data-title="Oil Temperatur"
```

24 Datei-Dokumentation

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

```
00137
                  data-value='%sBordspannung%'
00138
00139
00140
          <canvas data-type="radial-gauge"</pre>
                  data-width="300"
00141
                  data-height="300"
00142
                  data-units="%"
00143
00144
                  data-title="Fü llstand"
00145
                  data-min-value="0"
                  data-start-angle="70"
00146
                  data-ticks-angle="220"
00147
                  data-value-box="true"
00148
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"
              data-highlights='[
{"from": 0, "to": 10, "color": "rgba(255, 50, 50, .75)"},
{"from": 10, "to": 20, "color": "rgba(255, 165, 0, .75)"},
00153
00154
00155
              {"from": 20, "to": 100, "color": "rgba(0, 255, 0, .75)"}
00157
00158
                  data-color-plate="#fff"
                  \verb|data-border-shadow-width="0"|
00159
                  data-borders="false"
00160
00161
                  data-needle-type="arrow
                  data-needle-width="4"
00162
00163
                  data-needle-circle-size="7"
00164
                  data-needle-circle-outer="true"
                  data-needle-circle-inner="false"
00165
                  data-animation-duration="1500"
00166
                  data-animation-rule="linear"
00167
00168
                  data-value-text='%sFuellstand% %'
00169
                  data-value='%sFuellstand%'
00170
         ></canvas>

     <a class="active" href="/">Home</a>
00171
00172
              <a href="werte.html">Werte</a>
00173
              <a href="settings.html">Setting</a>
00175
              <a href="system.html">System</a>
00176
              <a href="ueber.html">About</a>
00177
         </111>
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>
       <meta charset="UTF-8">
00005
       <link rel="stylesheet" type="text/css" href="style.css">
00006 </head>
00007 <body>
80000
       <h1>
       Wartezeit für Reboot, WiFi und Webserver Initialisierung<br/>br>Aufruf der home page in <span
00009
     id="countdown">15</span> Sekunden...
00010
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 {
           document.getElementById("countdown").innerHTML = seconds;
00018
00019
           window.setTimeout("countdown()", 1000);
00020
00021
00022
       countdown();
00023 </script>
00024 
         <a href="/">Home</a>
00025
         <a href="werte.html">Werte</a>
00026
00027
         <a class="active" href="settings.html">Settings</a>
         <a href="system.html">System</a>
```

26 Datei-Dokumentation

7.5 data/settings.html-Dateireferenz

7.6 settings.html

gehe zur Dokumentation dieser Datei

```
00001 <!DOCTYPE html>
00003 <head>
00004
         <title>Settings</title>
         cmeta name="viewport" content="width=device-width, initial-scale=1">
clink rel="icon" href="data:,">
<link rel="stylesheet" type="text/css" href="style.css">
00005
00006
00007
00008 </head>
00009 <body>
00010
           Kü hlwassertemperatur: %sCoolantTemp% ° C<br>
00011
00012
                         Offset: %sCoolantOffset% °C<br>
                         Motortemperatur: %sMotorTemp% °C<br>
00013
00014
                         Offset: %sMotorOffset% °C
00015
         %CONFIGPLACEHOLDER%
00016
          <script>
00017
             function formToJson(form) {
00018
                     /*alert("formToJson wird aufgerufen!");*/
00019
                 var xhr = new XMLHttpRequest();
                 var jsonFormInfo = JSON.stringify({
00020
00021
                     wAP_SSID: form.SSID.value,
00022
                     wAP_IP: form.IP.value,
00023
                     wAP_Password: form.Password.value,
00024
                     wMotor_Offset: form.MotorOffset.value,
                     wCoolant_Offset: form.CoolantOffset.value,
00025
00026
                     wFuellstandmax: form.Fuellstandmax.value,
00027
                     wADC1_Cal: form.ADC1_Cal.value,
00028
                     wADC2_Cal: form.ADC2_Cal.value
00029
             });
00030
                 xhr.open("POST", "/settings.html", true);
00031
                 xhr.setRequestHeader("Content-Type", "application/json");
00032
00033
                 xhr.send(jsonFormInfo);
00034
                 window.alert("Gespeichert!");
00035
00036
         </script>
00037
00038
         Nach Änderungen neu starten!
00040
         <button class="button" onclick="reboot_handler()">Neustart</button>
00041
00042
         00043
         <script>
00044
             function reboot handler()
00045
00046
             document.getElementById("status").innerHTML = "Starte Reboot ...";
00047
             var xhr = new XMLHttpRequest();
             xhr.open("GET", "/reboot", true);
00048
00049
             xhr.send();
00050
             setTimeout(function() { window.open("/reboot", "_self"); }, 500);
00051
00052
         </script>
00053
00054
         00055
             <a href="/">Home</a>
00056
             <a href="werte.html">Werte</a>
00057
             <a class="active" href="settings.html">Settings</a>
00059
             <a href="system.html">System</a>
00060
             <a href="ueber.html">About</a>
00061
         </111>
00062 </body>
00063 </html >
```

7.7 data/system.html-Dateireferenz

7.8 system.html

gehe zur Dokumentation dieser Datei

```
00001 <HTML>
00002 <HEAD>
        <TITLE>Systeminfo</TITLE>
00003
        <meta name="viewport" content="width=device-width, initial-scale=1">
00004
        <
00005
00006
00007 </HEAD>
00008 <BODY>
00009
        <br />
        Eigene IP-Adresse - AP: %sAP_IP%<br />
00010
00011
                     Clients am AP: %sAP_Clients%
00012
        OneWire %sOneWire_Status% Sensoren gefunden
00013
        Informationen zum ESP32 - Board:<br />%sBoardInfo%
        LittleFS, benutzte Bytes: %sFS_USpace% <br />
00014
00015
                      LittleFS, gesamte Bytes: %sFS_TSpace% 
        Free Heapspace: %sHeapspace%
00016
00017
        <br />
00018
        <br />
00019
        00020
           <a href="/">Home</a>
           <a href="werte.html">Werte</a>
<a href="settings.html">Setting</a>
<a class="active" href="system.html">System</a>
00021
00022
00023
           <a href="ueber.html">About</a>
00025
        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>
        <meta name="viewport" content="width=device-width, initial-scale=1">
k rel="icon" href="data:,">
00004
00005
        rel="stylesheet" type="text/css" href="style.css">
00006
00007 </HEAD>
00008 <BODY>
00009
        %sVersionS%
00010
            <br />
        Autor: Gerry Sebb</br>
00011
        <a href="mailto: gerry@sebb.de">gerry@sebb.de</a>
00012
00013
           <br />
            <img src="/logo80.jpg" alt="Open-Boats Logo">
00015
00016
        00017
           <a href="/">Home</a>
            <a href="werte.html">Werte</a>
00018
           <a href="settings.html">Setting</a>
00019
           <a href="system.html">System</a>
00020
00021
            <a class="active" href="ueber.html">About</a>
00022
        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>
        <meta name="viewport" content="width=device-width, initial-scale=1">
k rel="shortcut icon" type="image/x-icon" href="favicon.ico">
k rel="icon" href="data:,">
k rel="stylesheet" type="text/css" href="style.css">
00005
00006
00007
00008
00009
         <script src='gauge.min.js'></script>
         <meta http-equiv="refresh" content="5">
00010
00011 </head>
00012 <body>
00013
         DC Status
         Bordspannung: %sBordspannung% V
00014
         Maschine
00015
00016
         Kühlwasser Temperatur: %sCoolantTemp% °C</br>
00017
                         Offset: %sCoolantOffset% °C</br>
00018
                         Fehler: %sCoolantError%
00019
                        Motor Temperatur: %sMotorTemp% °C</br>
00020
00021
                        Offset: %sMotorOffset% °</br>
                        Fehler: %sMotorError%
00023
00024
         Motor Drehzahl: %sDrehzahl% U ⁄ min
         Maschinenstunden: %sCounter% h
00025
         cp>Tank
Tank
Tank Fü llstand: %sFuellstand% %</br>
max. F&uuml; llstand:
00026
00027
     %sFuellstandmax% l
00028
         ADC Kalibrierung
00029
         ADC1: %sADC1_Cal%</br>ADC2: %sADC2_Cal%
00030
        00031
            <a href=""">Home</a>
<a class="active" href="/">Werte</a>

00032
00033
             <a href="settings.html">Setting</a>
00035
            <a href="system.html">System</a>
00036
             <a href="ueber.html">About</a>
        00037
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

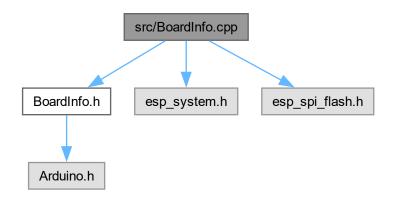
gehe zur Dokumentation dieser Datei

```
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 {
          // Konstruktor der Klasse
00038
00039
          // ChipID auslesen
          //The chip ID is essentially its MAC address(length: 6 bytes).
00040
00041
          m_chipid = 0;
          m_chipid = ESP.getEfuseMac(); //The chip ID is essentially its MAC address(length: 6 bytes).
00042
00043
          // Chip - Info auslesen
00044
          esp_chip_info(&m_chipinfo);
00046
00047 void BoardInfo::ShowChipID()
00048 {
```

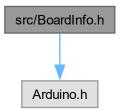
7.17 BoardInfo.cpp 31

```
00049
            if (m_chipid != 0)
00050
                Serial.printf("ESP32 Chip ID = %04X", (uint16_t) (m_chipid>32));
                                                                                                  //print High 2 bytes
00051
                Serial.printf("%08X\n", (uint32_t)m_chipid);
00052
                                                                                                    //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];
sprintf(string1, "ESP32 Chip ID = %04X%08X<br>", (uint16_t) (m_chipid»32), (uint32_t)m_chipid);
00067
00068
                msg = (char*)string1;
00069
                msg += "<br>";
                sprintf(string1, "%d CPU - Kerne<br/>br>WLAN: %s<br/>br>Bluetooth: %s%s",
00070
00071
                   m_chipinfo.cores,
                    (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
(m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
(m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00072
00073
00074
00075
                msg += (char*)string1;
                msg += "<br>";
00076
                sprintf(string1, "Silicon revision: %d", m_chipinfo.revision);
00077
               msg += (char*)string1;
msg += "<br/>;
00078
00079
08000
                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>";
                sprintf(string1, "Freier Speicher: %d bytes", ESP.getFreeHeap());
00085
                msg += (char*)string1;
msg += "<br/>';
00086
00087
00088
                sprintf(string1, "Min freier Speicher: %d bytes", esp_get_minimum_free_heap_size());
               msg += (char*)string1;
msg += "<br/>;
00089
00090
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 {
            // Infos zum Board
00102
           Serial.printf("Das ist ein Chip mit %d CPU - Kernen\nWLAN: %s\nBluetooth: %s%s\n",
00103
00104
                    m_chipinfo.cores,
00105
                     (m_chipinfo.features & CHIP_FEATURE_WIFI_BGN) ? "2.4GHz" : "nicht vorhanden",
                     (m_chipinfo.features & CHIP_FEATURE_BT) ? "/BT" : "",
(m_chipinfo.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
00106
00107
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),
                     (m_chipinfo.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");
00112
00113
00114
           Serial.printf("(Freier Speicher: %d bytes)\n", esp_get_free_heap_size());
Serial.printf("Freier Speicher: %d bytes\n", ESP.getFreeHeap());
00115
           Serial.printf("Minimum freier Speicher: %d bytes\n", esp_qet_minimum_free_heap_size());
00116
00117 }
00118
00119 void BoardInfo::ShowChipTemperature()
00120 {
00121
           uint8_t temp_farenheit;
00122
           float temp celsius;
00123
           temp_farenheit = temprature_sens_read();
00124
            if (128 == temp_farenheit)
00125
00126
                Serial.println("Kein Temperatur - Sensor vorhanden.");
00127
00128
           temp_celsius = ( temp_farenheit - 32 ) / 1.8;
Serial.printf("Temperatur Board: %i Fahrenheit\n", temp_farenheit);
Serial.printf("Temperatur Board: %.1f °C\n", temp_celsius);
00130
00131
00132 }
```

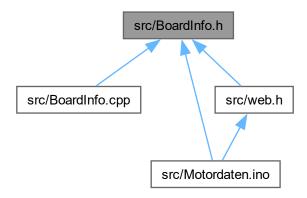
7.18 src/BoardInfo.h-Dateireferenz

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.

7.19 BoardInfo.h

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
00010 Class B
00019
          BoardInfo();
00020
          void ShowChipID();
void ShowChipInfo();
void ShowChipTemperature();
00021
00022
00023
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

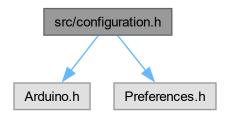
gehe zur Dokumentation dieser Datei

```
00001 #ifndef _BoatData_H_
00002 #define _BoatData_H_
00003
00004 struct tBoatData {
00005
                                              // Days since 1970-01-01
        unsigned long DaysSince1970;
00006
         double TrueHeading, SOG, COG, Variation,
80000
                  GPSTime, // Secs since midnight,
                 Latitude, Longitude, Altitude, HDOP, GeoidalSeparation, DGPSAge, WaterTemperature, WaterDepth, Offset, WindDirectionT, WindDirectionM, WindSpeedK, WindSpeedM,
00009
00010
00011
00012
                  WindAngle;
00013
         int GPSQualityIndicator, SatelliteCount, DGPSReferenceStationID;
00014
         bool MOBActivated;
00015
         char Status;
00016
00017 public:
         tBoatData() {
00018
00019
           TrueHeading=0;
00020
            SOG=0;
00021
           COG=0;
00022
           Variation=7.0;
           GPSTime=0;
00023
           Latitude = 0;
Longitude = 0;
00024
00025
00026
            Altitude=0;
00027
           HDOP=100000;
00028
           DGPSAge=100000;
00029
           WaterTemperature = 0;
DaysSince1970=0;
00030
00031
           MOBActivated=false;
00032
           SatelliteCount=0;
00033
           DGPSReferenceStationID=0;
00034
        };
00035 };
00036
00037 #endif // _BoatData_H_
```

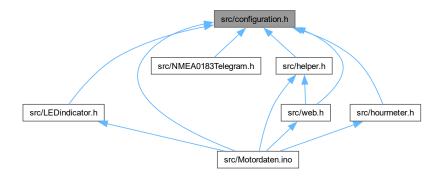
7.22 src/configuration.h-Dateireferenz

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 VersionSoftware "2.8.1.0 (2025-07-08)"
- #define VersionHardware "2.3.0.0 (2024-11-30)"
- #define ESP32_CAN_TX_PIN GPIO_NUM_4
 Config NMEA2000.
- #define ESP32_CAN_RX_PIN GPIO_NUM_5
- #define N2K_SOURCE 15
- #define EngineSendOffset 0
- #define TankSendOffset 40
- #define RPMSendOffset 80
- #define BatteryDCSendOffset 120
- #define BatteryDCStatusSendOffset 160
- #define SlowDataUpdatePeriod 1000
- #define PAGE REFRESH 10
- #define WEB TITEL "Motordaten"
- #define HostName "Motordaten"
- #define CL_SSID "NoWa"
- #define CL_PASSWORD "12345678"
- #define I2C_SDA 21
- #define I2C SCL 22
- #define SEALEVELPRESSURE_HPA (1013.25)
- #define RPM Calibration Value 7.0
- #define Eingine_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 sI2C_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 VersionSoftware

```
#define VersionSoftware "2.8.1.0 (2025-07-08)"
```

Definiert in Zeile 19 der Datei configuration.h.

7.22.2.2 VersionHardware

```
#define VersionHardware "2.3.0.0 (2024-11-30)"
```

Definiert in Zeile 20 der Datei configuration.h.

7.22.2.3 ESP32_CAN_TX_PIN

#define ESP32_CAN_TX_PIN GPIO_NUM_4

Config NMEA2000.

Definiert in Zeile 26 der Datei configuration.h.

7.22.2.4 ESP32_CAN_RX_PIN

```
#define ESP32_CAN_RX_PIN GPIO_NUM_5
```

Definiert in Zeile 27 der Datei configuration.h.

7.22.2.5 N2K_SOURCE

#define N2K_SOURCE 15

Definiert in Zeile 28 der Datei configuration.h.

7.22.2.6 EngineSendOffset

#define EngineSendOffset 0

Definiert in Zeile 34 der Datei configuration.h.

7.22.2.7 TankSendOffset

#define TankSendOffset 40

Definiert in Zeile 35 der Datei configuration.h.

7.22.2.8 RPMSendOffset

#define RPMSendOffset 80

Definiert in Zeile 36 der Datei configuration.h.

7.22.2.9 BatteryDCSendOffset

#define BatteryDCSendOffset 120

Definiert in Zeile 37 der Datei configuration.h.

7.22.2.10 BatteryDCStatusSendOffset

#define BatteryDCStatusSendOffset 160

Definiert in Zeile 38 der Datei configuration.h.

7.22.2.11 SlowDataUpdatePeriod

#define SlowDataUpdatePeriod 1000

Definiert in Zeile 39 der Datei configuration.h.

7.22.2.12 PAGE_REFRESH

```
#define PAGE_REFRESH 10
```

Definiert in Zeile 43 der Datei configuration.h.

7.22.2.13 WEB_TITEL

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

Definiert in Zeile 44 der Datei configuration.h.

7.22.2.14 HostName

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

Definiert in Zeile 62 der Datei configuration.h.

7.22.2.15 CL_SSID

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

Definiert in Zeile 79 der Datei configuration.h.

7.22.2.16 CL_PASSWORD

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

Definiert in Zeile 80 der Datei configuration.h.

7.22.2.17 I2C_SDA

#define I2C_SDA 21

Definiert in Zeile 90 der Datei configuration.h.

7.22.2.18 I2C_SCL

```
#define I2C_SCL 22
```

Definiert in Zeile 91 der Datei configuration.h.

7.22.2.19 SEALEVELPRESSURE_HPA

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

Definiert in Zeile 92 der Datei configuration.h.

7.22.2.20 RPM_Calibration_Value

```
#define RPM_Calibration_Value 7.0
```

Definiert in Zeile 115 der Datei configuration.h.

7.22.2.21 Eingine_RPM_Pin

```
#define Eingine_RPM_Pin 19
```

Definiert in Zeile 116 der Datei configuration.h.

7.22.2.22 ONE_WIRE_BUS

```
#define ONE_WIRE_BUS 14
```

Definiert in Zeile 125 der Datei configuration.h.

7.22.2.23 SERVER_HOST_NAME

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

Definiert in Zeile 148 der Datei configuration.h.

7.22.2.24 TCP PORT

```
#define TCP_PORT 6666
```

Definiert in Zeile 149 der Datei configuration.h.

7.22.2.25 DNS PORT

```
#define DNS_PORT 53
```

Definiert in Zeile 150 der Datei configuration.h.

7.22.3 Dokumentation der Aufzählungstypen

7.22.3.1 EngineStatus

enum EngineStatus

Aufzählungswerte

Off	
On	

Definiert in Zeile 114 der Datei configuration.h.

```
00114 { Off = 0, On = 1, };
```

7.22.4 Variablen-Dokumentation

7.22.4.1 NodeAddress

int NodeAddress

Definiert in Zeile 29 der Datei configuration.h.

7.22.4.2 preferences

Preferences preferences

Definiert in Zeile 30 der Datei configuration.h.

7.22.4.3 chipid

uint8_t chipid[6]

Definiert in Zeile 31 der Datei configuration.h.

7.22.4.4 id

```
uint32\_t id = 0
```

Definiert in Zeile 32 der Datei configuration.h.

7.22.4.5 i

int i = 0

Definiert in Zeile 33 der Datei configuration.h.

7.22.4.6 sHeapspace

String sHeapspace = ""

Definiert in Zeile 45 der Datei configuration.h.

7.22.4.7 tAP_Config

```
Web_Config tAP_Config
```

Definiert in Zeile 59 der Datei configuration.h.

7.22.4.8 channel

```
const int channel = 10
```

Definiert in Zeile 63 der Datei configuration.h.

7.22.4.9 hide_SSID

```
const bool hide_SSID = false
```

Definiert in Zeile 64 der Datei configuration.h.

7.22.4.10 max_connection

```
const int max\_connection = 2
```

Definiert in Zeile 65 der Datei configuration.h.

7.22.4.11 IP

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

Definiert in Zeile 68 der Datei configuration.h.

7.22.4.12 Gateway

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

Definiert in Zeile 69 der Datei configuration.h.

7.22.4.13 NMask

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

Definiert in Zeile 70 der Datei configuration.h.

7.22.4.14 AP_SSID

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

Definiert in Zeile 71 der Datei configuration.h.

7.22.4.15 AP_PASSWORD

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

Definiert in Zeile 72 der Datei configuration.h.

7.22.4.16 AP_IP

```
IPAddress AP_IP
```

Definiert in Zeile 73 der Datei configuration.h.

7.22.4.17 CL_IP

```
IPAddress CL_IP
```

Definiert in Zeile 74 der Datei configuration.h.

7.22.4.18 SELF_IP

```
IPAddress SELF_IP
```

Definiert in Zeile 75 der Datei configuration.h.

7.22.4.19 sAP_Station

```
String sAP_Station = ""
```

Definiert in Zeile 76 der Datei configuration.h.

7.22.4.20 iSTA_on

```
int iSTA_on = 0
```

Definiert in Zeile 81 der Datei configuration.h.

7.22.4.21 bConnect_CL

```
int bConnect_CL = 0
```

Definiert in Zeile 82 der Datei configuration.h.

7.22.4.22 bClientConnected

```
bool bClientConnected = 0
```

Definiert in Zeile 83 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 86 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 87 der Datei configuration.h.

7.22.4.25 fbmp_temperature

```
float fbmp_temperature = 0
```

Definiert in Zeile 93 der Datei configuration.h.

7.22.4.26 fbmp_pressure

```
float fbmp\_pressure = 0
```

Definiert in Zeile 94 der Datei configuration.h.

7.22.4.27 fbmp_altitude

```
float fbmp_altitude = 0
```

Definiert in Zeile 95 der Datei configuration.h.

7.22.4.28 sl2C_Status

```
String sI2C_Status = ""
```

Definiert in Zeile 96 der Datei configuration.h.

7.22.4.29 bl2C_Status

```
bool bI2C_Status = 0
```

Definiert in Zeile 97 der Datei configuration.h.

7.22.4.30 iMaxSonar

```
const int iMaxSonar = 35
```

Definiert in Zeile 100 der Datei configuration.h.

7.22.4.31 iDistance

```
int iDistance = 0
```

Definiert in Zeile 101 der Datei configuration.h.

7.22.4.32 FuelLevel

```
float FuelLevel = 0
```

Definiert in Zeile 104 der Datei configuration.h.

7.22.4.33 FuelLevelMax

```
float FuelLevelMax = 30
```

Definiert in Zeile 105 der Datei configuration.h.

7.22.4.34 CoolantTemp

```
float CoolantTemp = 0
```

Definiert in Zeile 106 der Datei configuration.h.

7.22.4.35 MotorTemp

```
float MotorTemp = 0
```

Definiert in Zeile 107 der Datei configuration.h.

7.22.4.36 EngineRPM

```
float EngineRPM = 0
```

Definiert in Zeile 108 der Datei configuration.h.

7.22.4.37 BordSpannung

```
float BordSpannung = 0
```

Definiert in Zeile 109 der Datei configuration.h.

7.22.4.38 EngineOn

```
bool EngineOn = false
```

Definiert in Zeile 110 der Datei configuration.h.

7.22.4.39 motorErrorReported

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

Definiert in Zeile 111 der Datei configuration.h.

7.22.4.40 coolantErrorReported

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

Definiert in Zeile 112 der Datei configuration.h.

7.22.4.41 Counter

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

Definiert in Zeile 113 der Datei configuration.h.

7.22.4.42 Bat1Capacity

```
int Bat1Capacity = 55
```

Definiert in Zeile 119 der Datei configuration.h.

7.22.4.43 Bat2Capacity

```
int Bat2Capacity = 90
```

Definiert in Zeile 120 der Datei configuration.h.

7.22.4.44 SoCError

```
int SoCError = 0
```

Definiert in Zeile 121 der Datei configuration.h.

7.22.4.45 BatSoC

```
float BatSoC = 0
```

Definiert in Zeile 122 der Datei configuration.h.

7.22.4.46 sOneWire_Status

```
String sOneWire_Status = ""
```

Definiert in Zeile 126 der Datei configuration.h.

7.22.4.47 fDrehzahl

```
float fDrehzahl = 0
```

Definiert in Zeile 129 der Datei configuration.h.

7.22.4.48 fGaugeDrehzahl

```
float fGaugeDrehzahl = 0
```

Definiert in Zeile 130 der Datei configuration.h.

7.22.4.49 fBordSpannung

```
float fBordSpannung = 0
```

Definiert in Zeile 131 der Datei configuration.h.

7.22.4.50 fCoolantTemp

```
float fCoolantTemp = 0
```

Definiert in Zeile 132 der Datei configuration.h.

7.22.4.51 fMotorTemp

```
float fMotorTemp = 0
```

Definiert in Zeile 133 der Datei configuration.h.

7.22.4.52 fCoolantOffset

```
float fCoolantOffset = 0
```

Definiert in Zeile 134 der Datei configuration.h.

7.22.4.53 fMotorOffset

```
float fMotorOffset = 0
```

Definiert in Zeile 135 der Datei configuration.h.

7.22.4.54 sSTBB

```
String sSTBB = ""
```

Definiert in Zeile 136 der Datei configuration.h.

7.22.4.55 sOrient

```
String sOrient = ""
```

Definiert in Zeile 137 der Datei configuration.h.

7.22.4.56 dMWV_WindDirectionT

```
double dMWV_WindDirectionT = 0
```

Definiert in Zeile 140 der Datei configuration.h.

7.22.4.57 dMWV_WindSpeedM

```
double dMWV_WindSpeedM = 0
```

Definiert in Zeile 141 der Datei configuration.h.

7.22.4.58 dVWR_WindDirectionM

```
double dVWR_WindDirectionM = 0
```

Definiert in Zeile 142 der Datei configuration.h.

7.22.4.59 dVWR_WindAngle

```
double dVWR_WindAngle = 0
```

Definiert in Zeile 143 der Datei configuration.h.

7.22.4.60 dVWR_WindSpeedkn

```
double dVWR\_WindSpeedkn = 0
```

Definiert in Zeile 144 der Datei configuration.h.

7.22.4.61 dVWR_WindSpeedms

```
double dVWR_WindSpeedms = 0
```

Definiert in Zeile 145 der Datei configuration.h.

7.23 configuration.h 49

7.22.4.62 udpAddress

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

Definiert in Zeile 153 der Datei configuration.h.

7.22.4.63 udpPort

```
const int udpPort = 4444
```

Definiert in Zeile 154 der Datei configuration.h.

7.23 configuration.h

gehe zur Dokumentation dieser Datei

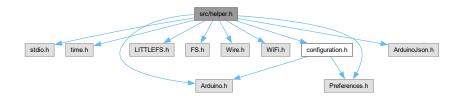
```
00001 #ifndef __configuration__H_
00002 #define __configuration__H_
00003
00014
00015 #include <Arduino.h>
00016 #include <Preferences.h>
00018 // Versionierung
00019 #define VersionSoftware "2.8.1.0 (2025-07-08)" // Version Software 00020 #define VersionHardware "2.3.0.0 (2024-11-30)" // Version HArdware
00021
00026 #define ESP32_CAN_TX_PIN GPIO_NUM_4 // Set CAN TX port to 4 00027 #define ESP32_CAN_RX_PIN GPIO_NUM_5 // Set CAN RX port to 5
00028 #define N2K_SOURCE 15
00029 int NodeAddress;
                                                    // To store Last Node Address
                                                   // Nonvolatile storage on ESP32 - To store LastDeviceAddress
00030 Preferences preferences;
00031 uint8_t chipid[6];
00032 uint32_t id = 0;
00033 int i = 0;
00034 #define EngineSendOffset 0
00035 #define TankSendOffset 40
00036 #define RPMSendOffset 80
00037 #define BatteryDCSendOffset 120
00038 #define BatteryDCStatusSendOffset 160
00039 #define SlowDataUpdatePeriod 1000 // Time between CAN Messages sent
00040
00041
00042 //Configuration Website
00043 #define PAGE_REFRESH 10 // x Sec. 00044 #define WEB_TITEL "Motordaten"
00045 String sHeapspace = "";
00046
00047 //Configuration mit Webinterface
00048 struct Web_Config
00049 {
00050
            char wAP_IP[20];
           char wAP_SSID[64];
00051
00052
           char wAP_Password[12];
00053
           char wMotor_Offset[6];
00054
           char wCoolant_Offset[6];
00055
           char wFuellstandmax[6];
           char wADC1_Cal[6];
00056
00057
           char wADC2_Cal[6];
00058 };
00059 Web_Config tAP_Config;
00060
00061 //Configuration AP
                                    "Motordaten"
00062 #define HostName
00063 const int channel
00064 const bool hide_SSID
                                                                 // WiFi Channel number between 1 and 13
                                     = 10;
                                       = false;
                                                                 // To disable SSID broadcast -> SSID will not appear
      in a basic WiFi scan
00065 const int max_connection = 2;
                                                                  // Maximum simultaneous connected clients on the AP
00066
00067 // Variables for WIFI-AP
00068 IPAddress IP = IPAddress(192, 168, 15, 30);
00069 IPAddress Gateway = IPAddress(192, 168, 15, 30);
00070 IPAddress NMask = IPAddress(255, 255, 255, 0);
```

```
00071 const char* AP_SSID = "Motordaten";
00072 const char* AP_PASSWORD = "12345678";
00073 IPAddress AP_IP;
00074 IPAddress CL IP;
00075 IPAddress SELF_IP;
00076 String sAP_Station = "";
00078 //Configuration Client (Network Data Windsensor)
00079 #define CL_SSID "NoWa" //Windmesser
00080 #define CL_PASSWORD "12345678"
                                                           // Status STA-Mode
00081 int iSTA_on = 0;
00082 int bConnect_CL = 0;
00083 bool bClientConnected = 0;
00084
00085 // Calibration data variable definition for ADC1 and ADC2 Input
00086 double ADC_Calibration_Value1 = 170.0;
00087 double ADC_Calibration_Value2 = 19.0;
00088
00089 //Confuration Sensors I2C
00090 #define I2C_SDA 21
                                                        //Standard 21
00091 #define I2C_SCL 22
                                                        //Standard 22
00092 #define SEALEVELPRESSURE_HPA (1013.25) //1013.25
00093 float fbmp_temperature = 0;
00094 float fbmp_pressure = 0;
00095 float fbmp_altitude = 0;
00096 String sI2C_Status = "";
00097 bool bI2C_Status = 0;
00008
00099 // Global Data Sonar
00100 const int iMaxSonar = 35;
                                                //Analoginput
00101 int iDistance = 0;
00103 // Global Data Motordata Sensor
00104 float FuelLevel = 0;
00105 float FuelLevelMax = 30;
00106 float CoolantTemp = 0;
00107 float MotorTemp = 0;
00108 float EngineRPM = 0;
00109 float BordSpannung = 0;
00110 bool EngineOn = false;
00111 String motorErrorReported = "Aus";
00112 String coolantErrorReported = "Aus";
00113 static unsigned long Counter;  // Enginehours
00114 enum EngineStatus { Off = 0, On = 1, };
00115 #define RPM_Calibration_Value 7.0 // Translates Generator RPM to Engine RPM
                                                        // Enginehours
00116 #define Eingine_RPM_Pin 19 \, // Engine RPM is measured as interrupt on GPIO 23
00117
00118 // Global Data Battery
00119 int BatlCapacity = 55; // Starterbatterie
00120 int Bat2Capacity = 90; // Versorgerbatterie
00121 int SoCError = 0;
00122 float BatSoC = 0;
00123
00124 // Data wire for teperature (Dallas DS18B20) 00125 #define ONE_WIRE_BUS 14 // Data wire
UU125 #define ONE_WIRE_BUS 14 // Data wire for teperature (Dallas DS18B20) is plugged into GPIO 13 00126 String sOneWire_Status = "";
00128 // Variables Website
00129 float fDrehzahl = 0;
00130 float fGaugeDrehzahl = 0;
00131 float fBordSpannung = 0;
00132 float fCoolantTemp = 0;
00133 float fMotorTemp = 0;
00134 float fCoolantOffset = 0;
00135 float fMotorOffset = 0;
00136 String sSTBB = "";
00137 String sOrient = "";
00138
00139 //Definiton NMEA0183 MWV
00140 double dMWV_WindDirectionT = 0;
00141 double dMWV_WindSpeedM = 0;
00142 double dVWR_WindDirectionM = 0;
00143 double dVWR_WindAngle = 0;
00144 double dVWR_WindSpeedkn = 0;
00145 double dVWR_WindSpeedms = 0;
00146
00147 //Configuration NMEA0183
00148 #define SERVER_HOST_NAME "192.168.30.15"
00149 #define TCP_PORT 6666
                                                                 //"192.168.76.34"
                                                           //6666
00150 #define DNS PORT 53
00151
00152 //Variable NMEA 0183 Stream
00153 const char *udpAddress = "192.168.30.255"; // Set network address for broadcast 00154 const int udpPort = 4444; // UDP port
00154 const int udpPort = 4444;
00155
00156 #endif
```

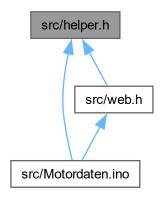
7.24 src/helper.h-Dateireferenz

Hilfsfunktionen.

```
#include <stdio.h>
#include <time.h>
#include <Arduino.h>
#include <LITTLEFS.h>
#include <FS.h>
#include <Wire.h>
#include <WiFi.h>
#include "configuration.h"
#include <ArduinoJson.h>
#include <Preferences.h>
Include-Abhängigkeitsdiagramm für helper.h:
```



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



Funktionen

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

LittleFS, Dateien auflisten.

• void readConfig (String filename)

Konfiguration aus Json-Datei lesen.

• bool writeConfig (String json)

Webseiten Eingabe in Json-Datei schreiben.

- void 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.

7.24.2.2 freeHeapSpace()

```
void freeHeapSpace ()
```

Freie Speichergroesse aller 5s lesen

Definiert in Zeile 37 der Datei helper.h.

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

```
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



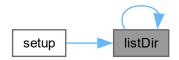
Definiert in Zeile 92 der Datei helper.h.

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

Konfiguration aus Json-Datei lesen.

Parameter

filename

Definiert in Zeile 129 der Datei helper.h.

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

```
strcpy(tAP_Config.wAP_IP, testDocument["wAP_IP"] | "192.168.15.30");
                         strcpy(tAP_Config.wAP_IP, testDocument["wAP_IP"] | "192.168.15.30");
strcpy(tAP_Config.wAP_Password, testDocument["wAP_Password"] | "12345678");
strcpy(tAP_Config.wMotor_Offset, testDocument["wMotorOffset"] | "0.0");
strcpy(tAP_Config.wCoolant_Offset, testDocument["wCoolantOffset"] | "0.0");
strcpy(tAP_Config.wFuellstandmax, testDocument["wFuellstandmax"] | "0.0");
strcpy(tAP_Config.wADC1_Cal, testDocument["wADC1_Cal"] | "0.0");
strcpy(tAP_Config.wADC2_Cal, testDocument["wADC2_Cal"] | "0.0");
00150
00151
00152
00153
00154
00155
00156
00157
                               configFile.close();
                              Serial.println("Config - File closed");
00158
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()

Webseiten Eingabe in Json-Datei schreiben.

Parameter



Rückgabe

true

false

Definiert in Zeile 175 der Datei helper.h.

```
00176 {
00177
          Serial.println(json);
00178
          Serial.println("neue Konfiguration speichern");
00179
00180
          // Datei zum Schreiben öffnen (überschreibt alte Datei)
          File configFile = LittleFS.open("/config.json", "w");
if (!configFile) {
00181
00182
              Serial.println("Config - Datei konnte nicht geöffnet werden!");
00183
00184
              return false;
00185
00186
00187
          // JSON-Dokument parsen
00188
          JsonDocument testDocument;
00189
          DeserializationError error = deserializeJson(testDocument, json);
00190
          if (error) {
00191
              Serial.print(F("deserializeJson() failed: "));
00192
              Serial.println(error.f_str());
```

```
00193
              configFile.close();
00194
              return false;
00195
00196
         // JSON in Datei schreiben
00197
00198
          serializeJson(testDocument, configFile);
00199
          Serial.println("Konfiguration geschrieben...");
00200
00201
          // Kontrolle
00202
          serializeJsonPretty(testDocument, Serial);
00203
00204
          configFile.close();
          Serial.println("Config - Datei geschlossen");
00205
00206
          return true;
00207 }
```

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



7.24.2.7 I2C_scan()

```
void I2C_scan (
     void )
```

I2C Bus auslesen, alle Geräte mit Adresse ausgegeben

Definiert in Zeile 212 der Datei helper.h.

```
00212
00213
        byte error, address;
00214
        int nDevices;
00215
        Serial.println("Scanning...");
00216
00217
        for(address = 1; address < 127; address++ )</pre>
00218
00219
          Wire.beginTransmission(address);
          error = Wire.endTransmission();
if (error == 0)
00220
00221
00222
00223
            Serial.print("I2C device found at address 0x");
00224
            if (address<16)
00225
            {
00226
              Serial.print("0");
00227
00228
            Serial.println(address, HEX);
00229
            nDevices++;
00230
00231
          else if (error==4)
00232
00233
             Serial.print("Unknow error at address 0x");
00234
             if (address<16)
00235
00236
               Serial.print("0");
00237
00238
            Serial.println(address, HEX);
00239
            nDevices++;
00240
00241
          else if (error==4) {
            Serial.print("Unknow error at address 0x");
00242
             if (address<16) {
   Serial.print("0");</pre>
00243
00244
00245
00246
            Serial.println(address, HEX);
00247
```

7.24.2.8 sWifiStatus()

```
String sWifiStatus (
int Status)
```

WIFI Status lesen.

Parameter

Status

Rückgabe

String

Definiert in Zeile 264 der Datei helper.h.

```
00265 {
00266
              switch(Status){
00267
                case WL_IDLE_STATUS:return "Warten";
                case WL_NO_SSID_AVAIL:return "Weine SSID vorhanden";
case WL_SCAN_COMPLETED:return "Scan komlett";
case WL_CONNECTED:return "Verbunden";
case WL_CONNECT_FAILED:return "Verbindung fehlerhaft";
case WL_CONNECTION_LOST:return "Verbindung verloren";
00268
00269
00270
00271
00272
00273
                 case WL_DISCONNECTED:return "Nicht verbunden";
00274
                 default:return "unbekannt";
00275
00276 }
```

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



7.24.2.9 toChar()

Convert string to char.

7.25 helper.h 59

Parameter

command

Rückgabe

char*

Definiert in Zeile 285 der Datei helper.h.

```
00285
00286     if(command.length()!=0) {
          char *p = const_cast<char*>(command.c_str());
          return p;
00289     }
00290     else{
          return 0;
00292     }
00293 }
```

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);
            struct tm tm_now;
00029
00030
            localtime_r(&now, &tm_now);
00031
            char buff[100];
           strftime(buff, sizeof(buff), "%d-%m-%Y %H:%M:%S", &tm_now);
printf("Zeit: %s\n", buff);
00032
00033
00034 }
00035
00037 void freeHeapSpace(){
00038
         static unsigned long last = millis();
00039
            if (millis() - last > 5000) {
00040
            last = millis();
sHeapspace = ESP.getFreeHeap();
00041
                Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00042
00043
00044 }
00045
00047 void WiFiDiag(void) {
00048 Serial.println("\nWifi-Diag:");
00049 AP_IP = WiFi.softAPIP();
00050 CL_IP = WiFi.localIP();
         Serial.print("AP IP address: ");
Serial.println(AP_IP.toString());
Serial.print("Client IP address: ");
00051
00052
00053
         Serial.println(CL_IP.toString());
00054
         WiFi.printDiag(Serial);
Serial.print("\nScan AP's ");
00055
00056
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");
            } else {
```

```
Serial.print(n);
Serial.println(" networks found");
00065
00066
                 for (int i = 0; i < n; ++i)
00067
                {
00068
                  // Print SSID and RSSI for each network found
                   Serial.print(i + 1);
Serial.print(": ");
00069
00070
00071
                   Serial.print(WiFi.SSID(i));
00072
                   Serial.print(" (");
                   Serial.print(WiFi.RSSI(i));
00073
00074
                   Serial.print(")");
                   Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
00075
00076
                   delay(10);
00077
00078
00079
        }
00080 }
00081
00082 /
         **************************** Filesystem ********************/
00083
00091
00092 void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
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
            if(!root.isDirectory()) {
    Serial.println(" - not a directory");
00100
00101
00102
                return;
00103
00104
00105
           File file = root.openNextFile();
           while(file){
00106
                if(file.isDirectory()){
    Serial.print(" DIR
00107
                                        DIR : ");
00109
                     Serial.println(file.name());
00110
                     if(levels){
00111
                          listDir(fs, file.path(), levels -1);
00112
                     }
00113
                } else {
                     Serial.print(" FILE: ");
00114
                     Serial.print(file.name());
00115
00116
                     Serial.print("\tSIZE: ");
00117
                     Serial.println(file.size());
00118
00119
                file = root.openNextFile();
00120
           }
00121 }
00122
00128
00129 void readConfig(String filename) {
00130
           JsonDocument testDocument:
00131
           File configFile = LittleFS.open(filename);
            if (configFile)
00133
                Serial.println("open config file");
DeserializationError error = deserializeJson(testDocument, configFile);
00134
00135
00136
                // Test if parsing succeeds.
00137
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("Read Config - File");
                     strcpy(tAP_Config.wAP_SSID, testDocument["wAP_SSID"] | "Motordaten");
strcpy(tAP_Config.wAP_IP, testDocument["wAP_IP"] | "192.168.15.30");
00148
00149
                     strcpy(tAP_Config.wAP_Password, testDocument["wAP_Password"] | "12345678");
strcpy(tAP_Config.wMotor_Offset, testDocument["wMPdotorOffset"] | "0.0");
00150
00151
00152
              strcpy(tAP_Config.wCoolant_Offset, testDocument["wCoolantOffset"] | "0.0");
              strcpy(tAP_Config.wFuellstandmax, testDocument["wFuellstandmax"] | "0.0");
strcpy(tAP_Config.wADC1_Cal, testDocument["wADC1_Cal"] | "0.0");
strcpy(tAP_Config.wADC2_Cal, testDocument["wADC2_Cal"] | "0.0");
00153
00154
00155
00156
00157
                 configFile.close();
                Serial.println("Config - File closed");
00158
00159
            }
00160
00161
            else
00162
            {
```

7.25 helper.h 61

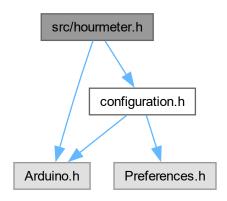
```
00163
              Serial.println("failed to load json config");
00164
00165 }
00166
00174
00175 bool writeConfig(String json)
00176 {
00177
          Serial.println(json);
00178
          Serial.println("neue Konfiguration speichern");
00179
00180
          // Datei zum Schreiben öffnen (überschreibt alte Datei)
          File configFile = LittleFS.open("/config.json", "w");
00181
00182
          if (!configFile) {
00183
              Serial.println("Config - Datei konnte nicht geöffnet werden!");
00184
              return false;
00185
00186
00187
          // JSON-Dokument parsen
          JsonDocument testDocument;
00188
00189
          DeserializationError error = deserializeJson(testDocument, json);
00190
              Serial.print(F("deserializeJson() failed: "));
00191
00192
              Serial.println(error.f_str());
00193
              configFile.close();
00194
              return false;
00195
         }
00196
00197
          // JSON in Datei schreiben
00198
          serializeJson(testDocument, configFile);
00199
          Serial.println("Konfiguration geschrieben...");
00200
00201
          // Kontrolle
00202
          serializeJsonPretty(testDocument, Serial);
00203
00204
          configFile.close();
          Serial.println("Config - Datei geschlossen");
00205
00206
          return true;
00207 }
00208
00211
00212 void T2C scan(void) {
00213
       byte error, address;
00214
        int nDevices;
00215
        Serial.println("Scanning...");
00216
        nDevices = 0;
00217
        for(address = 1; address < 127; address++ )</pre>
00218
00219
          Wire.beginTransmission(address);
00220
          error = Wire.endTransmission();
00221
          if (error == 0)
00222
00223
            Serial.print("I2C device found at address 0x");
00224
            if (address<16)</pre>
00225
00226
              Serial.print("0");
00227
00228
            Serial.println(address, HEX);
00229
            nDevices++;
00230
00231
          else if (error==4)
00232
00233
            Serial.print("Unknow error at address 0x");
00234
            if (address<16)</pre>
00235
00236
              Serial.print("0");
00237
00238
            Serial.println(address, HEX);
00239
            nDevices++;
00240
          else if (error==4) {
   Serial.print("Unknow error at address 0x");
00241
00242
            if (address<16) {
   Serial.print("0");</pre>
00243
00244
00245
00246
            Serial.println(address, HEX);
00247
00248
        if (nDevices == 0) {
00249
         Serial.println("No I2C devices found\n");
00250
00251
00252
       else {
00253
         Serial.println("done\n");
00254
00255 }
00256
00263
```

```
00264 String sWifiStatus(int Status)
00266
            switch(Status){
            case WL_IDLE_STATUS:return "Warten";
           case WL_IDLE_STATUS:return "Warten";
case WL_NO_SSID_AVAIL:return "Keine SSID vorhanden";
case WL_SCAN_COMPLETED:return "Scan komlett";
case WL_CONNECTED:return "Verbunden";
case WL_CONNECT_FAILED:return "Verbindung fehlerhaft";
case WL_CONNECTION_LOST:return "Verbindung verloren";
case WL_DISCONNECTED:return "Nicht verbunden";
default:return "unbekannt";
}
00267
00268
00269
00270
00271
00272
00273
00274
00276 }
00277
00284
00289
00290
            else{
              return 0;
00291
00292
00293 }
00294
00295
00296 #endif
```

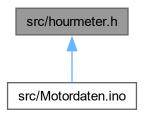
7.26 src/hourmeter.h-Dateireferenz

Betriebstundenzähler.

```
#include <Arduino.h>
#include "configuration.h"
Include-Abhängigkeitsdiagramm für hourmeter.h:
```



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



Funktionen

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

Variablen

- Preferences bsz1
- static unsigned long lastRun
- static unsigned long CounterOld
- · static unsigned long milliRest
- int state1 = LOW
- int laststate1 = LOW

7.26.1 Ausführliche Beschreibung

Betriebstundenzähler.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei hourmeter.h.

7.26.2 Dokumentation der Funktionen

7.26.2.1 EngineHours()

```
unsigned long EngineHours ( bool \ \textit{CountOn} \ = \ 0)
```

Betriebstundenzähler Berechnet Betriebstunden, wenn Anlage eingeschaltet ist.

Parameter

CountOn

Rückgabe

unsigned long

Definiert in Zeile 29 der Datei hourmeter.h.

```
00029
00030
                 unsigned long now = millis();
                milliRest += now - lastRun;
if (CountOn == 1) {
   while (milliRest >= 1000) {
00031
00032
00033
                             Counter++;
00034
00035
                              milliRest -= 1000;
00036
00037
                } else {
00038
                       milliRest = 0;
00039
00040
                lastRun = now;
00041
00042
                 state1 = CountOn;
                if (laststate1 == HIGH && state1 == LOW) { // speichern bei Flanke negativ
  bsz1.begin("bsz", false); // NVS nutzen, BSZ erstellen, lesen und schreiben (false)
  CounterOld = bsz1.getLong("Start"); // Speicher auslesen
  Counter = CounterOld + Counter; // Laufzeit alt + aktuell
  bsz1.putLong("Start", Counter); // Speicher schreiben
  bsz1.and(); // Preferences becoden
00043
00044
00045
00046
00047
00048
                       bsz1.end(); // Preferences beenden
00049
                 laststate1 = state1; // Aktualisiere laststate1
00050
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 state1 = LOW
```

Definiert in Zeile 21 der Datei hourmeter.h.

7.26.3.6 laststate1

```
int laststate1 = LOW
```

Definiert in Zeile 21 der Datei hourmeter.h.

7.27 hourmeter.h

gehe zur Dokumentation dieser Datei

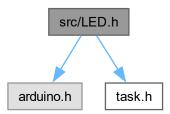
```
00001 #ifndef _HOURMETER_H_
00002 #define _HOURMETER_H_
00003
00014
00015 #include <Arduino.h>
00016 #include "configuration.h"
00017
00018 Preferences bsz1;
00019
00020 static unsigned long lastRun, CounterOld, milliRest; 00021 int state1 = LOW, laststate1 = LOW;
00022
00029 unsigned long EngineHours(bool CountOn = 0) {
           unsigned long now = millis();
milliRest += now - lastRun;
if (CountOn == 1) {
00030
00031
00032
                     while (milliRest >= 1000) {
00033
                           Counter++;
00034
00035
                            milliRest -= 1000;
00036
00037
               } else {
                     milliRest = 0;
00038
00039
00040
               lastRun = now;
00041
00042
               if (laststate1 == HIGH && state1 == LOW) { // speichern bei Flanke negativ
  bsz1.begin("bsz", false); // NVS nutzen, BSZ erstellen, lesen und schreiben (false)
  CounterOld = bsz1.getLong("Start"); // Speicher auslesen
  Counter = CounterOld + Counter; // Laufzeit alt + aktuell
  bsz1.putLong("Start", Counter); // Speicher schreiben
00043
00044
00045
00046
00047
00048
                     bsz1.end(); // Preferences beenden
00049
00050
                laststate1 = state1; // Aktualisiere laststate1
00051
                return Counter;
00052 }
00053
00054 #endif
```

7.28 src/LED.h-Dateireferenz

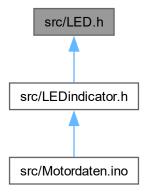
LED Ansteuerung.

```
#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()

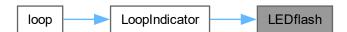
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



7.28.3.2 LEDflash()

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

Definiert in Zeile 37 der Datei LED.h.



7.28.3.3 flashLED()

Definiert in Zeile 49 der Datei LED.h.

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
00067
        delay(250);
digitalWrite(LED::Red, LOW);
        digitalWrite(LED::Blue, HIGH);
00068
        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 }
```

7.29 LED.h 71

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



7.28.3.6 LEDoff()

```
void LEDoff ( \label{eq:pin} \text{int } \textit{PIN} = \text{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
                                    //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) {
       taskBegin();
while (1) { // blockiert dank der TaskPause nicht
00027
00028
        digitalWrite(PIN, HIGH); // LED ein
00029
          taskPause(250); // gibt Rechenzeit ab
digitalWrite(PIN, LOW); // LED aus
00030
00031
00032
         taskPause(1000); // gibt Rechenzeit ab
00034
       taskEnd();
00035 }
00036
00037 void LEDflash(int PIN = LEDBoard) {
00038 taskBegin();
00039 while (1) { // blockiert dank der TaskPause nicht
        digitalWrite(PIN, HIGH); // LED ein
00041
          delay(5);
00042
          //taskPause(2);
                             // gibt Rechenzeit ab
         digitalWrite(PIN, LOW); // LED aus taskPause(3000); // gibt Rechenzeit ab
00043
00044
00045
00046
       taskEnd();
00047 }
00048
00051
         digitalWrite(PIN, HIGH);
       } 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);
        delay(250);
00070
        digitalWrite(LED::Blue, LOW);
00071
        digitalWrite(LED::Green, HIGH);
00072
        delay(250);
00073
       digitalWrite(LED::Green, LOW);
00074 }
00076 void LEDon(int PIN = LEDBoard) {
00077 digitalWrite(PIN, HIGH);
00078 }
00079
00080 void LEDoff(int PIN = LEDBoard) {
       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
```

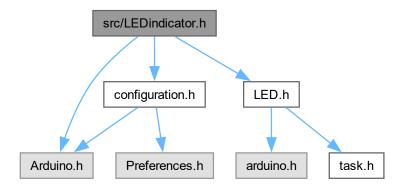
7.30 src/LEDindicator.h-Dateireferenz

LED Betriebsanzeige.

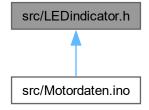
```
#include <Arduino.h>
#include "configuration.h"
```

#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

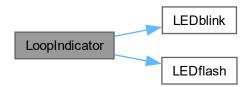
7.30.2.1 LoopIndicator()

```
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));
         LEDflash(LED(Green)); // flash for loop run without temp-failure
00040 }
00041 if (ErrorOn == true) {
         LEDblink(LED(Red));
00042
00043 }
00044 }
```

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



7.31 LEDindicator.h 75

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

```
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){
00038 if (ErrorOff == true ){
00039    LEDflash(LED(Green)); // flash for loop run without temp-failure
00040 }
00041 if (ErrorOn == true) {
         LEDblink (LED (Red));
00042
00043 }
00044 }
00045
00046 #endif
```

7.32 src/Motordaten.ino-Dateireferenz

Motordaten NMEA2000.

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

Include-Abhängigkeitsdiagramm für Motordaten.ino:



Makrodefinitionen

• #define ENABLE_DEBUG_LOG 0

Funktionen

- OneWire oneWire (ONE_WIRE_BUS)
- void debug_log (char *str)
- void IRAM_ATTR handleInterrupt ()

RPM Event Interrupt Enters on falling edge.

- · void setup ()
- void GetTemperature (void *parameter)

Get the Temperature object This task runs isolated on core 0 because sensors.requestTemperatures() is slow and blocking for about 750 ms With error on Sensor set output to -5 $^{\circ}$ C.

• double ReadRPM ()

Calculate engine RPM from number of interupts per time.

- bool IsTimeToUpdate (unsigned long NextUpdate)
- unsigned long InitNextUpdate (unsigned long Period, unsigned long Offset=0)
- void SetNextUpdate (unsigned long &NextUpdate, unsigned long Period)
- void SendN2kDCStatus (double BatteryVoltage, double SoC, double BatCapacity)

Send PGN127506.

· void SendN2kBattery (double BatteryVoltage)

Send PGN127508.

void SendN2kTankLevel (double level, double capacity)

Send PGN 127505.

- void SendN2kEngineData (double Oiltemp, double 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: $https://github.com/G6EJD/ \leftarrow ESP32-ADC-Accuracy-Improvement-function.$

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

2.8.1

Datum

2025-07-08

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()

Definiert in Zeile 88 der Datei Motordaten.ino.

```
00088  #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.

```
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
       portEXIT_CRITICAL_ISR(&mux);
00106
00107
       Last_int_time = millis();
00108 }
```



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.

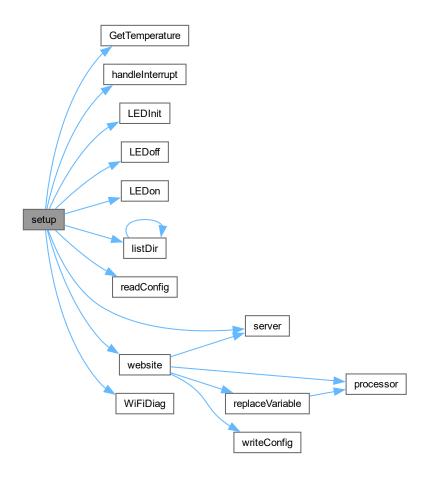
```
00112
00113
          // Init USB serial port
00114
          Serial.begin(115200);
00115
00116
          Serial.printf("Motordaten setup %s startn", VersionSoftware);
00117
             if (!LittleFS.begin(true)) {
00123
                  Serial.println(F("An Error has occurred while mounting LittleFS"));
00124
                  return;
00125
             Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00126
00127
00128
             File root = LittleFS.open("/");
00129
          listDir(LittleFS, "/", 3);
00130
00135
          readConfig("/config.json");
             IP = inet_addr(tAP_Config.wAP_IP);
AP_SSID = tAP_Config.wAP_SSID;
00136
00137
             AP_PASSWORD = tAP_Config.wAP_Password;
00138
00139
             fMotorOffset = atof(tAP_Config.wMotor_Offset);
00140
             fCoolantOffset = atof(tAP_Config.wCoolant_Offset);
       FuelLevelMax = atof(tAP_Config.wFuellstandmax);
ADC_Calibration_Value1 = atof(tAP_Config.wADC1_Cal);
ADC_Calibration_Value2 = atof(tAP_Config.wADC2_Cal);
Serial.println("\nAP-Configdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + ",
Passwort: " + AP_PASSWORD + " read from file");
00141
00142
00143
00144
       Serial.println("Configdata : TMotorOffset: " + String(fMotorOffset) + " , TCoolantOffset: " + String(fCoolantOffset) + " read from file");
00145
       Serial.println("Configdata: Füllstandmax: " + String(FuelLevelMax) + " , ADC1: " + String(ADC_Calibration_Value1) + " , ADC2: " + String(ADC_Calibration_Value2)+ " read from file");
00146
00147
00148
             File f = LittleFS.open("/config.json", "r");
00149
00150
            Serial.println("config.json konnte nicht geöffnet werden!");
00151
            String content = f.readString();
Serial.println("\nconfig.json Inhalt:");
00152
00153
00154
             Serial.println(content);
00155
             f.close();
00156
00157
           // LED
00158
          LEDInit();
00159
00160
          // Boardinfo
             sBoardInfo = boardInfo.ShowChipIDtoString();
00165
00166
00167
             //Wifi
          WiFi.mode(WIFI_AP_STA);
00168
00169
          WiFi.softAPdisconnect();
          if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
  WiFi.softAPConfig(IP, Gateway, NMask);
  Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00171
00172
       Serial.println("\nSet IP " + IP.toString() + " ,Gateway: " + Gateway.toString() + " ,NetMask: " + NMask.toString() + " ready");

LEDon(LED(Green));
00173
00174
00175
             delay(1000);
00176
             LEDoff(LED(Green));
```

```
} else {
00178
           Serial.println(F("Starting AP failed."));
00179
           LEDon (LED (Red));
            delay(1000);
00180
00181
           ESP.restart();
00182
00183
00184
       WiFi.setHostname(HostName);
00185
       Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00186
       delav(1000);
00187
00188
       WiFiDiag();
00189
00190
          if (!MDNS.begin(AP_SSID)) {
00191
              Serial.println(F("Error setting up MDNS responder!"));
00192
              while (1) {
                 delay(1000);
00193
00194
00195
00196
       Serial.println(F("mDNS responder started\n"));
00197
00198 // Start TCP (HTTP) server
00199
         server.begin();
         Serial.println(F("TCP server started\n"));
00200
00201
00202 // Add service to MDNS-SD
00203 MDNS.addService("http", "tcp", 80);
00204 MDNS.addService("ws", "tcp", 81);
00205
00206 // Webconfig laden
00207
       website();
00208
00213
       pinMode(Eingine_RPM_Pin, INPUT_PULLUP);
00214
       attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
     to interrupt on Falling Edge
00215
       timer = timerBegin(0, 80, true);
                                                                                        // this returns a
     pointer to the hw_timer_t global variable
00216
       // 0 = first timer
00217
       // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00218
        // true - counts up
00219
       timerStart(timer);
                                                                                        // starts the timer
00220
       sensors.begin():
00225
00226
       oneWire.reset();
        Serial.print("OneWire: Found ");
00227
00228
          Serial.print(sensors.getDeviceCount(), DEC);
00229
         Serial.println(" devices.");
         Serial.print("Parasite power is: ");
00230
       if (sensors.isParasitePowerMode()) Serial.println("ON");
00231
00232
         else Serial.println("OFF");
00233
       sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00234
00235
00236
       byte present = 0;
00237
       byte data[12];
00238
       byte addr[8];
00239
00240
       Serial.print("Looking for 1-Wire devices...\n\r");
       00241
00242
00243
00244
00245
00246
             Serial.print('0');
00247
00248
            Serial.print(addr[i], HEX);
00249
           if (i < 7) {
             Serial.print(", ");
00250
00251
00252
00253
          if ( OneWire::crc8( addr, 7) != addr[7]) {
00254
              Serial.print("CRC is not valid!\n");
00255
             return;
00256
         }
00257
00258
       Serial.print("\n\rNo more sensors!\n\r");
00259
        oneWire.reset_search();
00260
       delay(250);
00261
00262 // search for devices on the bus and assign based on an index
       if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00263
00264
       if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00265
00266
00267
00268 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
     or Mega
```

```
00269
        NMEA2000.SetN2kCANMsgBufSize(8);
00270
        NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00271
        NMEA2000.SetN2kCANSendFrameBufSize(250);
00272
00273
        esp_efuse_mac_get_default(chipid);
00274
        for (i = 0; i < 6; i++) id += (chipid[i] « (7 * i));
00275
00280
        NMEA2000.SetProductInformation("MD01.2507", // Manufacturer's Model serial code
                                          100, // Manufacturer's product code
"MD Sensor Module", // Manufacturer's Model ID
00281
00282
                                                                 // Manufacturer's Software version code
00283
                                          VersionSoftware,
00284
                                          VersionHardware
                                                                 // Manufacturer's Model version
00285
                                          );
00286 // Set device information
00287
        NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00288
                                          132, // Device function=Analog to NMEA 2000 Gateway. See codes on
      \texttt{http://www.nmea.org/Assets/20120726\%20nmea\%202000\%20class\%20\&\%20function\%20codes\%20v\%202.00.pdf}
00289
      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
00290
                                          2046 // Just choosen free from code list on
      http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00291
00292
00293 // If you also want to see all traffic on the bus use N2km\_ListenAndNode instead of N2km\_NodeOnly
      below
00294
00295
        NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
      Actisense format.
00296
         preferences.begin("nvs", false);
00297
                                                                        // Open nonvolatile storage (nvs)
        NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
00298
      33
00299
00300
        Serial.printf("NodeAddress=%d\n", NodeAddress);
00301
        NMEA2000.SetMode(tNMEA2000::N2km ListenAndNode, NodeAddress);
00302
00303
        NMEA2000.ExtendTransmitMessages(TransmitMessages);
        NMEA2000.Open();
00304
00305
00306
       xTaskCreatePinnedToCore(
00307
           GetTemperature, /\star Function to implement the task \star/
           "Task1", /* Name of the task \star/
00308
           10000, /* Stack size in words */
NULL, /* Task input parameter */
00309
          NULL,
00310
           0, /* Priority of the task */
00311
00312
           &Task1, /* Task handle. */
00313
          0); /* Core where the task should run */
00314
00315
        delav(200);
00316
00321
        ArduinoOTA
00322
          .onStart([]() {
00323
             String type;
             if (ArduinoOTA.getCommand() == U_FLASH)
  type = "sketch";
00324
00325
            else // U_SPIFFS
type = "filesystem";
00326
00327
00328
             // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end() Serial.println("Start updating " + type);
00329
00330
00331
           })
00332
           .onEnd([]() {
00333
             Serial.println("\nEnd");
00334
00335
           .onProgress([](unsigned int progress, unsigned int total) {
00336
             Serial.printf("Progress: u% \r", (progress / (total / 100)));
00337
00338
           .onError([](ota_error_t error) {
             Serial.printf("Error[%u]: ", error);
00339
             if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00340
00341
             else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00342
             else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
             else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00343
             else if (error == OTA_END_ERROR) Serial.println("End Failed");
00344
00345
00346
00347
        ArduinoOTA.begin();
00348
        printf("Setup end\n");
00349
00350 }
```

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 $-5\,^{\circ}$ C.

Parameter

parameter

Definiert in Zeile 359 der Datei Motordaten.ino.

```
if (motorErrorReported == "Aus") {
                                                                                   // Nur einmal melden
00368
               Serial.print("Error read Motor Temp\n");
00369
                motorErrorReported = "Ein";}
               MotorTemp = -5.0;
00370
00371
           } else {
               MotorTemp = tmp0 + fMotorOffset;
00372
00373
               motorErrorReported = "Aus";
                                                                         // Fehler wurde behoben
00374
00375
           vTaskDelay(100);
           tmp1 = sensors.getTempC(MotorCoolant);
if (tmp1 == DEVICE_DISCONNECTED_C) {
   if (coolantErrorReported == "Aus") {
00376
00377
00378
                                                                                  // Nur einmal melden
00379
               Serial.print("Error read Coolant Temp\n");
00380
               coolantErrorReported = "Ein";}
00381
               CoolantTemp = -5.0;
00382
           } else {
               CoolantTemp = tmp1 + fCoolantOffset;
coolantErrorReported = "Aus";
00383
00384
                                                                          // Fehler wurde behoben
00385
00386
           vTaskDelay(100);
00387 }
00388 }
```

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



7.32.3.6 ReadRPM()

```
double ReadRPM ()
```

Calculate engine RPM from number of interupts per time.

Rückgabe

double

Definiert in Zeile 395 der Datei Motordaten.ino.

```
00395
         double RPM = 0;
00396
00397
00398
         portENTER_CRITICAL(&mux);
         if (PeriodCount != 0) {
   RPM = 1000000.00 / PeriodCount;
                                                                      // 0 means no signals measured
// PeriodCount in 0.000001 of a second
00399
00400
00401
00402
         portEXIT_CRITICAL(&mux);
00403
         if (millis() > Last_int_time + 200) RPM = 0;
                                                                    // No signals RPM=0;
00404
         return (RPM);
00405 }
```

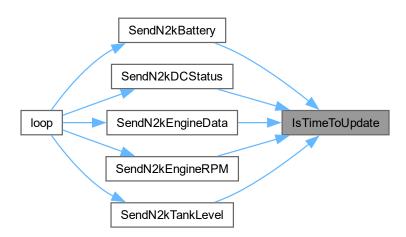


7.32.3.7 IsTimeToUpdate()

```
bool IsTimeToUpdate ( unsigned \ long \ \textit{NextUpdate})
```

Definiert in Zeile 408 der Datei Motordaten.ino.

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

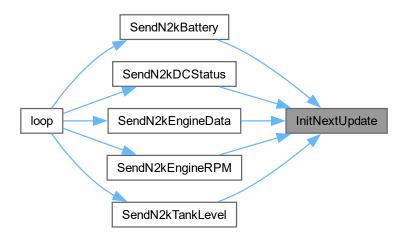


7.32.3.8 InitNextUpdate()

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

Definiert in Zeile 411 der Datei Motordaten.ino.

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

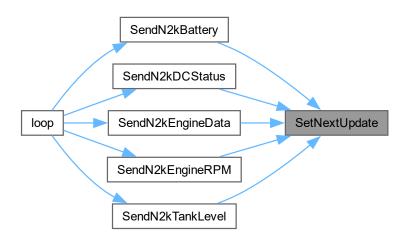


7.32.3.9 SetNextUpdate()

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

Definiert in Zeile 415 der Datei Motordaten.ino.

```
00415
00416  while ( NextUpdate < millis() ) NextUpdate += Period;
00417 }</pre>
```



7.32.3.10 SendN2kDCStatus()

Send PGN127506.

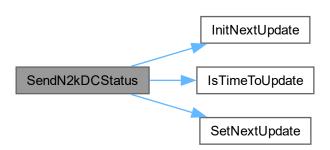
Parameter

BatteryVoltage	
SoC	
BatCapacity	

Definiert in Zeile 427 der Datei Motordaten.ino.

```
00427
00428
            static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
         BatteryDCStatusSendOffset);
00429
            tN2kMsg N2kMsg;
00430
00431
            if ( IsTimeToUpdate(SlowDataUpdated) ) {
00432
               SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00433
              Serial.printf("Voltage : %3.1f V\n", BatteryVoltage);
Serial.printf("SoC : %3.1f %\n", SoC);
Serial.printf("Capacity : %3.1f Ah\n", BatCapacity);
// SetN2kDCStatus(N2kMsg,1,1,N2kDCt_Battery,56,92,38500,0.012, AhToCoulomb(420));
SetN2kDCStatus(N2kMsg, 1, 2, N2kDCt_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
(Soulomb(45)).
00434
00435
00436
00437
00438
        AhToCoulomb(55));
00439
              NMEA2000.SendMsg(N2kMsg);
00440
00441 }
```

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





7.32.3.11 SendN2kBattery()

```
\begin{tabular}{ll} {\tt void SendN2kBattery} & ( \\ & {\tt double } \begin{tabular}{ll} {\tt BatteryVoltage} \end{tabular} \label{table}
```

Send PGN127508.

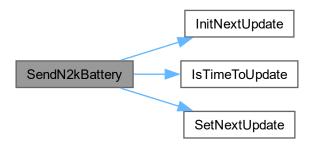
Parameter

BatteryVoltage 1 8 1

Definiert in Zeile 448 der Datei Motordaten.ino.

```
00448
        static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00449
00450
        tN2kMsg N2kMsg;
00451
00452
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00453
         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00454
          Serial.printf("Voltage
                                      : %3.1f V\n", BatteryVoltage);
00455
00456
          SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1); NMEA2000.SendMsg(N2kMsg);
00457
00458
00459
00460 }
```

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





7.32.3.12 SendN2kTankLevel()

Send PGN 127505.

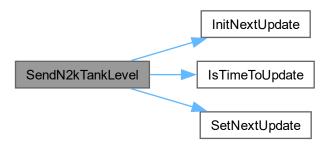
Parameter

level	
capacity	

Definiert in Zeile 468 der Datei Motordaten.ino.

```
00468
00469
         static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00470
         tN2kMsg N2kMsg;
00471
00472
         if ( IsTimeToUpdate(SlowDataUpdated) ) {
00473
           SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00474
            Serial.printf("Fuel Level : \$3.1f \$\$\n", level); \\ Serial.printf("Fuel Capacity: \$3.1f 1\n", capacity); \\ 
00475
00476
00477
           SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity);
00478
00479
           NMEA2000.SendMsg(N2kMsg);
00480
00481 }
```

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





7.32.3.13 SendN2kEngineData()

Send PGN 127489.

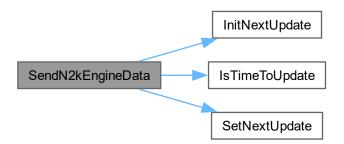
Parameter

Oiltemp	
Coolanttemp	
rpm	
hours	
voltage	

Definiert in Zeile 492 der Datei Motordaten.ino.

```
00492
                       static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00493
00494
                      tN2kMsq N2kMsq;
00495
                      tN2kEngineDiscreteStatus1 Status1;
00496
                       tN2kEngineDiscreteStatus2 Status2;
                      Status1.Bits.OverTemperature = Oiltemp > 90;
Status1.Bits.LowCoolantLevel = Coolanttemp > 90;
                                                                                                                                                                  // Alarm Motor over temp
00497
00498
                                                                                                                                                                          // Alarm low cooling
                      Status1.Bits.LowSystemVoltage = voltage < 11;
Status2.Bits.EngineShuttingDown = rpm < 100;
00499
00500
                                                                                                                                                                    // Alarm Motor off
00501
                      EngineOn = !Status2.Bits.EngineShuttingDown;
00502
00503
                      if ( IsTimeToUpdate(SlowDataUpdated) ) {
00504
                            SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00505
                            Serial.printf("Oil Temp : %3.1f °C \n", Oiltemp);
Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00506
00507
                            Serial.printf("Overtemp Oil: %s \n", Statusl.Bits.OverTemperature ? "Yes" : "No"); Serial.printf("Overtemp Mot: %s \n", Statusl.Bits.LowCoolantLevel ? "Yes" : "No");
00509
00510
                            Serial.printf("Engine Off : %s \n", Status2.Bits.EngineShuttingDown ? "Yes": "No");
00511
00512
                             // SetN2kTemperatureExt (N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
00513
                // PGN130312, uncomment the PGN to be used
00514
00515
                            \tt Set N2 k Engine Dynamic Param (N2 k Msg, 0, N2 k Double NA, CTo Kelvin (Oiltemp), CTo Kelvin (Coolant temp), CTo Kelvin (Cool
                N2kDoubleNA, N2kDoubleNA, hours ,N2kDoubleNA ,N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00516
00517
                            NMEA2000.SendMsg(N2kMsg);
00518
```

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 ( \label{eq:cond} \mbox{double $\it RPM$)}
```

Send PGN 127488.

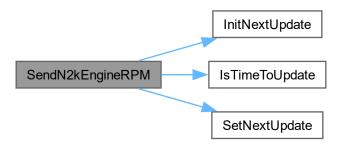
Parameter

RPM

Definiert in Zeile 526 der Datei Motordaten.ino.

```
00526
00527
        static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMSendOffset);
00528
        tN2kMsg N2kMsg;
00529
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
   SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00530
00531
00532
00533
           Serial.printf("Engine RPM : %4.0f RPM \n", RPM);
00534
00535
           SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00536
          NMEA2000.SendMsg(N2kMsg);
00537
00538
00539 }
```

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 ( \label{eq:condition} \text{byte } pin \text{)}
```

ReadVoltage is used to improve the linearity of the ESP32 ADC see: $https://github.com/G6EJD/ \\com/G6EJD/ \\com/G6$

Parameter



Rückgabe

double

Definiert in Zeile 547 der Datei Motordaten.ino.

```
00547 {
00548 double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in range 0 to 4095
00549 if (reading < 1 || reading > 4095) return 0;
00550 // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) + 0.000854596860691 * reading + 0.065440348345433;
00551 return (-0.000000000000016 * pow(reading, 4) + 0.00000000118171 * pow(reading, 3) - 0.00000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;
00552 } // Added an improved polynomial, use either, comment out as required
```



7.32.3.16 loop()

```
void loop ()
```

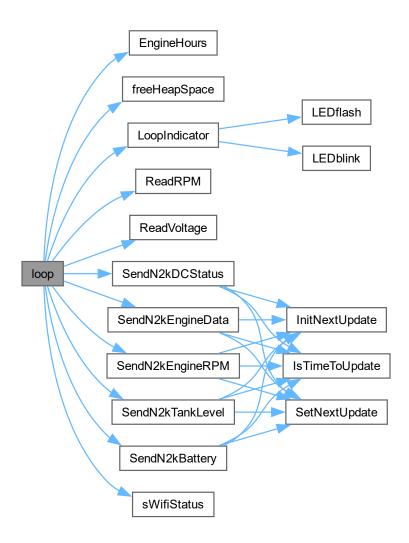
Actual Website Data

Construct a new if object Reboot from Website

Definiert in Zeile 555 der Datei Motordaten.ino.

```
00556
00557
        LoopIndicator();
00558
        BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
00559
     // This implements a low pass filter to eliminate spike for ADC readings
00560
        FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
00561
      This implements a low pass filter to eliminate spike for ADC readings
00562
00563
       EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6; // This implements a low
     pass filter to eliminate spike for RPM measurements
00564
        BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
00565
      Zustand über Spannung
00566
       // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00567
00568
        EngineHours (EngineOn);
00569
00570
        SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00571
        SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00572
        SendN2kEngineRPM(EngineRPM);
00573
        SendN2kBattery(BordSpannung);
        SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00574
00575
        NMEA2000.ParseMessages();
00576
        int SourceAddress = NMEA2000.GetN2kSource();
00578
        if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
                                              // Set new Node Address (to save only once)
00579
          NodeAddress = SourceAddress;
          preferences.begin("nvs", false);
00580
          preferences.putInt("LastNodeAddress", SourceAddress);
00581
00582
          preferences.end();
00583
          Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00584
00585
00586
        // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00587
        if ( Serial.available() ) {
00588
          Serial.read():
00589
00590
00591
00592 // OTA
00593
          ArduinoOTA.handle();
00594
00599
          webSocket.loop();
          fCoolantTemp = CoolantTemp;
fMotorTemp = MotorTemp;
00600
00601
00602
          fBordSpannung = BordSpannung;
00603
          fDrehzahl = EngineRPM;
sCL_Status = sWifiStatus(WiFi.status());
00604
          sAP_Station = WiFi.softAPgetStationNum();
00605
00606
          freeHeapSpace();
00607
00612
        if (IsRebootRequired) {
            Serial.println("Rebooting ESP32: ");
00613
            delay(1000); // give time for reboot page to load ESP.restart();
00614
00615
00616
00617
00618
00619 }
```

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:

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

Definiert in Zeile 50 der Datei Motordaten.ino.

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

```
00002
        This code is free software; you can redistribute it and/or
        modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either
00003
00004
00005
         version 2.1 of the License, or (at your option) any later version.
         This code is distributed in the hope that it will be useful,
00007
         but WITHOUT ANY WARRANTY; without even the implied warranty of
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
00008
00009
         Lesser General Public License for more details.
00010
         You should have received a copy of the GNU Lesser General Public
        License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
00011
00012
00013 */
00014
00025
00026 #include <Arduino.h>
00027 #include "configuration.h"
00028 #include <Preferences.h>
00029 #include <ArduinoOTA.h>
00030 #include <OneWire.h>
00031 #include <DallasTemperature.h>
00032 #include <ESP_WiFi.h>
00033 #include <ESPAsyncWebServer.h>
00034 #include <NMEA2000_CAN.h> // This will automatically choose right CAN library and create suitable
     NMEA2000 object
00035 #include <N2kMessages.h>
00036 #include <ESPmDNS.h>
00037 #include <arpa/inet.h>
00038 #include "BoardInfo.h"
00039 #include "helper.h"
00040 #include "web.h"
00041 #include "hourmeter.h"
00042 #include "LEDindicator.h"
00043
00044 #define ENABLE_DEBUG_LOG 0 // Debug log
00045
00046
00050 const unsigned long TransmitMessages[] PROGMEM = {127488L, // Engine Rapid / RPM
00051
                                                                127489L, // Engine parameters dynamic
                                                                127505L, // Fluid Level
127506L, // Battery
127508L, // Battery Status
00052
00053
00054
00055
00056
                                                               };
00057
00058
00062
00063 volatile uint64_t StartValue = 0;
00064 volatile uint64_t PeriodCount = 0;
00065 unsigned long Last_int_time = 0;
00066 hw_timer_t * timer = NULL;
00067 portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
00068
00072 OneWire oneWire(ONE WIRE BUS);
00073 DallasTemperature sensors(&oneWire);
00074 // DeviceAddress MotorThermometer; /**< arrays to hold device addresses 00075 uint8_t MotorCoolant[8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
00076 uint8_t MotorOil[8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
00077
00078 const int ADCpin2 = 35;
00079 const int ADCpin1 = 34;
08000
00082 TaskHandle_t Task1;
00083
00085 const int baudrate = 38400;
00086 const int rs_config = SERIAL_8N1;
00087
00088 void debug_log(char* str) {
00089 #if ENABLE_DEBUG_LOG == 1
```

7.33 Motordaten.ino 97

```
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
        portEXIT_CRITICAL_ISR(&mux);
00106
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", VersionSoftware);
00117
00122
           if (!LittleFS.begin(true)) {
00123
                Serial.println(F("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
         readConfig("/config.json");
00135
           IP = inet_addr(tAP_Config.wAP_IP);
00136
           AP_SSID = tAP_Config.wAP_SSID;
00137
           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);
           ADC_Calibration_value1 = atof(tAP_Config.wADC1_Cal);
ADC_Calibration_value2 = atof(tAP_Config.wADC2_Cal);
Serial.println("\nAP-Configdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + " ,
00142
00143
00144
      Passwort: " + AP_PASSWORD + " read from file");
Serial.println("Configdata: TMotorOffset: " + String(fMotorOffset) + " , TCoolantOffset: " +
00145
      String(fCoolantOffset) + " read from file");
      Serial.println("Configdata : Füllstandmax: " + String(FuelLevelMax) + " , ADC1: " + String(ADC_Calibration_Value1) + " , ADC2: " + String(ADC_Calibration_Value2) + " read from file");
00146
00147
00148
           File f = LittleFS.open("/config.json", "r");
00149
00150
           Serial.println("config.json konnte nicht geöffnet werden!");
00151
        } else {
          String content = f.readString();
00152
           Serial.println("\nconfig.json Inhalt:");
00153
00154
           Serial.println(content);
00155
           f.close();
00156
         // LED
00157
00158
        LEDInit();
00159
00160
        // Boardinfo
           sBoardInfo = boardInfo.ShowChipIDtoString();
00165
00166
00167
            //Wifi
         WiFi.mode(WIFI_AP_STA);
00168
      wirl.softAPGreen):
if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
    WiFi.softAPConfig(IP, Gateway, NMask);
    Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
    Serial.println("\nSc IP " + IP.toString() + " ,Gateway: " + Gateway.toString() + " ,NetMask: " +
NMask.toString() + " ready");
    LEDon(LED(Green)):
         WiFi.softAPdisconnect();
00169
00170
00171
00172
00173
00174
           LEDon (LED (Green));
00175
           delay(1000);
00176
           LEDoff(LED(Green));
00177
00178
              Serial.println(F("Starting AP failed."));
              LEDon (LED (Red));
00179
              delay(1000);
00180
00181
              ESP.restart();
00182
00183
00184
         WiFi.setHostname(HostName);
         Serial.println("Set Hostname" + String(WiFi.getHostname()) + "done\n");
00185
00186
00187
         delav(1000);
```

```
WiFiDiag();
00188
00189
00190
           if (!MDNS.begin(AP_SSID)) {
00191
               Serial.println(F("Error setting up MDNS responder!"));
00192
               while (1) {
00193
                   delay(1000);
00194
00195
00196
        Serial.println(F("mDNS responder started\n"));
00197
00198 // Start TCP (HTTP) server
00199
          server.begin();
00200
           Serial.println(F("TCP server started\n"));
00201
00202 // Add service to MDNS-SD
        MDNS.addService("http", "tcp", 80);
MDNS.addService("ws", "tcp", 81);
00203
00204
00205
00206 // Webconfig laden
00207
        website();
00208
00213
        pinMode(Eingine_RPM_Pin, INPUT_PULLUP);
                                                                                                      // sets pin high
       attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
00214
      to interrupt on Falling Edge
00215
        timer = timerBegin(0, 80, true);
                                                                                                  // this returns a
      pointer to the hw_timer_t global variable
00216
        // 0 = first timer
00217
         // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
         // true - counts up
00218
        timerStart(timer);
00219
                                                                                                  // starts the timer
00220
00225
        sensors.begin();
00226
        oneWire.reset();
00227
           Serial.print("OneWire: Found ");
          Serial.print(sensors.getDeviceCount(), DEC);
Serial.println(" devices.");
Serial.print("Parasite power is: ");
00228
00229
00230
00231
        if (sensors.isParasitePowerMode()) Serial.println("ON");
00232
           else Serial.println("OFF");
00233
        sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00234
00235
        byte i:
00236
        byte present = 0;
00237
        byte data[12];
00238
        byte addr[8];
00239
00240
        Serial.print("Looking for 1-Wire devices...\n\r");
        while (oneWire.search(addr)) {
   Serial.print("\n\rFound \'1-Wire\' device with address:\n\r");
   for( i = 0; i < 8; i++) {</pre>
00241
00242
00243
             Serial.print("0x");
00244
00245
             if (addr[i] < 16)
00246
               Serial.print('0');
00247
00248
             Serial.print(addr[i], HEX);
00249
             if (i < 7) {
00250
               Serial.print(", ");
00251
00252
           if ( OneWire::crc8( addr, 7) != addr[7]) {
    Serial.print("CRC is not valid!\n");
00253
00254
00255
               return;
00256
          }
00257
00258
        Serial.print("\n\rNo more sensors!\n\r");
00259
        oneWire.reset_search();
00260
        delay(250);
00261
00262 // search for devices on the bus and assign based on an index
        if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00263
00264
         if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00265
00266
00267
00268 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
      or Mega
        NMEA2000.SetN2kCANMsgBufSize(8);
00269
00270
        NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00271
        NMEA2000.SetN2kCANSendFrameBufSize(250);
00272
00273
        esp_efuse_mac_get_default(chipid);
for (i = 0; i < 6; i++) id += (chipid[i] « (7 * i));</pre>
00274
00275
00280
        NMEA2000.SetProductInformation("MD01.2507", // Manufacturer's Model serial code
                                           100, // Manufacturer's product code
"MD Sensor Module", // Manufacturer's Model ID
00281
00282
                                                                   // Manufacturer's Software version code
00283
                                            VersionSoftware.
```

7.33 Motordaten.ino 99

```
00284
                                            VersionHardware
                                                                   // Manufacturer's Model version
00285
00286 // Set device information
00287
        {\tt NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.} \\
                                           132, // Device function=Analog to NMEA 2000 Gateway. See codes on
00288
      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
00290
                                           2046 // Just choosen free from code list on
      http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00291
                                         );
00292
00293 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
00294
00295
        NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
      Actisense format.
00296
00297
        preferences.begin("nvs", false);
                                                                          // Open nonvolatile storage (nvs)
         NodeAddress = preferences.getInt("LastNodeAddress", 33); // Read stored last NodeAddress, default
00298
      33
00299
        preferences.end();
00300
        Serial.printf("NodeAddress=%d\n", NodeAddress);
00301
00302
        NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00303
        NMEA2000.ExtendTransmitMessages(TransmitMessages);
00304
        NMEA2000.Open();
00305
00306
       xTaskCreatePinnedToCore(
          GetTemperature, /* Function to implement the task */ "Task1", /* Name of the task */
00307
00308
           10000, /* Stack size in words */
NULL, /* Task input parameter */
00309
00310
00311
           0, /* Priority of the task */
          &Task1, /* Task handle. */
0); /* Core where the task should run */
00312
00313
00314
00315
        delay(200);
00316
00321
        ArduinoOTA
00322
          .onStart([]() {
00323
             String type;
             if (ArduinoOTA.getCommand() == U FLASH)
00324
             type = "sketch";
else // U_SPIFFS
00325
00326
00327
               type = "filesystem";
00328
             // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end() Serial.println("Start updating " + type);
00329
00330
00331
           1)
00332
           .onEnd([]() {
00333
             Serial.println("\nEnd");
00334
           .onProgress([](unsigned int progress, unsigned int total) {
   Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00335
00336
00337
00338
           .onError([](ota_error_t error) {
00339
             Serial.printf("Error[%u]: ", error);
             if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00340
00341
             else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
             else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00342
00343
00344
             else if (error == OTA_END_ERROR) Serial.println("End Failed");
00345
00346
00347
        ArduinoOTA.begin();
00348
        printf("Setup end\n");
00349
00350 }
00351
00358
00359 void GetTemperature(void * parameter) {
00360
        float tmp0 = 0;
         float tmp1 = 0;
00361
00362
         for (;;) {
00363
           sensors.requestTemperatures();
                                                                       // Send the command to get temperatures
00364
           vTaskDelay(100);
00365
           tmp0 = sensors.getTempC(MotorOil);
           if (tmp0 == DEVICE_DISCONNECTED_C) {
   if (motorErrorReported == "Aus") {
00366
00367
                                                                                // Nur einmal melden
               Serial.print("Error read Motor Temp\n");
00368
               motorErrorReported = "Ein";}
00369
00370
               MotorTemp = -5.0;
00371
           } else {
00372
               MotorTemp = tmp0 + fMotorOffset;
               motorErrorReported = "Aus";
                                                                       // Fehler wurde behoben
00373
00374
           }
```

```
00375
          vTaskDelay(100);
00376
          tmp1 = sensors.getTempC(MotorCoolant);
00377
          if (tmp1 == DEVICE_DISCONNECTED_C) {
            if (coolantErrorReported == "Aus") {
00378
                                                                      // Nur einmal melden
             Serial.print("Error read Coolant Temp\n");
00379
00380
              coolantErrorReported = "Ein";}
              CoolantTemp = -5.0;
00382
         } else {
00383
           CoolantTemp = tmp1 + fCoolantOffset;
                                                                // Fehler wurde behoben
00384
              coolantErrorReported = "Aus";
00385
00386
         vTaskDelav(100);
00387
       }
00388 }
00389
00395 double ReadRPM() {
00396
       double RPM = 0;
00397
       portENTER_CRITICAL(&mux);
00399
       if (PeriodCount != 0) {
                                                           // 0 means no signals measured
                                                           // PeriodCount in 0.000001 of a second
00400
        RPM = 1000000.00 / PeriodCount;
00401
00402
       portEXIT_CRITICAL(&mux);
       if (millis() > Last_int_time + 200) RPM = 0;
                                                        // No signals RPM=0:
00403
00404
       return (RPM);
00405 }
00406
00407
00408 bool IsTimeToUpdate(unsigned long NextUpdate) {
00409
       return (NextUpdate < millis());</pre>
00410 }
00411 unsigned long InitNextUpdate(unsigned long Period, unsigned long Offset = 0) {
00412 return millis() + Period + Offset;
00413 }
00414
00415 void SetNextUpdate(unsigned long &NextUpdate, unsigned long Period) {
       while ( NextUpdate < millis() ) NextUpdate += Period;</pre>
00416
00418
00427 void SendN2kDCStatus(double BatteryVoltage, double SoC, double BatCapacity) {
       static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
00428
     BatteryDCStatusSendOffset);
00429
       tN2kMsg N2kMsg;
00430
00431
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00432
         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00433
         Serial.printf("Voltage : %3.1f V\n", BatteryVoltage);
Serial.printf("SoC : %3.1f %\n", SoC);
Serial.printf("Capacity : %3.1f Ah\n", BatCapacity);
00434
00435
00436
00437
          // SetN2kDCStatus(N2kMsg,1,1,N2kDCt_Battery,56,92,38500,0.012, AhToCoulomb(420));
AhToCoulomb(55));

00439 NMF7200
00438
          SetN2kDCStatus(N2kMsg, 1, 2, N2kDCt_Battery, SoC, 0, N2kDoubleNA, BatteryVoltage,
         NMEA2000.SendMsg(N2kMsg);
00440
00441 }
00442
00448 void SendN2kBattery(double BatteryVoltage) {
00449
       static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00450
       tN2kMsq N2kMsq;
00451
00452
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00453
         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00454
00455
         Serial.printf("Voltage
                                     : %3.1f V\n", BatteryVoltage);
00456
00457
          SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00458
         NMEA2000.SendMsq(N2kMsq);
00459
       }
00460 }
00461
00468 void SendN2kTankLevel(double level, double capacity) {
       static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00469
00470
        tN2kMsg N2kMsg;
00471
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00472
00473
         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00474
00475
          Serial.printf("Fuel Level : %3.1f %%\n", level);
00476
          Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00477
00478
          SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00479
          NMEA2000.SendMsg(N2kMsg);
00480
00481 }
00482
```

7.33 Motordaten.ino 101

```
00492 void SendN2kEngineData(double Oiltemp, double Coolanttemp, double rpm, double hours, double voltage) {
        static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00493
00494
        tN2kMsg N2kMsg;
00495
        tN2kEngineDiscreteStatus1 Status1;
00496
        tN2kEngineDiscreteStatus2 Status2;
        Status1.Bits.OverTemperature = Oiltemp > 90;
00497
                                                          // Alarm Motor over temp
        Status1.Bits.LowCoolantLevel = Coolanttemp > 90;
00498
                                                           // Alarm low cooling
00499
        Status1.Bits.LowSystemVoltage = voltage < 11;</pre>
00500
        Status2.Bits.EngineShuttingDown = rpm < 100;</pre>
                                                           // Alarm Motor off
00501
        EngineOn = !Status2.Bits.EngineShuttingDown;
00502
00503
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00504
          SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00505
00506
          Serial.printf("Oil Temp
                                     : %3.1f °C \n", Oiltemp);
          Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00507
00508
          Serial.printf("Overtemp Oil: %s \n", Statusl.Bits.OverTemperature ? "Yes": "No");
Serial.printf("Overtemp Mot: %s \n", Statusl.Bits.LowCoolantLevel ? "Yes": "No");
00509
00510
          Serial.printf("Engine Off : %s \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00511
00512
00513
          // SetN2kTemperatureExt(N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
      // PGN130312, uncomment the PGN to be used
00514
00515
          SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Coolanttemp),
     N2kDoubleNA, N2kDoubleNA, hours ,N2kDoubleNA ,N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00516
00517
          NMEA2000.SendMsg(N2kMsg);
00518
00519 }
00520
00526 void SendN2kEngineRPM(double RPM) {
00527
        static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMSendOffset);
00528
        tN2kMsg N2kMsg;
00529
        if ( IsTimeToUpdate(SlowDataUpdated) ) {
00530
00531
         SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00533
          Serial.printf("Engine RPM : %4.0f RPM \n", RPM);
00534
          SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA, N2kInt8NA);
00535
00536
00537
          NMEA2000. SendMsa (N2kMsa):
00538
00539 }
00540
00547 double ReadVoltage(byte pin) {
00548 double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in
     range 0 to 4095
      if (reading < 1 || reading > 4095) return 0;
        // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +
00550
     0.000854596860691 * reading + 0.065440348345433;

return (-0.000000000000016 * pow(reading, 4) + 0.00000000118171 * pow(reading, 3) -
00551
      0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000; \\
00552 } // Added an improved polynomial, use either, comment out as required
00553
           00555 void loop() {
00556
00557
        LoopIndicator();
00558
        BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
00559
     // This implements a low pass filter to eliminate spike for ADC readings
00560
        FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
00561
     This implements a low pass filter to eliminate spike for ADC readings
00562
        EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM Calibration Value) / 6 : // This implements a low
00563
      pass filter to eliminate spike for RPM measurements
        BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
00565
      Zustand über Spannung
00566
        // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00567
00568
        EngineHours (EngineOn);
00569
00570
        SendN2kTankLevel(FuelLevel, FuelLevelMax); // Adjust max tank capacity
00571
        SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00572
        SendN2kEngineRPM(EngineRPM);
00573
        SendN2kBattery(BordSpannung);
00574
        SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00575
00576
        NMEA2000.ParseMessages();
00577
        int SourceAddress = NMEA2000.GetN2kSource();
        00578
00579
         preferences.begin("nvs", false);
00580
```

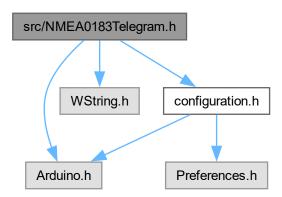
```
preferences.putInt("LastNodeAddress", SourceAddress);
00582
           preferences.end();
00583
           Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00584
00585
         // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00586
        if ( Serial.available() ) {
00588
           Serial.read();
00589
00590
00591
00592 // OTA
00593
          ArduinoOTA.handle();
00594
00599
           webSocket.loop();
           fCoolantTemp = CoolantTemp;
fMotorTemp = MotorTemp;
00600
00601
           fBordSpannung = BordSpannung;
00602
           fDrehzahl = EngineRPM;
00603
00604
           sCL_Status = sWifiStatus(WiFi.status());
00605
           sAP_Station = WiFi.softAPgetStationNum();
00606
           freeHeapSpace();
00607
        if (IsRebootRequired) {
   Serial.println("Rebooting ESP32: ");
   delay(1000); // give time for reboot page to load
00612
00613
00614
00615
             ESP.restart();
00616
00617
00618
00619 }
```

7.34 src/NMEA0183Telegram.h-Dateireferenz

NMEA0183 Telegrame senden.

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

Include-Abhängigkeitsdiagramm für NMEA0183Telegram.h:



Funktionen

• char CheckSum (String NMEAData)

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 Telegrame 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()

Checksum calculation for NMEA.

Parameter

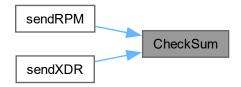
NMEAData

Rückgabe

char

Definiert in Zeile 23 der Datei NMEA0183Telegram.h.

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;
08000
          String SendSensor;
00081
            NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);
//NMEASensorKraeng += ",";
NMEASensor += String(fGaugeDrehzahl);
00082
00083
00084
00085
            NMEASensor += ",D,ROLL";
00086
00087
          // Build CheckSum
00088
00089
         HexCheckSum = String(CheckSum(NMEASensor), HEX);
         // Build complete NMEA string
SendSensor = "$" + NMEASensor;
SendSensor += "*";
00090
00091
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;
         String SendSensor;
00109
00110
00111
            NMEASensor = "IIRPM,E,1,"; //NMEASensor = "IIXDR,E,1," + String(SensorID);
           NMEASensor += String(fGaugeDrehzahl);
NMEASensor += ",15,A";
00112
00113
00114
00115
         // Build CheckSum
00116
         HexCheckSum = String(CheckSum(NMEASensor), HEX);
         // Build complete NMEA string
SendSensor = "$" + NMEASensor;
SendSensor += "*";
00117
00118
00119
00120
00121
         SendSensor += HexCheckSum;
00122
         Serial.println(SendSensor);
00123
00124
         return SendSensor;
00125 }
```

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



7.35 NMEA0183Telegram.h

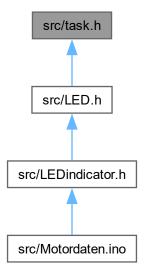
gehe zur Dokumentation dieser Datei

```
00012 #include <Arduino.h>
00013 #include <WString.h> // Needs for structures 00014 #include "configuration.h"
00022
00023 char CheckSum(String NMEAData) {
00024 char checksum = 0;
        // Iterate over the string, XOR each byte with the total sum
for (int c = 0; c < NMEAData.length(); c++) {
   checksum = char(checksum ^ NMEAData.charAt(c));</pre>
00025
00026
00027
00029 // Return the result return checksum;
00031 }
00032
00033 /*
00035 Transducer Values
         1 2 3 4
00036
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
           Druck $IIXDR,P,1.02481,B,Barometer*29
Kraengung $IIXDR,A,0,x.x,ROLL*hh<CR><LF>
00050
00051
00052
00053
00054
         RPM - Revolutions
00055
                1 2 3 4 5 6
00056
00057
00058 $--RPM,a,x,x,x,x,A*hh<CR><LF>
           Field Number:
00060
              1) Sourse, S = Shaft, E = Engine
00061
00062
              2) Engine or shaft number
              3) Speed, Revolutions per minute
4) Propeller pitch, % of maximum, "-" means astern
5) Status, A means data is valid
00063
00064
00065
00066
              6) Checksum
00067
00068 */
00069
00075
00076 String sendXDR()
00077 {
00078
         String HexCheckSum;
00079
         String NMEASensor;
08000
         String SendSensor;
00081
           NMEASensor = "IIXDR,A,"; //NMEASensor = "IIXDR,A," + String(SensorID);
//NMEASensorKraeng += ",";
NMEASensor += String(fGaugeDrehzahl);
NMEASensor += ",D,ROLL";
00082
00083
00084
00085
00086
         // Build CheckSum
00087
00088
         HexCheckSum = String(CheckSum(NMEASensor), HEX);
00089
         // Build complete NMEA string
         SendSensor = "$" + NMEASensor;
SendSensor += "*";
00090
00091
00092
         SendSensor += HexCheckSum;
00093
00094
         Serial.println(SendSensor);
00095
00096
         return SendSensor;
00097 }
00098
00104
00105 String sendRPM()
00106 {
         String HexCheckSum;
```

```
00108
         String NMEASensor;
00109
         String SendSensor;
00110
           NMEASensor = "IIRPM,E,1,"; //NMEASensor = "IIXDR,E,1," + String(SensorID);
00111
           NMEASensor += String(fGaugeDrehzahl);
NMEASensor += ",15,A";
00112
00113
00114
00115
00116
        HexCheckSum = String(CheckSum(NMEASensor), HEX);
        // Build complete NMEA string
SendSensor = "$" + NMEASensor;
SendSensor += "*";
00117
00118
00119
         SendSensor += HexCheckSum;
00120
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

Wert.

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

Definiert in Zeile 12 der Datei task.h.

7.36.1.5 taskWaitFor

Wert:

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

Definiert in Zeile 13 der Datei task.h.

7.37 task.h 109

7.36.1.6 taskStepName

Wert:

```
TASKSTEP_##STEPNAME :
```

Definiert in Zeile 16 der Datei task.h.

7.36.1.7 taskJumpTo

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() }
80000
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
\tt 00015 // Benennen und anspringen von Schrittketten Verzweigungen
00016 #define taskStepName(STEPNAME) TASKSTEP_##STEPNAME : 00017 #define taskJumpTo(STEPNAME) goto TASKSTEP_##STEPNAME
00018
00019 #endif
```

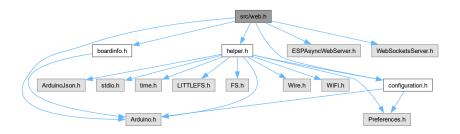
7.38 src/web.h-Dateireferenz

Webseite Variablen lesen und schreiben, Webseiten erstellen.

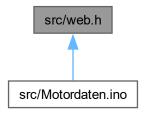
```
#include "helper.h"
#include "configuration.h"
#include "boardinfo.h"
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>
```

#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 ( {\rm const\ String\ \&\ } var)
```

Definiert in Zeile 29 der Datei web.h.

```
buttons += "<label>IP </label><input type = \"text\" name = \"IP\"
       value=\"";
                buttons += tAP_Config.wAP_IP;
buttons += "\"/>";
00039
00040
       buttons += "<label>Password </label><input type = \"text\" name = \"Password\" value=\"";
00041
00042
                buttons += tAP_Config.wAP_Password;
00043
                buttons += "\"/>";
       buttons += "\"/>";
buttons += "<label>Oil Offset </label><input type = \"text\" name =
\"MotorOffset\" value=\"";
buttons += tAP_Config.wMotor_Offset;</pre>
00044
00045
       buttons += "\"/> °C";
buttons += "\p class=\"CInput\"><label>K&uuml;hlwasser Offset </label><input type = \"text\"
name = \"CoolantOffset\" value=\"";
00046
00047
                buttons += tAP_Config.wCoolant_Offset;
buttons += "\"/> °C";
buttons += "p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name</pre>
00048
00049
00050
         \"Fuellstandmax\" value=\"";
00051
               buttons += tAP_Config.wFuellstandmax;
       buttons += "\"/> 1";
buttons += "class=\"CInput\"><label>ADC1 Kalibrierung </label><input type = \"text\" name = \"ADC1_Cal\" value=\"";</pre>
00052
                buttons += tAP_Config.wADC1_Cal;
buttons += "\"/>";
00054
00055
                buttons += "<label>ADC2 Kalibrierung </label><input type = \"text\" name =
00056
       \"ADC2_Cal\" value=\"";
00057
                buttons += tAP_Config.wADC2_Cal;
                buttons += "\"/>";
00058
                buttons += "<"/>/>;
buttons += "<input type=\"submit\" value=\"Speichern\">";
buttons += "</form>";
00059
00060
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);
                if (var == "sFuellstand") return String(FuelLevel, 1);
if (var == "sFuellstandmax") return String(FuelLevelMax, 1);
if (var == "sBordspannung") return String(fBordSpannung, 1);
if (var == "sCoolantTemp") return String(fCoolantTemp, 1);
00072
00073
00074
00075
                     (var == "sMotorTemp") return String(fMotorTemp, 1);
                if
00077
                      (var == "sCoolantOffset") return String(fCoolantOffset);
00078
                 if (var == "sMotorOffset") return String(fMotorOffset);
                if (var == "sMotorError") return String(motorErrorReported);
00079
                if (var == "sCoolantError") return String(coolantErrorReported);
08000
                if (var == "sBoardInfo") return string(coordinatifforkeported
if (var == "sBoardInfo") return sBoardInfo;
if (var == "sADC1_Cal") return String(ADC_Calibration_Value1);
if (var == "sADC2_Cal") return String(ADC_Calibration_Value2);
00081
00082
00083
                if (var == "sHeapspace") return sHeapspace;
if (var == "sFS_USpace") return String(LittleFS.usedBytes());
if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00084
00085
00086
                if (var == "sAP_IP") return WiFi.softAPIP().toString();
if (var == "sAP_Clients") return String(sAP_Station);
00087
00088
                if (var == "sCL_Addr") return WiFi.localIP().toString();
00089
```

```
00090     if (var == "sCL_Status") return String(sCL_Status);
00091     if (var == "sOneWire_Status") return String(sOneWire_Status);
00092     if (var == "sVersionS") return VersionSoftware;
00093     if (var == "sVersionH") return VersionHardware;
00094     if (var == "sCounter") return String(Counter);
00095     if (var == "CONFIGPLACEHOLDER") return processor(var);
00096     return "NoVariable";
```

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 99 der Datei web.h.

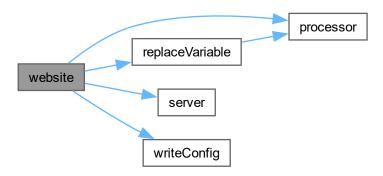
```
00100
              server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request){
00101
                   request->send(LittleFS, "/favicon.ico", "image/x-icon");
00102
              server.on("/logo80.jpg", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(LittleFS, "/logo80.jpg", "image/jpg");
00103
00104
00105
              });
00106
              server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {
00107
                   request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00108
              server.on("/system.html", HTTP_GET, [](AsyncWebServerRequest* request) {
    request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00109
00110
00111
              });
              server.on("/settings.html", HTTP_GET, [](AsyncWebServerRequest* request) {
   request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00112
00113
00114
              server.on("/werte.html", HTTP_GET, [](AsyncWebServerRequest* request) {
   request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00115
00116
00117
              server.on("/ueber.html", HTTP_GET, [](AsyncWebServerRequest* request) {
    request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00118
00119
00120
              server.on("/reboot", HTTP_GET, [](AsyncWebServerRequest * request) {
   request->send(LittleFS, "/reboot.html", String(), false, processor);
00121
00122
00123
                    IsRebootRequired = true;
00124
00125
              server.on("/gauge.min.js", HTTP_GET, [](AsyncWebServerRequest* request) {
```

```
request->send(LittleFS, "/gauge.min.js");
00127
           server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
   request->send(LittleFS, "/style.css", "text/css");
00128
00129
00130
           });
           server.on("/settings.html", HTTP_POST, [](AsyncWebServerRequest *request){
00131
00132
           // Body wird asynchron empfangen!
00133
                }, NULL, [](AsyncWebServerRequest *request, uint8_t *data, size_t len, size_t index, size_t
     total) {
00134
00135
           String json = "";

for (size_t i = 0; i < len; i++) {

    json += (char)data[i];
00136
00137
           }
00138
                    Serial.println("Empfangenes JSON (Body):");
00139
                    Serial.println(json);
00140
                    writeConfig(json);
                    request->send(200, "text/plain", "Daten gespeichert");
00141
00142 });
00143 }
```

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.

7.39 web.h 115

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
               String buttons = "";
buttons += "<form onsubmit = \"event.preventDefault(); formToJson(this);\">";
buttons += "<label>SSID </label><input type = \"text\" name = \"SSID\"</pre>
00033
00034
00035
      value=\"";
               buttons += tAP_Config.wAP_SSID;
buttons += "\"/>";
00036
00037
00038
                \label = "IP </label><input type = \"text\" name = \"IP\"
      value=\"";
               buttons += tAP_Config.wAP_IP;
buttons += "\"/>";
buttons += "class=\"CInput\"><label>Password </label><input type = \"text\" name =</pre>
00039
00040
00041
       \"Password\" value=\"";
```

```
00042
                 buttons += tAP_Config.wAP_Password;
       buttons += "\"/>";
buttons += "<label>Oil Offset </label><input type = \"text\" name =
\"MotorOffset\" value=\"";</pre>
00043
00044
                buttons += tAP_Config.wMotor_Offset;
buttons += "\"/> °C";
buttons += "<label>K&uuml;hlwasser Offset </label><input type = \"text\"</pre>
00045
00046
       name = \"CoolantOffset\" value=\"";
                 buttons += tAP_Config.wCoolant_Offset;
buttons += "\"/> °C";
00048
00049
                 buttons += "<label>max. F&uuml; llstand </label><input type = \"text\" name
00050
       = \"Fuellstandmax\" value=\"";
buttons += tAP_Config.wFuellstandmax;
buttons += "\"/> 1";
00051
00052
                 buttons += "<label>ADC1 Kalibrierung </label><input type = \"text\" name =
00053
       \"ADC1_Cal\" value=\"";
00054
                buttons += tAP_Config.wADC1_Cal;
                 buttons += "\"/>";
00055
                 buttons += "<label>ADC2 Kalibrierung </label><input type = \"text\" name =
00056
       \"ADC2_Cal\" value=\"";
                 buttons += tAP_Config.wADC2_Cal;
buttons += "\"/>";
00057
00058
                 buttons += "<input type=\"submit\" value=\"Speichern\">";
buttons += "</form>";
00059
00060
00061
                 return buttons;
00062
00063
            return String();
00064 }
00065
00066 //Variables for website
00067 String sCL_Status = sWifiStatus(WiFi.status());
00069 String replaceVariable(const String& var)
00070 {
            if (var == "sDrehzahl") return String(fDrehzahl, 1);
if (var == "sFuellstand") return String(FuelLevel, 1);
00071
00072
00073
            if (var == "sFuellstandmax") return String(FuelLevelMax, 1);
            if (var == "sBordspannung") return String(fBordSpannung, 1);
00075
            if (var == "sCoolantTemp") return String(fCoolantTemp, 1);
00076
            if (var == "sMotorTemp") return String(fMotorTemp, 1);
            if (var == "sCoolantOffset") return String(fCoolantOffset);
if (var == "sMotorOffset") return String(fMotorOffset);
if (var == "sMotorError") return String(motorErrorReported);
00077
00078
00079
            if (var == "sCoolantError") return String(coolantErrorReported);
00080
                (var == "sBoardInfo") return sBoardInfo;
00081
            if
            if (var == "sADC1_Cal") return String(ADC_Calibration_Value1);
if (var == "sADC2_Cal") return String(ADC_Calibration_Value2);
if (var == "sHeapspace") return String(ADC_Calibration_Value2);
if (var == "sFS_USpace") return String(LittleFS.usedBytes());
00082
00083
00084
00085
            if (var == "sFS_TSpace") return String(LittleFS.totalBytes());
00086
                (var == "sAP_IP") return WiFi.softAPIP().toString();
            if
00088
            if (var == "sAP_Clients") return String(sAP_Station);
00089
            if (var == "sCL_Addr") return WiFi.localIP().toString();
            if (var == "sCL_Status") return String(sCL_Status);
if (var == "sOneWire_Status") return String(sOneWire_Status);
00090
00091
00092
            if (var == "sVersionS") return VersionSoftware;
            if (var == "sVersionH") return VersionHardware;
            if (var == "sCounter") return String(Counter);
00094
            if (var == "CONFIGPLACEHOLDER") return processor(var);
return "NoVariable";
00095
00096
00097 }
00098
00099 void website() {
00100
           server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request){
00101
                 request->send(LittleFS, "/favicon.ico", "image/x-icon");
00102
00103
            server.on("/logo80.jpg", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(LittleFS, "/logo80.jpg", "image/jpg");
00104
00105
00106
            server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {
00107
                request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00108
            server.on("/system.html", HTTP_GET, [](AsyncWebServerRequest* request) {
    request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00109
00110
00111
            server.on("/settings.html", HTTP_GET, [](AsyncWebServerRequest* request) {
00112
00113
                 request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00114
            server.on("/werte.html", HTTP_GET, [](AsyncWebServerRequest* request) {
   request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00115
00116
00117
            server.on("/ueber.html", HTTP_GET, [](AsyncWebServerRequest* request) {
    request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00118
00119
00120
            server.on("/reboot", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(LittleFS, "/reboot.html", String(), false, processor);
00121
00122
00123
                  IsRebootRequired = true;
```

7.39 web.h 117

```
00124
             });
             server.on("/gauge.min.js", HTTP_GET, [](AsyncWebServerRequest* request) {
   request->send(LittleFS, "/gauge.min.js");
00125
00126
00127
             server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(LittleFS, "/style.css", "text/css");
});
00128
00129
00130
00131
             server.on("/settings.html", HTTP_POST, [](AsyncWebServerRequest *request){
      }, NULL, [](AsyncWebServerRequest *request, uint8_t *data, size_t len, size_t index, size_t total) {
00132
00133
             String json = "";

for (size_t i = 0; i < len; i++) {

    json += (char)data[i];
00134
00135
00136
00137
00138
                       Serial.println("Empfangenes JSON (Body):");
                       Serial.println(json);
writeConfig(json);
request->send(200, "text/plain", "Daten gespeichert");
00139
00140
00142 });
00143 }
00144
```

Index

ADC_Calibration_Value1	BordSpannung
configuration.h, 43	configuration.h, 45
ADC_Calibration_Value2	bsz1
configuration.h, 44	hourmeter.h, 65
ADCpin1	BUF
Motordaten.ino, 95	BoardInfo.cpp, 30
ADCpin2	
Motordaten.ino, 95	channel
Altitude	configuration.h, 42
tBoatData, 18	CheckSum
AP_IP	NMEA0183Telegram.h, 103
configuration.h, 43	chipid
AP_PASSWORD	configuration.h, 41
configuration.h, 42	CL_IP
AP_SSID	configuration.h, 43
configuration.h, 42	CL_PASSWORD
	configuration.h, 39
Bat1Capacity	CL_SSID
configuration.h, 46	configuration.h, 39
Bat2Capacity	COG
configuration.h, 46	tBoatData, 17
BatSoC	configuration.h
configuration.h, 46	ADC_Calibration_Value1, 43
BatteryDCSendOffset	ADC_Calibration_Value2, 44
configuration.h, 38	AP_IP, 43
BatteryDCStatusSendOffset	AP_PASSWORD, 42
configuration.h, 38	AP_SSID, 42
baudrate	Bat1Capacity, 46
Motordaten.ino, 95	Bat2Capacity, 46
bClientConnected	BatSoC, 46
configuration.h, 43	BatteryDCSendOffset, 38
bConnect_CL	BatteryDCStatusSendOffset, 38
configuration.h, 43	bClientConnected, 43
bl2C_Status	bConnect_CL, 43
configuration.h, 44	bl2C_Status, 44
Blue	BordSpannung, 45
LED.h, 68	channel, 42
BoardInfo, 13	chipid, 41
BoardInfo, 13	CL_IP, 43
m_chipid, 15	CL_PASSWORD, 39
m_chipinfo, 15	CL_SSID, 39
ShowChipID, 14	coolantErrorReported, 46
ShowChipIDtoString, 14	CoolantTemp, 45
ShowChipInfo, 14	Counter, 46
ShowChipTemperature, 14	dMWV_WindDirectionT, 48
boardInfo	dMWV_WindSpeedM, 48
web.h, 115	DNS_PORT, 40
BoardInfo.cpp	dVWR_WindAngle, 48
BUF, 30	dVWR_WindDirectionM, 48
temprature sens read, 30	dVWR WindSpeedkn, 48

dVWR_WindSpeedms, 48	udpAddress, 48
Eingine_RPM_Pin, 40	udpPort, 49
EngineOn, 45	VersionHardware, 37
EngineRPM, 45	VersionSoftware, 37
EngineSendOffset, 38	WEB_TITEL, 39
EngineStatus, 40	coolantErrorReported
ESP32_CAN_RX_PIN, 38	configuration.h, 46
ESP32_CAN_TX_PIN, 37	CoolantTemp
fbmp_altitude, 44	configuration.h, 45
fbmp_pressure, 44	Counter
fbmp_temperature, 44	configuration.h, 46
fBordSpannung, 47	CounterOld
fCoolantOffset, 47	hourmeter.h, 65
fCoolantTemp, 47	,
fDrehzahl, 47	data/index.html, 23
fGaugeDrehzahl, 47	data/reboot.html, 25
fMotorOffset, 47	data/settings.html, 26
fMotorTemp, 47	data/system.html, 27
FuelLevel, 45	data/ueber.html, 27
FuelLevelMax, 45	data/werte.html, 28
Gateway, 42	DaysSince1970
hide SSID, 42	tBoatData, 17
HostName, 39	debug_log
i, 41	Motordaten.ino, 78
I2C_SCL, 39	DGPSAge
I2C_SDA, 39	tBoatData, 18
id, 41	DGPSReferenceStationID
	tBoatData, 20
iDistance, 45	dMWV_WindDirectionT
iMaxSonar, 44	configuration.h, 48
IP, 42	dMWV_WindSpeedM
iSTA_on, 43	
max_connection, 42	configuration.h, 48
motorErrorReported, 46	DNS_PORT
MotorTemp, 45	configuration.h, 40 dVWR WindAngle
N2K_SOURCE, 38	configuration.h, 48
NMask, 42	•
NodeAddress, 41	dVWR_WindDirectionM
Off, 41	configuration.h, 48
On, 41	dVWR_WindSpeedkn
ONE_WIRE_BUS, 40	configuration.h, 48
PAGE_REFRESH, 39	dVWR_WindSpeedms
preferences, 41	configuration.h, 48
RPM_Calibration_Value, 40	Eingine RPM Pin
RPMSendOffset, 38	configuration.h, 40
sAP_Station, 43	ENABLE_DEBUG_LOG
SEALEVELPRESSURE_HPA, 40	Motordaten.ino, 78
SELF_IP, 43	EngineHours
SERVER_HOST_NAME, 40	•
sHeapspace, 41	hourmeter.h, 64
sI2C_Status, 44	EngineOn
SlowDataUpdatePeriod, 39	configuration.h, 45
SoCError, 46	EngineRPM
sOneWire_Status, 46	configuration.h, 45
sOrient, 48	EngineSendOffset
sSTBB, 47	configuration.h, 38
TankSendOffset, 38	EngineStatus
tAP_Config, 41	configuration.h, 40
TCP_PORT, 40	ErrorOff
	LEDindicator.h, 75

ErrorOn	sWifiStatus, 58
LEDindicator.h, 75	toChar, 58
ESP32_CAN_RX_PIN	WiFiDiag, 53
configuration.h, 38	writeConfig, 56
ESP32_CAN_TX_PIN	hide_SSID
configuration.h, 37	configuration.h, 42
coringarationini, or	HostName
fbmp_altitude	configuration.h, 39
configuration.h, 44	hourmeter.h
fbmp_pressure	
configuration.h, 44	bsz1, 65
fbmp_temperature	CounterOld, 65
configuration.h, 44	EngineHours, 64
fBordSpannung	lastRun, 65
configuration.h, 47	laststate1, 66
fCoolantOffset	milliRest, 66
	state1, 66
configuration.h, 47	:
fCoolantTemp	i
configuration.h, 47	configuration.h, 41
fDrehzahl	I2C_scan
configuration.h, 47	helper.h, 57
fGaugeDrehzahl	I2C_SCL
configuration.h, 47	configuration.h, 39
flashLED	I2C_SDA
LED.h, 69	configuration.h, 39
fMotorOffset	id
configuration.h, 47	configuration.h, 41
fMotorTemp	iDistance
configuration.h, 47	configuration.h, 45
freeHeapSpace	iMaxSonar
helper.h, 52	configuration.h, 44
FuelLevel	InitNextUpdate
configuration.h, 45	Motordaten.ino, 84
FuelLevelMax	IP
configuration.h, 45	configuration.h, 42
oomigaratoriii, io	IsRebootRequired
Gateway	web.h, 115
configuration.h, 42	iSTA_on
GeoidalSeparation	configuration.h, 43
tBoatData, 18	IsTimeToUpdate
GetTemperature	Motordaten.ino, 83
Motordaten.ino, 82	wotordateri.irio, 83
GPSQualityIndicator	Last_int_time
tBoatData, 19	Motordaten.ino, 94
GPSTime	lastRun
tBoatData, 18	hourmeter.h, 65
Green	laststate1
LED.h, 68	hourmeter.h, 66
handleInterrupt	Latitude
Motordaten.ino, 78	tBoatData, 18
	LED
HDOP	LED.h, 68
tBoatData, 18	LED.h
helper.h	Blue, 68
freeHeapSpace, 52	flashLED, 69
I2C_scan, 57	Green, 68
listDir, 54	LED, 68
readConfig, 55	LEDblink, 69
ShowTime, 52	LEDBoard, 68

LEDflash, 69	loop, 91
LEDInit, 70	MotorCoolant, 94
LEDoff, 71	MotorOil, 95
LEDoff RGB, 71	mux, 94
LEDon, 70	oneWire, 78, 94
Red, 68	PeriodCount, 94
LEDblink	PROGMEM, 93
-	ReadRPM, 83
LED.h, 69 LEDBoard	ReadVoltage, 91
	g ,
LED.h, 68	rs_config, 95
LEDflash	SendN2kBattery, 86
LED.h, 69	SendN2kDCStatus, 85
LEDindicator.h	SendN2kEngineData, 88
ErrorOff, 75	SendN2kEngineRPM, 90
ErrorOn, 75	SendN2kTankLevel, 87
LoopIndicator, 74	SetNextUpdate, 85
LEDInit	setup, 78
LED.h, 70	StartValue, 94
LEDoff	Task1, 95
LED.h, 71	timer, 94
LEDoff RGB	motorErrorReported
LED.h, 71	configuration.h, 46
LEDon	MotorOil
LED.h, 70	Motordaten.ino, 95
listDir	
	MotorTemp
helper.h, 54	configuration.h, 45
Longitude	mux
tBoatData, 18	Motordaten.ino, 94
loop	New Course
Motordaten.ino, 91	N2K_SOURCE
LoopIndicator	configuration.h, 38
LEDindicator.h, 74	NMask
	configuration.h, 42
m_chipid	NMEA0183Telegram.h
BoardInfo, 15	CheckSum, 103
m_chipinfo	sendRPM, 105
BoardInfo, 15	sendXDR, 104
max_connection	NodeAddress
configuration.h, 42	configuration.h, 41
milliRest	gerane, , ,
hourmeter.h, 66	Off
MKSPIFFSTOOL	configuration.h, 41
replace_fs, 11	Offset
MOBActivated	tBoatData, 19
tBoatData, 20	On
	configuration.h, 41
MotorCoolant	
Motordaten.ino, 94	ONE_WIRE_BUS
MotorData NMEA2000, 1	configuration.h, 40
Motordaten.ino	oneWire
ADCpin1, 95	Motordaten.ino, 78, 94
ADCpin2, 95	
baudrate, 95	PAGE_REFRESH
debug_log, 78	configuration.h, 39
ENABLE_DEBUG_LOG, 78	PeriodCount
GetTemperature, 82	Motordaten.ino, 94
handleInterrupt, 78	preferences
InitNextUpdate, 84	configuration.h, 41
IsTimeToUpdate, 83	processor
Last_int_time, 94	web.h, 111
∟asi_iii∟iiii⊡, 34	',

PROGMEM	configuration.h, 41
Motordaten.ino, 93	ShowChipID
	BoardInfo, 14
readConfig	ShowChipIDtoString
helper.h, 55	BoardInfo, 14
README.md, 28	ShowChipInfo
ReadRPM	BoardInfo, 14
Motordaten.ino, 83	ShowChipTemperature
ReadVoltage	BoardInfo, 14
Motordaten.ino, 91	ShowTime
Red	helper.h, 52
LED.h, 68	sl2C_Status
replace_fs, 11	configuration.h, 44
MKSPIFFSTOOL, 11	SlowDataUpdatePeriod
replace_fs.py, 28	configuration.h, 39
replaceVariable	SoCError
web.h, 112	configuration.h, 46
RPM_Calibration_Value	SOG
configuration.h, 40	tBoatData, 17
RPMSendOffset	sOneWire Status
configuration.h, 38	configuration.h, 46
rs_config	sOrient
Motordaten.ino, 95	configuration.h, 48
	src/BoardInfo.cpp, 29, 30
sAP_Station	src/BoardInfo.h, 32, 33
configuration.h, 43	src/BoatData.h, 33, 34
SatelliteCount	src/configuration.h, 34, 49
tBoatData, 20	src/helper.h, 51, 59
sBoardInfo	src/hourmeter.h, 62, 66
web.h, 114	src/LED.h, 67, 71
sCL Status	src/LEDindicator.h, 72, 75
web.h, 115	
SEALEVELPRESSURE HPA	src/Motordaten.ino, 76, 96
configuration.h, 40	src/NMEA0183Telegram.h, 102, 106
SELF IP	src/task.h, 107, 109
configuration.h, 43	src/web.h, 109, 115
SendN2kBattery	sSTBB
Motordaten.ino, 86	configuration.h, 47
SendN2kDCStatus	StartValue
Motordaten.ino, 85	Motordaten.ino, 94
SendN2kEngineData	state1
Motordaten.ino, 88	hourmeter.h, 66
SendN2kEngineRPM	Status
Motordaten.ino, 90	tBoatData, 20
	sWifiStatus
SendN2k Janki evel	
SendN2kTankLevel Motordaten inc. 87	helper.h, 58
Motordaten.ino, 87	helper.h, 58
Motordaten.ino, 87 sendRPM	helper.h, 58 TankSendOffset
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105	helper.h, 58 TankSendOffset configuration.h, 38
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME configuration.h, 40	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108 taskJumpTo, 109
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME configuration.h, 40 SetNextUpdate	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108 taskJumpTo, 109 taskPause, 108
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME configuration.h, 40 SetNextUpdate Motordaten.ino, 85	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108 taskJumpTo, 109 taskPause, 108 taskStepName, 108
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME configuration.h, 40 SetNextUpdate Motordaten.ino, 85 setup	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108 taskJumpTo, 109 taskPause, 108 taskStepName, 108 taskSwitch, 108
Motordaten.ino, 87 sendRPM NMEA0183Telegram.h, 105 sendXDR NMEA0183Telegram.h, 104 server web.h, 111 SERVER_HOST_NAME configuration.h, 40 SetNextUpdate Motordaten.ino, 85	helper.h, 58 TankSendOffset configuration.h, 38 tAP_Config configuration.h, 41 task.h taskBegin, 108 taskEnd, 108 taskJumpTo, 109 taskPause, 108 taskStepName, 108

Motordaten.ino, 95	tBoatData, 17
taskBegin	VersionHardware
task.h, 108	configuration.h, 37
taskEnd	VersionSoftware
task.h, 108	configuration.h, 37
taskJumpTo	1001.01
task.h, 109	wADC1_Cal
taskPause	Web_Config, 21
task.h, 108	wADC2_Cal
taskStepName	Web_Config, 21
task.h, 108	wAP_IP
taskSwitch	Web_Config, 21
task.h, 108	wAP_Password
taskWaitFor	Web_Config, 21
task.h, 108	wAP_SSID
tBoatData, 16	Web_Config, 21
Altitude, 18	WaterDepth
COG, 17	tBoatData, 19
DaysSince1970, 17	WaterTemperature
DGPSAge, 18	tBoatData, 18
DGPSReferenceStationID, 20	wCoolant_Offset
GeoidalSeparation, 18	Web_Config, 21
GPSQualityIndicator, 19	web.h
GPSTime, 18	boardInfo, 115
HDOP, 18	IsRebootRequired, 115
Latitude, 18	processor, 111
Longitude, 18	replaceVariable, 112
MOBActivated, 20	sBoardInfo, 114
Offset, 19	sCL_Status, 115
SatelliteCount, 20	server, 111
SOG, 17	website, 113
Status, 20	webSocket, 114
tBoatData, 17	Web_Config, 20
TrueHeading, 17	wADC1_Cal, 21
Variation, 17	wADC2_Cal, 21
WaterDepth, 19	wAP IP, 21
•	wAP_Password, 21
WaterTemperature, 18	wAP_SSID, 21
WindAngle, 19 WindDirectionM, 19	wCoolant_Offset, 21
•	wFuellstandmax, 21
WindDirectionT, 19	wMotor_Offset, 21
WindSpeedK, 19	WEB TITEL
WindSpeedM, 19	configuration.h, 39
TCP_PORT	website
configuration.h, 40	web.h, 113
temprature_sens_read	webSocket
BoardInfo.cpp, 30	web.h, 114
timer	
Motordaten.ino, 94	wFuellstandmax
	wFuellstandmax Web Config 21
toChar	Web_Config, 21
toChar helper.h, 58	Web_Config, 21 WiFiDiag
toChar helper.h, 58 TrueHeading	Web_Config, 21 WiFiDiag helper.h, 53
toChar helper.h, 58	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle
toChar helper.h, 58 TrueHeading tBoatData, 17	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19
toChar helper.h, 58 TrueHeading tBoatData, 17 udpAddress	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19 WindDirectionM
toChar helper.h, 58 TrueHeading tBoatData, 17 udpAddress configuration.h, 48	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19 WindDirectionM tBoatData, 19
toChar helper.h, 58 TrueHeading tBoatData, 17 udpAddress configuration.h, 48 udpPort	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19 WindDirectionM tBoatData, 19 WindDirectionT
toChar helper.h, 58 TrueHeading tBoatData, 17 udpAddress configuration.h, 48	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19 WindDirectionM tBoatData, 19 WindDirectionT tBoatData, 19
toChar helper.h, 58 TrueHeading tBoatData, 17 udpAddress configuration.h, 48 udpPort	Web_Config, 21 WiFiDiag helper.h, 53 WindAngle tBoatData, 19 WindDirectionM tBoatData, 19 WindDirectionT

WindSpeedM tBoatData, 19 wMotor_Offset Web_Config, 21 writeConfig helper.h, 56