Motordaten

V 2.5

Erzeugt von Doxygen 1.13.2

Kapitel 1

MotorData NMEA2000

1.1 Description

This repository shows how to measure the

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

and send it as NNMEA2000 meassage.

- PGN 127488 // Engine Rapid / RPM
- PGN 127489 // Engine parameters dynamic
- PGN 127505 // Fluid Level

- PGN 127506 // Battery
- PGN 127508 // Battery Status

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

Doxygen Documentation

1.2 Based on the work of

NMEA2000-Data-Sender @AK-Homberger

NMEA 2000 @ttlappalainen

This project is part of OpenBoatProject

1.3 Website

1.4 Plotter

1.5 Wiring diagram

1.6 PCB Layout

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

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

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

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

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

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

It's also possible with Unisensor case.

- UNI sensor Link

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

1.7 Partlist:

- PCB by Aisler Link

Assembly: MD N2k__Assembly.pdf

- 1 C1 10µ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 2 C2 22µ CP_EIA-7343-15_Kemet-W_Pad2.25x2.55mm_HandSolder 1
- 3 R1 100k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 4 R2 27k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1

- 5 R3 300R R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 6 R4 10k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 7 R5 1k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 8 R6 4k7 R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 9 R7 2k R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal 1
- 10 D1 B360 B 360 F Schottkydiode, 60 V, 3 A, DO-214AB/SMC 1
- 11 D2 LED_RBKG RGB LED Kingbright 1
- 12 D3 PESD1CAN SOT-23 Dual bidirectional TVS diode 1
- 13 D4 ZPD3.3 D_DO-35_SOD27_P10.16mm_Horizontal 1 Link
- 14 D5 1N4148 D_DO-35_SOD27_P7.62mm_Horizontal 1 Link
- 15 D6 P4SMAJ26CA D_SMA_TVS 1
- 16 U1 TSR_1-2450 Converter_DCDC_TRACO_TSR-1_THT 1 Link
- 17 U2 ESP32-Huzzah Adafruit_ESP32 1
- 18 U3 SN65HVD230 SOIC-8_3.9x4.9mm_P1.27mm 1 Link
- 19 U4 H11L1 DIP-6_W7.62mm 1 Link
- 20 FL1 EPCO B82789C0513 B82789C0113N002 1
- 21 J2, J3 Conn_01x04_Pin PinHeader_1x04_P2.54mm_Vertical 2
- 22 J1 Conn_01x03_Pin PinHeader_1x03_P2.54mm_Vertical 1
- 23 Wago-Case: Link

1.8 Changes

- Version 2.6 Add value for ADC calibration to setting.html
- Version 2.5 Error handling OneWire-Temperatur (set sensor output to -5℃) and change PIN to GPIO14
- Version 2.4 add Doxygen
- Version 2.3 add Temperatur: Motor(Water)temp and OilTemp (2x OneWire), add Alarm Watertemp
- Version 2.2 add Motorparameter: EngineHours and Alarms (Oiltemp max / Engine Stop)
- Version 2.1 Minor updates website, change Engine Parameter to PGN127489 (Oil Temp)
- Version 2.0
  - update Website (code and html files)
  - change Hardware layout, add protection's and C's on Voltage input, add protection's for CanBus
  - change Webinterface, add calibration-offset for temperature

Kapitel 2

Verzeichnis der Namensbereiche

2.1 Liste aller Namensbereiche

Liste aller Namensbereiche mit Kurzbeschreibung:

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

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

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 5

Dokumentation der Namensbereiche

5.1 replace_fs-Namensbereichsreferenz

Variablen

- MKSPIFFSTOOL

5.1.1 Variablen-Dokumentation

5.1.1.1 MKSPIFFSTOOL

`replace_fs.MKSPIFFSTOOL`

Definiert in Zeile 3 der Datei replace_fs.py.

Kapitel 6

Klassen-Dokumentation

6.1 BoardInfo Klassenreferenz

`#include <BoardInfo.h>`

Öffentliche Methoden

- BoardInfo ()

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

Geschützte Attribute

- uint64_t m_chipid
- esp_chip_info_t m_chipinfo

6.1.1 Ausführliche Beschreibung

Definiert in Zeile 16 der Datei BoardInfo.h.

6.1.2 Beschreibung der Konstruktoren und Destruktoren

6.1.2.1 BoardInfo()

`BoardInfo::BoardInfo ()`

Construct a new Board Info:: Board Info object.

Definiert in Zeile 36 der Datei BoardInfo.cpp.

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

6.1.3 Dokumentation der Elementfunktionen

6.1.3.1 ShowChipID()

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

Definiert in Zeile 47 der Datei BoardInfo.cpp.

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

6.1.3.2 ShowChipInfo()

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

Definiert in Zeile 100 der Datei BoardInfo.cpp.

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

6.1.3.3 ShowChipTemperature()

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

Definiert in Zeile 119 der Datei BoardInfo.cpp.

```
00120 {
00121     uint8_t temp_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         return;
00128     }
00129     temp_celsius = ( temp_farenheit - 32 ) / 1.8;
00130     Serial.printf("Temperatur Board: %i Fahrenheit\n", temp_farenheit);
00131     Serial.printf("Temperatur Board: %.1f °C\n", temp_celsius);
00132 }
```

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

6.1.3.4 ShowChipIDtoString()

```
String BoardInfo::ShowChipIDtoString ()
```

Definiert in Zeile 61 der Datei BoardInfo.cpp.

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

6.1.4 Dokumentation der Datenelemente

6.1.4.1 m_chipid

```
uint64_t BoardInfo::m_chipid  [protected]
```

Definiert in Zeile 28 der Datei BoardInfo.h.

6.1.4.2 m_chipinfo

```
esp_chip_info_t BoardInfo::m_chipinfo  [protected]
```

Definiert in Zeile 29 der Datei BoardInfo.h.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Dateien:

- src/BoardInfo.h
- src/BoardInfo.cpp

6.2 tBoatData Strukturreferenz

`#include <BoatData.h>`

Öffentliche Methoden

- tBoatData ()

Öffentliche Attribute

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

6.2.1 Ausführliche Beschreibung

Definiert in Zeile 4 der Datei BoatData.h.

6.2.2 Beschreibung der Konstruktoren und Destruktoren

6.2.2.1 tBoatData()

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

Definiert in Zeile 18 der Datei BoatData.h.

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

6.2.3 Dokumentation der Datenelemente

6.2.3.1 DaysSince1970

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

Definiert in Zeile 5 der Datei BoatData.h.

6.2.3.2 TrueHeading

```
double tBoatData::TrueHeading
```

Definiert in Zeile 7 der Datei BoatData.h.

6.2.3.3 SOG

```
double tBoatData::SOG
```

Definiert in Zeile 7 der Datei BoatData.h.

6.2.3.4 COG

```
double tBoatData::COG
```

Definiert in Zeile 7 der Datei BoatData.h.

6.2.3.5 Variation

```
double tBoatData::Variation
```

Definiert in Zeile 7 der Datei BoatData.h.

6.2.3.6 GPSTime

```
double tBoatData::GPSTime
```

Definiert in Zeile 8 der Datei BoatData.h.

6.2.3.7 Latitude

```
double tBoatData::Latitude
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.8 Longitude

```
double tBoatData::Longitude
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.9 Altitude

```
double tBoatData::Altitude
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.10 HDOP

```
double tBoatData::HDOP
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.11 GeoidalSeparation

```
double tBoatData::GeoidalSeparation
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.12 DGPSAge

```
double tBoatData::DGPSAge
```

Definiert in Zeile 9 der Datei BoatData.h.

6.2.3.13 WaterTemperature

```
double tBoatData::WaterTemperature
```

Definiert in Zeile 10 der Datei BoatData.h.

6.2.3.14 WaterDepth

```
double tBoatData::WaterDepth
```

Definiert in Zeile 10 der Datei BoatData.h.

6.2.3.15 Offset

```
double tBoatData::Offset
```

Definiert in Zeile 10 der Datei BoatData.h.

6.2.3.16 WindDirectionT

```
double tBoatData::WindDirectionT
```

Definiert in Zeile 11 der Datei BoatData.h.

6.2.3.17 WindDirectionM

```
double tBoatData::WindDirectionM
```

Definiert in Zeile 11 der Datei BoatData.h.

6.2.3.18 WindSpeedK

```
double tBoatData::WindSpeedK
```

Definiert in Zeile 11 der Datei BoatData.h.

6.2.3.19 WindSpeedM

```
double tBoatData::WindSpeedM
```

Definiert in Zeile 11 der Datei BoatData.h.

6.2.3.20 WindAngle

```
double tBoatData::WindAngle
```

Definiert in Zeile 12 der Datei BoatData.h.

6.2.3.21 GPSQualityIndicator

```
int tBoatData::GPSQualityIndicator
```

Definiert in Zeile 13 der Datei BoatData.h.

6.2.3.22 SatelliteCount

```
int tBoatData::SatelliteCount
```

Definiert in Zeile 13 der Datei BoatData.h.

6.2.3.23 DGPSReferenceStationID

```
int tBoatData::DGPSReferenceStationID
```

Definiert in Zeile 13 der Datei BoatData.h.

6.2.3.24 MOBActivated

```
bool tBoatData::MOBActivated
```

Definiert in Zeile 14 der Datei BoatData.h.

6.2.3.25 Status

```
char tBoatData::Status
```

Definiert in Zeile 15 der Datei BoatData.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/BoatData.h

6.3 Web_Config Strukturreferenz

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

Öffentliche Attribute

- char wAP_IP [20]
- char wAP_SSID [64]
- char wAP_Password [12]
- char wMotor_Offset [6]
- char wCoolant_Offset [6]
- char wFuellstandmax [6]
- char wADC1_Cal [6]
- char wADC2_Cal [6]

6.3.1 Ausführliche Beschreibung

Definiert in Zeile 47 der Datei configuration.h.

6.3.2 Dokumentation der Datenelemente

6.3.2.1 wAP_IP

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

Definiert in Zeile 49 der Datei configuration.h.

6.3.2.2 wAP_SSID

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

Definiert in Zeile 50 der Datei configuration.h.

6.3.2.3 wAP_Password

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

Definiert in Zeile 51 der Datei configuration.h.

6.3.2.4 wMotor_Offset

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

Definiert in Zeile 52 der Datei configuration.h.

6.3.2.5 wCoolant_Offset

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

Definiert in Zeile 53 der Datei configuration.h.

6.3.2.6 wFuellstandmax

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

Definiert in Zeile 54 der Datei configuration.h.

6.3.2.7 wADC1_Cal

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

Definiert in Zeile 55 der Datei configuration.h.

6.3.2.8 wADC2_Cal

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

Definiert in Zeile 56 der Datei configuration.h.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- src/configuration.h

Kapitel 7

Datei-Dokumentation

7.1 data/index.html-Dateireferenz

7.2 index.html

gehe zur Dokumentation dieser Datei

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

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

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

7.3 data/reboot.html-Dateireferenz

7.4 reboot.html

gehe zur Dokumentation dieser Datei

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

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

7.5 data/settings.html-Dateireferenz

7.6 settings.html

gehe zur Dokumentation dieser Datei

```
00001 <!DOCTYPE html>
00002 <html>
00003 <head>
00004     <title>Settings</title>
00005     <meta name="viewport" content="width=device-width, initial-scale=1">
00006     <link rel="icon" href="data:,">
00007     <link rel="stylesheet" type="text/css" href="style.css">
00008 </head>
00009 <body>
00010     <br />
00011     <p class="label"> K&uuml;hlwassertemperatur: %sCoolantTemp% &deg;C<br>
00012                     Offset: %sCoolantOffset% &deg;C<br>
00013                     Motortemperatur: %sMotorTemp% &deg;C<br>
00014                     Offset: %sMotorOffset% &deg;C</p>
00015     %CONFIGPLACEHOLDER%
00016     <script>
00017         function formToJson(form) {
00018             var xhr = new XMLHttpRequest();
00019             var SSID = form.SSID.value;
00020             var IP = form.IP.value;
00021             var Password = form.Password.value;
00022             var CoolantOffset = form.CoolantOffset.value;
00023             var MotorOffset = form.MotorOffset.value;
00024             var Fuellstandmax = form.Fuellstandmax.value;
00025             var ADC1_Cal = form.ADC1_Cal.value;
00026             var ADC2_Cal = form.ADC2_Cal.value;
00027 
00028             var jsonFormInfo = JSON.stringify({
00029                 SSID: SSID,
00030                 IP: IP,
00031                 Password: Password,
00032                 CoolantOffset: CoolantOffset,
00033                 MotorOffset: MotorOffset,
00034                 Fuellstandmax: Fuellstandmax,
00035                 ADC1_Cal: ADC1_Cal,
00036                 ADC2_Cal: ADC2_Cal
00037             });
00038 
00039             xhr.open("POST", "/settings.html?save=" + jsonFormInfo, true);
00040             /* window.alert("Json function send end"); */
00041             xhr.send();
00042             window.alert("Gespeichert!");
00043         }
00044     </script>
00045 
00046     <p class="label">Nach &Auml;nderungen neu starten!</p>
00047 
00048     <button class="button" onclick="reboot_handler()">Neustart</button>
00049 
00050     <p id="status"></p>
00051     <script>
00052         function reboot_handler()
00053         {
00054         document.getElementById("status").innerHTML = "Starte Reboot ...";
00055         var xhr = new XMLHttpRequest();
00056         xhr.open("GET", "/reboot", true);
00057         xhr.send();
00058         setTimeout(function(){ window.open("/reboot","_self"); }, 500);
00059         }
00060     </script>
00061 
00062 
00063     <ul class="bottomnav">
00064         <li><a href="/">Home</a></li>
00065         <li><a href="werte.html">Werte</a></li>
00066         <li><a class="active" href="settings.html">Settings</a></li>
00067         <li><a href="system.html">System</a></li>
00068         <li class="right"><a href="ueber.html">About</a></li>
```

```
00069     </ul>
00070 </body>
00071 </html >
```

7.7 data/system.html-Dateireferenz

7.8 system.html

gehe zur Dokumentation dieser Datei

```
00001 <HTML>
00002 <HEAD>
00003     <TITLE>Systeminfo</TITLE>
00004     <meta name="viewport" content="width=device-width, initial-scale=1">
00005     <link rel="icon" href="data:,">
00006     <link rel="stylesheet" type="text/css" href="style.css">
00007 </HEAD>
00008 <BODY>
00009     <br />
00010     <p class="label">Eigene IP-Adresse - AP: %sAP_IP%<br />
00011                     Clients am AP: %sAP_Clients%</p>
00012     <p class="label">OneWire %sOneWire_Status% Sensor gefunden</p>
00013     <p class="label">Informationen zum ESP32 - Board:<br />%sBoardInfo%</p>
00014     <p class="label">LittleFS, benutzte Bytes: %sFS_USpace% <br />
00015                      LittleFS, gesamte Bytes: %sFS_TSpace% </p>
00016     <p class="label">Free Heapspace: %sHeapspace%</p>
00017     <br />
00018     <br />
00019     <ul class="bottomnav">
00020         <li><a href="/">Home</a></li>
00021         <li><a href="werte.html">Werte</a></li>
00022         <li><a href="settings.html">Setting</a></li>
00023         <li><a class="active" href="system.html">System</a></li>
00024         <li class="right"><a href="ueber.html">About</a></li>
00025     </ul>
00026 </BODY>
00027 </HTML>
```

7.9 data/ueber.html-Dateireferenz

7.10 ueber.html

gehe zur Dokumentation dieser Datei

```
00001 <HTML>
00002 <HEAD>
00003     <TITLE>Wer steckt dahinter</TITLE>
00004     <meta name="viewport" content="width=device-width, initial-scale=1">
00005     <link rel="icon" href="data:,">
00006     <link rel="stylesheet" type="text/css" href="style.css">
00007 </HEAD>
00008 <BODY>
00009     <p class="label">%sVersion%</p>
00010         <br />
00011     <p class="label">Autor: Gerry Sebb</br>
00012     <a href="mailto: gerry@sebb.de">gerry@sebb.de</a></p>
00013         <br />
00014         <img src="/logo80.jpg" alt="Open-Boats Logo">
00015         <br />
00016     <ul class="bottomnav">
00017         <li><a href="/">Home</a></li>
00018         <li><a href="werte.html">Werte</a></li>
00019         <li><a href="settings.html">Setting</a></li>
00020         <li><a href="system.html">System</a></li>
00021         <li class="right"><a class="active" href="ueber.html">About</a></li>
00022     </ul>
00023 </BODY>
00024 </HTML>
```

7.11 data/werte.html-Dateireferenz

7.12 werte.html

gehe zur Dokumentation dieser Datei

```
00001 <!DOCTYPE html>
00002 <html>
00003 <head>
00004     <title>Motordaten</title>
00005     <meta name="viewport" content="width=device-width, initial-scale=1">
00006     <link rel="shortcut icon" type="image/x-icon" href="favicon.ico">
00007     <link rel="icon" href="data:,">
00008     <link rel="stylesheet" type="text/css" href="style.css">
00009     <script src='gauge.min.js'></script>
00010     <meta http-equiv="refresh" content="5">
00011 </head>
00012 <body>
00013     <p>DC Status</p>
00014     <p class="label">Bordspannung: %sBordspannung% V</p>
00015     <p>Maschine</p>
00016     <p id="coolwater" class="label">K&uuml;hlwasser Temperatur: %sCoolantTemp% &deg;C</br>
00017                     Offset: %sCoolantOffset% &deg;C</br>
00018                     Fehler: %sCoolantError%
00019                    </p>
00020     <p class="label">Motor Temperatur: %sMotorTemp% &deg;C</br>
00021                     Offset: %sMotorOffset% &deg;</br>
00022                     Fehler: %sMotorError%
00023                    </p>
00024     <p class="label">Motor Drehzahl: %sDrehzahl% U &frasl; min</p>
00025     <p class="label">Maschinenstunden: %sCounter% h</p>
00026     <p>Tank</p>
00027     <p class="label">Tank F&uuml;llstand: %sFuellstand% &#37;</br>max. F&uuml;llstand:
      %sFuellstandmax% l</p>
00028     <p>ADC Kalibrierung</p>
00029     <p class="label">ADC1: %sADC1_Cal%</br>ADC2: %sADC2_Cal%</p>
00030
00031     <ul class="bottomnav">
00032         <li><a href="/">Home</a></li>
00033         <li><a class="active" href="/">Werte</a></li>
00034         <li><a href="settings.html">Setting</a></li>
00035         <li><a href="system.html">System</a></li>
00036         <li class="right"><a href="ueber.html">About</a></li>
00037     </ul>
00038 </body>
00039 </html>
```

7.13 README.md-Dateireferenz

7.14 replace_fs.py-Dateireferenz

Namensbereiche

- namespace replace_fs

Variablen

- replace_fs.MKSPIFFSTOOL

7.15 replace_fs.py

gehe zur Dokumentation dieser Datei

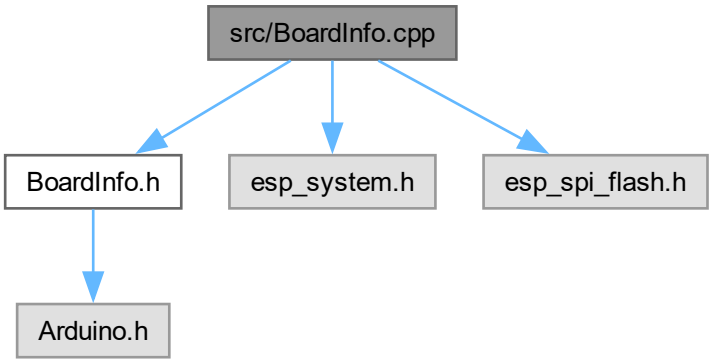
```
00001 Import("env")
00002 print("Replace MKSPIFFSTOOL with mklittlefs.exe")
00003 env.Replace (MKSPIFFSTOOL = "mklittlefs.exe")
```

7.16 src/BoardInfo.cpp-Dateireferenz

Boardinfo.

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

Include-Abhängigkeitsdiagramm für BoardInfo.cpp:



Makrodefinitionen

- #define BUF 255

Funktionen

- uint8_t temprature_sens_read ()

7.16.1 Ausführliche Beschreibung

Boardinfo.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei BoardInfo.cpp.

7.16.2 Makro-Dokumentation

7.16.2.1 BUF

```
#define BUF 255
```

Definiert in Zeile 29 der Datei BoardInfo.cpp.

7.16.3 Dokumentation der Funktionen

7.16.3.1 temprature_sens_read()

```
uint8_t temprature_sens_read ()
```

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

7.17 BoardInfo.cpp

gehe zur Dokumentation dieser Datei

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

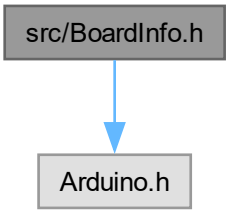
```
00049     if (m_chipid != 0)
00050     {
00051         Serial.printf("ESP32 Chip ID = %04X",(uint16_t)(m_chipid»32));        //print High 2 bytes
00052         Serial.printf("%08X\n",(uint32_t)m_chipid);                            //print Low 4bytes.
00053     }
00054     else
00055     {
00056 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

7.18 src/BoardInfo.h-Dateireferenz

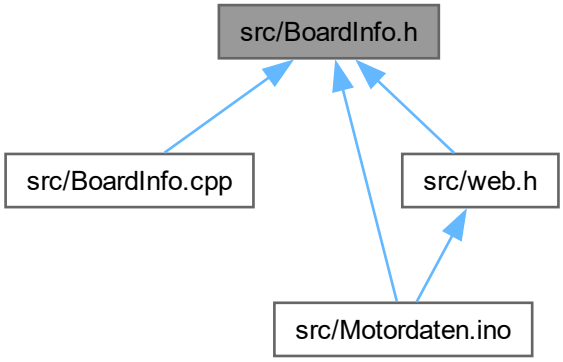
Hardwareinfo from ESP Board.

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

Include-Abhängigkeitsdiagramm für BoardInfo.h:



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



Klassen

- class BoardInfo

7.18.1 Ausführliche Beschreibung

Hardwareinfo from ESP Board.

Autor

Gerry Sebb

Version

1.0

Datum

2024-01-22

Copyright

Definiert in Datei BoardInfo.h.

7.19 BoardInfo.h

gehe zur Dokumentation dieser Datei

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

7.20 src/BoatData.h-Dateireferenz

Klassen

- struct tBoatData

7.21 BoatData.h

gehe zur Dokumentation dieser Datei

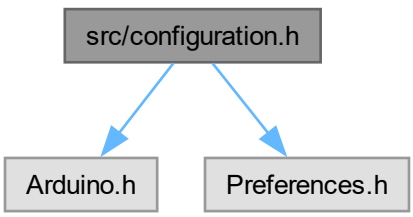
```
00001 #ifndef _BoatData_H_
00002 #define _BoatData_H_
00003
00004 struct tBoatData {
00005   unsigned long DaysSince1970;   // Days since 1970-01-01
00006
00007   double TrueHeading,SOG,COG,Variation,
00008          GPSTime,// Secs since midnight,
00009          Latitude, Longitude, Altitude, HDOP,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

7.22 src/configuration.h-Dateireferenz

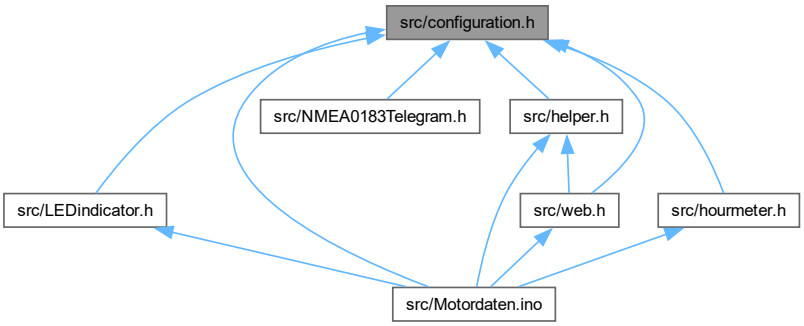
Konfiguration für GPIO und Variable.

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

Include-Abhängigkeitsdiagramm für configuration.h:



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



Klassen

- struct Web_Config

Makrodefinitionen

- #define Version "V2.6 vom 23.02.2025"
- #define ESP32_CAN_TX_PIN GPIO_NUM_4

  Config NMEA2000.
- #define ESP32_CAN_RX_PIN GPIO_NUM_5
- #define N2K_SOURCE 15
- #define EngineSendOffset 0
- #define TankSendOffset 40
- #define RPMSendOffset 80
- #define BatteryDCSendOffset 120
- #define BatteryDCStatusSendOffset 160
- #define SlowDataUpdatePeriod 1000
- #define PAGE_REFRESH 10
- #define WEB_TITEL "Motordaten"
- #define HostName "Motordaten"
- #define CL_SSID "NoWa"
- #define CL_PASSWORD "12345678"
- #define I2C_SDA 21
- #define I2C_SCL 22
- #define SEALEVELPRESSURE_HPA (1013.25)
- #define RPM_Calibration_Value 4.0
- #define 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 bI2C_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 = ""
- String coolantErrorReported = ""
- 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

Definiert in Datei configuration.h.

7.22.2 Makro-Dokumentation

7.22.2.1 Version

```
#define Version "V2.6 vom 23.02.2025"
```

Definiert in Zeile 19 der Datei configuration.h.

7.22.2.2 ESP32_CAN_TX_PIN

```
#define ESP32_CAN_TX_PIN GPIO_NUM_4
```

Config NMEA2000.

Definiert in Zeile 25 der Datei configuration.h.

7.22.2.3 ESP32_CAN_RX_PIN

```
#define ESP32_CAN_RX_PIN GPIO_NUM_5
```

Definiert in Zeile 26 der Datei configuration.h.

7.22.2.4 N2K_SOURCE

```
#define N2K_SOURCE 15
```

Definiert in Zeile 27 der Datei configuration.h.

7.22.2.5 EngineSendOffset

```
#define EngineSendOffset 0
```

Definiert in Zeile 33 der Datei configuration.h.

7.22.2.6 TankSendOffset

```
#define TankSendOffset 40
```

Definiert in Zeile 34 der Datei configuration.h.

7.22.2.7 RPMSendOffset

```
#define RPMSendOffset 80
```

Definiert in Zeile 35 der Datei configuration.h.

7.22.2.8 BatteryDCSendOffset

```
#define BatteryDCSendOffset 120
```

Definiert in Zeile 36 der Datei configuration.h.

7.22.2.9 BatteryDCStatusSendOffset

```
#define BatteryDCStatusSendOffset 160
```

Definiert in Zeile 37 der Datei configuration.h.

7.22.2.10 SlowDataUpdatePeriod

```
#define SlowDataUpdatePeriod 1000
```

Definiert in Zeile 38 der Datei configuration.h.

7.22.2.11 PAGE_REFRESH

```
#define PAGE_REFRESH 10
```

Definiert in Zeile 42 der Datei configuration.h.

7.22.2.12 WEB_TITEL

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

Definiert in Zeile 43 der Datei configuration.h.

7.22.2.13 HostName

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

Definiert in Zeile 61 der Datei configuration.h.

7.22.2.14 CL_SSID

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

Definiert in Zeile 78 der Datei configuration.h.

7.22.2.15 CL_PASSWORD

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

Definiert in Zeile 79 der Datei configuration.h.

7.22.2.16 I2C_SDA

```
#define I2C_SDA 21
```

Definiert in Zeile 89 der Datei configuration.h.

7.22.2.17 I2C_SCL

```
#define I2C_SCL 22
```

Definiert in Zeile 90 der Datei configuration.h.

7.22.2.18 SEALEVELPRESSURE_HPA

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

Definiert in Zeile 91 der Datei configuration.h.

7.22.2.19 RPM_Calibration_Value

```
#define RPM_Calibration_Value 4.0
```

Definiert in Zeile 114 der Datei configuration.h.

7.22.2.20 Eingine_RPM_Pin

```
#define Eingine_RPM_Pin 19
```

Definiert in Zeile 115 der Datei configuration.h.

7.22.2.21 ONE_WIRE_BUS

```
#define ONE_WIRE_BUS 14
```

Definiert in Zeile 124 der Datei configuration.h.

7.22.2.22 SERVER_HOST_NAME

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

Definiert in Zeile 147 der Datei configuration.h.

7.22.2.23 TCP_PORT

```
#define TCP_PORT 6666
```

Definiert in Zeile 148 der Datei configuration.h.

7.22.2.24 DNS_PORT

```
#define DNS_PORT 53
```

Definiert in Zeile 149 der Datei configuration.h.

7.22.3 Dokumentation der Aufzählungstypen

7.22.3.1 EngineStatus

```
enum EngineStatus
```

Aufzählungswerte

Off	
On	

Definiert in Zeile 113 der Datei configuration.h.

```
00113 { Off = 0,  On = 1, };
```

7.22.4 Variablen-Dokumentation

7.22.4.1 NodeAddress

```
int NodeAddress
```

Definiert in Zeile 28 der Datei configuration.h.

7.22.4.2 preferences

```
Preferences preferences
```

Definiert in Zeile 29 der Datei configuration.h.

7.22.4.3 chipid

```
uint8_t chipid[6]
```

Definiert in Zeile 30 der Datei configuration.h.

7.22.4.4 id

```
uint32_t id = 0
```

Definiert in Zeile 31 der Datei configuration.h.

7.22.4.5 i

```
int i = 0
```

Definiert in Zeile 32 der Datei configuration.h.

7.22.4.6 sHeapspace

```
String sHeapspace = ""
```

Definiert in Zeile 44 der Datei configuration.h.

7.22.4.7 tAP_Config

```
Web_Config tAP_Config
```

Definiert in Zeile 58 der Datei configuration.h.

7.22.4.8 channel

```
const int channel = 10
```

Definiert in Zeile 62 der Datei configuration.h.

7.22.4.9 hide_SSID

```
const bool hide_SSID = false
```

Definiert in Zeile 63 der Datei configuration.h.

7.22.4.10 max_connection

```
const int max_connection = 2
```

Definiert in Zeile 64 der Datei configuration.h.

7.22.4.11 IP

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

Definiert in Zeile 67 der Datei configuration.h.

7.22.4.12 Gateway

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

Definiert in Zeile 68 der Datei configuration.h.

7.22.4.13 NMask

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

Definiert in Zeile 69 der Datei configuration.h.

7.22.4.14 AP_SSID

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

Definiert in Zeile 70 der Datei configuration.h.

7.22.4.15 AP_PASSWORD

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

Definiert in Zeile 71 der Datei configuration.h.

7.22.4.16 AP_IP

```
IPAddress AP_IP
```

Definiert in Zeile 72 der Datei configuration.h.

7.22.4.17 CL_IP

```
IPAddress CL_IP
```

Definiert in Zeile 73 der Datei configuration.h.

7.22.4.18 SELF_IP

```
IPAddress SELF_IP
```

Definiert in Zeile 74 der Datei configuration.h.

7.22.4.19 sAP_Station

```
String sAP_Station = ""
```

Definiert in Zeile 75 der Datei configuration.h.

7.22.4.20 iSTA_on

```
int iSTA_on = 0
```

Definiert in Zeile 80 der Datei configuration.h.

7.22.4.21 bConnect_CL

```
int bConnect_CL = 0
```

Definiert in Zeile 81 der Datei configuration.h.

7.22.4.22 bClientConnected

```
bool bClientConnected = 0
```

Definiert in Zeile 82 der Datei configuration.h.

7.22.4.23 ADC_Calibration_Value1

```
double ADC_Calibration_Value1 = 170.0
```

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

Definiert in Zeile 85 der Datei configuration.h.

7.22.4.24 ADC_Calibration_Value2

```
double ADC_Calibration_Value2 = 19.0
```

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

Definiert in Zeile 86 der Datei configuration.h.

7.22.4.25 fbmp_temperature

```
float fbmp_temperature = 0
```

Definiert in Zeile 92 der Datei configuration.h.

7.22.4.26 fbmp_pressure

```
float fbmp_pressure = 0
```

Definiert in Zeile 93 der Datei configuration.h.

7.22.4.27 fbmp_altitude

```
float fbmp_altitude = 0
```

Definiert in Zeile 94 der Datei configuration.h.

7.22.4.28 sI2C_Status

```
String sI2C_Status = ""
```

Definiert in Zeile 95 der Datei configuration.h.

7.22.4.29 bI2C_Status

```
bool bI2C_Status = 0
```

Definiert in Zeile 96 der Datei configuration.h.

7.22.4.30 iMaxSonar

```
const int iMaxSonar = 35
```

Definiert in Zeile 99 der Datei configuration.h.

7.22.4.31 iDistance

```
int iDistance = 0
```

Definiert in Zeile 100 der Datei configuration.h.

7.22.4.32 FuelLevel

```
float FuelLevel = 0
```

Definiert in Zeile 103 der Datei configuration.h.

7.22.4.33 FuelLevelMax

```
float FuelLevelMax = 30
```

Definiert in Zeile 104 der Datei configuration.h.

7.22.4.34 CoolantTemp

```
float CoolantTemp = 0
```

Definiert in Zeile 105 der Datei configuration.h.

7.22.4.35 MotorTemp

```
float MotorTemp = 0
```

Definiert in Zeile 106 der Datei configuration.h.

7.22.4.36 EngineRPM

```
float EngineRPM = 0
```

Definiert in Zeile 107 der Datei configuration.h.

7.22.4.37 BordSpannung

```
float BordSpannung = 0
```

Definiert in Zeile 108 der Datei configuration.h.

7.22.4.38 EngineOn

```
bool EngineOn = false
```

Definiert in Zeile 109 der Datei configuration.h.

7.22.4.39 motorErrorReported

```
String motorErrorReported = ""
```

Definiert in Zeile 110 der Datei configuration.h.

7.22.4.40 coolantErrorReported

```
String coolantErrorReported = ""
```

Definiert in Zeile 111 der Datei configuration.h.

7.22.4.41 Counter

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

Definiert in Zeile 112 der Datei configuration.h.

7.22.4.42 Bat1Capacity

```
int Bat1Capacity = 55
```

Definiert in Zeile 118 der Datei configuration.h.

7.22.4.43 Bat2Capacity

```
int Bat2Capacity = 90
```

Definiert in Zeile 119 der Datei configuration.h.

7.22.4.44 SoCError

```
int SoCError = 0
```

Definiert in Zeile 120 der Datei configuration.h.

7.22.4.45 BatSoC

```
float BatSoC = 0
```

Definiert in Zeile 121 der Datei configuration.h.

7.22.4.46 sOneWire_Status

```
String sOneWire_Status = ""
```

Definiert in Zeile 125 der Datei configuration.h.

7.22.4.47 fDrehzahl

```
float fDrehzahl = 0
```

Definiert in Zeile 128 der Datei configuration.h.

7.22.4.48 fGaugeDrehzahl

```
float fGaugeDrehzahl = 0
```

Definiert in Zeile 129 der Datei configuration.h.

7.22.4.49 fBordSpannung

```
float fBordSpannung = 0
```

Definiert in Zeile 130 der Datei configuration.h.

7.22.4.50 fCoolantTemp

```
float fCoolantTemp = 0
```

Definiert in Zeile 131 der Datei configuration.h.

7.22.4.51 fMotorTemp

```
float fMotorTemp = 0
```

Definiert in Zeile 132 der Datei configuration.h.

7.22.4.52 fCoolantOffset

```
float fCoolantOffset = 0
```

Definiert in Zeile 133 der Datei configuration.h.

7.22.4.53 fMotorOffset

```
float fMotorOffset = 0
```

Definiert in Zeile 134 der Datei configuration.h.

7.22.4.54 sSTBB

```
String sSTBB = ""
```

Definiert in Zeile 135 der Datei configuration.h.

7.22.4.55 sOrient

```
String sOrient = ""
```

Definiert in Zeile 136 der Datei configuration.h.

7.22.4.56 dMWV_WindDirectionT

```
double dMWV_WindDirectionT = 0
```

Definiert in Zeile 139 der Datei configuration.h.

7.22.4.57 dMWV_WindSpeedM

```
double dMWV_WindSpeedM = 0
```

Definiert in Zeile 140 der Datei configuration.h.

7.22.4.58 dVWR_WindDirectionM

```
double dVWR_WindDirectionM = 0
```

Definiert in Zeile 141 der Datei configuration.h.

7.22.4.59 dVWR_WindAngle

```
double dVWR_WindAngle = 0
```

Definiert in Zeile 142 der Datei configuration.h.

7.22.4.60 dVWR_WindSpeedkn

```
double dVWR_WindSpeedkn = 0
```

Definiert in Zeile 143 der Datei configuration.h.

7.22.4.61 dVWR_WindSpeedms

```
double dVWR_WindSpeedms = 0
```

Definiert in Zeile 144 der Datei configuration.h.

7.22.4.62 udpAddress

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

Definiert in Zeile 152 der Datei configuration.h.

7.22.4.63 udpPort

```
const int udpPort = 4444
```

Definiert in Zeile 153 der Datei configuration.h.

7.23 configuration.h

gehe zur Dokumentation dieser Datei

```
00001 #ifndef __configuration__H__
00002 #define __configuration__H__
00003
00014
00015 #include <Arduino.h>
00016 #include <Preferences.h>
00017
00018 // Versionierung
00019 #define Version "V2.6 vom 23.02.2025"  // Version
00020
00025 #define ESP32_CAN_TX_PIN GPIO_NUM_4  // Set CAN TX port to 4
00026 #define ESP32_CAN_RX_PIN GPIO_NUM_5  // Set CAN RX port to 5
00027 #define N2K_SOURCE 15
00028 int NodeAddress;                     // To store Last Node Address
00029 Preferences preferences;             // Nonvolatile storage on ESP32 - To store LastDeviceAddress
00030 uint8_t chipid[6];
00031 uint32_t id = 0;
00032 int i = 0;
00033 #define EngineSendOffset 0
00034 #define TankSendOffset 40
00035 #define RPMSendOffset 80
00036 #define BatteryDCSendOffset 120
00037 #define BatteryDCStatusSendOffset 160
00038 #define SlowDataUpdatePeriod 1000  // Time between CAN Messages sent
00039
00040
00041 //Configuration Website
00042 #define PAGE_REFRESH 10 // x Sec.
00043 #define WEB_TITEL "Motordaten"
00044 String sHeapspace = "";
00045
00046 //Configuration mit Webinterface
00047 struct Web_Config
00048 {
00049     char wAP_IP[20];
00050     char wAP_SSID[64];
00051     char wAP_Password[12];
00052     char wMotor_Offset[6];
00053     char wCoolant_Offset[6];
00054     char wFuellstandmax[6];
00055     char wADC1_Cal[6];
00056     char wADC2_Cal[6];
00057 };
00058 Web_Config tAP_Config;
00059
00060 //Configuration AP
00061 #define HostName        "Motordaten"
00062 const int   channel        = 10;               // WiFi Channel number between 1 and 13
00063 const bool  hide_SSID      = false;            // To disable SSID broadcast -> SSID will not appear
      in a basic WiFi scan
00064 const int   max_connection = 2;                // Maximum simultaneous connected clients on the AP
00065
00066 // Variables for WIFI-AP
00067 IPAddress IP = IPAddress(192, 168, 15, 30);
00068 IPAddress Gateway = IPAddress(192, 168, 15, 30);
00069 IPAddress NMask = IPAddress(255, 255, 255, 0);
00070 const char* AP_SSID = "Motordaten";
```

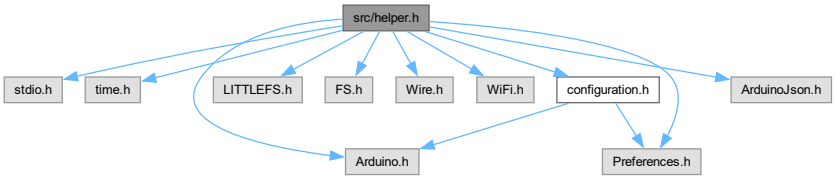
```
00071 const char* AP_PASSWORD  = "12345678";
00072 IPAddress AP_IP;
00073 IPAddress CL_IP;
00074 IPAddress SELF_IP;
00075 String sAP_Station = "";
00076
00077 //Configuration Client (Network Data Windsensor)
00078 #define CL_SSID      "NoWa"                //Windmesser
00079 #define CL_PASSWORD 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

7.24 src/helper.h-Dateireferenz

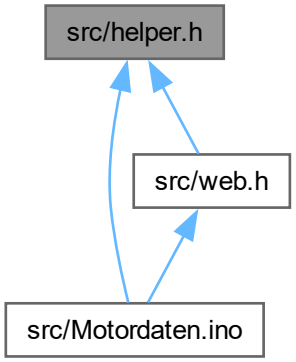
Hilfsfunktionen.

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

Include-Abhängigkeitsdiagramm für helper.h:



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



Funktionen

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

  LittleFS, Dateien auflisten.

- void readConfig (String filename)

  Konfiguration aus Json-Datei lesen.
- bool writeConfig (String json)

  Webseiten Eingabe in Json-Datei schreiben.
- void I2C_scan (void)
- String sWifiStatus (int Status)

  WIFI Status lesen.
- char * toChar (String command)

  Convert string to char.

7.24.1 Ausführliche Beschreibung

Hilfsfunktionen.

Autor

Gerry Sebb

Version

1.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei helper.h.

7.24.2 Dokumentation der Funktionen

7.24.2.1 ShowTime()

```
void ShowTime ()
```

Definiert in Zeile 27 der Datei helper.h.

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

7.24.2.2 freeHeapSpace()

```
void freeHeapSpace ()
```

Freie Speichergroesse aller 5s lesen

Definiert in Zeile 37 der Datei helper.h.

```
00037                     {
00038     static unsigned long last = millis();
00039     if (millis() - last > 5000) {
00040         last = millis();
00041     sHeapspace = ESP.getFreeHeap();
00042         Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00043     }
00044 }
```

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

7.24.2.3 WiFiDiag()

```
void WiFiDiag (
              void )
```

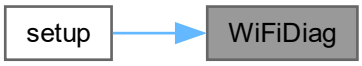
Ausgabe WIFI Parameter und Netzwerk scannen

Definiert in Zeile 47 der Datei helper.h.

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

```
00078     }
00079   }
00080 }
```

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

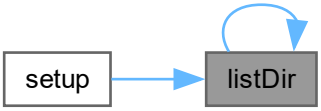
fs	
dirname	
levels	

Definiert in Zeile 92 der Datei helper.h.

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

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

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



7.24.2.5 readConfig()

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

Konfiguration aus Json-Datei lesen.

Parameter

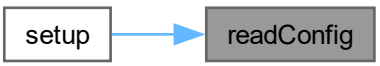
filename	

Definiert in Zeile 129 der Datei helper.h.

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

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

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



7.24.2.6 writeConfig()

```
bool writeConfig (
             String json)
```

Webseiten Eingabe in Json-Datei schreiben.

Parameter

json	

Rückgabe

true

false

Definiert in Zeile 175 der Datei helper.h.

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

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

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

7.24.2.7 I2C_scan()

```
void I2C_scan (
             void )
```

I2C Bus auslesen, alle Geräte mit Adresse ausgegeben

Definiert in Zeile 217 der Datei helper.h.

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

```
00248       if (address<16) {
00249         Serial.print("0");
00250       }
00251       Serial.println(address,HEX);
00252     }
00253   }
00254   if (nDevices == 0) {
00255     Serial.println("No I2C devices found\n");
00256   }
00257   else {
00258     Serial.println("done\n");
00259   }
00260 }
```

7.24.2.8 sWifiStatus()

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

WIFI Status lesen.

Parameter

Status	

Rückgabe

String

Definiert in Zeile 269 der Datei helper.h.

```
00270 {
00271   switch(Status){
00272     case WL_IDLE_STATUS:return "Warten";
00273     case WL_NO_SSID_AVAIL:return "Keine SSID vorhanden";
00274     case WL_SCAN_COMPLETED:return "Scan komlett";
00275     case WL_CONNECTED:return "Verbunden";
00276     case WL_CONNECT_FAILED:return "Verbindung fehlerhaft";
00277     case WL_CONNECTION_LOST:return "Verbindung verloren";
00278     case WL_DISCONNECTED:return "Nicht verbunden";
00279     default:return "unbekannt";
00280   }
00281 }
```

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

7.24.2.9 toChar()

```
char * toChar (
            String command)
```

Convert string to char.

Parameter

command	

Rückgabe

char*

Definiert in Zeile 290 der Datei helper.h.

```
00290                                {
00291     if(command.length()!=0){
00292         char *p = const_cast<char*>(command.c_str());
00293         return p;
00294     }
00295     else{
00296       return 0;
00297     }
00298 }
```

7.25 helper.h

gehe zur Dokumentation dieser Datei

```
00001 #ifndef _HELPER_H_
00002 #define _HELPER_H_
00003
00014
00015
00016 #include <stdio.h>
00017 #include <time.h>
00018 #include <Arduino.h>
00019 #include <LITTLEFS.h>
00020 #include <FS.h>
00021 #include <Wire.h>
00022 #include <WiFi.h>
00023 #include "configuration.h"
00024 #include <ArduinoJson.h>
00025 #include <Preferences.h>
00026
00027 void ShowTime(){
00028     time_t now = time(NULL);
00029     struct tm tm_now;
00030     localtime_r(&now, &tm_now);
00031     char buff[100];
00032     strftime(buff, sizeof(buff), "%d-%m-%Y %H:%M:%S", &tm_now);
00033     printf("Zeit: %s\n", buff);
00034 }
00035
00037 void freeHeapSpace(){
00038     static unsigned long last = millis();
00039     if (millis() - last > 5000) {
00040         last = millis();
00041     sHeapspace = ESP.getFreeHeap();
00042         Serial.printf("\n[MAIN] Free heap: %d bytes\n", ESP.getFreeHeap());
00043     }
00044 }
00045
00047 void WiFiDiag(void) {
00048   Serial.println("\nWifi-Diag:");
00049   AP_IP = WiFi.softAPIP();
00050   CL_IP = WiFi.localIP();
00051   Serial.print("AP IP address: ");
00052   Serial.println(AP_IP.toString());
00053   Serial.print("Client IP address: ");
00054   Serial.println(CL_IP.toString());
00055   WiFi.printDiag(Serial);
00056   Serial.print("\nScan AP's ");
00057   {
00058     // WiFi.scanNetworks will return the number of networks found
00059     int n = WiFi.scanNetworks();
00060     Serial.println("scan done");
00061     if (n == 0) {
00062         Serial.println("no networks found");
00063     } else {
```

```
00064         Serial.print(n);
00065         Serial.println(" networks found");
00066         for (int i = 0; i < n; ++i)
00067         {
00068           // Print SSID and RSSI for each network found
00069           Serial.print(i + 1);
00070           Serial.print(": ");
00071           Serial.print(WiFi.SSID(i));
00072           Serial.print(" (");
00073           Serial.print(WiFi.RSSI(i));
00074           Serial.print(")");
00075           Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
00076           delay(10);
00077         }
00078     }
00079   }
00080 }
00081
00082 /***************************** Filesystem *************************/
00083
00091
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     }
00100     if(!root.isDirectory()){
00101         Serial.println(" - not a directory");
00102         return;
00103     }
00104
00105     File file = root.openNextFile();
00106     while(file){
00107         if(file.isDirectory()){
00108             Serial.print("  DIR : ");
00109             Serial.println(file.name());
00110             if(levels){
00111                 listDir(fs, file.path(), levels -1);
00112             }
00113         } else {
00114             Serial.print("  FILE: ");
00115             Serial.print(file.name());
00116             Serial.print("\tSIZE: ");
00117             Serial.println(file.size());
00118         }
00119         file = root.openNextFile();
00120     }
00121 }
00122
00128
00129 void readConfig(String filename) {
00130     JsonDocument testDocument;
00131     File configFile = LittleFS.open(filename);
00132     if (configFile)
00133     {
00134         Serial.println("opened config file");
00135         DeserializationError error = deserializeJson(testDocument, configFile);
00136
00137         // Test if parsing succeeds.
00138         if (error)
00139         {
00140             Serial.print(F("deserializeJson() failed: "));
00141             Serial.println(error.f_str());
00142             return;
00143         }
00144
00145         Serial.println("deserializeJson ok");
00146         {
00147             Serial.println("Lese Daten aus Config - Datei");
00148             strcpy(tAP_Config.wAP_SSID, testDocument["SSID"] | "Motordaten");
00149             strcpy(tAP_Config.wAP_IP, testDocument["IP"] | "192.168.15.30");
00150             strcpy(tAP_Config.wAP_Password, testDocument["Password"] | "12345678");
00151             strcpy(tAP_Config.wMotor_Offset, testDocument["MotorOffset"] | "0.0");
00152       strcpy(tAP_Config.wCoolant_Offset, testDocument["CoolantOffset"] | "0.0");
00153       strcpy(tAP_Config.wFuellstandmax, testDocument["Fuellstandmax"] | "0.0");
00154       strcpy(tAP_Config.wADC1_Cal, testDocument["ADC1_Cal"] | "0.0");
00155       strcpy(tAP_Config.wADC2_Cal, testDocument["ADC2_Cal"] | "0.0");
00156         }
00157         configFile.close();
00158         Serial.println("Config - Datei geschlossen");
00159     }
00160
00161     else
00162     {
```

```
00163         Serial.println("failed to load json config");
00164     }
00165 }
00166 
00174 
00175 bool writeConfig(String json)
00176 {
00177     Serial.println(json);
00178 
00179     Serial.println("neue Konfiguration speichern");
00180 
00181     File configFile = LittleFS.open("/config.json", FILE_WRITE);
00182     if (configFile)
00183     {
00184         Serial.println("Config - Datei öffnen");
00185         File configFile = LittleFS.open("/config.json", FILE_WRITE);
00186         if (configFile)
00187         {
00188             Serial.println("Config - Datei zum Schreiben geöffnet");
00189             JsonDocument testDocument;
00190             Serial.println("JSON - Daten übergeben");
00191             DeserializationError error = deserializeJson(testDocument, json);
00192             // Test if parsing succeeds.
00193             if (error)
00194             {
00195                 Serial.print(F("deserializeJson() failed: "));
00196                 Serial.println(error.f_str());
00197                 // bei Memory - Fehler den <Wert> in StaticJsonDocument<200> testDocument; erhöhen
00198                 return false;
00199             }
00200             Serial.println("Konfiguration schreiben...");
00201             serializeJson(testDocument, configFile);
00202             Serial.println("Konfiguration geschrieben...");
00203 
00204             // neue Config in Serial ausgeben zur Kontrolle
00205             serializeJsonPretty(testDocument, Serial);
00206 
00207             Serial.println("Config - Datei geschlossen");
00208             configFile.close();
00209         }
00210     }
00211     return true;
00212 }
00213 
00214 /***************************** I2C Bus *************************/
00216 
00217 void I2C_scan(void){
00218   byte error, address;
00219   int nDevices;
00220   Serial.println("Scanning...");
00221   nDevices = 0;
00222   for(address = 1; address < 127; address++ )
00223   {
00224     Wire.beginTransmission(address);
00225     error = Wire.endTransmission();
00226     if (error == 0)
00227     {
00228       Serial.print("I2C device found at address 0x");
00229       if (address<16)
00230       {
00231         Serial.print("0");
00232       }
00233       Serial.println(address,HEX);
00234       nDevices++;
00235     }
00236     else if (error==4)
00237     {
00238       Serial.print("Unknow error at address 0x");
00239       if (address<16)
00240       {
00241         Serial.print("0");
00242       }
00243       Serial.println(address,HEX);
00244       nDevices++;
00245     }
00246     else if (error==4) {
00247       Serial.print("Unknow error at address 0x");
00248       if (address<16) {
00249         Serial.print("0");
00250       }
00251       Serial.println(address,HEX);
00252     }
00253   }
00254   if (nDevices == 0) {
00255     Serial.println("No I2C devices found\n");
00256   }
00257   else {
```

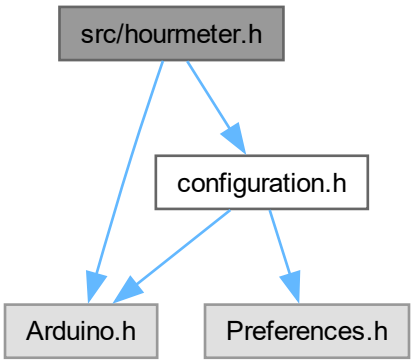
```
00258     Serial.println("done\n");
00259   }
00260 }
00261
00268
00269 String sWifiStatus(int Status)
00270 {
00271   switch(Status){
00272     case WL_IDLE_STATUS:return "Warten";
00273     case WL_NO_SSID_AVAIL:return "Keine SSID vorhanden";
00274     case WL_SCAN_COMPLETED:return "Scan komlett";
00275     case WL_CONNECTED:return "Verbunden";
00276     case WL_CONNECT_FAILED:return "Verbindung fehlerhaft";
00277     case WL_CONNECTION_LOST:return "Verbindung verloren";
00278     case WL_DISCONNECTED:return "Nicht verbunden";
00279     default:return "unbekannt";
00280   }
00281 }
00282
00289
00290 char* toChar(String command){
00291     if(command.length()!=0){
00292         char *p = const_cast<char*>(command.c_str());
00293         return p;
00294     }
00295     else{
00296       return 0;
00297     }
00298 }
00299
00300
00301 #endif
```

7.26 src/hourmeter.h-Dateireferenz

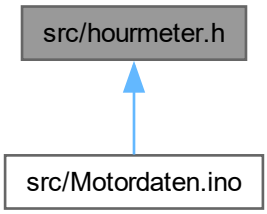
Betriebstundenzähler.

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

Include-Abhängigkeitsdiagramm für hourmeter.h:



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



Funktionen

- unsigned long EngineHours (bool CountOn=0)

  Betriebstundenzähler Berechnet Betriebstunden, wenn Anlage eingeschaltet ist.

Variablen

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

7.26.1 Ausführliche Beschreibung

Betriebstundenzähler.

Autor

Gerry Sebb

Version

1.0

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei hourmeter.h.

7.26.2 Dokumentation der Funktionen

7.26.2.1 EngineHours()

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

Betriebstundenzähler Berechnet Betriebstunden, wenn Anlage eingeschaltet ist.

Parameter

CountOn	

Rückgabe

unsigned long

< speichern bei Flanke negativ

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

< Speicher auslesen

< Laufzeit alt + aktuell

< Speicher schreiben

< Preferences beenden

Definiert in Zeile 30 der Datei hourmeter.h.

```
00030                                           {
00031     {
00032     long now = millis();
00033     milliRest += now - lastRun;
00034         if (CountOn == 1)
00035         {
00036             while (milliRest>=1000){
00037                 Counter++;
00038                 milliRest-=1000;
00039             }
00040         }
00041             else milliRest=0;
00042             lastRun = now;
00043             return Counter;
00044     }
00045     state1 = CountOn;
00046         if (laststate1 == HIGH && state1 == LOW)
00047         {
00048             bsz1.begin("bsz", false);
00049             CounterOld = preferences.getUInt("Start", 0);
00050             Counter = CounterOld + Counter;
00051             bsz1.putUInt("Start", Counter);
00052             bsz1.end();
00053             state1 = LOW;
00054         }
00055 }
```

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

7.26.3 Variablen-Dokumentation

7.26.3.1 bsz1

```
Preferences bsz1
```

Definiert in Zeile 18 der Datei hourmeter.h.

7.26.3.2 lastRun

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

Definiert in Zeile 20 der Datei hourmeter.h.

7.26.3.3 CounterOld

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

Definiert in Zeile 20 der Datei hourmeter.h.

7.26.3.4 milliRest

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

Definiert in Zeile 20 der Datei hourmeter.h.

7.26.3.5 state1

```
int state1 = LOW
```

Definiert in Zeile 21 der Datei hourmeter.h.

7.26.3.6 laststate1

```
int laststate1 = LOW
```

Definiert in Zeile 21 der Datei hourmeter.h.

7.27 hourmeter.h

gehe zur Dokumentation dieser Datei

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

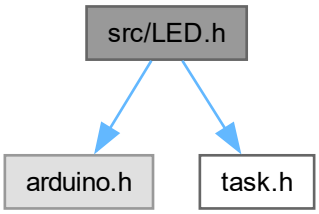
```
00040         }
00041             else milliRest=0;
00042             lastRun = now;
00043             return Counter;
00044     }
00045     state1 = CountOn;
00046         if (laststate1 == HIGH && state1 == LOW)
00047         {
00048             bsz1.begin("bsz", false);
00049             CounterOld = preferences.getUInt("Start", 0);
00050             Counter = CounterOld + Counter;
00051             bsz1.putUInt("Start", Counter);
00052             bsz1.end();
00053             state1 = LOW;
00054         }
00055 }
00056
00057 #endif
```

7.28 src/LED.h-Dateireferenz

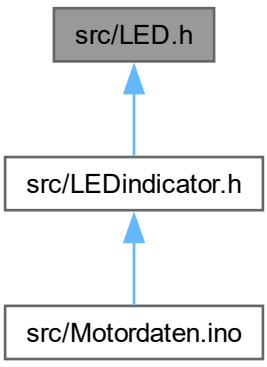
LED Ansteuerung.

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

Include-Abhängigkeitsdiagramm für LED.h:



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



Aufzählungen

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

Funktionen

- void LEDblink (int PIN=LED())
- void LEDflash (int PIN=LED())
- void flashLED (int PIN=LED())
- void LEDInit ()

  Start Initialisierung LEDtest.
- void LEDon (int PIN=LED())
- void LEDoff (int PIN=LED())
- void LEDoff_RGB ()

7.28.1 Ausführliche Beschreibung

LED Ansteuerung.

Autor

Gerry Sebb

Version

2.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei LED.h.

7.28.2 Dokumentation der Aufzählungstypen

7.28.2.1 LED

```
enum LED
```

Aufzählungswerte

Red	
Green	
Blue	
LEDBoard	

Definiert in Zeile 19 der Datei LED.h.

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

7.28.3 Dokumentation der Funktionen

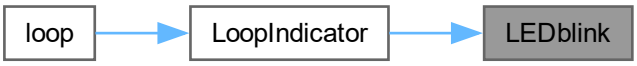
7.28.3.1 LEDblink()

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

Definiert in Zeile 26 der Datei LED.h.

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

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



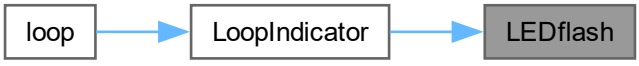
7.28.3.2 LEDflash()

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

Definiert in Zeile 38 der Datei LED.h.

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

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



7.28.3.3 flashLED()

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

Definiert in Zeile 51 der Datei LED.h.

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

7.28.3.4 LEDInit()

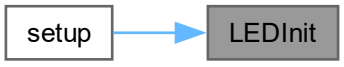
```
void LEDInit ()
```

Start Initialisierung LEDtest.

Definiert in Zeile 63 der Datei LED.h.

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

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



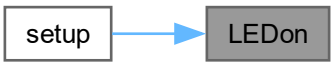
7.28.3.5 LEDon()

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

Definiert in Zeile 78 der Datei LED.h.

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

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



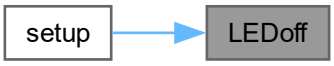
7.28.3.6 LEDoff()

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

Definiert in Zeile 82 der Datei LED.h.

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

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



7.28.3.7 LEDoff_RGB()

```
void LEDoff_RGB ()
```

Definiert in Zeile 86 der Datei LED.h.

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

7.29 LED.h

gehe zur Dokumentation dieser Datei

```
00001
00011
00012 #include <arduino.h>
00013 #include "task.h"
00014
```

```
00015 //Configuration LED
00016 //const int LEDBoard = 2;  //DevModule
00017 //const int LEDBoard = 13;   //Adafruit Huzzah32
00018
00019 enum LED {
00020   Red = 25,
00021   Green = 26,
00022   Blue = 33,
00023   LEDBoard = 13 //Adafruit Huzzah32
00024   };
00025
00026 void LEDblink(int PIN = LED()){
00027  taskBegin();
00028    while(1)   // blockiert dank der TaskPause nicht
00029    {
00030       digitalWrite(PIN,HIGH);  // LED ein
00031       taskPause(250);   // gibt Rechenzeit ab
00032       digitalWrite(PIN,LOW);   // LED aus
00033       taskPause(1000);   // gibt Rechenzeit ab
00034    }
00035    taskEnd();
00036 }
00037
00038 void LEDflash(int PIN = LED()){
00039    taskBegin();
00040    while(1)   // blockiert dank der TaskPause nicht
00041    {
00042       digitalWrite(PIN,HIGH);  // LED ein
00043       delay (5);
00044       //taskPause(2);   // gibt Rechenzeit ab
00045       digitalWrite(PIN,LOW);   // LED aus
00046       taskPause(3000);   // gibt Rechenzeit ab
00047    }
00048    taskEnd();
00049 }
00050
00051 void flashLED(int PIN = LED()) {
00052   if (millis() % 1000 > 500) {
00053     digitalWrite(PIN, HIGH);
00054   } else {
00055     digitalWrite(PIN, LOW);
00056   }
00057 }
00058
00063 void LEDInit() {
00064   pinMode(LED(Red),   OUTPUT);
00065   pinMode(LED(Blue),  OUTPUT);
00066   pinMode(LED(Green), OUTPUT);
00067   digitalWrite(LED(Red), HIGH);
00068   delay(250);
00069   digitalWrite(LED(Red), LOW);
00070   digitalWrite(LED(Blue), HIGH);
00071   delay(250);
00072   digitalWrite(LED(Blue), LOW);
00073   digitalWrite(LED(Green), HIGH);
00074   delay(250);
00075   digitalWrite(LED(Green), LOW);
00076 }
00077
00078 void LEDon(int PIN = LED()) {
00079   digitalWrite(PIN, HIGH);
00080 }
00081
00082 void LEDoff(int PIN = LED()) {
00083   digitalWrite(PIN, LOW);
00084 }
00085
00086 void LEDoff_RGB() {
00087   digitalWrite(LED(Blue), LOW);
00088   digitalWrite(LED(Green),LOW);
00089   digitalWrite(LED(Red), LOW);
00090 }
00091
```

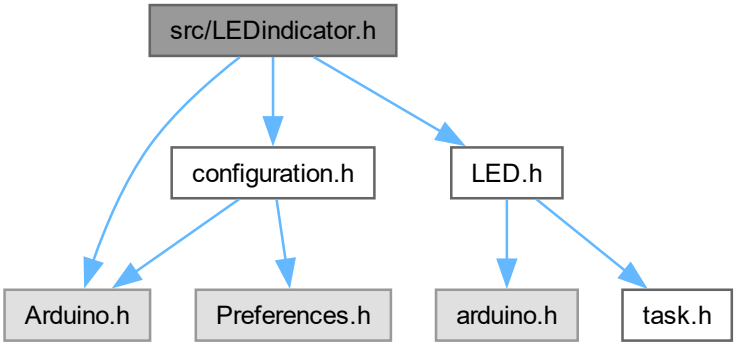
7.30 src/LEDindicator.h-Dateireferenz

LED Betriebsanzeige.

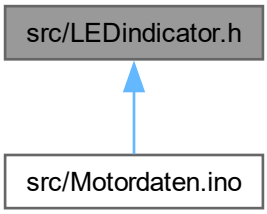
```
#include <Arduino.h>
#include "configuration.h"
```

```
#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

Definiert in Datei LEDindicator.h.

7.30.2 Dokumentation der Funktionen

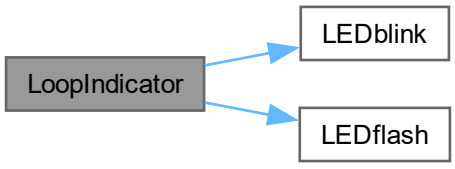
7.30.2.1 LoopIndicator()

```
void LoopIndicator ()
```

Definiert in Zeile 27 der Datei LEDindicator.h.

```
00027                     {
00028 if (motorErrorReported == "Aus" && coolantErrorReported == "Aus"){
00029   LEDflash(LED(Green)); // flash for loop run
00030   ErrorOff = true;
00031 }
00032 if (motorErrorReported == "Ein" || coolantErrorReported == "Ein"){
00033   LEDblink(LED(Red));
00034   ErrorOn = true;
00035 }
00036 if (!(ErrorOff && ErrorOn)){
00037       LEDflash(LED(Green));
00038       LEDflash(LED(Red));
00039 }
00040 }
```

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



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

7.30.3 Variablen-Dokumentation

7.30.3.1 ErrorOff

```
bool ErrorOff = false
```

Sensor failure switch LED green/red.

Definiert in Zeile 24 der Datei LEDindicator.h.

7.30.3.2 ErrorOn

```
bool ErrorOn = false
```

Definiert in Zeile 25 der Datei LEDindicator.h.

7.31 LEDindicator.h

gehe zur Dokumentation dieser Datei

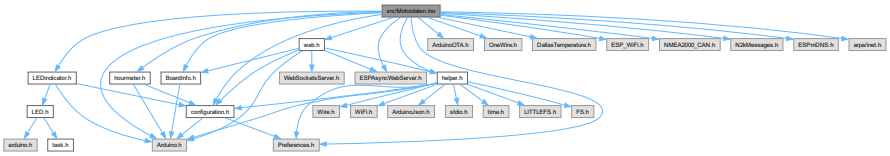
```
00001
00011
00012 #ifndef _LEDINDICATOR_
00013 #define _LEDINDICATOR_
00014
00015 # include <Arduino.h>
00016 # include "configuration.h"
00017 # include "LED.h"
00018
00019
00024 bool ErrorOff = false;
00025 bool ErrorOn = false;
00026
00027 void LoopIndicator(){
00028 if (motorErrorReported == "Aus" && coolantErrorReported == "Aus"){
00029   LEDflash(LED(Green)); // flash for loop run
00030   ErrorOff = true;
00031 }
00032 if (motorErrorReported == "Ein" || coolantErrorReported == "Ein"){
00033   LEDblink(LED(Red));
00034   ErrorOn = true;
00035 }
00036 if (!(ErrorOff && ErrorOn)){
00037       LEDflash(LED(Green));
00038       LEDflash(LED(Red));
00039 }
00040 }
00041
00042 #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 °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/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

2.4

Datum

2025-01-06

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 45 der Datei Motordaten.ino.

7.32.3 Dokumentation der Funktionen

7.32.3.1 oneWire()

```
OneWire oneWire (
            ONE_WIRE_BUS )
```

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

7.32.3.2 debug_log()

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

Definiert in Zeile 89 der Datei Motordaten.ino.

```
00089                           {
00090 #if ENABLE_DEBUG_LOG == 1
00091   Serial.println(str);
00092 #endif
00093 }
```

7.32.3.3 handleInterrupt()

```
void IRAM_ATTR handleInterrupt ()
```

RPM Event Interrupt Enters on falling edge.

Rückgabe

* void

Definiert in Zeile 101 der Datei Motordaten.ino.

```
00102 {
00103   portENTER_CRITICAL_ISR(&mux);
00104   uint64_t TempVal = timerRead(timer);        // value of timer at interrupt
00105   PeriodCount = TempVal - StartValue;         // period count between rising edges in 0.000001 of a
      second
00106   StartValue = TempVal;                       // puts latest reading as start for next calculation
00107   portEXIT_CRITICAL_ISR(&mux);
00108   Last_int_time = millis();
00109 }
```

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

7.32.3.4 setup()

```
void setup ()
```

Filesystem prepare for Webfiles

file exists, reading and loading config file

Read Boardinfo for output

Construct a new pin Mode object

Start OneWire

Set NMEA2000 product information

OTA

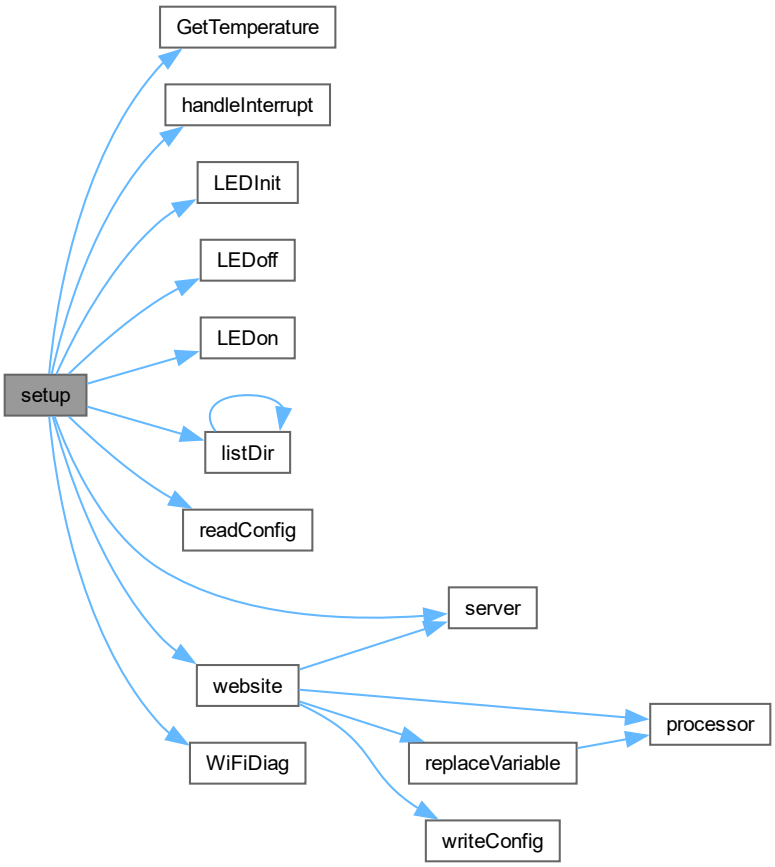
Definiert in Zeile 112 der Datei Motordaten.ino.

```
00112             {
00113
00114   // Init USB serial port
00115   Serial.begin(115200);
00116
00117   Serial.printf("Motordaten setup %s start\n", Version);
00118
00123     if (!LittleFS.begin(true)) {
00124         Serial.println("An Error has occurred while mounting LittleFS");
00125         return;
00126     }
00127     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00128
00129     File root = LittleFS.open("/");
00130   listDir(LittleFS, "/", 3);
00131
00136   readConfig("/config.json");
00137     IP = inet_addr(tAP_Config.wAP_IP);
00138     AP_SSID = tAP_Config.wAP_SSID;
00139     AP_PASSWORD = tAP_Config.wAP_Password;
00140     fMotorOffset = atof(tAP_Config.wMotor_Offset);
00141     fCoolantOffset = atof(tAP_Config.wCoolant_Offset);
00142     FuelLevelMax = atof(tAP_Config.wFuellstandmax);
00143     ADC_Calibration_Value1 = atof(tAP_Config.wADC1_Cal);
00144     ADC_Calibration_Value2 = atof(tAP_Config.wADC2_Cal);
00145     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + " , Passwort:
      " + AP_PASSWORD + " , MotorTOffset: " + fMotorOffset + " , CoolantTOffset: " + fCoolantOffset + " read
      from file");
00146
00147   // LED
00148   LEDInit();
00149
00150   // Boardinfo
00155     sBoardInfo = boardInfo.ShowChipIDtoString();
00156
00157     //Wifi
00158   WiFi.mode(WIFI_AP_STA);
00159   WiFi.softAPdisconnect();
00160   if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00161     WiFi.softAPConfig(IP, Gateway, NMask);
00162     Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00163     Serial.println("\nSet IP " + IP.toString() + " ,Gateway: " + Gateway.toString() + " ,NetMask: " +
      NMask.toString() + " ready");
00164     LEDon(LED(Green));
00165     delay(1000);
00166   } else {
00167       Serial.println("Starting AP failed.");
00168       LEDoff(LED(Green));
00169       LEDon(LED(Red));
00170       delay(1000);
00171       ESP.restart();
00172   }
00173
00174   WiFi.setHostname(HostName);
00175   Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00176
00177   delay(1000);
00178   WiFiDiag();
```

```
00179
00180     if (!MDNS.begin(AP_SSID)) {
00181         Serial.println("Error setting up MDNS responder!");
00182         while (1) {
00183             delay(1000);
00184         }
00185     }
00186   Serial.println("mDNS responder started\n");
00187
00188 // Start TCP (HTTP) server
00189     server.begin();
00190     Serial.println("TCP server started\n");
00191
00192 // Add service to MDNS-SD
00193     MDNS.addService("http", "tcp", 80);
00194   MDNS.addService("ws", "tcp", 81);
00195
00196 // Webconfig laden
00197   website();
00198
00203   pinMode(Eingine_RPM_Pin, INPUT_PULLUP);                                          // sets pin high
00204   attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
      to interrupt on Falling Edge
00205   timer = timerBegin(0, 80, true);                                                 // this returns a
      pointer to the hw_timer_t global variable
00206   // 0 = first timer
00207   // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00208   // true - counts up
00209   timerStart(timer);                                                               // starts the timer
00210
00215   sensors.begin();
00216   oneWire.reset();
00217     Serial.print("OneWire: Found ");
00218     Serial.print(sensors.getDeviceCount(), DEC);
00219     Serial.println(" devices.");
00220     Serial.print("Parasite power is: ");
00221   if (sensors.isParasitePowerMode()) Serial.println("ON");
00222     else Serial.println("OFF");
00223   sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00224
00225   byte i;
00226   byte present = 0;
00227   byte data[12];
00228   byte addr[8];
00229
00230   Serial.print("Looking for 1-Wire devices...\n\r");
00231   while(oneWire.search(addr)) {
00232     Serial.print("\n\rFound \'1-Wire\' device with address:\n\r");
00233     for( i = 0; i < 8; i++) {
00234       Serial.print("0x");
00235       if (addr[i] < 16) {
00236         Serial.print('0');
00237       }
00238       Serial.print(addr[i], HEX);
00239       if (i < 7) {
00240         Serial.print(", ");
00241       }
00242     }
00243     if ( OneWire::crc8( addr, 7) != addr[7]) {
00244         Serial.print("CRC is not valid!\n");
00245         return;
00246     }
00247   }
00248   Serial.print("\n\rNo more sensors!\n\r");
00249   oneWire.reset_search();
00250   delay(250);
00251
00252 // search for devices on the bus and assign based on an index
00253   if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00254   if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00255
00256
00257
00258 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
      or Mega
00259   NMEA2000.SetN2kCANMsgBufSize(8);
00260   NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00261   NMEA2000.SetN2kCANSendFrameBufSize(250);
00262
00263   esp_efuse_mac_get_default(chipid);
00264   for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00265
00270   NMEA2000.SetProductInformation("MD01.2501", // Manufacturer's Model serial code
00271                                  100, // Manufacturer's product code
00272                                  "MD Sensor Module",  // Manufacturer's Model ID
00273                                  "2.5.1.0 (2025-02-20)",  // Manufacturer's Software version code
00274                                  "2.0.0.0 (2024-11-30)" // Manufacturer's Model version
```

```
00275                                 );
00276 // Set device information
00277   NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00278                                 132, // Device function=Analog to NMEA 2000 Gateway. See codes on
      http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20&%20function%20codes%20v%202.00.pdf
00279                                 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
00280                                 2046 // Just choosen free from code list on
      http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00281                                );
00282
00283 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
      below
00284
00285   NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
      Actisense format.
00286
00287   preferences.begin("nvs", false);                          // Open nonvolatile storage (nvs)
00288   NodeAddress = preferences.getInt("LastNodeAddress", 33);  // Read stored last NodeAddress, default
      33
00289   preferences.end();
00290   Serial.printf("NodeAddress=%d\n", NodeAddress);
00291
00292   NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00293   NMEA2000.ExtendTransmitMessages(TransmitMessages);
00294   NMEA2000.Open();
00295
00296  xTaskCreatePinnedToCore(
00297     GetTemperature, /* Function to implement the task */
00298     "Task1", /* Name of the task */
00299     10000,  /* Stack size in words */
00300     NULL,  /* Task input parameter */
00301     0,  /* Priority of the task */
00302     &Task1,  /* Task handle. */
00303     0); /* Core where the task should run */
00304
00305   delay(200);
00306
00311   ArduinoOTA
00312     .onStart([]() {
00313       String type;
00314       if (ArduinoOTA.getCommand() == U_FLASH)
00315         type = "sketch";
00316       else // U_SPIFFS
00317         type = "filesystem";
00318
00319       // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00320       Serial.println("Start updating " + type);
00321     })
00322     .onEnd([]() {
00323       Serial.println("\nEnd");
00324     })
00325     .onProgress([](unsigned int progress, unsigned int total) {
00326       Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00327     })
00328     .onError([](ota_error_t error) {
00329       Serial.printf("Error[%u]: ", error);
00330       if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00331       else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00332       else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00333       else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00334       else if (error == OTA_END_ERROR) Serial.println("End Failed");
00335     });
00336
00337   ArduinoOTA.begin();
00338
00339   printf("Setup end\n");
00340 }
```

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℃.

Parameter

parameter	

Definiert in Zeile 349 der Datei Motordaten.ino.

```
00349                                       {
00350   float tmp0 = 0;
00351   float tmp1 = 0;
00352   for (;;) {
00353     sensors.requestTemperatures();                    // Send the command to get temperatures
00354     vTaskDelay(100);
00355     tmp0 = sensors.getTempC(MotorOil);
00356     if (tmp0 == DEVICE_DISCONNECTED_C) {
```

```
00357         if (motorErrorReported == "Aus") {                          // Nur einmal melden
00358          Serial.print("Error read Motor Temp\n");
00359          motorErrorReported = "Ein";}
00360      MotorTemp = -5.0;
00361      } else {
00362          MotorTemp = tmp0 + fMotorOffset;
00363          motorErrorReported = "Aus";                          // Fehler wurde behoben
00364      }
00365      vTaskDelay(100);
00366      tmp1 = sensors.getTempC(MotorCoolant);
00367      if (tmp1 == DEVICE_DISCONNECTED_C) {
00368         if (coolantErrorReported == "Aus") {                          // Nur einmal melden
00369          Serial.print("Error read Coolant Temp\n");
00370          coolantErrorReported = "Ein";}
00371      CoolantTemp = -5.0;
00372      } else {
00373          CoolantTemp = tmp1 + fCoolantOffset;
00374          coolantErrorReported = "Aus";                         // Fehler wurde behoben
00375      }
00376      vTaskDelay(100);
00377    }
00378
00379    }
```

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.

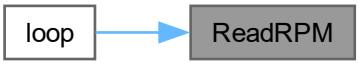
Rückgabe

double

Definiert in Zeile 386 der Datei Motordaten.ino.

```
00386                 {
00387    double RPM = 0;
00388
00389    portENTER_CRITICAL(&mux);
00390    if (PeriodCount != 0) {                          // 0 means no signals measured
00391      RPM = 1000000.00 / PeriodCount;                // PeriodCount in 0.000001 of a second
00392    }
00393    portEXIT_CRITICAL(&mux);
00394    if (millis() > Last_int_time + 200) RPM = 0;       // No signals RPM=0;
00395    return (RPM);
00396  }
```

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



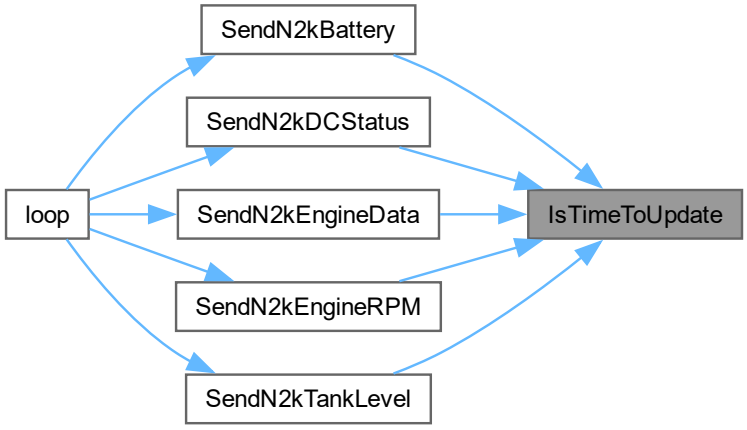
7.32.3.7 IsTimeToUpdate()

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

Definiert in Zeile 399 der Datei Motordaten.ino.

```
00399                                           {
00400   return (NextUpdate < millis());
00401 }
```

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



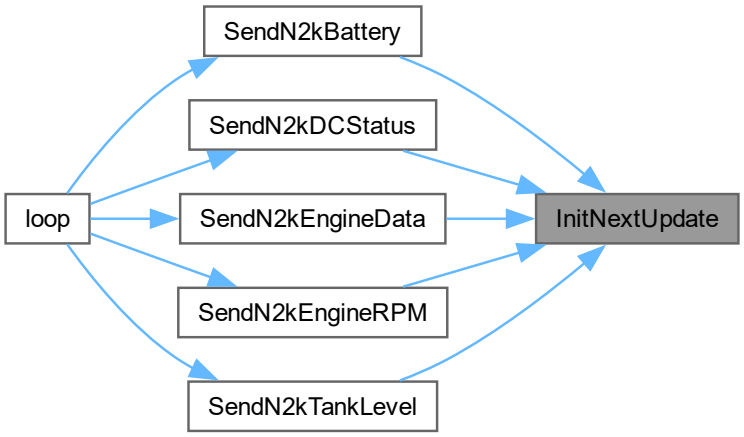
7.32.3.8 InitNextUpdate()

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

Definiert in Zeile 402 der Datei Motordaten.ino.

```
00402                                                                   {
00403   return millis() + Period + Offset;
00404 }
```

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



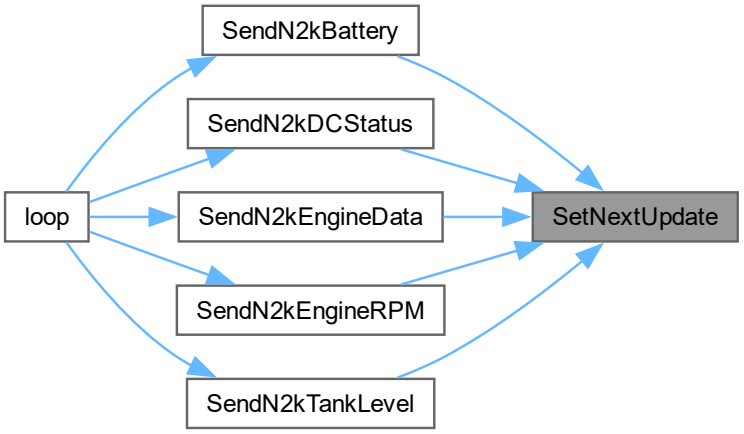
7.32.3.9 SetNextUpdate()

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

Definiert in Zeile 406 der Datei Motordaten.ino.

```
00406                                                                   {
00407   while ( NextUpdate < millis() ) NextUpdate += Period;
00408 }
```

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



7.32.3.10 SendN2kDCStatus()

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

Send PGN127506.

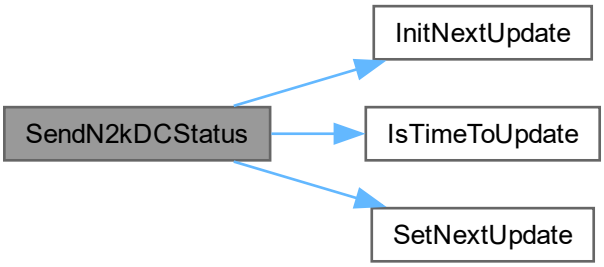
Parameter

BatteryVoltage	
SoC	
BatCapacity	

Definiert in Zeile 418 der Datei Motordaten.ino.

```
00418                                                                                {
00419   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
      BatteryDCStatusSendOffset);
00420   tN2kMsg N2kMsg;
00421
00422   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00423     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00424
00425     Serial.printf("Voltage     : %3.1f V\n", BatteryVoltage);
00426     Serial.printf("SoC         : %3.1f %\n", SoC);
00427     Serial.printf("Capacity    : %3.1f Ah\n", BatCapacity);
00428     // SetN2kDCStatus(N2kMsg,1,1,N2kDCt_Battery,56,92,38500,0.012, AhToCoulomb(420));
00429     SetN2kDCStatus(N2kMsg, 1, 2, N2kDCt_Battery, SoC, 0,  N2kDoubleNA, BatteryVoltage,
      AhToCoulomb(55));
00430     NMEA2000.SendMsg(N2kMsg);
00431   }
00432 }
```

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



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

7.32.3.11 SendN2kBattery()

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

Send PGN127508.

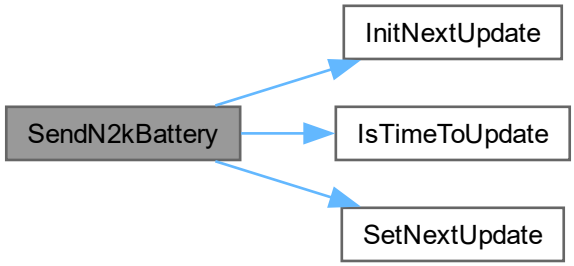
Parameter

BatteryVoltage	

Definiert in Zeile 439 der Datei Motordaten.ino.

```
00439                                          {
00440   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00441   tN2kMsg N2kMsg;
00442
00443   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00444     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00445
00446     Serial.printf("Voltage     : %3.1f V\n", BatteryVoltage);
00447
00448     SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00449     NMEA2000.SendMsg(N2kMsg);
00450   }
00451 }
```

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



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

7.32.3.12 SendN2kTankLevel()

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

Send PGN 127505.

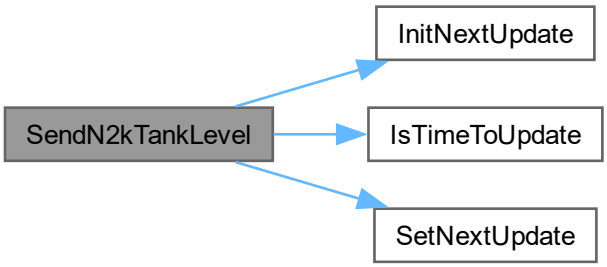
Parameter

level	
capacity	

Definiert in Zeile 459 der Datei Motordaten.ino.

```
00459                                                 {
00460   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00461   tN2kMsg N2kMsg;
00462
00463   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00464     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00465
00466     Serial.printf("Fuel Level    : %3.1f %%\n", level);
00467     Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00468
00469     SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00470     NMEA2000.SendMsg(N2kMsg);
00471   }
00472 }
```

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



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

7.32.3.13 SendN2kEngineData()

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

Send PGN 127489.

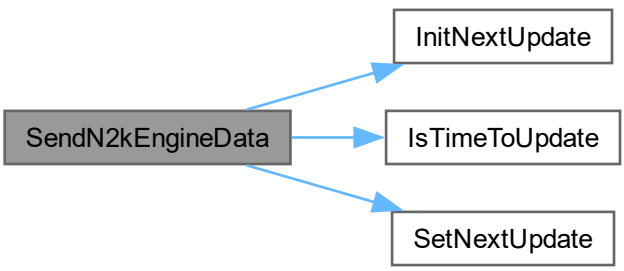
Parameter

Oiltemp	
Coolanttemp	
rpm	
hours	
voltage	

Definiert in Zeile 483 der Datei Motordaten.ino.

```
00483                                                                                                                {
00484   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00485   tN2kMsg N2kMsg;
00486   tN2kEngineDiscreteStatus1 Status1;
00487   tN2kEngineDiscreteStatus2 Status2;
00488   Status1.Bits.OverTemperature = Oiltemp > 90;      // Alarm Motor over temp
00489   Status1.Bits.LowCoolantLevel = Coolanttemp > 90;    // Alarm low cooling
00490   Status1.Bits.LowSystemVoltage = voltage < 11;
00491   Status2.Bits.EngineShuttingDown = rpm < 100;      // Alarm Motor off
00492   EngineOn = !Status2.Bits.EngineShuttingDown;
00493
00494   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00495     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00496
00497     Serial.printf("Oil Temp    : %3.1f °C \n", Oiltemp);
00498     Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
00499     Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00500     Serial.printf("Overtemp Oil: %s  \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00501     Serial.printf("Overtemp Mot: %s  \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00502     Serial.printf("Engine Off  : %s  \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00503
00504     // SetN2kTemperatureExt(N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
      // PGN130312, uncomment the PGN to be used
00505
00506     SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Coolanttemp),
      N2kDoubleNA, N2kDoubleNA, hours ,N2kDoubleNA ,N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00507
00508     NMEA2000.SendMsg(N2kMsg);
00509   }
00510 }
```

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



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

7.32.3.14 SendN2kEngineRPM()

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

Send PGN 127488.

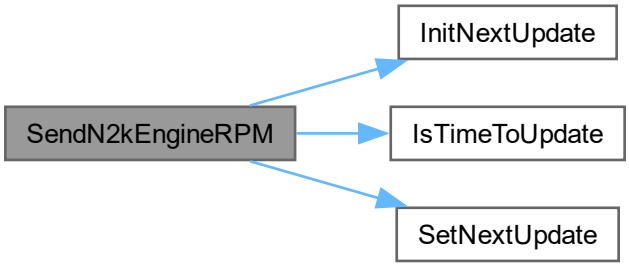
Parameter

RPM	

Definiert in Zeile 517 der Datei Motordaten.ino.

```
00517                                 {
00518   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMSendOffset);
00519   tN2kMsg N2kMsg;
00520
00521   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00522     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00523
00524     Serial.printf("Engine RPM  : %4.0f RPM \n", RPM);
00525
00526     SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA,  N2kInt8NA);
00527
00528     NMEA2000.SendMsg(N2kMsg);
00529   }
00530 }
```

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



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

7.32.3.15 ReadVoltage()

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

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

Parameter

pin	

Rückgabe

double

Definiert in Zeile 538 der Datei Motordaten.ino.

```
00538                                {
00539   double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in
      range 0 to 4095
00540   if (reading < 1 || reading > 4095) return 0;
00541   // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +
      0.000854596860691 * reading + 0.065440348345433;
00542   return (-0.000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -
      0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.0341435246340 89) * 1000;
00543 } // Added an improved polynomial, use either, comment out as required
```

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

7.32.3.16 loop()

```
void loop ()
```

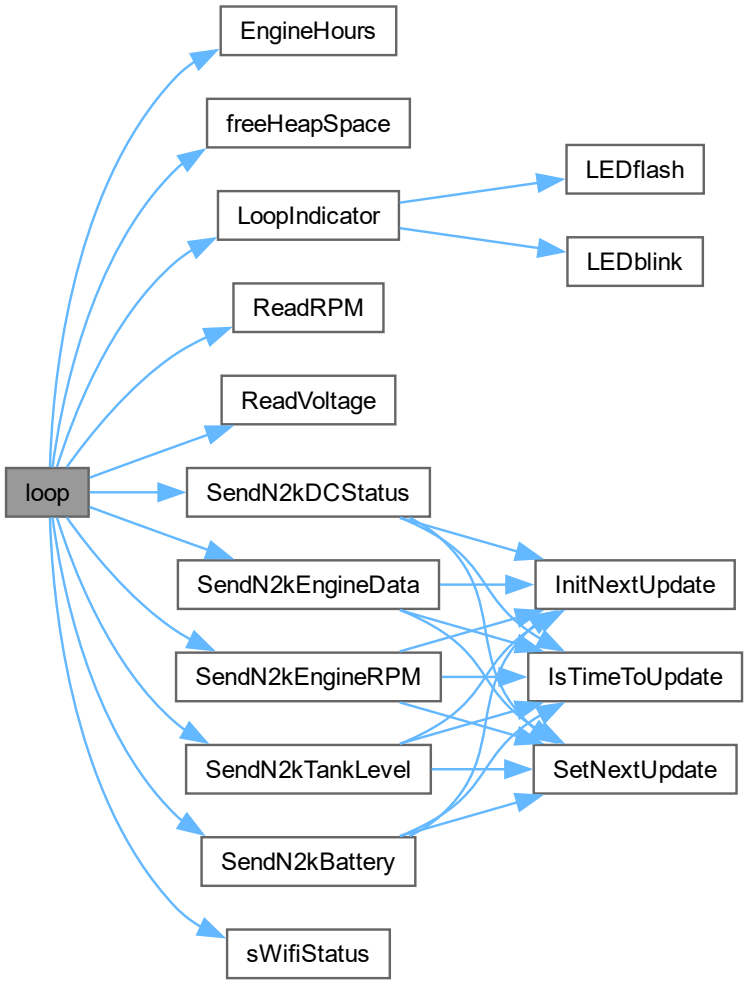
Actual Website Data

Construct a new if object Reboot from Website

Definiert in Zeile 546 der Datei Motordaten.ino.

```
00546             {
00547
00548   LoopIndicator();
00549
00550   BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
      // This implements a low pass filter to eliminate spike for ADC readings
00551
00552   FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
      This implements a low pass filter to eliminate spike for ADC readings
00553
00554   EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
      pass filter to eliminate spike for RPM measurements
00555
00556   BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
      Zustand über Spannung
00557   // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00558
00559   EngineHours(EngineOn);
00560
00561   SendN2kTankLevel(FuelLevel, FuelLevelMax);  // Adjust max tank capacity
00562   SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00563   SendN2kEngineRPM(EngineRPM);
00564   SendN2kBattery(BordSpannung);
00565   SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00566
00567   NMEA2000.ParseMessages();
00568   int SourceAddress = NMEA2000.GetN2kSource();
00569   if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00570     NodeAddress = SourceAddress;      // Set new Node Address (to save only once)
00571     preferences.begin("nvs", false);
00572     preferences.putInt("LastNodeAddress", SourceAddress);
00573     preferences.end();
00574     Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00575   }
00576
00577   // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00578   if ( Serial.available() ) {
00579     Serial.read();
00580   }
00581
00582
00583 // OTA
00584     ArduinoOTA.handle();
00585
00590     webSocket.loop();
00591     fCoolantTemp = CoolantTemp;
00592     fMotorTemp = MotorTemp;
00593     fBordSpannung = BordSpannung;
00594     fDrehzahl = EngineRPM;
00595     sCL_Status = sWifiStatus(WiFi.status());
00596     sAP_Station = WiFi.softAPgetStationNum();
00597     freeHeapSpace();
00598
00603   if (IsRebootRequired) {
00604       Serial.println("Rebooting ESP32: ");
00605       delay(1000); // give time for reboot page to load
00606       ESP.restart();
00607       }
00608
00609
00610 }
```

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



7.32.4 Variablen-Dokumentation

7.32.4.1 PROGMEM

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

Initialisierung:

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

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

Definiert in Zeile 51 der Datei Motordaten.ino.

```
00051                                                 {127488L, // Engine Rapid / RPM
00052                                                  127489L, // Engine parameters dynamic
00053                                                  127505L, // Fluid Level
00054                                                  127506L, // Battery
00055                                                  127508L, // Battery Status
00056                                                  0
00057                                                 };
```

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 64 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 65 der Datei Motordaten.ino.

7.32.4.4 Last_int_time

```
unsigned long Last_int_time = 0
```

Definiert in Zeile 66 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 67 der Datei Motordaten.ino.

7.32.4.6 mux

```
portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED
```

synchs between maon cose and interrupt?

Definiert in Zeile 68 der Datei Motordaten.ino.

7.32.4.7 oneWire

```
DallasTemperature sensors& oneWire
```

Pass our oneWire reference to Dallas Temperature.

Definiert in Zeile 74 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 76 der Datei Motordaten.ino.

```
00076 { 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 77 der Datei Motordaten.ino.

```
00077 { 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 79 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 80 der Datei Motordaten.ino.

7.32.4.12 Task1

```
TaskHandle_t Task1
```

Task handle for OneWire read (Core 0 on ESP32)

Definiert in Zeile 83 der Datei Motordaten.ino.

7.32.4.13 baudrate

```
const int baudrate = 38400
```

Serial port 2 config (GPIO 16)

Definiert in Zeile 86 der Datei Motordaten.ino.

7.32.4.14 rs_config

```
const int rs_config = SERIAL_8N1
```

Definiert in Zeile 87 der Datei Motordaten.ino.

7.33 Motordaten.ino

gehe zur Dokumentation dieser Datei

```
00001
00002 /*
00003   This code is free software; you can redistribute it and/or
00004   modify it under the terms of the GNU Lesser General Public
00005   License as published by the Free Software Foundation; either
00006   version 2.1 of the License, or (at your option) any later version.
00007   This code is distributed in the hope that it will be useful,
00008   but WITHOUT ANY WARRANTY; without even the implied warranty of
00009   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
00010   Lesser General Public License for more details.
00011   You should have received a copy of the GNU Lesser General Public
00012   License along with this library; if not, write to the Free Software
00013   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
00014 */
00015
00026
00027 #include <Arduino.h>
00028 #include "configuration.h"
00029 #include <Preferences.h>
00030 #include <ArduinoOTA.h>
00031 #include <OneWire.h>
00032 #include <DallasTemperature.h>
00033 #include <ESP_WiFi.h>
00034 #include <ESPAsyncWebServer.h>
00035 #include <NMEA2000_CAN.h>  // This will automatically choose right CAN library and create suitable
      NMEA2000 object
00036 #include <N2kMessages.h>
00037 #include <ESPmDNS.h>
00038 #include <arpa/inet.h>
00039 #include "BoardInfo.h"
00040 #include "helper.h"
00041 #include "web.h"
00042 #include "hourmeter.h"
00043 #include "LEDindicator.h"
00044
00045 #define ENABLE_DEBUG_LOG 0 // Debug log
00046
00047
00051 const unsigned long TransmitMessages[] PROGMEM = {127488L, // Engine Rapid / RPM
00052                                                   127489L, // Engine parameters dynamic
00053                                                   127505L, // Fluid Level
00054                                                   127506L, // Battery
00055                                                   127508L, // Battery Status
00056                                                   0
00057                                                  };
00058
00059
00063
00064 volatile uint64_t StartValue = 0;
00065 volatile uint64_t PeriodCount = 0;
00066 unsigned long Last_int_time = 0;
00067 hw_timer_t * timer = NULL;
00068 portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
00069
00073 OneWire oneWire(ONE_WIRE_BUS);
00074 DallasTemperature sensors(&oneWire);
00075 // DeviceAddress MotorThermometer;    /**< arrays to hold device addresses
00076 uint8_t MotorCoolant[8] = { 0x28, 0xD3, 0x81, 0xCF, 0x0F, 0x0, 0x0, 0x79 };
00077 uint8_t MotorOil[8] = { 0x28, 0xB0, 0x3C, 0x1A, 0xF, 0x0, 0x0, 0xC0 };
00078
00079 const int ADCpin2 = 35;
00080 const int ADCpin1 = 34;
00081
00083 TaskHandle_t Task1;
00084
00086 const int baudrate = 38400;
00087 const int rs_config = SERIAL_8N1;
00088
00089 void debug_log(char* str) {
```

```
00090 #if ENABLE_DEBUG_LOG == 1
00091   Serial.println(str);
00092 #endif
00093 }
00094
00100 //=====================================
00101 void IRAM_ATTR handleInterrupt()
00102 {
00103   portENTER_CRITICAL_ISR(&mux);
00104   uint64_t TempVal = timerRead(timer);        // value of timer at interrupt
00105   PeriodCount = TempVal - StartValue;         // period count between rising edges in 0.000001 of a
      second
00106   StartValue = TempVal;                       // puts latest reading as start for next calculation
00107   portEXIT_CRITICAL_ISR(&mux);
00108   Last_int_time = millis();
00109 }
00110
00111 /****************************************** Setup
      ******************************************************/
00112 void setup() {
00113
00114   // Init USB serial port
00115   Serial.begin(115200);
00116
00117   Serial.printf("Motordaten setup %s start\n", Version);
00118
00123     if (!LittleFS.begin(true)) {
00124         Serial.println("An Error has occurred while mounting LittleFS");
00125         return;
00126     }
00127     Serial.println("\nBytes LittleFS used:" + String(LittleFS.usedBytes()));
00128
00129     File root = LittleFS.open("/");
00130   listDir(LittleFS, "/", 3);
00131
00136   readConfig("/config.json");
00137     IP = inet_addr(tAP_Config.wAP_IP);
00138     AP_SSID = tAP_Config.wAP_SSID;
00139     AP_PASSWORD = tAP_Config.wAP_Password;
00140     fMotorOffset = atof(tAP_Config.wMotor_Offset);
00141     fCoolantOffset = atof(tAP_Config.wCoolant_Offset);
00142     FuelLevelMax = atof(tAP_Config.wFuellstandmax);
00143     ADC_Calibration_Value1 = atof(tAP_Config.wADC1_Cal);
00144     ADC_Calibration_Value2 = atof(tAP_Config.wADC2_Cal);
00145     Serial.println("\nConfigdata : AP IP: " + IP.toString() + ", AP SSID: " + AP_SSID + " , Passwort:
      " + AP_PASSWORD + " , MotorTOffset: " + fMotorOffset + " , CoolantTOffset: " + fCoolantOffset + " read
      from file");
00146
00147   // LED
00148   LEDInit();
00149
00150   // Boardinfo
00155     sBoardInfo = boardInfo.ShowChipIDtoString();
00156
00157     //Wifi
00158   WiFi.mode(WIFI_AP_STA);
00159   WiFi.softAPdisconnect();
00160   if(WiFi.softAP(AP_SSID, AP_PASSWORD, channel, hide_SSID, max_connection)){
00161     WiFi.softAPConfig(IP, Gateway, NMask);
00162     Serial.println("\nAccesspoint " + String(AP_SSID) + " running");
00163     Serial.println("\nSet IP " + IP.toString() + " ,Gateway: " + Gateway.toString() + " ,NetMask: " +
      NMask.toString() + " ready");
00164     LEDon(LED(Green));
00165     delay(1000);
00166   } else {
00167       Serial.println("Starting AP failed.");
00168       LEDoff(LED(Green));
00169       LEDon(LED(Red));
00170       delay(1000);
00171       ESP.restart();
00172   }
00173
00174   WiFi.setHostname(HostName);
00175   Serial.println("Set Hostname " + String(WiFi.getHostname()) + " done\n");
00176
00177   delay(1000);
00178   WiFiDiag();
00179
00180     if (!MDNS.begin(AP_SSID)) {
00181         Serial.println("Error setting up MDNS responder!");
00182         while (1) {
00183             delay(1000);
00184         }
00185     }
00186   Serial.println("mDNS responder started\n");
00187
00188 // Start TCP (HTTP) server
```

```
00189     server.begin();
00190     Serial.println("TCP server started\n");
00191
00192 // Add service to MDNS-SD
00193     MDNS.addService("http", "tcp", 80);
00194   MDNS.addService("ws", "tcp", 81);
00195
00196 // Webconfig laden
00197   website();
00198
00203   pinMode(Eingine_RPM_Pin, INPUT_PULLUP);                                    // sets pin high
00204   attachInterrupt(digitalPinToInterrupt(Eingine_RPM_Pin), handleInterrupt, FALLING); // attaches pin
      to interrupt on Falling Edge
00205   timer = timerBegin(0, 80, true);                                          // this returns a
      pointer to the hw_timer_t global variable
00206   // 0 = first timer
00207   // 80 is prescaler so 80MHZ divided by 80 = 1MHZ signal ie 0.000001 of a second
00208   // true - counts up
00209   timerStart(timer);                                                        // starts the timer
00210
00215   sensors.begin();
00216   oneWire.reset();
00217     Serial.print("OneWire: Found ");
00218     Serial.print(sensors.getDeviceCount(), DEC);
00219     Serial.println(" devices.");
00220     Serial.print("Parasite power is: ");
00221   if (sensors.isParasitePowerMode()) Serial.println("ON");
00222     else Serial.println("OFF");
00223   sOneWire_Status = String(sensors.getDeviceCount(), DEC);
00224
00225   byte i;
00226   byte present = 0;
00227   byte data[12];
00228   byte addr[8];
00229
00230   Serial.print("Looking for 1-Wire devices...\n\r");
00231   while(oneWire.search(addr)) {
00232     Serial.print("\n\rFound \'1-Wire\' device with address:\n\r");
00233     for( i = 0; i < 8; i++) {
00234       Serial.print("0x");
00235       if (addr[i] < 16) {
00236         Serial.print('0');
00237       }
00238       Serial.print(addr[i], HEX);
00239       if (i < 7) {
00240         Serial.print(", ");
00241       }
00242     }
00243     if ( OneWire::crc8( addr, 7) != addr[7]) {
00244         Serial.print("CRC is not valid!\n");
00245         return;
00246     }
00247   }
00248   Serial.print("\n\rNo more sensors!\n\r");
00249   oneWire.reset_search();
00250   delay(250);
00251
00252 // search for devices on the bus and assign based on an index
00253   if (!sensors.getAddress(MotorOil, 0)) Serial.println("Unable to find address for Device 0");
00254   if (!sensors.getAddress(MotorCoolant, 1)) Serial.println("Unable to find address for Device 1");
00255
00256
00257
00258 // Reserve enough buffer for sending all messages. This does not work on small memory devices like Uno
      or Mega
00259   NMEA2000.SetN2kCANMsgBufSize(8);
00260   NMEA2000.SetN2kCANReceiveFrameBufSize(250);
00261   NMEA2000.SetN2kCANSendFrameBufSize(250);
00262
00263   esp_efuse_mac_get_default(chipid);
00264   for (i = 0; i < 6; i++) id += (chipid[i] << (7 * i));
00265
00270   NMEA2000.SetProductInformation("MD01.2501", // Manufacturer's Model serial code
00271                                  100, // Manufacturer's product code
00272                                  "MD Sensor Module",  // Manufacturer's Model ID
00273                                  "2.5.1.0 (2025-02-20)",  // Manufacturer's Software version code
00274                                  "2.0.0.0 (2024-11-30)" // Manufacturer's Model version
00275                                 );
00276 // Set device information
00277   NMEA2000.SetDeviceInformation(id, // Unique number. Use e.g. Serial number.
00278                                 132, // Device function=Analog to NMEA 2000 Gateway. See codes on
      http://www.nmea.org/Assets/20120726%20nmea%202000%20class%20&%20function%20codes%20v%202.00.pdf
00279                                 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
00280                                 2046 // Just choosen free from code list on
      http://www.nmea.org/Assets/20121020%20nmea%202000%20registration%20list.pdf
00281                                );
```

```
00282
00283 // If you also want to see all traffic on the bus use N2km_ListenAndNode instead of N2km_NodeOnly
      below
00284
00285   NMEA2000.SetForwardType(tNMEA2000::fwdt_Text); // Show in clear text. Leave uncommented for default
      Actisense format.
00286
00287   preferences.begin("nvs", false);                          // Open nonvolatile storage (nvs)
00288   NodeAddress = preferences.getInt("LastNodeAddress", 33);  // Read stored last NodeAddress, default
      33
00289   preferences.end();
00290   Serial.printf("NodeAddress=%d\n", NodeAddress);
00291
00292   NMEA2000.SetMode(tNMEA2000::N2km_ListenAndNode, NodeAddress);
00293   NMEA2000.ExtendTransmitMessages(TransmitMessages);
00294   NMEA2000.Open();
00295
00296  xTaskCreatePinnedToCore(
00297     GetTemperature, /* Function to implement the task */
00298     "Task1", /* Name of the task */
00299     10000,  /* Stack size in words */
00300     NULL,  /* Task input parameter */
00301     0,  /* Priority of the task */
00302     &Task1,  /* Task handle. */
00303     0); /* Core where the task should run */
00304
00305   delay(200);
00306
00311   ArduinoOTA
00312     .onStart([]() {
00313       String type;
00314       if (ArduinoOTA.getCommand() == U_FLASH)
00315         type = "sketch";
00316       else // U_SPIFFS
00317         type = "filesystem";
00318
00319       // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
00320       Serial.println("Start updating " + type);
00321     })
00322     .onEnd([]() {
00323       Serial.println("\nEnd");
00324     })
00325     .onProgress([](unsigned int progress, unsigned int total) {
00326       Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
00327     })
00328     .onError([](ota_error_t error) {
00329       Serial.printf("Error[%u]: ", error);
00330       if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
00331       else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
00332       else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
00333       else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
00334       else if (error == OTA_END_ERROR) Serial.println("End Failed");
00335     });
00336
00337   ArduinoOTA.begin();
00338
00339   printf("Setup end\n");
00340 }
00341
00348
00349 void GetTemperature( void * parameter) {
00350   float tmp0 = 0;
00351   float tmp1 = 0;
00352   for (;;) {
00353     sensors.requestTemperatures();                        // Send the command to get temperatures
00354     vTaskDelay(100);
00355     tmp0 = sensors.getTempC(MotorOil);
00356     if (tmp0 == DEVICE_DISCONNECTED_C) {
00357        if (motorErrorReported == "Aus") {                          // Nur einmal melden
00358         Serial.print("Error read Motor Temp\n");
00359         motorErrorReported = "Ein";}
00360     MotorTemp = -5.0;
00361     } else {
00362         MotorTemp = tmp0 + fMotorOffset;
00363         motorErrorReported = "Aus";                       // Fehler wurde behoben
00364     }
00365     vTaskDelay(100);
00366     tmp1 = sensors.getTempC(MotorCoolant);
00367     if (tmp1 == DEVICE_DISCONNECTED_C) {
00368        if (coolantErrorReported == "Aus") {                        // Nur einmal melden
00369         Serial.print("Error read Coolant Temp\n");
00370         coolantErrorReported = "Ein";}
00371     CoolantTemp = -5.0;
00372     } else {
00373         CoolantTemp = tmp1 + fCoolantOffset;
00374         coolantErrorReported = "Aus";                      // Fehler wurde behoben
00375     }
```

```
00376     vTaskDelay(100);
00377   }
00378
00379   }
00380
00386 double ReadRPM() {
00387   double RPM = 0;
00388
00389   portENTER_CRITICAL(&mux);
00390   if (PeriodCount != 0) {                          // 0 means no signals measured
00391     RPM = 1000000.00 / PeriodCount;                // PeriodCount in 0.000001 of a second
00392   }
00393   portEXIT_CRITICAL(&mux);
00394   if (millis() > Last_int_time + 200) RPM = 0;       // No signals RPM=0;
00395   return (RPM);
00396 }
00397
00398
00399 bool IsTimeToUpdate(unsigned long NextUpdate) {
00400   return (NextUpdate < millis());
00401 }
00402 unsigned long InitNextUpdate(unsigned long Period, unsigned long Offset = 0) {
00403   return millis() + Period + Offset;
00404 }
00405
00406 void SetNextUpdate(unsigned long &NextUpdate, unsigned long Period) {
00407   while ( NextUpdate < millis() ) NextUpdate += Period;
00408 }
00409
00410 /*********************** n2k Datenfunktionen ***************************/
00418 void SendN2kDCStatus(double BatteryVoltage, double SoC, double BatCapacity) {
00419   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod,
      BatteryDCStatusSendOffset);
00420   tN2kMsg N2kMsg;
00421
00422   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00423     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00424
00425     Serial.printf("Voltage     : %3.1f V\n", BatteryVoltage);
00426     Serial.printf("SoC         : %3.1f %\n", SoC);
00427     Serial.printf("Capacity    : %3.1f Ah\n", BatCapacity);
00428     // SetN2kDCStatus(N2kMsg,1,1,N2kDCt_Battery,56,92,38500,0.012, AhToCoulomb(420));
00429     SetN2kDCStatus(N2kMsg, 1, 2, N2kDCt_Battery, SoC, 0,  N2kDoubleNA, BatteryVoltage,
      AhToCoulomb(55));
00430     NMEA2000.SendMsg(N2kMsg);
00431   }
00432 }
00433
00439 void SendN2kBattery(double BatteryVoltage) {
00440   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, BatteryDCSendOffset);
00441   tN2kMsg N2kMsg;
00442
00443   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00444     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00445
00446     Serial.printf("Voltage     : %3.1f V\n", BatteryVoltage);
00447
00448     SetN2kDCBatStatus(N2kMsg, 2, BatteryVoltage, N2kDoubleNA, N2kDoubleNA, 1);
00449     NMEA2000.SendMsg(N2kMsg);
00450   }
00451 }
00452
00459 void SendN2kTankLevel(double level, double capacity) {
00460   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, TankSendOffset);
00461   tN2kMsg N2kMsg;
00462
00463   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00464     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00465
00466     Serial.printf("Fuel Level   : %3.1f %%\n", level);
00467     Serial.printf("Fuel Capacity: %3.1f l\n", capacity);
00468
00469     SetN2kFluidLevel(N2kMsg, 0, N2kft_Fuel, level, capacity );
00470     NMEA2000.SendMsg(N2kMsg);
00471   }
00472 }
00473
00483 void SendN2kEngineData(double Oiltemp, double Coolanttemp, double rpm, double hours, double voltage) {
00484   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, EngineSendOffset);
00485   tN2kMsg N2kMsg;
00486   tN2kEngineDiscreteStatus1 Status1;
00487   tN2kEngineDiscreteStatus2 Status2;
00488   Status1.Bits.OverTemperature = Oiltemp > 90;      // Alarm Motor over temp
00489   Status1.Bits.LowCoolantLevel = Coolanttemp > 90;    // Alarm low cooling
00490   Status1.Bits.LowSystemVoltage = voltage < 11;
00491   Status2.Bits.EngineShuttingDown = rpm < 100;      // Alarm Motor off
00492   EngineOn = !Status2.Bits.EngineShuttingDown;
```

```
00493
00494   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00495     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00496
00497     Serial.printf("Oil Temp    : %3.1f °C \n", Oiltemp);
00498     Serial.printf("Coolant Temp: %3.1f °C \n", Coolanttemp);
00499     Serial.printf("Engine Hours: %3.1f hrs \n", hours);
00500     Serial.printf("Overtemp Oil: %s  \n", Status1.Bits.OverTemperature ? "Yes" : "No");
00501     Serial.printf("Overtemp Mot: %s  \n", Status1.Bits.LowCoolantLevel ? "Yes" : "No");
00502     Serial.printf("Engine Off  : %s  \n", Status2.Bits.EngineShuttingDown ? "Yes" : "No");
00503
00504     // SetN2kTemperatureExt(N2kMsg, 0, 0, N2kts_ExhaustGasTemperature, CToKelvin(temp), N2kDoubleNA);
      // PGN130312, uncomment the PGN to be used
00505
00506     SetN2kEngineDynamicParam(N2kMsg, 0, N2kDoubleNA, CToKelvin(Oiltemp), CToKelvin(Coolanttemp),
      N2kDoubleNA, N2kDoubleNA, hours ,N2kDoubleNA ,N2kDoubleNA, N2kInt8NA, N2kInt8NA, Status1, Status2);
00507
00508     NMEA2000.SendMsg(N2kMsg);
00509   }
00510 }
00511
00517 void SendN2kEngineRPM(double RPM) {
00518   static unsigned long SlowDataUpdated = InitNextUpdate(SlowDataUpdatePeriod, RPMSendOffset);
00519   tN2kMsg N2kMsg;
00520
00521   if ( IsTimeToUpdate(SlowDataUpdated) ) {
00522     SetNextUpdate(SlowDataUpdated, SlowDataUpdatePeriod);
00523
00524     Serial.printf("Engine RPM  : %4.0f RPM \n", RPM);
00525
00526     SetN2kEngineParamRapid(N2kMsg, 0, RPM, N2kDoubleNA,  N2kInt8NA);
00527
00528     NMEA2000.SendMsg(N2kMsg);
00529   }
00530 }
00531
00538 double ReadVoltage(byte pin) {
00539   double reading = analogRead(pin); // Reference voltage is 3v3 so maximum reading is 3v3 = 4095 in
      range 0 to 4095
00540   if (reading < 1 || reading > 4095) return 0;
00541   // return -0.000000000009824 * pow(reading,3) + 0.000000016557283 * pow(reading,2) +
      0.000854596860691 * reading + 0.065440348345433;
00542   return (-0.000000000000016 * pow(reading, 4) + 0.000000000118171 * pow(reading, 3) -
      0.000000301211691 * pow(reading, 2) + 0.001109019271794 * reading + 0.034143524634089) * 1000;
00543 } // Added an improved polynomial, use either, comment out as required
00544
00545 /********************************** Loop **********************************/
00546 void loop() {
00547
00548   LoopIndicator();
00549
00550   BordSpannung = ((BordSpannung * 15) + (ReadVoltage(ADCpin2) * ADC_Calibration_Value2 / 4096)) / 16;
      // This implements a low pass filter to eliminate spike for ADC readings
00551
00552   FuelLevel = ((FuelLevel * 15) + (ReadVoltage(ADCpin1) * ADC_Calibration_Value1 / 4096)) / 16; //
      This implements a low pass filter to eliminate spike for ADC readings
00553
00554   EngineRPM = ((EngineRPM * 5) + ReadRPM() * RPM_Calibration_Value) / 6 ; // This implements a low
      pass filter to eliminate spike for RPM measurements
00555
00556   BatSoC = (BordSpannung - 10.5) * (100.0 - 0.0) / (14.9 - 10.5) + 0.0; // PB-Batterie im unbelasteten
      Zustand über Spannung
00557   // float BatSoC = analogInScale(BordSpannung, 15, 10, 100.0, 0.0, SoCError);
00558
00559   EngineHours(EngineOn);
00560
00561   SendN2kTankLevel(FuelLevel, FuelLevelMax);  // Adjust max tank capacity
00562   SendN2kEngineData(MotorTemp, CoolantTemp, EngineRPM, Counter, BordSpannung);
00563   SendN2kEngineRPM(EngineRPM);
00564   SendN2kBattery(BordSpannung);
00565   SendN2kDCStatus(BordSpannung, BatSoC, Bat1Capacity);
00566
00567   NMEA2000.ParseMessages();
00568   int SourceAddress = NMEA2000.GetN2kSource();
00569   if (SourceAddress != NodeAddress) { // Save potentially changed Source Address to NVS memory
00570     NodeAddress = SourceAddress;      // Set new Node Address (to save only once)
00571     preferences.begin("nvs", false);
00572     preferences.putInt("LastNodeAddress", SourceAddress);
00573     preferences.end();
00574     Serial.printf("Address Change: New Address=%d\n", SourceAddress);
00575   }
00576
00577   // Dummy to empty input buffer to avoid board to stuck with e.g. NMEA Reader
00578   if ( Serial.available() ) {
00579     Serial.read();
00580   }
00581
```

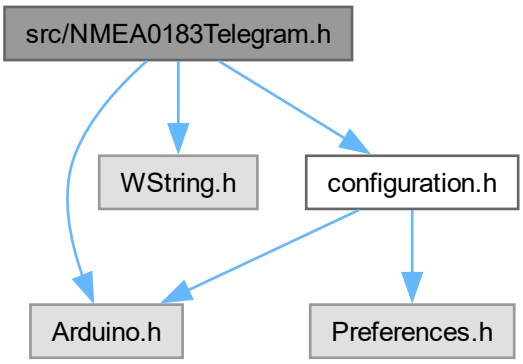
```
00582
00583 // OTA
00584     ArduinoOTA.handle();
00585
00590     webSocket.loop();
00591     fCoolantTemp = CoolantTemp;
00592     fMotorTemp = MotorTemp;
00593     fBordSpannung = BordSpannung;
00594     fDrehzahl = EngineRPM;
00595     sCL_Status = sWifiStatus(WiFi.status());
00596     sAP_Station = WiFi.softAPgetStationNum();
00597     freeHeapSpace();
00598
00603   if (IsRebootRequired) {
00604       Serial.println("Rebooting ESP32: ");
00605       delay(1000); // give time for reboot page to load
00606       ESP.restart();
00607       }
00608
00609
00610 }
```

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()

```
char CheckSum (
            String NMEAData)
```

Checksum calculation for NMEA.

Parameter

NMEAData	

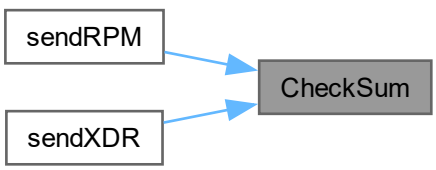
Rückgabe

char

Definiert in Zeile 23 der Datei NMEA0183Telegram.h.

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

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



7.34.2.2 sendXDR()

```
String sendXDR ()
```

Send NMEA0183 Send XDR Sensor data.

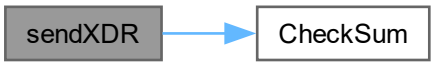
Rückgabe

String

Definiert in Zeile 76 der Datei NMEA0183Telegram.h.

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

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



7.34.2.3 sendRPM()

```
String sendRPM ()
```

Send NMEA0183 Send RPM Sensor data.

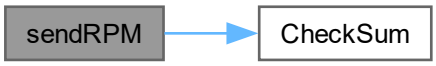
Rückgabe

String

Definiert in Zeile 105 der Datei NMEA0183Telegram.h.

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

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



7.35 NMEA0183Telegram.h

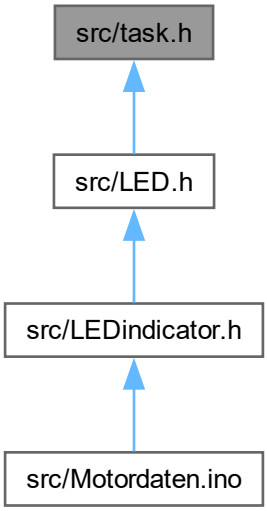
gehe zur Dokumentation dieser Datei

```
00001
00011
00012 #include <Arduino.h>
00013 #include <WString.h>        // Needs for structures
00014 #include "configuration.h"
00015
00022
00023 char CheckSum(String NMEAData) {
00024   char checksum = 0;
00025   // Iterate over the string, XOR each byte with the total sum
00026   for (int c = 0; c < NMEAData.length(); c++) {
00027     checksum = char(checksum ^ NMEAData.charAt(c));
00028   }
00029   // Return the result
00030   return checksum;
00031 }
00032
00033 /*
00034 XDR
00035 Transducer Values
00036             1 2   3 4       n
00037 |   |   |   |       | \\
00038 *  $--XDR,a,x.x,a,c--c, ..... *hh<CR><LF> \\
00039
00040     Field Number:
00041       1) Transducer Type
00042       2) Measurement Data
00043       3) Units of measurement
00044       4) Name of transducer
00045       x) More of the same
```

```
00046       n) Checksum
00047
00048     Example:
00049     Temperatur $IIXDR,C,19.52,C,TempAir*19
00050     Druck      $IIXDR,P,1.02481,B,Barometer*29
00051     Kraengung  $IIXDR,A,0,x.x,ROLL*hh<CR><LF>
00052
00053
00054   RPM - Revolutions
00055
00056         1 2 3   4   5 6
00057         | | |   |   | |
00058  $--RPM,a,x,x.x,x.x,A*hh<CR><LF>
00059
00060     Field Number:
00061       1) Sourse, S = Shaft, E = Engine
00062       2) Engine or shaft number
00063       3) Speed, Revolutions per minute
00064       4) Propeller pitch, % of maximum, "-" means astern
00065       5) Status, A means data is valid
00066       6) Checksum
00067
00068 */
00069
00075
00076 String sendXDR()
00077 {
00078   String HexCheckSum;
00079   String NMEASensor;
00080   String SendSensor;
00081
00082     NMEASensor = "IIXDR,A,";  //NMEASensor = "IIXDR,A," + String(SensorID);
00083     //NMEASensorKraeng += ",";
00084     NMEASensor += String(fGaugeDrehzahl);
00085     NMEASensor += ",D,ROLL";
00086
00087   // Build CheckSum
00088   HexCheckSum = String(CheckSum(NMEASensor), HEX);
00089   // Build complete NMEA string
00090   SendSensor = "$" + NMEASensor;
00091   SendSensor += "*";
00092   SendSensor += HexCheckSum;
00093
00094   Serial.println(SendSensor);
00095
00096   return SendSensor;
00097 }
00098
00104
00105 String sendRPM()
00106 {
00107   String HexCheckSum;
00108   String NMEASensor;
00109   String SendSensor;
00110
00111     NMEASensor = "IIRPM,E,1,";  //NMEASensor = "IIXDR,E,1," + String(SensorID);
00112     NMEASensor += String(fGaugeDrehzahl);
00113     NMEASensor += ",15,A";
00114
00115   // Build CheckSum
00116   HexCheckSum = String(CheckSum(NMEASensor), HEX);
00117   // Build complete NMEA string
00118   SendSensor = "$" + NMEASensor;
00119   SendSensor += "*";
00120   SendSensor += HexCheckSum;
00121
00122   Serial.println(SendSensor);
00123
00124   return SendSensor;
00125 }
```

7.36 src/task.h-Dateireferenz

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



Makrodefinitionen

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

7.36.1 Makro-Dokumentation

7.36.1.1 taskBegin

```
#define taskBegin()
```

Wert:

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

Definiert in Zeile 6 der Datei task.h.

7.36.1.2 taskEnd

```
#define taskEnd()
```

Wert:

```
}
```

Definiert in Zeile 7 der Datei task.h.

7.36.1.3 taskSwitch

```
#define taskSwitch()
```

Wert:

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

Definiert in Zeile 11 der Datei task.h.

7.36.1.4 taskPause

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

Wert:

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

Definiert in Zeile 12 der Datei task.h.

7.36.1.5 taskWaitFor

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

Wert:

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

Definiert in Zeile 13 der Datei task.h.

7.36.1.6 taskStepName

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

Wert:

```
TASKSTEP_##STEPNAME  :
```

Definiert in Zeile 16 der Datei task.h.

7.36.1.7 taskJumpTo

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

Wert:

```
goto  TASKSTEP_##STEPNAME
```

Definiert in Zeile 17 der Datei task.h.

7.37 task.h

gehe zur Dokumentation dieser Datei

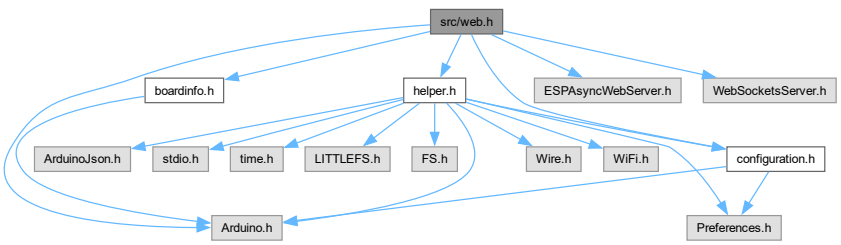
```
00001 #ifndef _TASK_H_
00002 #define _TASK_H_
00003
00004
00005 // grundlegene Worte um einen Task Bereich einzugrenzen
00006 #define taskBegin() static int mark = 0; static unsigned long __attribute__((unused)) timeStamp = 0;
      switch(mark){ case 0:
00007 #define taskEnd() }
00008
00009
00010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

7.38 src/web.h-Dateireferenz

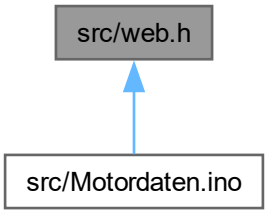
Webseite Variablen lesen und schreiben, Webseiten erstellen.

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

Include-Abhängigkeitsdiagramm für web.h:



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



Funktionen

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

Variablen

- WebSocketsServer webSocket = WebSocketsServer(81)
- String sBoardInfo
- BoardInfo boardInfo
- bool IsRebootRequired = false
- String sCL_Status = sWifiStatus(WiFi.status())

7.38.1 Ausführliche Beschreibung

Webseite Variablen lesen und schreiben, Webseiten erstellen.

Autor

Gerry Sebb

Version

0.1

Datum

2025-01-06

Copyright

Copyright (c) 2025

Definiert in Datei web.h.

7.38.2 Dokumentation der Funktionen

7.38.2.1 server()

```
AsyncWebServer server (
            80 )
```

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

7.38.2.2 processor()

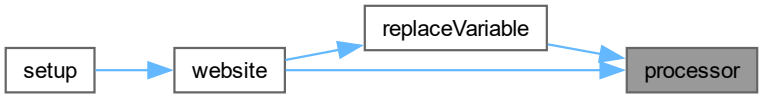
```
String processor (
            const String & var)
```

Definiert in Zeile 29 der Datei web.h.

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

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

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



7.38.2.3 replaceVariable()

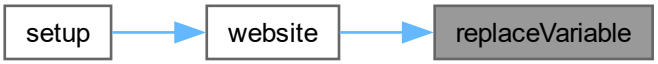
```
String replaceVariable (
            const String & var)
```

Definiert in Zeile 69 der Datei web.h.

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

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

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



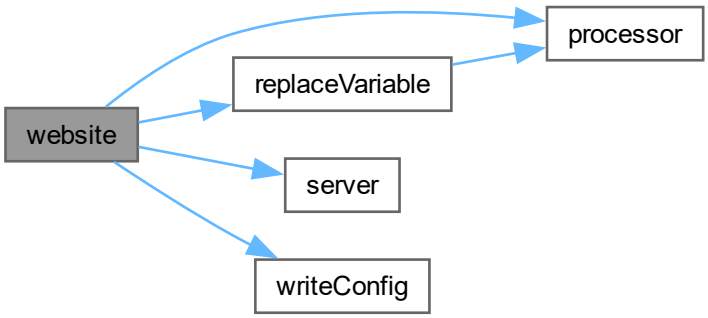
7.38.2.4 website()

```
void website ()
```

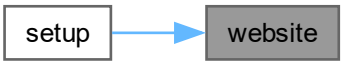
Definiert in Zeile 98 der Datei web.h.

```
00098               {
00099     server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request){
00100     request->send(LittleFS, "/favicon.ico", "image/x-icon");
00101     });
00102     server.on("/logo80.jpg", HTTP_GET, [](AsyncWebServerRequest *request){
00103     request->send(LittleFS, "/logo80.jpg", "image/jpg");
00104     });
00105     server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {
00106         request->send(LittleFS, "/index.html", String(), false, replaceVariable);
00107     });
00108     server.on("/system.html", HTTP_GET, [](AsyncWebServerRequest* request) {
00109         request->send(LittleFS, "/system.html", String(), false, replaceVariable);
00110     });
00111     server.on("/settings.html", HTTP_GET, [](AsyncWebServerRequest* request) {
00112         request->send(LittleFS, "/settings.html", String(), false, replaceVariable);
00113     });
00114     server.on("/werte.html", HTTP_GET, [](AsyncWebServerRequest* request) {
00115         request->send(LittleFS, "/werte.html", String(), false, replaceVariable);
00116     });
00117     server.on("/ueber.html", HTTP_GET, [](AsyncWebServerRequest* request) {
00118         request->send(LittleFS, "/ueber.html", String(), false, replaceVariable);
00119     });
00120     server.on("/reboot", HTTP_GET, [](AsyncWebServerRequest * request) {
00121         request->send(LittleFS, "/reboot.html", String(), false, processor);
00122         IsRebootRequired = true;
00123     });
00124     server.on("/gauge.min.js", HTTP_GET, [](AsyncWebServerRequest* request) {
00125         request->send(LittleFS, "/gauge.min.js");
00126     });
00127     server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
00128         request->send(LittleFS, "/style.css", "text/css");
00129     });
00130     server.on("/settings.html", HTTP_POST, [](AsyncWebServerRequest *request)
00131     {
00132         int count = request->params();
00133         Serial.printf("Anzahl: %i\n", count);
00134         for (int i = 0;i < count;i++)
00135         {
00136             AsyncWebParameter* p = request->getParam(i);
00137             Serial.print("PWerte von der Internet - Seite: ");
00138             Serial.print("Param name: ");
00139             Serial.println(p->name());
00140             Serial.print("Param value: ");
00141             Serial.println(p->value());
00142             Serial.println("------");
00143             // p->value in die config schreiben
00144             writeConfig(p->value());
00145         }
00146         request->send(200, "text/plain", "Daten gespeichert");
00147     });
00148
00149 }
```

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



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



7.38.3 Variablen-Dokumentation

7.38.3.1 webSocket

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

Definiert in Zeile 22 der Datei web.h.

7.38.3.2 sBoardInfo

```
String sBoardInfo
```

Definiert in Zeile 25 der Datei web.h.

7.38.3.3 boardInfo

```
BoardInfo boardInfo
```

Definiert in Zeile 26 der Datei web.h.

7.38.3.4 IsRebootRequired

```
bool IsRebootRequired = false
```

Definiert in Zeile 27 der Datei web.h.

7.38.3.5 sCL_Status

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

Definiert in Zeile 67 der Datei web.h.

7.39 web.h

gehe zur Dokumentation dieser Datei

```
00001
00012
00013 #include "helper.h"
00014 #include "configuration.h"
00015 #include "boardinfo.h"
00016 #include <ESPAsyncWebServer.h>
00017 #include <WebSocketsServer.h>
00018 #include <Arduino.h>
00019
00020 // Set web server port number to 80
00021 AsyncWebServer server(80);
00022 WebSocketsServer webSocket = WebSocketsServer(81);  // WebSocket server on port 81
00023
00024 // Info Board for HTML-Output
00025 String sBoardInfo;
00026 BoardInfo boardInfo;
00027 bool IsRebootRequired = false;
00028
00029 String processor(const String& var)
00030 {
00031     if (var == "CONFIGPLACEHOLDER")
00032     {
00033         String buttons = "";
00034         buttons += "<form onSubmit = \"event.preventDefault(); formToJson(this);\">";
00035         buttons += "<p class=\"CInput\"><label>SSID </label><input type = \"text\" name = \"SSID\"
      value=\"";
00036         buttons += tAP_Config.wAP_SSID;
00037         buttons += "\"/></p>";
00038         buttons += "<p class=\"CInput\"><label>IP </label><input type = \"text\" name = \"IP\"
      value=\"";
00039         buttons += tAP_Config.wAP_IP;
00040         buttons += "\"/></p>";
00041         buttons += "<p class=\"CInput\"><label>Password </label><input type = \"text\" name =
      \"Password\" value=\"";
00042         buttons += tAP_Config.wAP_Password;
00043         buttons += "\"/></p>";
00044         buttons += "<p class=\"CInput\"><label>Oil Offset </label><input type = \"text\" name =
      \"MotorOffset\" value=\"";
00045         buttons += tAP_Config.wMotor_Offset;
00046         buttons += "\"/> &deg;C</p>";
00047         buttons += "<p class=\"CInput\"><label>K&uuml;hlwasser Offset </label><input type = \"text\"
      name = \"CoolantOffset\" value=\"";
00048         buttons += tAP_Config.wCoolant_Offset;
00049         buttons += "\"/> &deg;C</p>";
00050         buttons += "<p class=\"CInput\"><label>max. F&uuml;llstand </label><input type = \"text\" name
      = \"Fuellstandmax\" value=\"";
00051         buttons += tAP_Config.wFuellstandmax;
00052         buttons += "\"/> l</p>";
00053         buttons += "<p class=\"CInput\"><label>ADC1 Kalibrierung </label><input type = \"text\" name =
      \"ADC1_Cal\" value=\"";
00054         buttons += tAP_Config.wADC1_Cal;
00055         buttons += "\"/></p>";
00056         buttons += "<p class=\"CInput\"><label>ADC2 Kalibrierung </label><input type = \"text\" name =
      \"ADC2_Cal\" value=\"";
00057         buttons += tAP_Config.wADC2_Cal;
00058         buttons += "\"/></p>";
00059         buttons += "<p class=\"button\"><input type=\"submit\" value=\"Speichern\"></p>";
```

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

```
00147     });
00148
00149 }
00150
```

Index