FFRESW

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 calcModule Namespace Reference

Namespace for the calculation module.

Classes

- class CalcModuleInternals

Enumerations

- enum Type { **General** , **Pressure** , **Position** }

  Enum for the different Types we want to extract from a response.
- enum class PressureUnit { **Pascal** , **Atmosphere** , **Psi** , **Bar** }

  Enum class for different Pressure Units aviable.

5.1.1 Detailed Description

Namespace for the calculation module.

5.2 comModule Namespace Reference

Namespace for the communication module.

Classes

- class ComModuleInternals
- class EthernetCommunication

  Class to handle Ethernet communication.
- class I2CCommunication

  Class to handle I2C communication.
- class SerialCommunication

  Class to handle Serial communication.
- class SPICommunication

  Class to handle SPI communication.

Enumerations

- enum class Service : uint8_t {
  SET = 0x01 , **GET** = 0x0B , **SET_COMPOUND** = 0x28 , **GET_COMPOUND** = 0x29 ,
  SETGET = 0x30 }

  Enum class for the service types.
- enum class Compound1 : uint32_t { **CONTROL_MODE** = 0x0F020000 , **TARGET_POSITION** = 0x11020000 , **TARGET_PRESSURE** = 0x07020000 , **NOT_USED** = 0x00000000 }

  Enum class for the compound 1 types.
- enum class Compound2 : uint32_t {
  ACCESS_MODE = 0x0F0B0000 , **CONTROL_MODE** = 0x0F020000 , **TARGET_POSITION** = 0x11020000 ,
  TARGET_PRESSURE = 0x07020000 ,
  ACTUAL_POSITION = 0x10010000 , **POSITION_STATE** = 0x00100000 , **ACTUAL_PRESSURE** = 0x07010000 , **TARGET_PRESSURE_USED** = 0x07030000 ,
  WARNING_BITMAP = 0x0F300100 , **NOT_USED** = 0x00000000 }

  Enum class for the compound 2 types.
- enum class Compound3 : uint32_t {
  CONTROL_MODE = 0x0F020000 , **TARGET_POSITION** = 0x11020000 , **TARGET_PRESSURE** = 0x07020000 , **SEPARATION** = 0x00000000 ,
  ACCESS_MODE = 0x0F0B0000 , **ACTUAL_POSITION** = 0x10010000 , **POSITION_STATE** = 0x00100000 ,
  ACTUAL_PRESSURE = 0x07010000 ,
  TARGET_PRESSURE_USED = 0x07030000 , **WARNING_BITMAP** = 0x0F300100 , **NOT_USED** = 0x00000000 }

  Enum class for the compound 3 types.
- enum class Error_Codes : uint8_t {
  NO_ERROR = 0x00 , **WRONG_COMMAND_LENGTH** = 0x0C , **VALUE_TOO_LOW** = 0x1C , **VALUE_↩ TOO_HIGH** = 0x1D ,
  RESULTING_ZERO_ADJUST_OFFSET = 0x20 , **NO_SENSOR_ENABLED** = 0x21 , **WRONG_ACCESS↩ _MODE** = 0x50 , **TIMEOUT** = 0x51 ,
  NV_MEMORY_NOT_READY = 0x6D , **WRONG_PARAMETER_ID** = 0x6E , **PARAMETER_NOT_↩ SETTABLE** = 0x70 , **PARAMETER_NOT_READABLE** = 0x71 ,
  WRONG_PARAMETER_INDEX = 0x73 , **WRONG_VALUE_WITHIN_RANGE** = 0x76 , **NOT_ALLOWED_↩ IN_THIS_STATE** = 0x78 , **SETTING_LOCK** = 0x79 ,
  WRONG_SERVICE = 0x7A , **PARAMETER_NOT_ACTIVE** = 0x7B , **PARAMETER_SYSTEM_ERROR** = 0x7C , **COMMUNICATION_ERROR** = 0x7D ,
  UNKNOWN_SERVICE = 0x7E , **UNEXPECTED_CHARACTER** = 0x7F , **NO_ACCESS_RIGHTS** = 0x80 , **NO_ADEQUATE_HARDWARE** = 0x81 ,
  WRONG_OBJECT_STATE = 0x82 , **NO_SLAVE_COMMAND** = 0x84 , **COMMAND_TO_UNKNOWN_↩ SLAVE** = 0x85 , **COMMAND_TO_MASTER_ONLY** = 0x87 ,
  ONLY_G_COMMAND_ALLOWED = 0x88 , **NOT_SUPPORTED** = 0x89 , **FUNCTION_DISABLED** = 0xA0 , **ALREADY_DONE** = 0xA1 }

  Enum class for the error codes.

5.2.1 Detailed Description

Namespace for the communication module.

5.3 flybackModule Namespace Reference

Namespace for the Flyback module.

Classes

- class Flyback

  Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.
- struct Measurement

  Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.

Typedefs

- typedef struct flybackModule::Measurement **meas**

Enumerations

- enum class MainSwitchStates : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }

  enum for different SwitchStates of HVModule
- enum class HVSwitchStates : int { **HV_Module_OFF** , **HV_Module_ON** }

  enum for SwitchStates of HV_Module on/off
- enum class **HVModule** : int { **powerSupply_OFF** , **powerSupply_ON** }

5.3.1 Detailed Description

Namespace for the Flyback module.

5.4 jsonModule Namespace Reference

Namespace for the JSON module.

Classes

- class JsonModuleInternals

  Class for the JSON module internals.

5.4.1 Detailed Description

Namespace for the JSON module.

5.5 locker Namespace Reference

Namespace for the locker system.

Classes

- class ScopedLock

  Scoped lock class for mutexes.

5.5.1 Detailed Description

Namespace for the locker system.

5.6 reportSystem Namespace Reference

Namespace for the report system.

Classes

- class ReportSystem

  Class for the report system.

5.6.1 Detailed Description

Namespace for the report system.

5.7 sensorModule Namespace Reference

Namespace for the sensor module.

Classes

- class SensorModuleInternals

  Class for the sensor module internals.

Enumerations

- enum class SensorType {
  TEMPERATURE , **OBJECTTEMPERATURE** , **AMBIENTTEMPERATURE** , **PRESSURE** ,
  DHT11 , **MCP9601_Celsius_Indoor** , **MCP9601_Fahrenheit_Indoor** , **MCP9601_Kelvin_Indoor** ,
  MCP9601_Celsius_Outdoor , **MCP9601_Fahrenheit_Outdoor** , **MCP9601_Kelvin_Outdoor** , **UNKNOWN**
  }

  Enum class for the sensor types.

5.7.1 Detailed Description

Namespace for the sensor module.

5.8 timeModule Namespace Reference

namespace for the timeModule

Classes

- struct DateTimeStruct

    Struct to hold the date and time.
- class TimeModuleInternals

    Class to handle Systemtime.

Typedefs

- typedef struct timeModule::DateTimeStruct **DateTimeStruct**

5.8.1 Detailed Description

namespace for the timeModule

5.9 vacControlModule Namespace Reference

Namespace for the VacControl module.

Classes

- struct Pressure
- class VacControl

    VacControl class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

Typedefs

- typedef struct vacControlModule::Pressure **meas**

Enumerations

- enum class MainSwitchStates : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }

    Enum to represent the states of the main switch and pump.
- enum class PumpSwitchStates : int { **Switch_Pump_OFF** , **Switch_Pump_ON** }

    Enum for extern Setup for pinMode for the PumpState.
- enum class PumpState : int { **pump_OFF** , **pump_ON** }

    Enum for extern Setup for pinMode for the PumpState.
- enum Scenarios {
  Scenario_1 = 0 , **Scenario_2** = 1 , **Scenario_3** = 2 , **Scenario_4** = 3 ,
  Scenario_5 = 4 , **Invalid_Scenario** = -1 }

    Enum to represent the different operating scenarios of the VacControl system.

5.9.1 Detailed Description

Namespace for the VacControl module.

Chapter 6

Class Documentation

6.1 calcModule::CalcModuleInternals Class Reference

Static Public Member Functions

- static float calculateAverage (const float ∗data, int length)

  Function to calculate the average of a data set.
- static float findMaximum (const float ∗data, int length)

  Function to calculate the maximum value of a data set.
- static float findMinimum (const float ∗data, int length)

  Function to calculate the minimum value of a data set.
- static float calculateStandardDeviation (const float ∗data, int length)

  Function to calculate the standard deviation of a data set.
- static float findMedian (float ∗data, int length)

  Function to calculate the median of a data set.
- static float celsiusToFahrenheit (float celsius)

  Function to convert celsius to fahrenheit.
- static float fahrenheitToCelsius (float fahrenheit)

  Function to convert fahrenheit to celsius.
- static float celsiusToKelvin (float celsius)

  Function to convert celsius to kelvin.
- static float kelvinToCelsius (float kelvin)

  Function to convert kelvin to celsius.
- static float pascalToAtm (float pascal)

  Function to convert pascal to atm.
- static float atmToPascal (float atm)

  Function to convert atm to pascal.
- static float pascalToPsi (float pascal)

  Function to convert pascal to psi.
- static float psiToPascal (float psi)

  Function to convert psi to pascal.
- static float calculatePower (float voltage, float current)

  Function to calculate the power.
- static float calculateCurrent (float voltage, float resistance)

  Funcion to calculate the current.
- static float calculateResistance (float voltage, float current)

Function to caculate the Resistance.
- static float extractFloat (String response, int id)

  Extract the float from a VAT uC eth frame.
- static float extractFloatFromResponse (const String &response, Type type)

  Extract the float from a VAT uC eth frame, specific for positions and pressures.
- static float **calculatePressureFromSensor** (int sensorValue, PressureUnit unit=PressureUnit::Pascal)

6.1.1 Member Function Documentation

6.1.1.1 atmToPascal()

```
float CalcModuleInternals::atmToPascal (
             float atm )  [static]
```

Function to convert atm to pascal.

Parameters

atm	-> The pressure in atm.

Returns

float -> The pressure in pascal.

6.1.1.2 calculateAverage()

```
float CalcModuleInternals::calculateAverage (
             const float * data,
             int length )  [static]
```

Function to calculate the average of a data set.

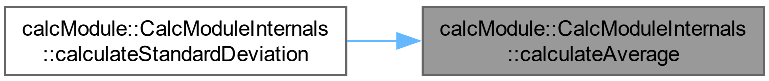
Parameters

data	-> The data set to calculate the average from.
length	-> The length of the data set.

Returns

float -> The average of the data set.

Here is the caller graph for this function:


calcModule::CalcModuleInternals ::calculateStandardDeviation
calcModule::CalcModuleInternals ::calculateAverage

6.1.1.3 calculateCurrent()

```
float CalcModuleInternals::calculateCurrent (
             float voltage,
             float resistance )  [static]
```

Funcion to calculate the current.

Parameters

voltage	-> The voltage.
resistance	-> The resistance.

Returns

float -> The calculated current.

6.1.1.4 calculatePower()

```
float CalcModuleInternals::calculatePower (
             float voltage,
             float current )  [static]
```

Function to calculate the power.

Parameters

voltage	-> The voltage.
current	-> The current.

Returns

float -> The calculated power.

6.1.1.5 calculateResistance()

```
float CalcModuleInternals::calculateResistance (
            float voltage,
            float current ) [static]
```

Function to caculate the Resistance.

Parameters

voltage	-> The voltage.
current	-> The current.

Returns

float -> The calculated resistance.

6.1.1.6 calculateStandardDeviation()

```
float CalcModuleInternals::calculateStandardDeviation (
            const float * data,
            int length ) [static]
```

Function to calculate the standard deviation of a data set.

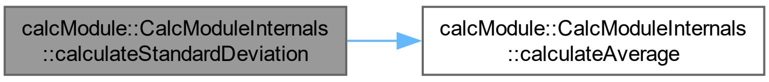
Parameters

data	-> The data set to calculate the standard deviation from.
length	-> The length of the data set.

Returns

float -> The standard deviation of the data set.

Here is the call graph for this function:


calcModule::CalcModuleInternals ::calculateStandardDeviation
calcModule::CalcModuleInternals ::calculateAverage

6.1.1.7 celsiusToFahrenheit()

```
float CalcModuleInternals::celsiusToFahrenheit (
            float celsius ) [static]
```

Function to convert celsius to fahrenheit.

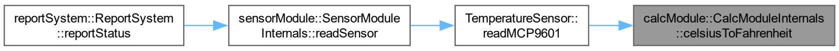
Parameters

celsius	-> The temperature in celsius.

Returns

float -> The temperature in fahrenheit.

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readMCP9601
calcModule::CalcModuleInternals ::celsiusToFahrenheit

6.1.1.8 celsiusToKelvin()

```
float CalcModuleInternals::celsiusToKelvin (
            float celsius )  [static]
```

Function to convert celsius to kelvin.

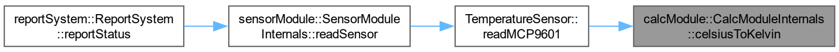
Parameters

celsius	-> The temperature in celsius.

Returns

float -> The temperature in kelvin.

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readMCP9601
calcModule::CalcModuleInternals ::celsiusToKelvin

6.1.1.9 extractFloat()

```
float CalcModuleInternals::extractFloat (
            String response,
            int id )  [static]
```

Extract the float from a VAT uC eth frame.

Parameters

response	-> The response from the VAT uC.
id	-> The id of the compound.

0 -> Simple GET/SET 1 -> Compound 1 1 -> Compound 2 1 -> Compound 3

Returns

float -> The extracted float.

6.1.1.10 extractFloatFromResponse()

```
float CalcModuleInternals::extractFloatFromResponse (
            const String & response,
            Type type ) [static]
```

Extract the float from a VAT uC eth frame, specific for positions and pressures.

Parameters

response	-> The response from the VAT uC.
type	-> The type to extract Pressure, Position, General

Returns

float -> The extracted float.

6.1.1.11 fahrenheitToCelsius()

```
float CalcModuleInternals::fahrenheitToCelsius (
            float fahrenheit ) [static]
```

Function to convert fahrenheit to celsius.

Parameters

fahrenheit	-> The temperature in fahrenheit.

Returns

float -> The temperature in celsius.

6.1.1.12 findMaximum()

```
float CalcModuleInternals::findMaximum (
            const float * data,
            int length ) [static]
```

Function to calculate the maximum value of a data set.

Parameters

data	-> The data set to calculate the maximum value from.
length	-> The length of the data set.

Returns

float -> The maximum value of the data set.

6.1.1.13 findMedian()

```
float CalcModuleInternals::findMedian (
            float * data,
            int length )  [static]
```

Function to calculate the median of a data set.

Parameters

data	-> The data set to calculate the median from.
length	-> The length of the data set.

Returns

float -> The median of the data set.

6.1.1.14 findMinimum()

```
float CalcModuleInternals::findMinimum (
            const float * data,
            int length )  [static]
```

Function to calculate the minimum value of a data set.

Parameters

data	-> The data set to calculate the minimum value from.
length	-> The length of the data set.

Returns

float -> The minimum value of the data set.

6.1.1.15 kelvinToCelsius()

```
float CalcModuleInternals::kelvinToCelsius (
            float kelvin )  [static]
```

Function to convert kelvin to celsius.

Parameters

kelvin	-> The temperature in kelvin.

Returns

float -> The temperature in celsius.

6.1.1.16 pascalToAtm()

```
float CalcModuleInternals::pascalToAtm (
            float pascal )  [static]
```

Function to convert pascal to atm.

Parameters

pascal	-> The pressure in pascal.

Returns

float -> The pressure in atm.

6.1.1.17 pascalToPsi()

```
float CalcModuleInternals::pascalToPsi (
            float pascal )  [static]
```

Function to convert pascal to psi.

Parameters

pascal	-> The pressure in pascal.

Returns

float -> The pressure in psi.

6.1.1.18 psiToPascal()

```
float CalcModuleInternals::psiToPascal (
            float psi )  [static]
```

Function to convert psi to pascal.

Parameters

psi	-> The pressure in psi.

Returns

float -> The pressure in pascal.

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp

6.2 calcModuleInternals Class Reference

Class for the calculation module internals.

```
#include <calcModule.h>
```

6.2.1 Detailed Description

Class for the calculation module internals.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h

6.3 comModule::ComModuleInternals Class Reference

Public Member Functions

- EthernetCommunication & getEthernet ()

  Get the Ethernet object.
- I2CCommunication & getI2C ()

  Get the I2C object.
- SPICommunication & getSPI ()

  Get the SPI object.
- SerialCommunication & getSerial ()

  Get the Serial object.

6.3.1 Member Function Documentation

6.3.1.1 getEthernet()

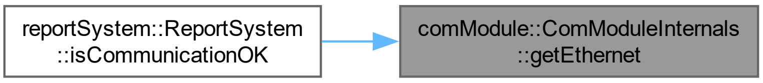
```
EthernetCommunication & ComModuleInternals::getEthernet ( )
```

Get the Ethernet object.

Returns

EthernetCommunication&

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::ComModuleInternals ::getEthernet

6.3.1.2 getI2C()

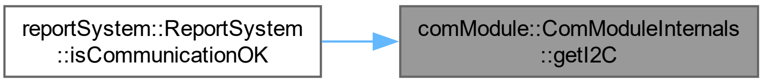
```
I2CCommunication & ComModuleInternals::getI2C ( )
```

Get the I2C object.

Returns

I2CCommunication&

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::ComModuleInternals ::getI2C

6.3.1.3 getSerial()

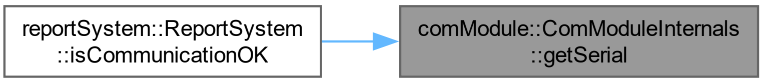
```
SerialCommunication & ComModuleInternals::getSerial ( )
```

Get the Serial object.

Returns

SerialCommunication&

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::ComModuleInternals ::getSerial

6.3.1.4 getSPI()

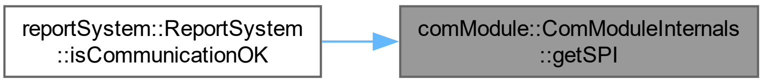
```
SPICommunication & ComModuleInternals::getSPI ( )
```

Get the SPI object.

Returns

SPICommunication&

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::ComModuleInternals ::getSPI

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp

6.4 comModuleInternals Class Reference

Class for the communication module internals.

```
#include <comModule.h>
```

6.4.1 Detailed Description

Class for the communication module internals.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h

6.5 timeModule::DateTimeStruct Struct Reference

Struct to hold the date and time.

```
#include <timeModule.h>
```

Public Attributes

- int **year**
- int **month**
- int **day**
- int **hour**
- int **minute**
- int **second**

6.5.1 Detailed Description

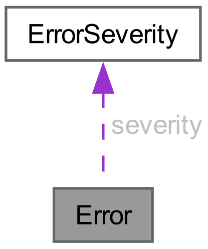
Struct to hold the date and time.

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h

6.6 Error Struct Reference

Collaboration diagram for Error:



Public Member Functions

- **Error** (int c, const char ∗m, ErrorSeverity s, const char ∗file, int line)
- void **addTrace** (const char ∗file, int line)
- bool **hasError** () const
- bool **isFatal** () const
- void **log** () const

Public Attributes

- int **code**
- ErrorSeverity **severity**
- const char ∗ **msg**
- const char ∗ **traceFiles** [TRACE_DEPTH]
- int **traceLines** [TRACE_DEPTH]
- int **traceDepth**

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.7 ErrorSeverity Struct Reference

Struct to defined the error settings.

```
#include <ptrUtils.h>
```

6.7.1 Detailed Description

Struct to defined the error settings.

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.8 comModule::EthernetCommunication Class Reference

Class to handle Ethernet communication.

```
#include <ETHH.h>
```

Public Member Functions

- void beginEthernet (uint8_t ∗macAddress, IPAddress ip)

  Function to initialize the Ethernet communication.
- void sendEthernetData (const char ∗endpoint, const char ∗data)

  Function to send data over Ethernet.
- void receiveEthernetData (char ∗buffer, size_t length)

  Function to receive data over Ethernet.
- void **handleEthernetClient** ()

  Function to handle the Ethernet client.
- String getRequestedEndpoint ()

  Function to get the requested endpoint.
- String getSpecificEndpoint (const String &jsonBody)

  Function to get the specific endpoint.
- void sendJsonResponse (const String &jsonBody)

  Function to send the json response with the measurment data.
- EthernetClient & getClient ()

  Get the currently active Ethernet client.
- bool isInitialized () const

  Function to check if the Ethernet communication is initialized.
- bool getSendDataFlag () const

  Function to get the current status of the flag.
- void setSendDataFlag (bool flag)

  Function to get the current status of the flag.
- void setCompound (Compound1 id, int index, String value)

  Function to set a compound command for the valve uC Slave (Compound1)
- void setCompound (Compound2 id, int index, String value)

  Function to set a compound command for the valve uC Slave (Compound2)
- void setCompound (Compound3 id, int index, String value)

  Function to set a compound command for the valve uC Slave (Compound3)
- void setCompoundInternal (String compoundType, unsigned long id, int index, String value)

  Function to set the Interal compound command for the valve uC Slave.
- String getCompound (Compound1 id, int index)

  Getter for a compound command response from the valve uC Slave (Compound1)
- String getCompound (Compound2 id, int index)

Getter for a compound command response from the valve uC Slave (Compound2)

- String getCompound (Compound3 id, int index)

  Getter for a compound command response from the valve uC Slave (Compound3)
- String getCompoundInternal (String compoundType, unsigned long id, int index)

  Getter for the internal compound command response from the valve uC Slave.
- Vector< float > getParsedCompound (Compound1 id, int index)

  Function to get a compound command response from the valve uC Slave (Compound1)
- Vector< float > getParsedCompound (Compound2 id, int index)

  Function to get a compound command response from the valve uC Slave (Compound2)
- Vector< float > getParsedCompound (Compound3 id, int index)

  Function to get a compound command response from the valve uC Slave (Compound3)
- Vector< float > parseCompoundResponse (String response)

  Function to parse a compound response into a vector (Compound1)
- void setParameter (Compound2 id, String value)

  Setter for a parameter from the VAT slave.
- String getParameter (Compound2 id)

  Getter for a parameter from the VAT slave.
- void sendCommand (String command)

  Helper function to send a command to the VAT slave controller.

6.8.1 Detailed Description

Class to handle Ethernet communication.

6.8.2 Member Function Documentation

6.8.2.1 beginEthernet()

```
void EthernetCommunication::beginEthernet (
            uint8_t * macAddress,
            IPAddress ip )
```

Function to initialize the Ethernet communication.

Parameters

macAddress	-> The MAC address to use for the Ethernet communication
ip	-> The IP address to use for the Ethernet communication

6.8.2.2 getClient()

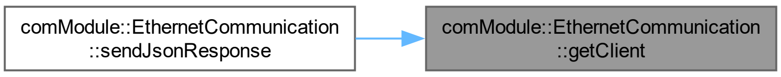
```
EthernetClient & EthernetCommunication::getClient ( )
```

Get the currently active Ethernet client.

Returns

EthernetClient& , Reference to the active Ethernet client

Here is the caller graph for this function:


comModule::EthernetCommunication ::sendJsonResponse
comModule::EthernetCommunication ::getClient

6.8.2.3 getCompound() `[1/3]`

```
String EthernetCommunication::getCompound (
              Compound1 id,
              int index )
```

Getter for a compound command response from the valve uC Slave (Compound1)

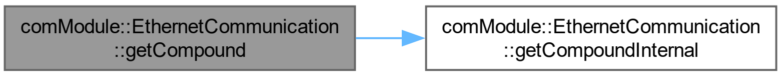
Parameters

id	-> Enum ID from Compound1
index	-> Index of the command

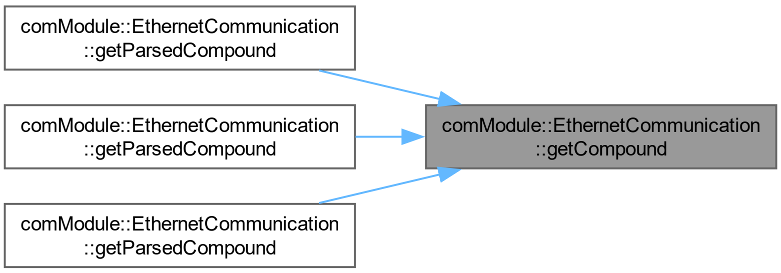
Returns

String -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal

Here is the caller graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getCompound

6.8.2.4 getCompound() [2/3]

```
String EthernetCommunication::getCompound (
             Compound2 id,
             int index )
```

Getter for a compound command response from the valve uC Slave (Compound2)

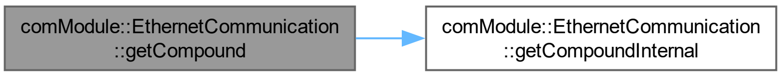
Parameters

id	-> Enum ID from Compound2
index	-> Index of the command

Returns

String -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal

6.8.2.5 getCompound() [3/3]

```
String EthernetCommunication::getCompound (
             Compound3 id,
             int index )
```

Getter for a compound command response from the valve uC Slave (Compound3)

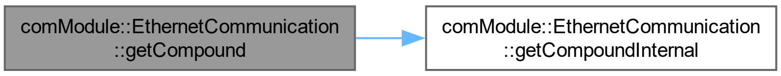
Parameters

id	-> Enum ID from Compound3
index	-> Index of the command

Returns

String -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal

6.8.2.6 getCompoundInternal()

```
String EthernetCommunication::getCompoundInternal (
             String compoundType,
             unsigned long id,
             int index )
```

Getter for the internal compound command response from the valve uC Slave.

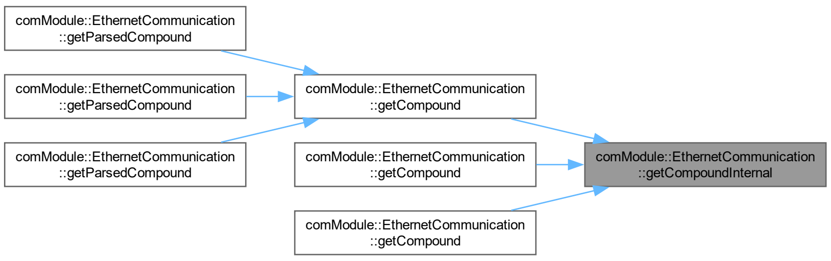
Parameters

compoundType	-> The type of the compound
id	-> The ID of the compound
index	-> The index of the compound

Returns

String -> Response from the valve uC slave

Here is the caller graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal

6.8.2.7 getParameter()

```
String EthernetCommunication::getParameter (
            Compound2 id )
```

Getter for a parameter from the VAT slave.

Parameters

id	-> The ID of the parameter to get

Returns

-> String will return the value of the parameter as a string, otherwise an empty string or error message.

6.8.2.8 getParsedCompound() [1/3]

```
Vector< float > EthernetCommunication::getParsedCompound (
            Compound1 id,
            int index )
```

Function to get a compound command response from the valve uC Slave (Compound1)

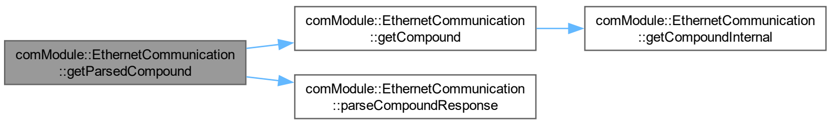
Parameters

id	-> Enum ID from Compound1
index	-> Index of the command

Returns

Vector<float> -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal
comModule::EthernetCommunication ::parseCompoundResponse

6.8.2.9 getParsedCompound() `[2/3]`

```
Vector< float > EthernetCommunication::getParsedCompound (
              Compound2 id,
              int index )
```

Function to get a compound command response from the valve uC Slave (Compound2)

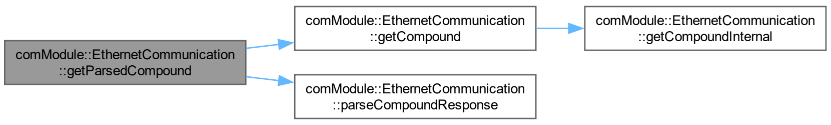
Parameters

id	-> Enum ID from Compound1
index	-> Index of the command

Returns

Vector<float> -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal
comModule::EthernetCommunication ::parseCompoundResponse

6.8.2.10 getParsedCompound() `[3/3]`

```
Vector< float > EthernetCommunication::getParsedCompound (
              Compound3 id,
              int index )
```

Function to get a compound command response from the valve uC Slave (Compound3)

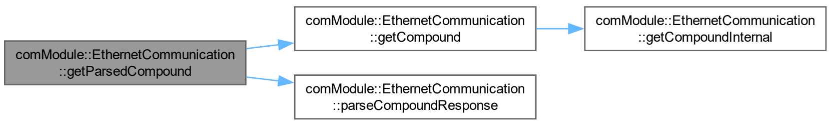
Parameters

id	-> Enum ID from Compound1
index	-> Index of the command

Returns

Vector<float> -> Response from the valve uC slave

Here is the call graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getCompound
comModule::EthernetCommunication ::getCompoundInternal
comModule::EthernetCommunication ::parseCompoundResponse

6.8.2.11 getRequestedEndpoint()

```
String EthernetCommunication::getRequestedEndpoint ( )
```

Function to get the requested endpoint.

Returns

String -> The requested endpoint

6.8.2.12 getSendDataFlag()

```
bool EthernetCommunication::getSendDataFlag ( ) const
```

Function to get the current status of the flag.

Returns

true -> if data should be sent

false -> if data should not be sent

6.8.2.13 getSpecificEndpoint()

```
String EthernetCommunication::getSpecificEndpoint (
             const String & jsonBody )
```

Function to get the specific endpoint.

Parameters

jsonBody	-> The json body to get the endpoint from

Returns

String -> The specific endpoint

6.8.2.14 isInitialized()

```
bool EthernetCommunication::isInitialized ( ) const
```

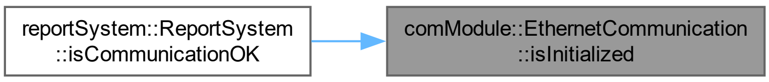
Function to check if the Ethernet communication is initialized.

Returns

true -> if the Ethernet communication is initialized

false -> if the Ethernet communication is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::EthernetCommunication ::isInitialized

6.8.2.15 parseCompoundResponse()

```
Vector< float > EthernetCommunication::parseCompoundResponse (
            String response )
```

Function to parse a compound response into a vector (Compound1)

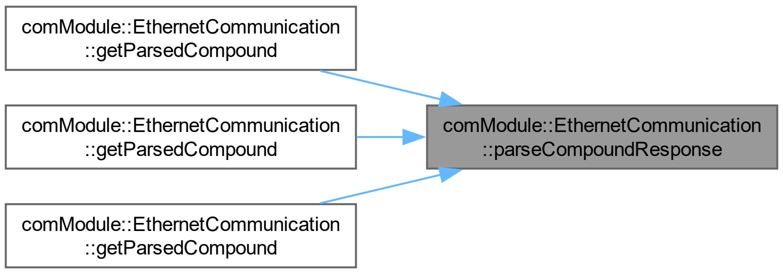
Parameters

response	-> Raw response string containing IEEE-754 hex values.

Returns

Vector<float> -> containing parsed float values.

Here is the caller graph for this function:


comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::getParsedCompound
comModule::EthernetCommunication ::parseCompoundResponse

6.8.2.16 receiveEthernetData()

```
void EthernetCommunication::receiveEthernetData (
             char * buffer,
             size_t length )
```

Function to receive data over Ethernet.

Parameters

buffer	-> The buffer to read the data into
length	-> The length of the data to read

6.8.2.17 sendCommand()

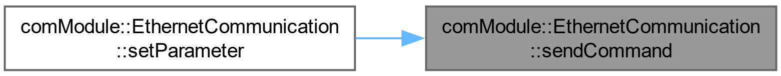
```
void EthernetCommunication::sendCommand (
             String command )
```

Helper function to send a command to the VAT slave controller.

Parameters

command	-> The command to send to the VAT slave controller

Here is the caller graph for this function:


comModule::EthernetCommunication ::setParameter
comModule::EthernetCommunication ::sendCommand

6.8.2.18 sendEthernetData()

```
void EthernetCommunication::sendEthernetData (
             const char * endpoint,
             const char * data )
```

Function to send data over Ethernet.

Parameters

endpoint	-> endpoint to send data to
data	-> The data to send

6.8.2.19 sendJsonResponse()

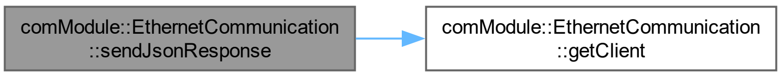
```
void EthernetCommunication::sendJsonResponse (
             const String & jsonBody )
```

Function to send the json response with the measurment data.

Parameters

jsonBody	-> jsonstring with the content needed

Here is the call graph for this function:


comModule::EthernetCommunication ::sendJsonResponse
comModule::EthernetCommunication ::getClient

6.8.2.20 setCompound() [1/3]

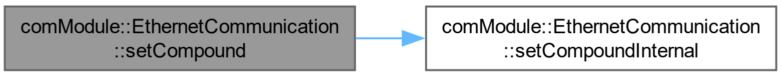
```
void EthernetCommunication::setCompound (
             Compound1 id,
             int index,
             String value )
```

Function to set a compound command for the valve uC Slave (Compound1)

Parameters

id	-> Enum ID from Compound1
index	-> Index of the command
value	-> Value of the command

Here is the call graph for this function:


comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompoundInternal

6.8.2.21 setCompound() [2/3]

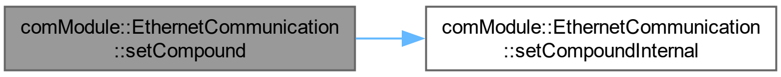
```
void EthernetCommunication::setCompound (
             Compound2 id,
             int index,
             String value )
```

Function to set a compound command for the valve uC Slave (Compound2)

Parameters

id	-> Enum ID from Compound2
index	-> Index of the command
value	-> Value of the command

Here is the call graph for this function:


comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompoundInternal

6.8.2.22 setCompound() [3/3]

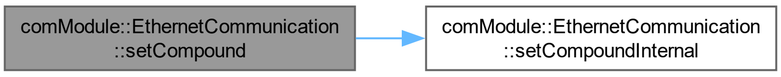
```
void EthernetCommunication::setCompound (
             Compound3 id,
             int index,
             String value )
```

Function to set a compound command for the valve uC Slave (Compound3)

Parameters

id	-> Enum ID from Compound3
index	-> Index of the command
value	-> Value of the command

Here is the call graph for this function:


comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompoundInternal

6.8.2.23 setCompoundInternal()

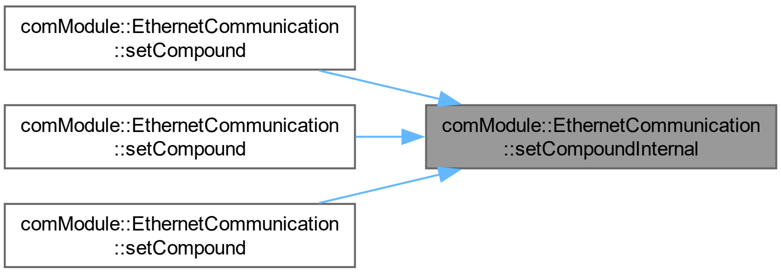
```
void EthernetCommunication::setCompoundInternal (
             String compoundType,
             unsigned long id,
             int index,
             String value )
```

Function to set the Interal compound command for the valve uC Slave.

Parameters

compoundType	-> The type of the compound
id	-> The ID of the compound
index	-> The index of the compound
value	-> The value of the compound

Here is the caller graph for this function:


comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompound
comModule::EthernetCommunication ::setCompoundInternal

6.8.2.24 setParameter()

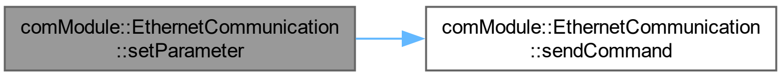
```
void EthernetCommunication::setParameter (
             Compound2 id,
             String value )
```

Setter for a parameter from the VAT slave.

Parameters

id	-> The ID of the parameter to set
value	-> The value to set the parameter to

Here is the call graph for this function:


comModule::EthernetCommunication ::setParameter
comModule::EthernetCommunication ::sendCommand

6.8.2.25 setSendDataFlag()

```
void EthernetCommunication::setSendDataFlag (
             bool flag )
```

Function to get the current status of the flag.

Parameters

flag	-> set the flag to true if data sent, false otherwise

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.cpp

6.9 flybackModule::Flyback Class Reference

Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

```
#include <flyback.h>
```

Public Member Functions

- void **initialize** ()

  Initialize the Flyback system This method sets up the pins and prepares the system for operation.
- void **deinitialize** ()

  denitialize the Flyback System This method shuts down the pins and prepares graceful restart.
- bool isInitialized () const

  Get the state of the Flyback system.
- bool getTimerState ()

  Returns the state of the timer.
- void setTimerState (bool state)

  Sets the state of the timer.
- MainSwitchStates getMainSwitchState ()

  Get the state of the Main-Switch.
- HVSwitchStates getHVSwitchState ()

  Get the state of the HV-Switch.
- HVModule getHVState ()

  Get the state of the HV-Output.
- Measurement measure ()

  Measures the voltage, current, power, digitalValue and frequency of the system.
- void **run** ()

  Executes logic depending on which Main-Switch state is active.
- void setExternFrequency (uint32_t frequency)

  Function to get the desired Frequency from HAS.
- uint32_t **getExternFrequency** ()

  Getter Function to get the Frequency.
- void setExternDutyCycle (int dutyCycle)

  Function to get the desired DutyCycle from HAS.
- int **getExternDutyCycle** ()

  Getter Function to get the DutyCycle $*$.
- void setExternPSU (int state)

  Function to get the desired PinMode from HAS.
- int getExternPSU ()

  Getter Function to get the PinMode.
- void regulateVoltage (float targetVoltage, float hysteresis)

  Function to regulate voltage to prevent swinging using a hyseresis.
- void setTargetVoltage (float voltage)

Setter Function for the HAS to regulate the target Voltage.
- float getTargetVoltage () const

  Getter Function for the HAS to know what the current targetVoltage is.
- void setHysteresis (float hysteresis)

  Setter Function for the HAS to regulate the hysteresis.
- float getHysteresis () const

  Getter Function for the HAS to know what the current hysteresisVoltage is.

6.9.1 Detailed Description

Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

6.9.2 Member Function Documentation

6.9.2.1 getExternPSU()

```
int Flyback::getExternPSU ( )
```

Getter Function to get the PinMode.

Returns

integer with state

6.9.2.2 getHVState()

```
HVModule Flyback::getHVState ( )
```

Get the state of the HV-Output.

Returns

Enum -> The current state of the HV-Output (e.g., "PowerSupply_OFF", "PowerSupply_ON")

6.9.2.3 getHVSwitchState()

```
HVSwitchStates Flyback::getHVSwitchState ( )
```

Get the state of the HV-Switch.

Returns

Enum -> The current state of the HV-Switch (e.g., "HV_Module OFF", "HV_Module ON")

6.9.2.4 getHysteresis()

```
float Flyback::getHysteresis ( ) const
```

Getter Function for the HAS to know what the current hysteresisVoltage is.

Returns

The Voltage the system is using to prevent swinging.

6.9.2.5 getMainSwitchState()

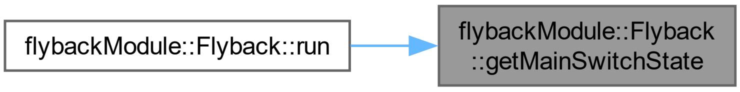
```
MainSwitchStates Flyback::getMainSwitchState ( )
```

Get the state of the Main-Switch.

Returns

Enum -> The current state of the Main-Switch (e.g., "HV_Module OFF", "HV_Module MANUAL", "HV_Module REMOTE", "Invalid Switch Position")

Here is the caller graph for this function:


flybackModule::Flyback::run
flybackModule::Flyback ::getMainSwitchState

6.9.2.6 getTargetVoltage()

```
float Flyback::getTargetVoltage ( ) const
```

Getter Function for the HAS to know what the current targetVoltage is.

Returns

The Voltage the system is trying to reach.

6.9.2.7 getTimerState()

```
bool Flyback::getTimerState ( )
```

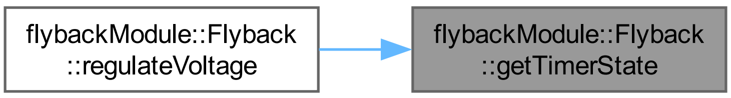
Returns the state of the timer.

Returns

true -> if the timer is initialized

false -> if the timer is not initialized

Here is the caller graph for this function:


flybackModule::Flyback ::regulateVoltage
flybackModule::Flyback ::getTimerState

6.9.2.8 isInitialized()

```
bool Flyback::isInitialized ( ) const
```

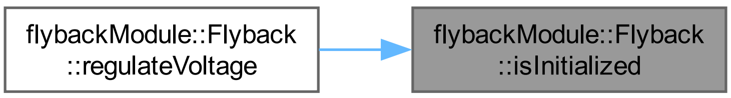
Get the state of the Flyback system.

Returns

true -> Flyback is initialized

false -> Flyback is not initialized

Here is the caller graph for this function:


flybackModule::Flyback ::regulateVoltage
flybackModule::Flyback ::isInitialized

6.9.2.9 measure()

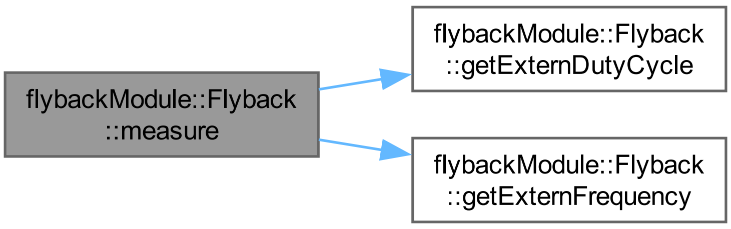
```
Measurement Flyback::measure ( )
```

Measures the voltage, current, power, digitalValue and frequency of the system.

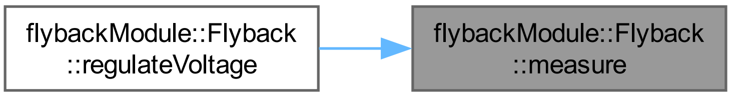
Returns

Measurement -> A Measurement object containing voltage, current, and power

Here is the call graph for this function:


flybackModule::Flyback ::measure
flybackModule::Flyback ::getExternDutyCycle
flybackModule::Flyback ::getExternFrequency

Here is the caller graph for this function:


flybackModule::Flyback ::regulateVoltage
flybackModule::Flyback ::measure

6.9.2.10 regulateVoltage()

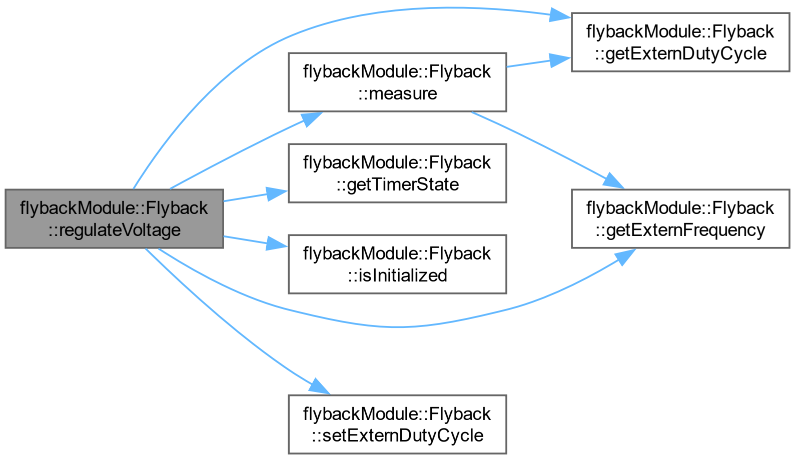
```
void Flyback::regulateVoltage (
            float targetVoltage,
            float hysteresis )
```

Function to regulate voltage to prevent swinging using a hyseresis.

Parameters

targetVoltage	-> The requested voltage.
hysteresis	-> The hysteresis we create to prevent swinging in the system.

Here is the call graph for this function:


flybackModule::Flyback ::regulateVoltage
flybackModule::Flyback ::measure
flybackModule::Flyback ::getTimerState
flybackModule::Flyback ::isInitialized
flybackModule::Flyback ::setExternDutyCycle
flybackModule::Flyback ::getExternDutyCycle
flybackModule::Flyback ::getExternFrequency

6.9.2.11 setExternDutyCycle()

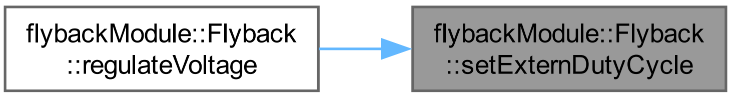
```
void Flyback::setExternDutyCycle (
             int dutyCycle )
```

Function to get the desired DutyCycle from HAS.

Parameters

dutyCycle	-> the dutyCycle to change

Here is the caller graph for this function:


flybackModule::Flyback ::regulateVoltage
flybackModule::Flyback ::setExternDutyCycle

6.9.2.12 setExternFrequency()

```
void Flyback::setExternFrequency (
             uint32_t frequency )
```

Function to get the desired Frequency from HAS.

Parameters

frequency	-> the frequency to change

6.9.2.13 setExternPSU()

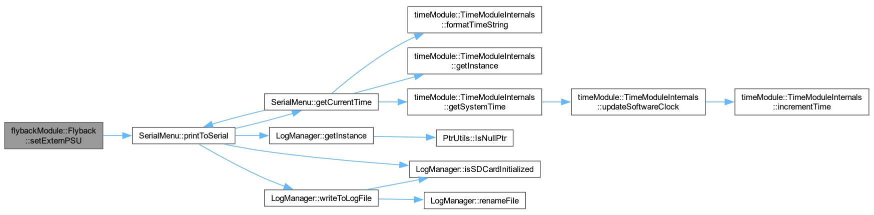
```
void Flyback::setExternPSU (
             int state )
```

Function to get the desired PinMode from HAS.

Parameters

state	-> the state to change the PinMode

Here is the call graph for this function:


flybackModule::Flyback ::setExternPSU
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.9.2.14 setHysteresis()

```
void Flyback::setHysteresis (
             float hysteresis )
```

Setter Function for the HAS to regulate the hysteresis.

Parameters

hysteresis	-> The desired hysteresis.

6.9.2.15 setTargetVoltage()

```
void Flyback::setTargetVoltage (
             float voltage )
```

Setter Function for the HAS to regulate the target Voltage.

Parameters

voltage	-> The desired Voltage to reach.

6.9.2.16 setTimerState()

```
void Flyback::setTimerState (
            bool state )
```

Sets the state of the timer.

Parameters

state	-> If true, the timer will be enabled, otherwise disabled

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.cpp

6.10 comModule::I2CCommunication Class Reference

Class to handle I2C communication.

```
#include <I2CC.h>
```

Public Member Functions

- void beginI2C (uint8_t address)

  Function to initialize the I2C communication.
- void **beginI2CGlobal** ()

  Function to initialize the I2C communication, without spefifing I2C address.
- void **endI2C** ()

  Function to end the I2C communication.
- void i2cWrite (uint8_t deviceAddress, uint8_t ∗data, size_t length)

  Function to write data to the I2C device.
- size_t i2cRead (uint8_t deviceAddress, uint8_t ∗buffer, size_t length)

  Function to read data from the I2C device.
- bool isInitialized () const

  Function to check if the I2C communication is initialized.

6.10.1 Detailed Description

Class to handle I2C communication.

6.10.2 Member Function Documentation

6.10.2.1 beginI2C()

```
void I2CCommunication::beginI2C (
             uint8_t address )
```

Function to initialize the I2C communication.

Parameters

address	-> The address of the I2C device

6.10.2.2 i2cRead()

```
size_t I2CCommunication::i2cRead (
            uint8_t deviceAddress,
            uint8_t * buffer,
            size_t length )
```

Function to read data from the I2C device.

Parameters

deviceAddress	-> The address of the I2C device
buffer	-> The buffer to read the data into
length	-> The length of the data to read

Returns

size_t -> The number of bytes read

6.10.2.3 i2cWrite()

```
void I2CCommunication::i2cWrite (
            uint8_t deviceAddress,
            uint8_t * data,
            size_t length )
```

Function to write data to the I2C device.

Parameters

deviceAddress	-> The address of the I2C device
data	-> The data to write
length	-> The length of the data

6.10.2.4 isInitialized()

```
bool I2CCommunication::isInitialized ( ) const
```

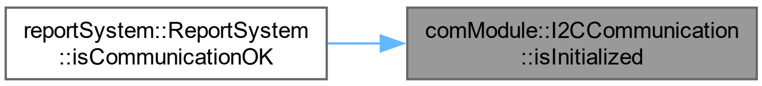
Function to check if the I2C communication is initialized.

Returns

true -> if the I2C communication is initialized

false -> if the I2C communication is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::I2CCommunication ::isInitialized

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC.cpp

6.11 jsonModule::JsonModuleInternals Class Reference

Class for the JSON module internals.

```
#include <jsonModule.h>
```

Public Member Functions

- template<typename T >
  void createJson (const char *key, T value)

  Add a key-value pair to the Json document.
- void **sendJsonSerial** ()

  Function to send the Json object over the Serial connection.
- String getJsonString () const

  Get the Json String object.
- void **clearJson** ()

  Clear the Json object.
- void **printJsonDocMemory** ()

  Prints information about the Json object.
- bool hasCapacityFor (size_t additionalSize) const

  Check if there is enough capacity left in the JsonDocument.

Public Attributes

- size_t **jsonBufferSize**

6.11.1 Detailed Description

Class for the JSON module internals.

6.11.2 Member Function Documentation

6.11.2.1 createJson()

```
template<typename T >
void jsonModule::JsonModuleInternals::createJson (
            const char * key,
            T value )  [inline]
```

Add a key-value pair to the Json document.

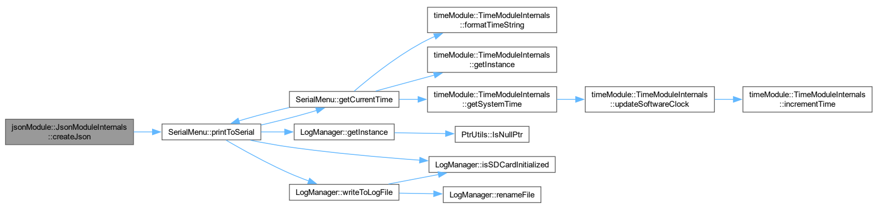
Template Parameters

T	Type of the value.

Parameters

key	Key to set.
value	Value to associate with the key.

Here is the call graph for this function:


timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
jsonModule::JsonModuleInternals ::createJson
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

6.11.2.2 getJsonString()

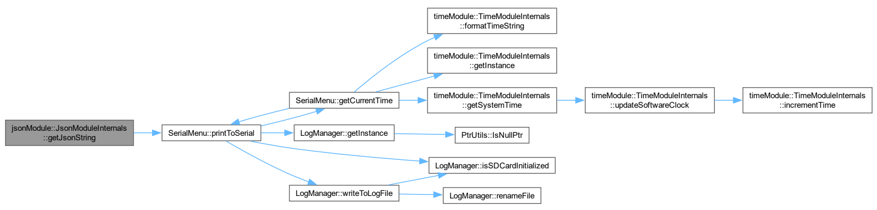
```
String JsonModuleInternals::getJsonString ( ) const
```

Get the Json String object.

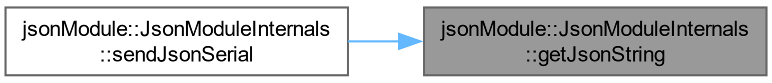
Returns

String -> The Json String object.

Here is the call graph for this function:


jsonModule::JsonModuleInternals ::getJsonString
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile

Here is the caller graph for this function:


jsonModule::JsonModuleInternals ::sendJsonSerial
jsonModule::JsonModuleInternals ::getJsonString

6.11.2.3 hasCapacityFor()

```
bool JsonModuleInternals::hasCapacityFor (
            size_t additionalSize ) const
```

Check if there is enough capacity left in the JsonDocument.

Parameters

additionalSize	Approximate size of the data to be added.

Returns

true If there is enough space.

false If adding may overflow the buffer.

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp

6.12 LockerBase Class Reference

Base class for the locker system.

```
#include <lockerBase.h>
```

Public Member Functions

- locker::ScopedLock lockEthernetScoped ()
- locker::ScopedLock lockTemperatureScoped ()
- locker::ScopedLock lockSerialScoped ()

6.12.1 Detailed Description

Base class for the locker system.

6.12.2 Member Function Documentation

6.12.2.1 lockEthernetScoped()

```
locker::ScopedLock LockerBase::lockEthernetScoped ( )  [inline]
```

Returns

locker::ScopedLock

6.12.2.2 lockSerialScoped()

```
locker::ScopedLock LockerBase::lockSerialScoped ( )  [inline]
```

Returns

locker::ScopedLock

6.12.2.3 lockTemperatureScoped()

```
locker::ScopedLock LockerBase::lockTemperatureScoped ( )  [inline]
```

Returns

locker::ScopedLock

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h

6.13 LogManager Class Reference

Public Member Functions

- void initSDCard (int cs)

  Function to initialize the SD card.
- void **shutdownSDCard** ()

  Function to shut down the SD card.
- void **flushLogs** ()

  Function to flush the current Logs in special cases.
- bool isSDCardInitialized () const

  Function to check if the SD card is initialized.
- void setLogFileName (const String &fileName)

  Set the Log File Name object.
- bool writeToLogFile (const String &logMessage)

  Function to write a log message to the log file.
- void renameFile (const String &oldName, const String &newName)

  Function to rename the currently written to file.

Static Public Member Functions

- static LogManager ∗ getInstance ()

  Get the Instance object.
- static String getCurrentTime ()

  Getter for the current time.

6.13.1 Member Function Documentation

6.13.1.1 getCurrentTime()

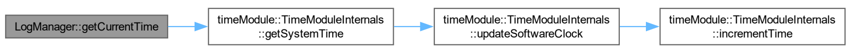
```
String LogManager::getCurrentTime ( )  [static]
```

Getter for the current time.

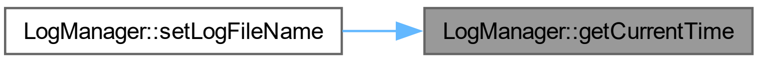
Returns

The current time as a String

Here is the call graph for this function:


LogManager::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

Here is the caller graph for this function:



6.13.1.2 getInstance()

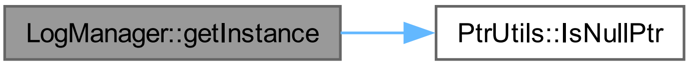
```
LogManager * LogManager::getInstance ( )  [static]
```

Get the Instance object.

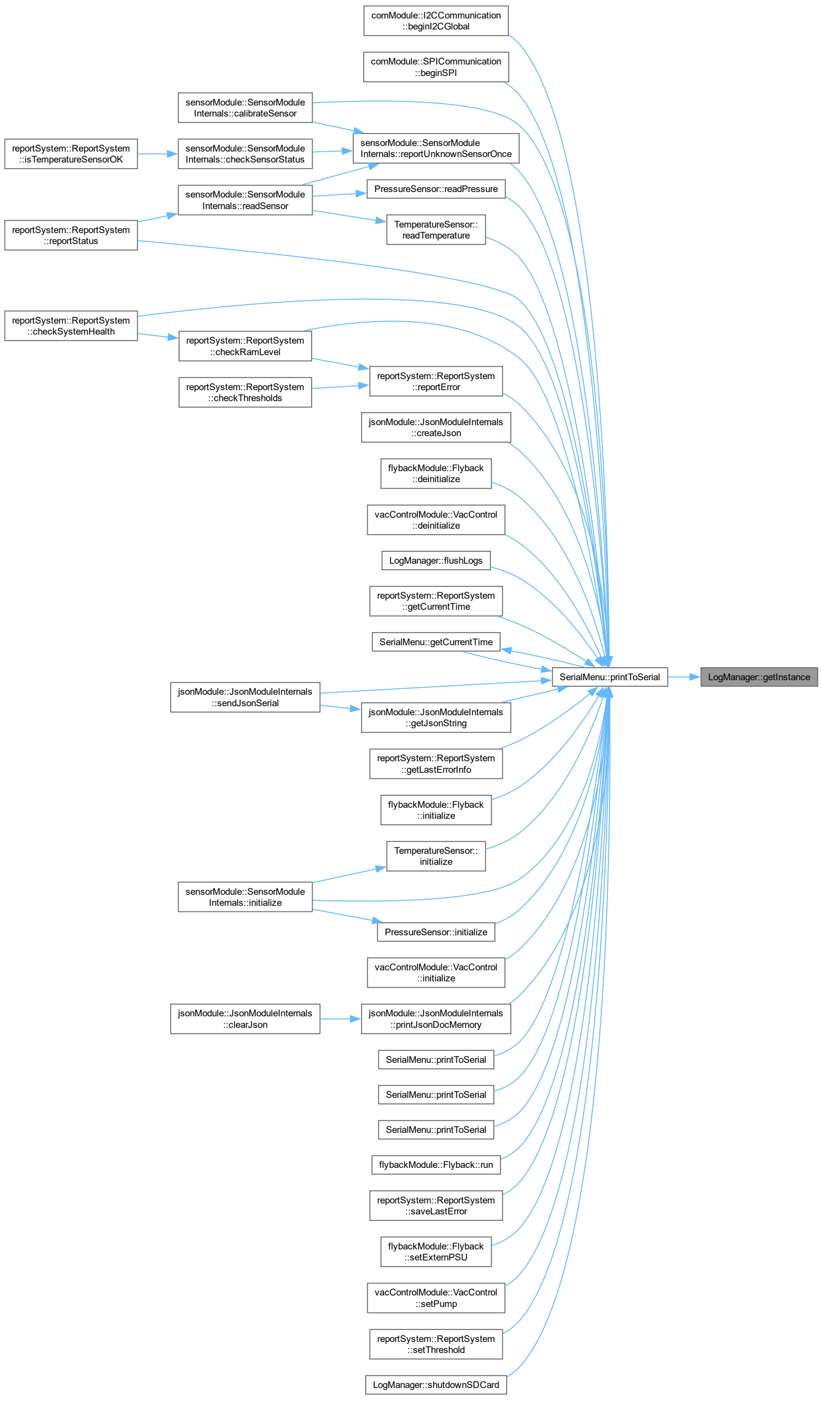
Returns

LogManager∗

Here is the call graph for this function:



Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
SerialMenu::printToSerial
LogManager::getInstance
jsonModule::JsonModuleInternals ::sendJsonSerial
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

6.13.1.3 initSDCard()

```
void LogManager::initSDCard (
                int cs )
```

Function to initialize the SD card.

Parameters

cs	-> The chip select pin for the SD card.

6.13.1.4 isSDCardInitialized()

`bool LogManager::isSDCardInitialized ( ) const`

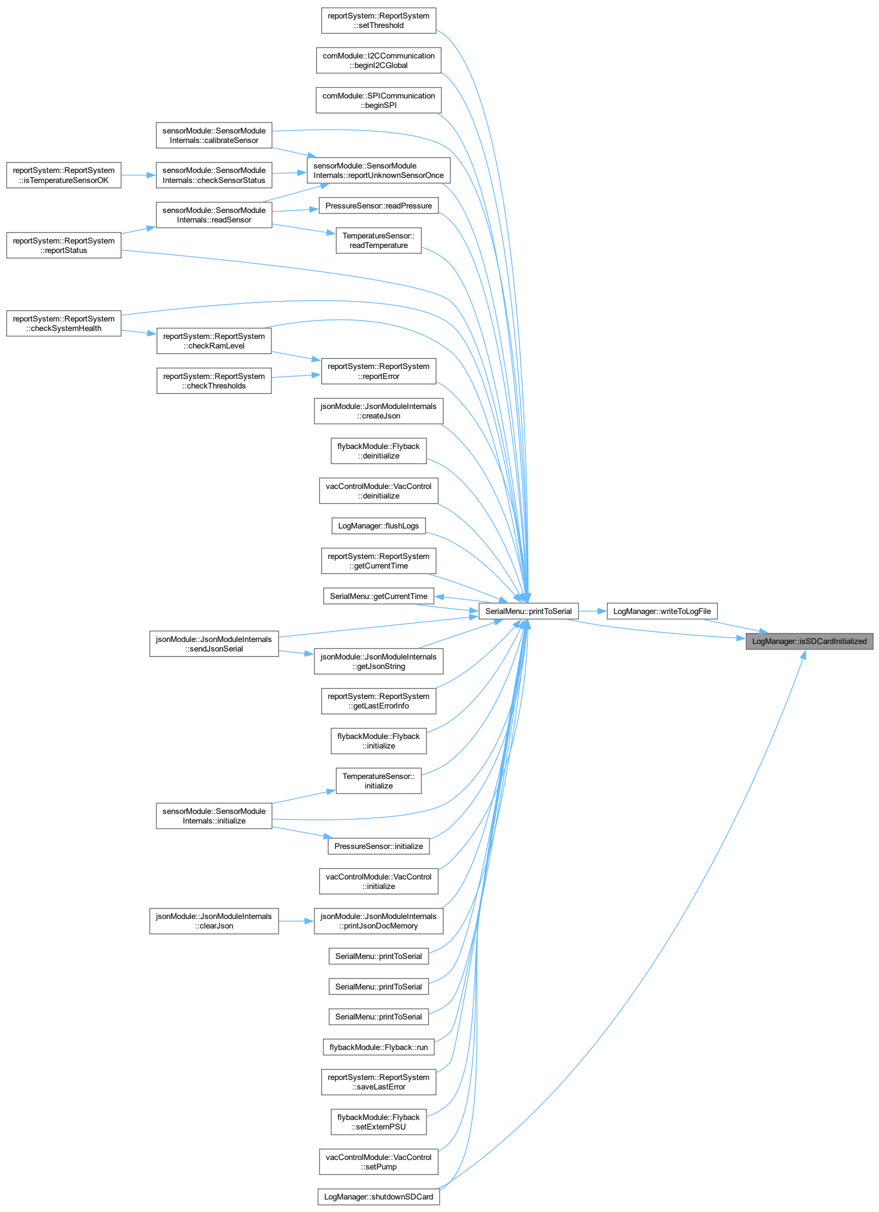
Function to check if the SD card is initialized.

Returns

true -> if the SD card is initialized

false -> if the SD card is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::setThreshold
comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
SerialMenu::printToSerial
LogManager::writeToLogFile
LogManager::isSDCardInitialized
jsonModule::JsonModuleInternals ::sendJsonSerial
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
LogManager::shutdownSDCard

6.13.1.5 renameFile()

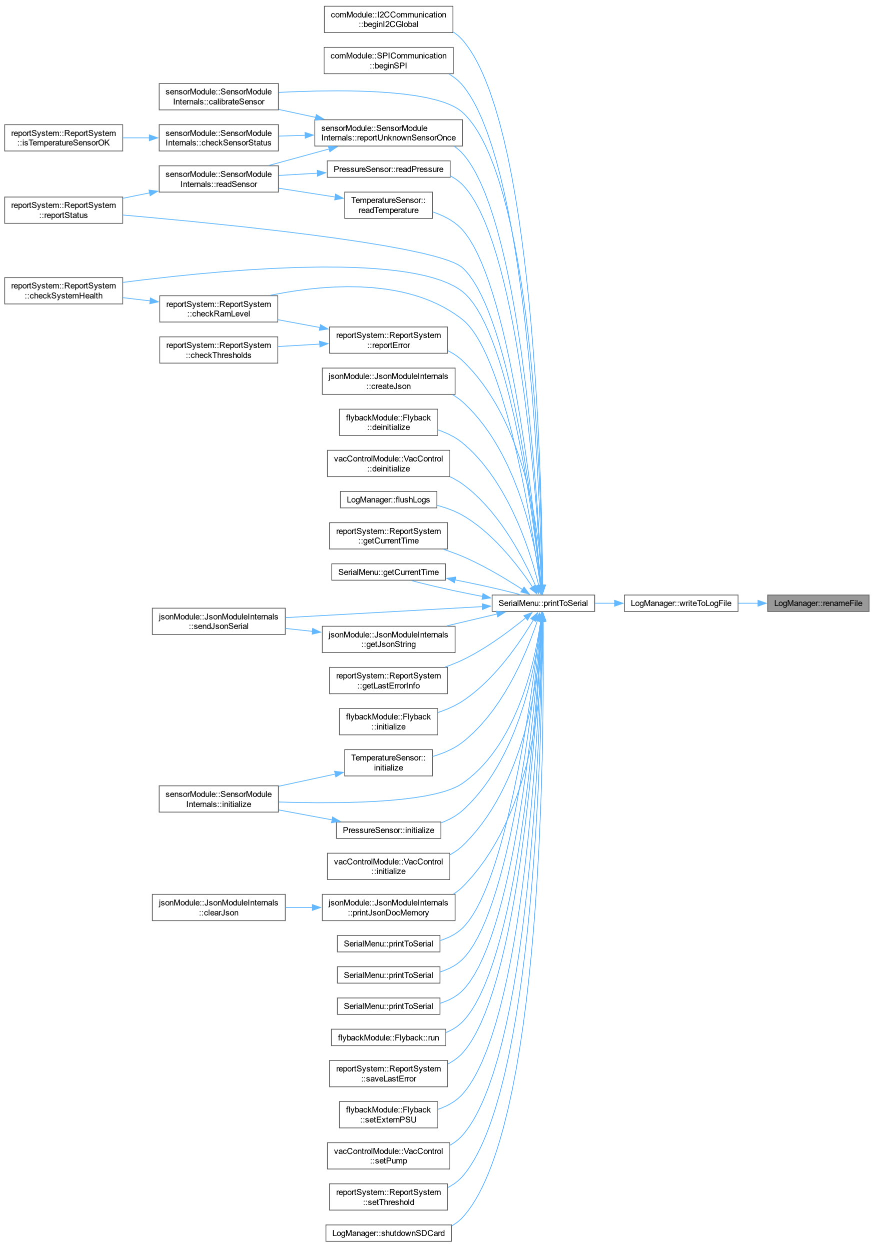
```
void LogManager::renameFile (
            const String & oldName,
            const String & newName )
```

Function to rename the currently written to file.

Parameters

oldName	-> This is the oldName of the file
newName	-> This is the newName of the file

Here is the caller graph for this function:



comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
jsonModule::JsonModuleInternals ::sendJsonSerial
SerialMenu::printToSerial
LogManager::writeToLogFile
LogManager::renameFile
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

6.13.1.6 setLogFileName()

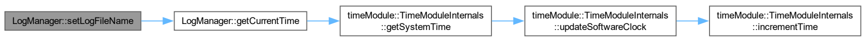
```
void LogManager::setLogFileName (
             const String & fileName )
```

Set the Log File Name object.

Parameters

fileName	-> The file name to set the log file name to.

Here is the call graph for this function:



LogManager::setLogFileName
LogManager::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.13.1.7 writeToLogFile()

```
bool LogManager::writeToLogFile (
             const String & logMessage )
```

Function to write a log message to the log file.

Parameters

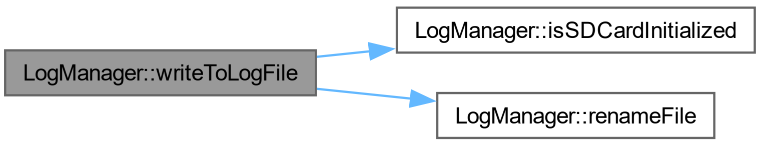
logMessage	-> The log message to write to the log file.

Returns

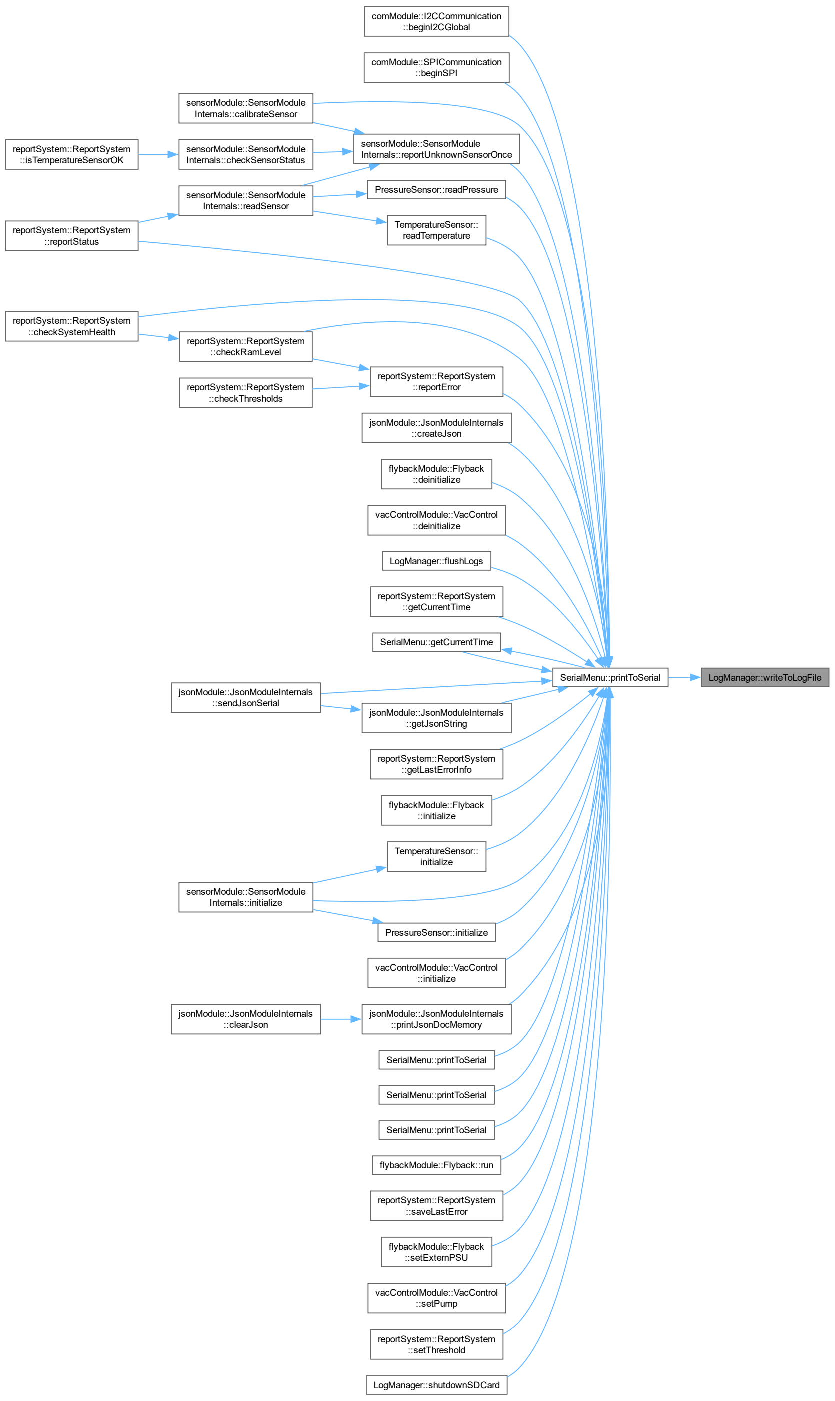
true -> if the log message was written successfully

false -> if the log message was not written successfully

Here is the call graph for this function:



Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
jsonModule::JsonModuleInternals ::sendJsonSerial
SerialMenu::printToSerial
LogManager::writeToLogFile
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.cpp

6.14 LogMapper Class Reference

Class which handle the printed log messages, maps aka parses them and saves them to the SD card.

```
#include <logManager.h>
```

6.14.1 Detailed Description

Class which handle the printed log messages, maps aka parses them and saves them to the SD card.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h

6.15 flybackModule::Measurement Struct Reference

Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.

```
#include <flyback.h>
```

Public Attributes

- float **voltage**
- float **current**
- float **power**
- int **digitalFreqValue**
- int **digitalDutyValue**
- int **dutyCycle**
- uint32_t **frequency**

6.15.1 Detailed Description

Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h

6.16 Measurement Struct Reference

Structure to store the measured values of the system This structure holds the pressure values measured from the system.

```
#include <vacControl.h>
```

6.16.1 Detailed Description

Structure to store the measured values of the system This structure holds the pressure values measured from the system.

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h

6.17 MenuItem Struct Reference

Serial menu structure.

```
#include <serialMenu.h>
```

Public Attributes

- const char ∗ **label**
- char **key**
- void(∗ **callback**)()

6.17.1 Detailed Description

Serial menu structure.

The documentation for this struct was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h

6.18 Outputlevel Class Reference

Enum Class for the differnet Outputlevels.

```
#include <serialMenu.h>
```

6.18.1 Detailed Description

Enum Class for the differnet Outputlevels.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h

6.19 PointerWrapper< T > Class Template Reference

Tempalte class for wrapping a pointer.

```
#include <ptrUtils.h>
```

Public Member Functions

- **PointerWrapper** (T ∗p=nullptr)
- T ∗ get () const

  Function to get the pointer.
- T ∗ release ()

  Function to release the pointer.
- void reset (T ∗p=nullptr)

  Function to reset the pointer.
- T & operator∗ ()

  Operator to dereference the pointer.
- T ∗ operator-> ()

  Operator to access the pointer.

6.19.1 Detailed Description

template<typename T>
class PointerWrapper< T >

Tempalte class for wrapping a pointer.

Template Parameters

T	

6.19.2 Member Function Documentation

6.19.2.1 get()

```
template<typename T >
T ∗ PointerWrapper< T >::get ( ) const  [inline]
```

Function to get the pointer.

Returns

T∗ -> The pointer.

6.19.2.2 operator∗()

```
template<typename T >
T & PointerWrapper< T >::operator∗ ( )  [inline]
```

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.19.2.3 operator->()

```
template<typename T >
T ∗ PointerWrapper< T >::operator-> ( )  [inline]
```

Operator to access the pointer.

Returns

T∗ -> The pointer.

6.19.2.4 release()

```
template<typename T >
T ∗ PointerWrapper< T >::release ( )  [inline]
```

Function to release the pointer.

Returns

T∗ -> The released pointer.

6.19.2.5 reset()

```
template<typename T >
void PointerWrapper< T >::reset (
             T ∗ p = nullptr )  [inline]
```

Function to reset the pointer.

Parameters

p	-> The pointer to reset to.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.20 vacControlModule::Pressure Struct Reference

Public Attributes

- float **pressure**

The documentation for this struct was generated from the following file:

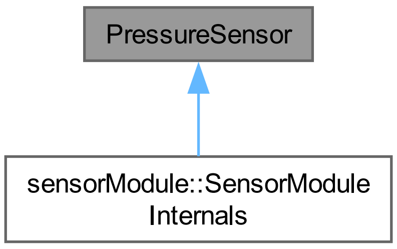
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h

6.21 PressureSensor Class Reference

Pressure sensor class.

```
#include <pressure.h>
```

Inheritance diagram for PressureSensor:



Public Member Functions

- void **initialize** ()

    Function to initialize the pressure sensor.
- float readPressure ()

    Function to read the pressure from the sensor.
- bool isInitialized () const

    Function to check if the pressure sensor is initialized.

6.21.1 Detailed Description

Pressure sensor class.

6.21.2 Member Function Documentation

6.21.2.1 isInitialized()

```
bool PressureSensor::isInitialized ( ) const
```

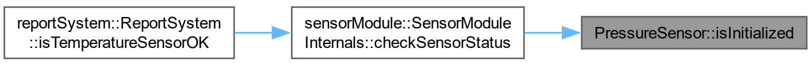
Function to check if the pressure sensor is initialized.

Returns

true -> if the pressure sensor is initialized

false -> if the pressure sensor is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
PressureSensor::isInitialized

6.21.2.2 readPressure()

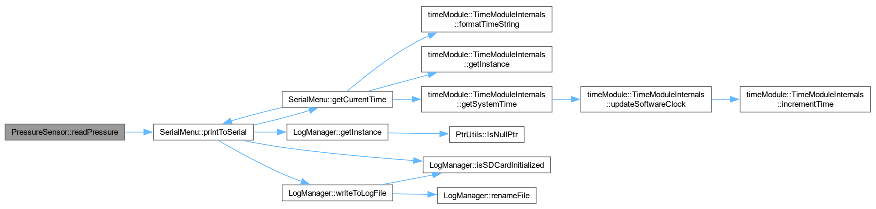
```
float PressureSensor::readPressure ( )
```

Function to read the pressure from the sensor.

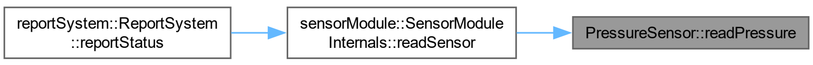
Returns

float -> The pressure value.

Here is the call graph for this function:


timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PressureSensor::readPressure
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp

6.22 PtrUtils Class Reference

Utility class for pointer operations.

```
#include <ptrUtils.h>
```

Static Public Member Functions

- template<typename T >
  static bool IsNullPtr (T *ptr)
- template<typename T >
  static bool IsValidPtr (T *ptr)

6.22.1 Detailed Description

Utility class for pointer operations.

6.22.2 Member Function Documentation

6.22.2.1 IsNullPtr()

```
template<typename T >
static bool PtrUtils::IsNullPtr (
            T * ptr )  [inline], [static]
```

Check if a pointer is nullptr.

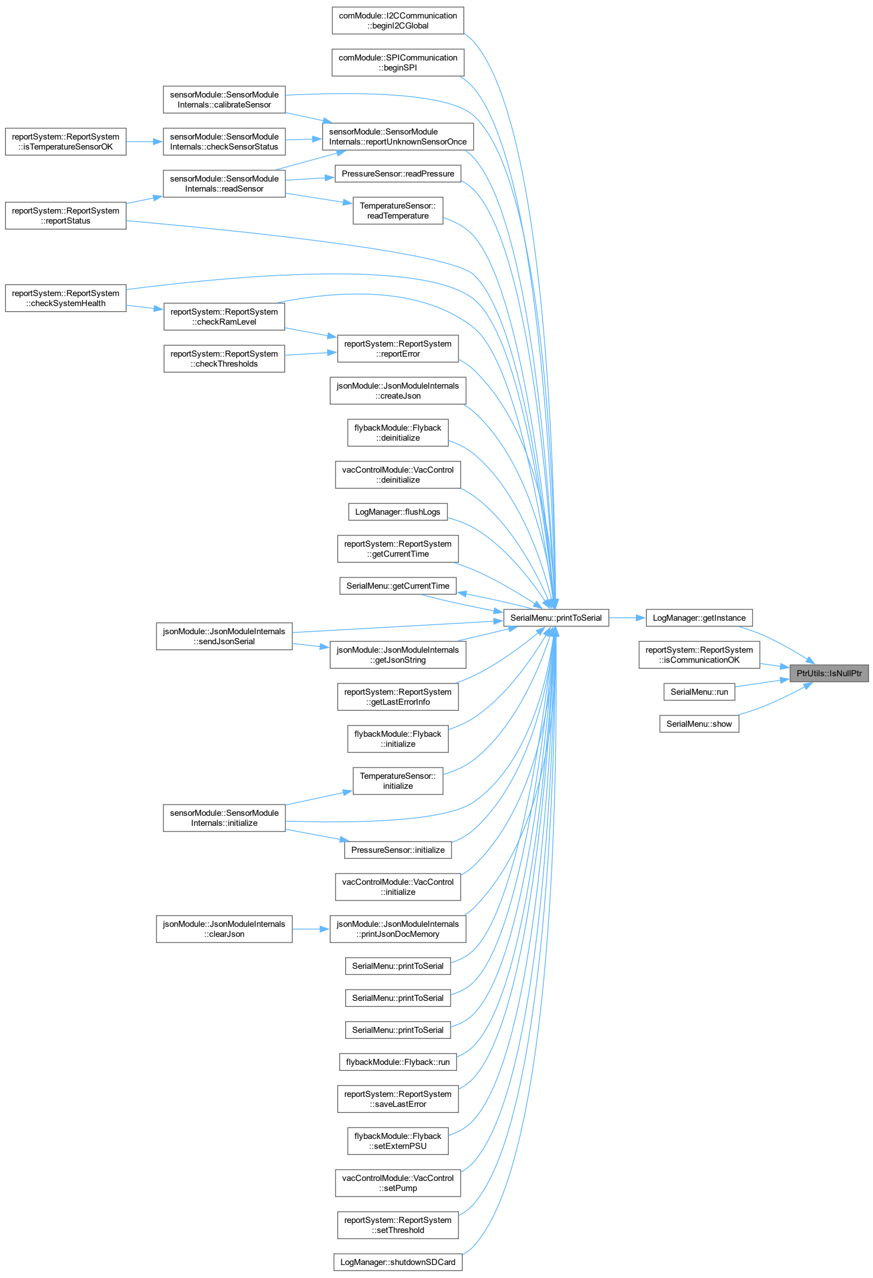
Parameters

ptr	Pointer to check.

Returns

true if the pointer is nullptr, false otherwise.

Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
SerialMenu::printToSerial
LogManager::getInstance
jsonModule::JsonModuleInternals ::sendJsonSerial
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::isCommunicationOK
PtrUtils::IsNullPtr
SerialMenu::run
SerialMenu::show
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

6.22.2.2 IsValidPtr()

```
template<typename T >
static bool PtrUtils::IsValidPtr (
            T * ptr )  [inline], [static]
```

Check if a pointer is valid (not nullptr).

Parameters

ptr	Pointer to check.

Returns

true if the pointer is not nullptr, false otherwise.

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.23 reportSystem::ReportSystem Class Reference

Class for the report system.

```
#include <reportSystem.h>
```

Public Member Functions

- void reportError (const char ∗errorMessage)
  Function to log an error message.
- bool checkSystemHealth (size_t memoryThreshold, bool checkEth, bool checkSpi, bool checkI2c, bool checkTemp, bool checkPress)
  Function to check the system health of the uC.
- bool reportStatus (bool active)
  Function to report the status of the system.
- void setThreshold (float tempThreshold, float pressureThreshold)
  Set Thresholds for the pressure and temperature sensors.
- bool checkThresholds (float currentTemp, float currentPressure)
  Check the thresholds for the temperature and pressure sensors.
- String getCurrentTime ()
  Get the Current Time of the system.
- String getMemoryStatus ()
  Get the Memory Status of the system.
- String getStackDump ()
  Get the Stack Dump of the system.
- void **startBusyTime** ()
  For Stack Guarding.
- void **startIdleTime** ()
  For Stack Guarding.
- float getCPULoad ()
  Getter for the CPU Load.
- void **resetUsage** ()
  Start the CPU Load Calculation.
- void saveLastError (const char ∗error)
  Saves last error message to EEPROM.
- String getLastError ()

Get the Last Error message from EEPROM.

- bool getLastErrorInfo ()

  Get the Last Error message from EEPROM.
- bool checkRamLevel (unsigned int warningThreshold, unsigned int criticalThreshold)

  Function to check the SRAM level on the hostsystem.
- bool isTemperatureSensorOK () const

  Function to report to the HAS via Endpoint if TempSensor is initialized and ok.
- bool isCommunicationOK () const

  Function to report to the HAS via Endpoint if Communication is initialized and ok.
- bool isMemoryOK () const

  Function to report to the HAS via Endpoint if Memory is in between the required bounds.
- bool isRamOK () const

  Function to report to the HAS via Endpoint if RAM is in between the required bounds.
- bool isStackSafe () const

  Function to report to the HAS via Endpoint if Stack is not overflowing or has other issues.
- bool hasNoSavedErrors () const

  Function to report to the HAS via Endpoint if any errors were written to the EEPROM.

Static Public Member Functions

- static void **initStackGuard** ()

  Initialize the Stack Guard.
- static bool detectStackOverflow ()

  Detect Stack Overflow.

6.23.1 Detailed Description

Class for the report system.

6.23.2 Member Function Documentation

6.23.2.1 checkRamLevel()

```
bool ReportSystem::checkRamLevel (
            unsigned int warningThreshold,
            unsigned int criticalThreshold )
```

Function to check the SRAM level on the hostsystem.

Parameters

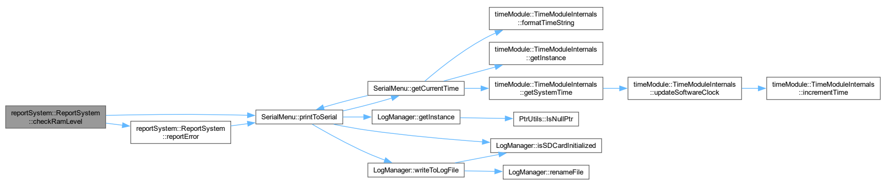
warningThreshold	-> first warning to get
criticalThreshold	-> last warning to get

Returns

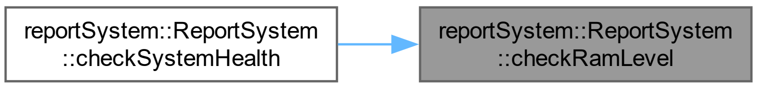
true -> if the level exceeded

false -> if the levels are withing the thresholds

Here is the call graph for this function:


reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::reportError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile

Here is the caller graph for this function:


reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel

6.23.2.2 checkSystemHealth()

```
bool ReportSystem::checkSystemHealth (
             size_t memoryThreshold,
             bool checkEth,
             bool checkSpi,
             bool checkI2c,
             bool checkTemp,
             bool checkPress )
```

Function to check the system health of the uC.

Parameters

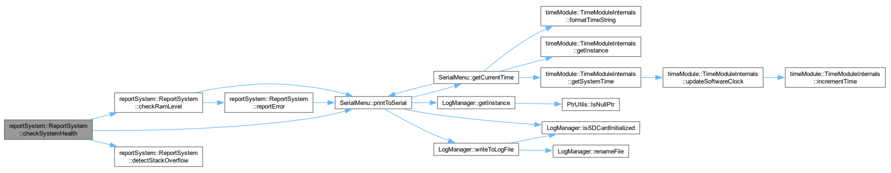
memoryThreshold	-> The memory threshold to check
checkEth	-> Check the Ethernet connection
checkSpi	-> Check the SPI connection
checkI2c	-> Check the I2C connection
checkTemp	-> Check the temperature sensor
checkPress	-> Check the pressure sensor

Returns

true -> if the system is healthy

false -> if the system is not healthy

Here is the call graph for this function:


reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::detectStackOverflow
reportSystem::ReportSystem ::reportError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.23.2.3 checkThresholds()

```
bool ReportSystem::checkThresholds (
             float currentTemp,
             float currentPressure )
```

Check the thresholds for the temperature and pressure sensors.

Parameters

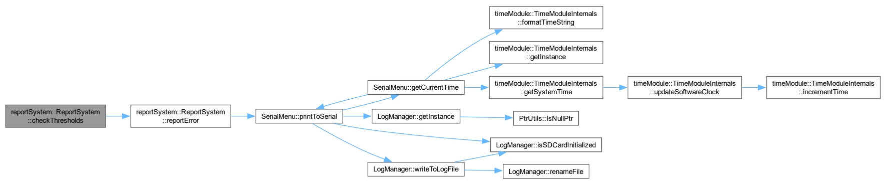
currentTemp	-> The current temperature
currentPressure	-> The current pressure

Returns

true -> if the thresholds are met

false -> if the thresholds are not met

Here is the call graph for this function:


reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.23.2.4 detectStackOverflow()

```
bool ReportSystem::detectStackOverflow ( )  [static]
```

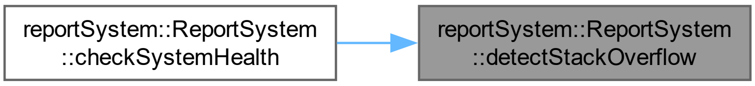
Detect Stack Overflow.

Returns

true -> if the stack has overflowed

false -> if the stack has not overflowed

Here is the caller graph for this function:


reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::detectStackOverflow

6.23.2.5 getCPULoad()

```
float ReportSystem::getCPULoad ( )
```

Getter for the CPU Load.

Returns

float -> The CPU Load

6.23.2.6 getCurrentTime()

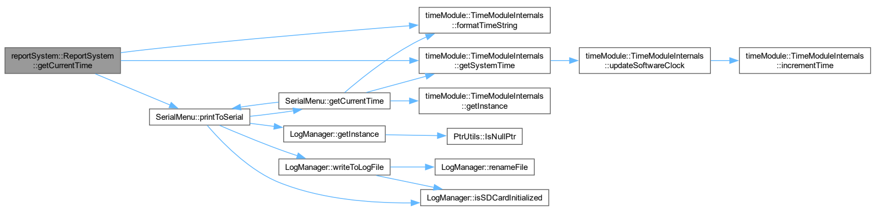
```
String ReportSystem::getCurrentTime ( )
```

Get the Current Time of the system.

Returns

String -> The current time

Here is the call graph for this function:


reportSystem::ReportSystem ::getCurrentTime
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getInstance
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::writeToLogFile
LogManager::renameFile
LogManager::isSDCardInitialized

6.23.2.7 getLastError()

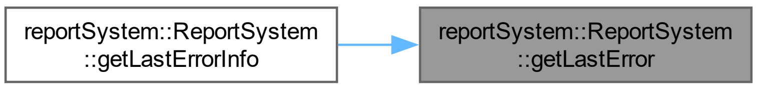
```
String ReportSystem::getLastError ( )
```

Get the Last Error message from EEPROM.

Returns

String -> The last error message

Here is the caller graph for this function:


reportSystem::ReportSystem ::getLastErrorInfo
reportSystem::ReportSystem ::getLastError

6.23.2.8 getLastErrorInfo()

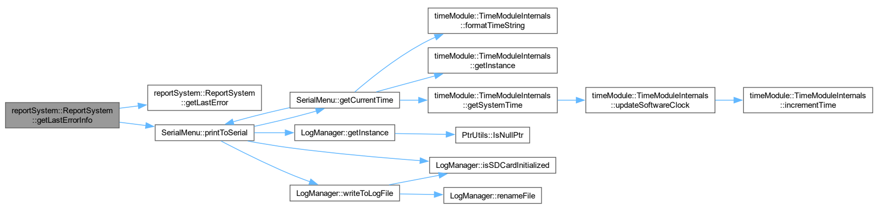
```
bool ReportSystem::getLastErrorInfo ( )
```

Get the Last Error message from EEPROM.

Returns

bool -> used by the Endpoint to report to HAS

Here is the call graph for this function:


reportSystem::ReportSystem ::getLastErrorInfo
reportSystem::ReportSystem ::getLastError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.23.2.9 getMemoryStatus()

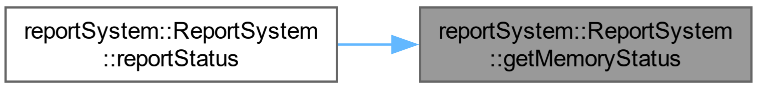
```
String ReportSystem::getMemoryStatus ( )
```

Get the Memory Status of the system.

Returns

String -> The memory status

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
reportSystem::ReportSystem ::getMemoryStatus

6.23.2.10 getStackDump()

```
String ReportSystem::getStackDump ( )
```

Get the Stack Dump of the system.

Returns

String -> The stack dump

6.23.2.11 hasNoSavedErrors()

```
bool ReportSystem::hasNoSavedErrors ( ) const
```

Function to report to the HAS via Endpoint if any errors were written to the EEPROM.

Returns

True if no errors in EEPROM, false otherwise

6.23.2.12 isCommunicationOK()

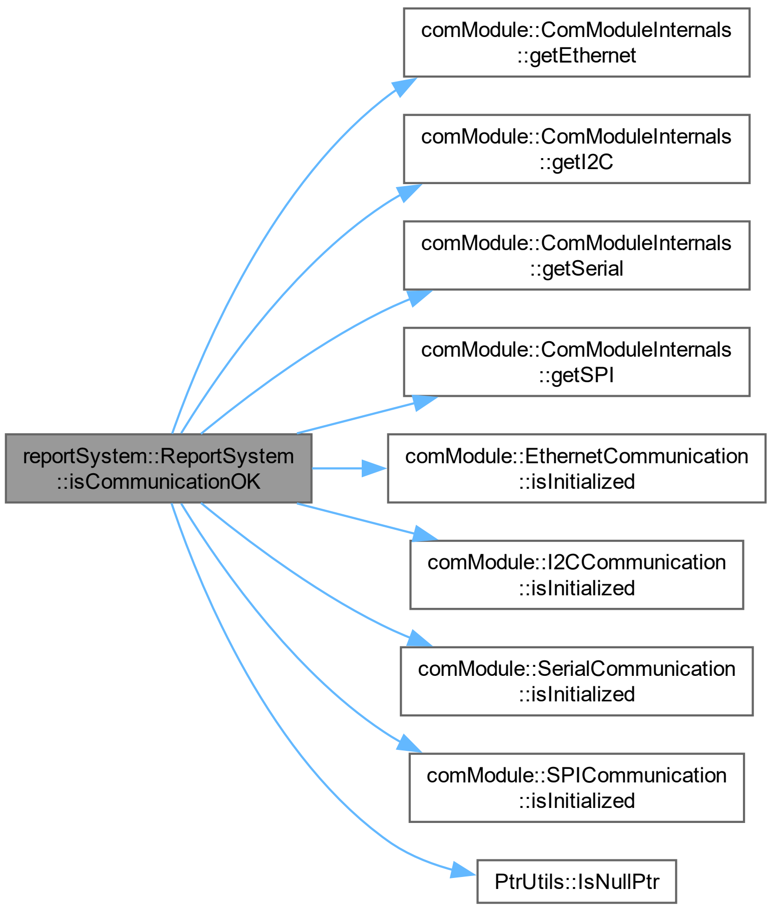
```
bool ReportSystem::isCommunicationOK ( ) const
```

Function to report to the HAS via Endpoint if Communication is initialized and ok.

Returns

True if communication is up, false otherwise

Here is the call graph for this function:


comModule::ComModuleInternals ::getEthernet
comModule::ComModuleInternals ::getI2C
comModule::ComModuleInternals ::getSerial
comModule::ComModuleInternals ::getSPI
reportSystem::ReportSystem ::isCommunicationOK
comModule::EthernetCommunication ::isInitialized
comModule::I2CCommunication ::isInitialized
comModule::SerialCommunication ::isInitialized
comModule::SPICommunication ::isInitialized
PtrUtils::IsNullPtr

6.23.2.13 isMemoryOK()

```
bool ReportSystem::isMemoryOK ( ) const
```

Function to report to the HAS via Endpoint if Memory is in between the required bounds.

Returns

True if memory is in between legal bounds, false otherwise

6.23.2.14 isRamOK()

```
bool ReportSystem::isRamOK ( ) const
```

Function to report to the HAS via Endpoint if RAM is in between the required bounds.

Returns

True if RAM is in between legal bounds, false otherwise

6.23.2.15 isStackSafe()

```
bool ReportSystem::isStackSafe ( ) const
```

Function to report to the HAS via Endpoint if Stack is not overflowing or has other issues.

Returns

True if stack if safe, false otherwise

6.23.2.16 isTemperatureSensorOK()

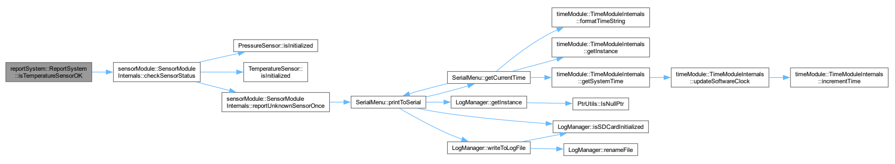
```
bool ReportSystem::isTemperatureSensorOK ( ) const
```

Function to report to the HAS via Endpoint if TempSensor is initialized and ok.

Returns

True if sensor is ok, false otherwise

Here is the call graph for this function:


reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
PressureSensor::isInitialized
TemperatureSensor:: isInitialized
sensorModule::SensorModule Internals::reportUnknownSensorOnce
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.23.2.17 reportError()

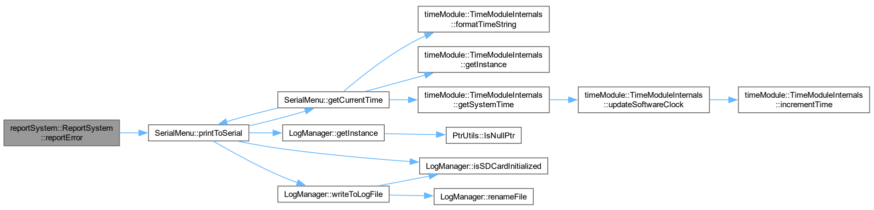
```
void ReportSystem::reportError (
            const char * errorMessage )
```

Function to log an error message.

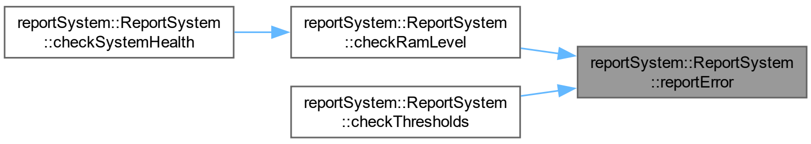
Parameters

errorMessage	-> The error message to log

Here is the call graph for this function:


reportSystem::ReportSystem ::reportError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

Here is the caller graph for this function:


reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError

6.23.2.18 reportStatus()

```
bool ReportSystem::reportStatus (
                bool active )
```

Function to report the status of the system.

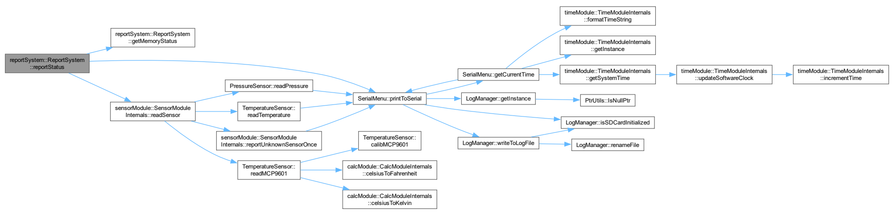
Parameters

active	-> The status of the system

Returns

bool -> The status of the system

Here is the call graph for this function:


reportSystem::ReportSystem ::reportStatus
reportSystem::ReportSystem ::getMemoryStatus
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
TemperatureSensor:: readTemperature
sensorModule::SensorModule Internals::reportUnknownSensorOnce
TemperatureSensor:: readMCP9601
TemperatureSensor:: calibMCP9601
calcModule::CalcModuleInternals ::celsiusToFahrenheit
calcModule::CalcModuleInternals ::celsiusToKelvin
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile

6.23.2.19 saveLastError()

```
void ReportSystem::saveLastError (
             const char * error )
```

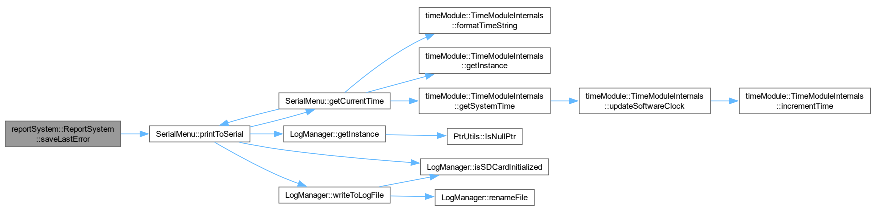
Saves last error message to EEPROM.

HINT: KEEP IN MIND ∼100 000 write cycles per cell!

Parameters

error	-> The error message to save

Here is the call graph for this function:


reportSystem::ReportSystem ::saveLastError
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile

6.23.2.20 setThreshold()

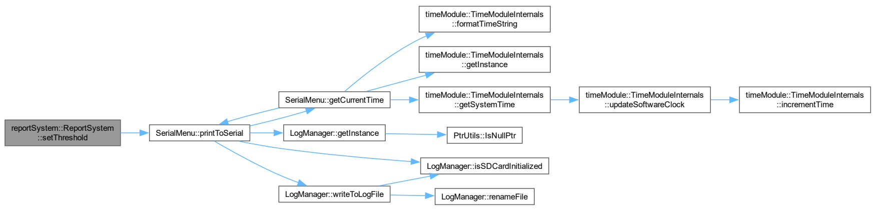
```
void ReportSystem::setThreshold (
             float tempThreshold,
             float pressureThreshold )
```

Set Thresholds for the pressure and temperature sensors.

Parameters

tempThreshold	-> The temperature threshold
pressureThreshold	-> The pressure threshold

Here is the call graph for this function:


reportSystem::ReportSystem ::setThreshold
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp

6.24 locker::ScopedLock Class Reference

Scoped lock class for mutexes.

```
#include <scopedLock.h>
```

Public Member Functions

- ScopedLock (frt::Mutex &mutex)

    Construct a new Scoped Lock object.
- **ScopedLock** (const ScopedLock &)=delete
- ScopedLock & **operator=** (const ScopedLock &)=delete
- ScopedLock (ScopedLock &&other) noexcept

    Construct a new Scoped Lock object.
- ScopedLock & **operator=** (ScopedLock &&)=delete

6.24.1 Detailed Description

Scoped lock class for mutexes.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 ScopedLock() [1/2]

```
locker::ScopedLock::ScopedLock (
            frt::Mutex & mutex )  [inline], [explicit]
```

Construct a new Scoped Lock object.

Parameters

mutex	-> The mutex to lock

6.24.2.2 ScopedLock() `[2/2]`

```
locker::ScopedLock::ScopedLock (
            ScopedLock && other )  [inline], [noexcept]
```

Construct a new Scoped Lock object.

Parameters

other	-> The other ScopedLock object to move from

The documentation for this class was generated from the following file:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h

6.25 ScopedPointer< T > Class Template Reference

Template class for a Scoped Pointer.

`#include <ptrUtils.h>`

Public Member Functions

- **ScopedPointer** (T ∗p=nullptr)
- T ∗ get () const

    Function to get the pointer.
- T ∗ release ()

    Function to release the pointer.
- void reset (T ∗p=nullptr)

    Function to reset the pointer.
- T & operator∗ () const

    Operator to dereference the pointer.
- T ∗ operator-> () const

    Operator to access the pointer.

6.25.1 Detailed Description

template<typename T>
class ScopedPointer< T >

Template class for a Scoped Pointer.

Template Parameters

T	-> The type of the pointer.

6.25.2 Member Function Documentation

6.25.2.1 get()

```
template<typename T >
T * ScopedPointer< T >::get ( ) const  [inline]
```

Function to get the pointer.

Returns

T* -> The pointer.

6.25.2.2 operator*()

```
template<typename T >
T & ScopedPointer< T >::operator* ( ) const  [inline]
```

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.25.2.3 operator->()

```
template<typename T >
T * ScopedPointer< T >::operator-> ( ) const  [inline]
```

Operator to access the pointer.

Returns

T* -> The pointer.

6.25.2.4 release()

```
template<typename T >
T * ScopedPointer< T >::release ( )  [inline]
```

Function to release the pointer.

Returns

T* -> The released pointer.

6.25.2.5 reset()

```
template<typename T >
void ScopedPointer< T >::reset (
             T * p = nullptr )  [inline]
```

Function to reset the pointer.

Parameters

p	-> The pointer to reset to.

The documentation for this class was generated from the following file:

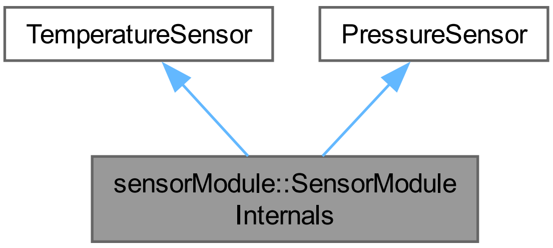
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.26 sensorModule::SensorModuleInternals Class Reference

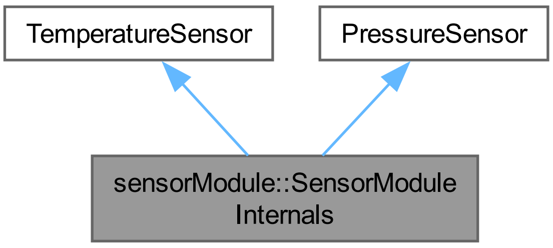
Class for the sensor module internals.

```
#include <sensorModule.h>
```

Inheritance diagram for sensorModule::SensorModuleInternals:



Collaboration diagram for sensorModule::SensorModuleInternals:



Public Member Functions

- void **initialize** ()

  Initialize the sensors.
- float readSensor (SensorType type)

  Function to read the sensor.
- bool calibrateSensor (SensorType type)

  Function to calibrate the sensor.
- bool checkSensorStatus (SensorType type)

  Function to check the status of the sensor.
- void reportUnknownSensorOnce (SensorType type, const __FlashStringHelper ∗context)

  Function to Report the faulty sensor only if it changed, saves us some String and doesn't shred the SRAM that much.

Public Member Functions inherited from TemperatureSensor

- void **initialize** ()

  Function to initialize the temperature sensor.
- float readTemperature ()

  Function to read the temperature from the sensor.
- float readMCP9601 (Units unit, SensorID sensor)

  Function to read form specific sensor MCP9601.
- bool isInitialized () const

  Check if the temperature sensor is initialized.
- uint8_t calibMCP9601 (SensorID sensor)

  Method to calibrate the MCP9601 sensor, Indoor and Outdoor Env.

Public Member Functions inherited from PressureSensor

- void **initialize** ()

  Function to initialize the pressure sensor.
- float readPressure ()

  Function to read the pressure from the sensor.
- bool isInitialized () const

  Function to check if the pressure sensor is initialized.

6.26.1 Detailed Description

Class for the sensor module internals.

6.26.2 Member Function Documentation

6.26.2.1 calibrateSensor()

```
bool SensorModuleInternals::calibrateSensor (
            SensorType type )
```

Function to calibrate the sensor.

Parameters

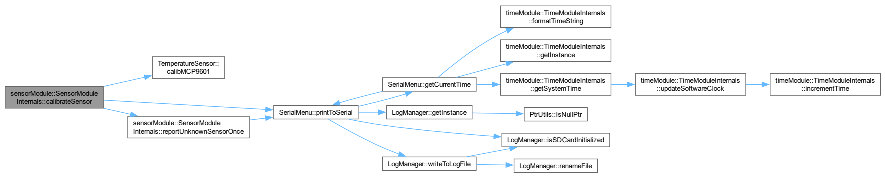
type	-> The type of the sensor to calibrate.

Returns

true -> if the sensor was calibrated successfully

false -> if the sensor was not calibrated successfully

Here is the call graph for this function:


sensorModule::SensorModule Internals::calibrateSensor
TemperatureSensor:: calibMCP9601
sensorModule::SensorModule Internals::reportUnknownSensorOnce
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.26.2.2 checkSensorStatus()

```
bool SensorModuleInternals::checkSensorStatus (
            SensorType type )
```

Function to check the status of the sensor.

Parameters

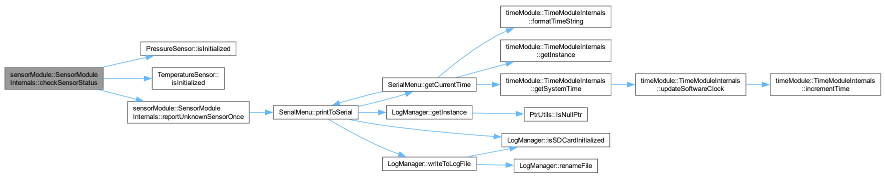
type	-> The type of the sensor to check.

Returns

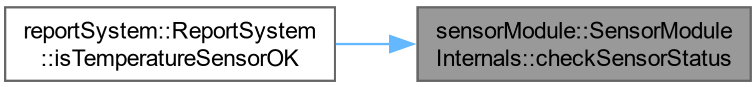
true -> if the sensor is healthy

false -> if the sensor is not healthy

Here is the call graph for this function:


sensorModule::SensorModule Internals::checkSensorStatus
PressureSensor::isInitialized
TemperatureSensor:: isInitialized
sensorModule::SensorModule Internals::reportUnknownSensorOnce
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

Here is the caller graph for this function:


reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus

6.26.2.3 readSensor()

```
float SensorModuleInternals::readSensor (
            SensorType type )
```

Function to read the sensor.

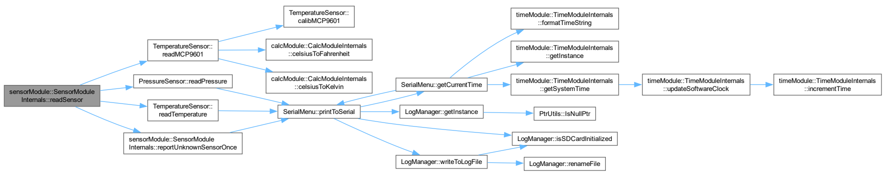
Parameters

type	-> The type of the sensor to read.

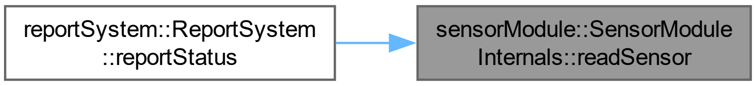
Returns

float -> The value of the sensor.

Here is the call graph for this function:


sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readMCP9601
PressureSensor::readPressure
TemperatureSensor:: readTemperature
sensorModule::SensorModule Internals::reportUnknownSensorOnce
TemperatureSensor:: calibMCP9601
calcModule::CalcModuleInternals ::celsiusToFahrenheit
calcModule::CalcModuleInternals ::celsiusToKelvin
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor

6.26.2.4 reportUnknownSensorOnce()

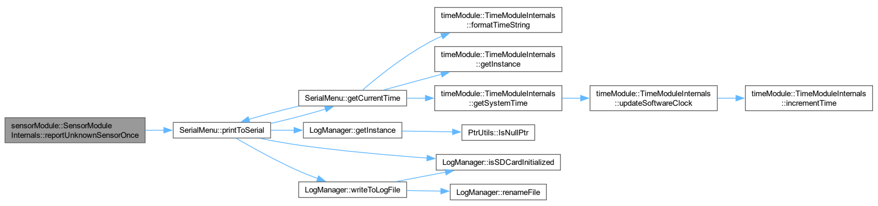
```
void SensorModuleInternals::reportUnknownSensorOnce (
            SensorType type,
            const __FlashStringHelper * context )
```

Function to Report the faulty sensor only if it changed, saves us some String and doesn't shred the SRAM that much.

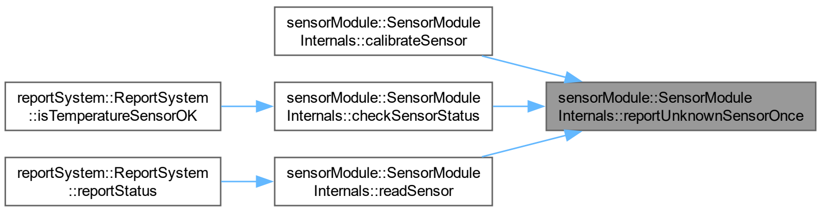
Parameters

type	-> The type of the sensor to check.
context	-> the actual context, so the method which reports from.

Here is the call graph for this function:


sensorModule::SensorModule Internals::reportUnknownSensorOnce
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile

Here is the caller graph for this function:


sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.cpp

6.27 comModule::SerialCommunication Class Reference

Class to handle Serial communication.

```
#include <SER.h>
```

Public Member Functions

- void beginSerial (long baudRate)

  Function to start the serial communication.
- void **endSerial** ()

  Function to end the serial communication.
- void sendSerialData (const char ∗data)

  Function to end the serial communication.
- void receiveSerialData (char ∗buffer, size_t length)

  Function to receive data over serial.
- bool isInitialized () const

  Function to check if the serial communication is initialized.

6.27.1 Detailed Description

Class to handle Serial communication.

6.27.2 Member Function Documentation

6.27.2.1 beginSerial()

```
void SerialCommunication::beginSerial (
             long baudRate )
```

Function to start the serial communication.

Parameters

baudRate	-> The baud rate to use for the serial communication

6.27.2.2 isInitialized()

```
bool SerialCommunication::isInitialized ( ) const
```

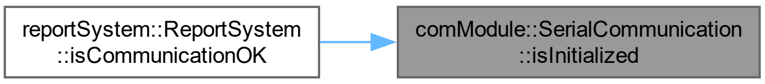
Function to check if the serial communication is initialized.

Returns

true -> if the serial communication is initialized

false -> if the serial communication is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::SerialCommunication ::isInitialized

6.27.2.3 receiveSerialData()

```
void SerialCommunication::receiveSerialData (
             char * buffer,
             size_t length )
```

Function to receive data over serial.

Parameters

buffer	-> The buffer to read the data into
length	-> The length of the data to read

6.27.2.4 sendSerialData()

```
void SerialCommunication::sendSerialData (
             const char * data )
```

Function to end the serial communication.

Parameters

data	-> The data to send

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER.cpp

6.28 SerialMenu Class Reference

Class for the serial menu.

```
#include <serialMenu.h>
```

Public Types

- enum class **OutputLevel** {
  DEBUG , **INFO** , **WARNING** , **ERROR** ,
  CRITICAL , **STATUS** , **PLAIN** }

Public Member Functions

- void load (MenuItem *items, size_t size)

  Function to load the menu items.
- void **show** ()

  Function to show the menu.
- void **run** ()

  Function to run the menu.

Static Public Member Functions

- static void printToSerial (OutputLevel level, const String &message, bool newLine=true, bool log←
Message=false)

    Function to print a message to the serial port, using mutexes, output level and new line options.
- static void printToSerial (OutputLevel level, const __FlashStringHelper ∗message, bool newLine=true, bool logMessage=false)

    Function to print a message to the serial port, using mutexes, output level and new line options.
- static void printToSerial (const String &message, bool newLine=true, bool logMessage=false)

    Funtion to print a message to the serial port, using mutexes, output level and new line options.
- static void printToSerial (const __FlashStringHelper ∗message, bool newLine=true, bool logMessage=false)

    Function to print a message to the serial port, using mutexes, output level and new line options.
- static String getCurrentTime ()

    Getter for the current time.

6.28.1 Detailed Description

Class for the serial menu.

6.28.2 Member Function Documentation

6.28.2.1 getCurrentTime()

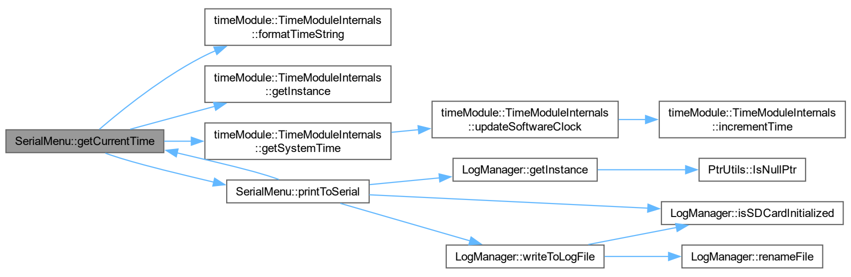
```
String SerialMenu::getCurrentTime ( )  [static]
```

Getter for the current time.

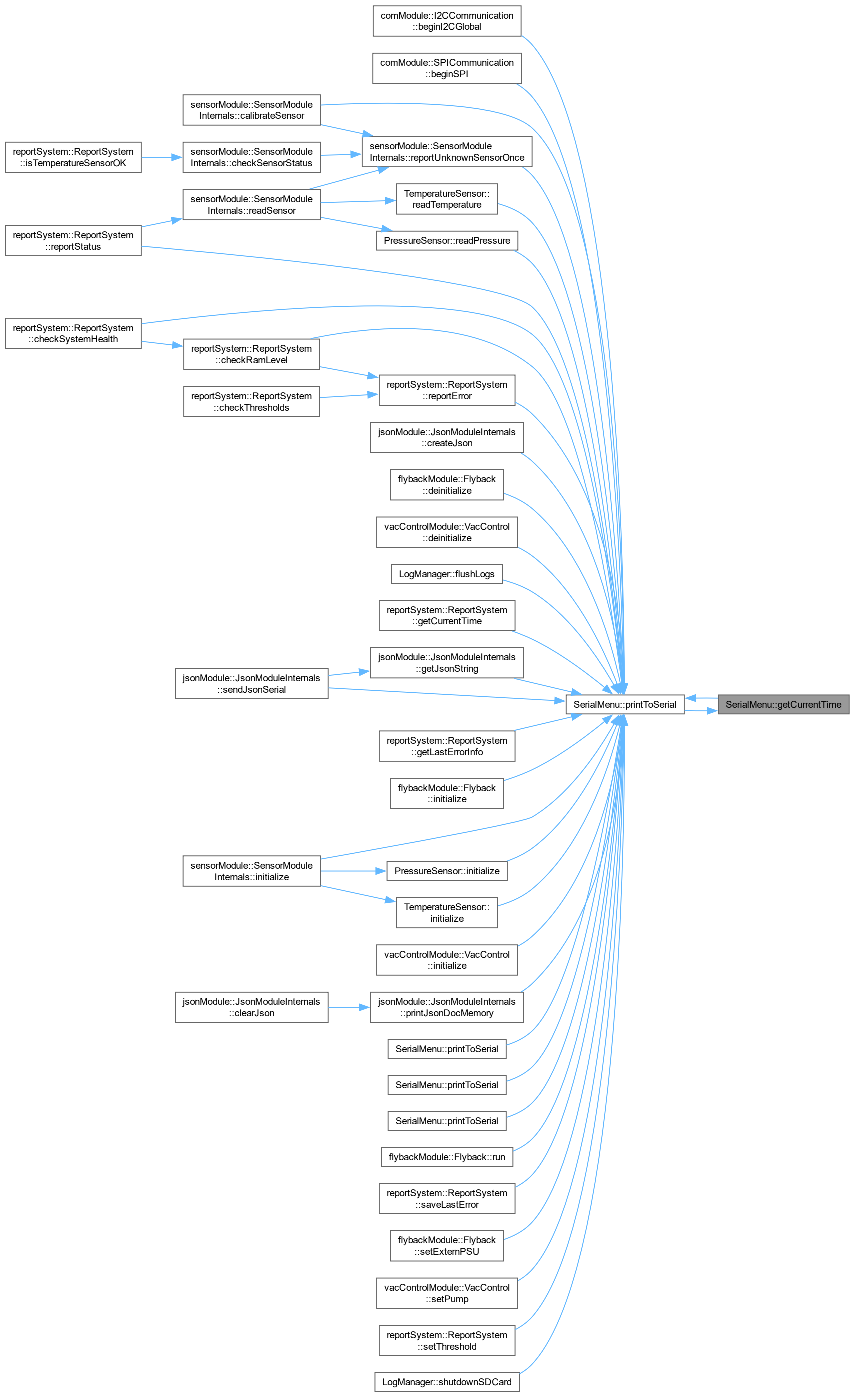
Returns

The current time as a String

Here is the call graph for this function:


timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
LogManager::getInstance
PtrUtils::IsNullPtr
SerialMenu::printToSerial
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::reportStatus
PressureSensor::readPressure
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::reportError
reportSystem::ReportSystem ::checkThresholds
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
jsonModule::JsonModuleInternals ::getJsonString
jsonModule::JsonModuleInternals ::sendJsonSerial
SerialMenu::printToSerial
SerialMenu::getCurrentTime
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
TemperatureSensor:: initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

6.28.2.2 load()

```
void SerialMenu::load (
            MenuItem * items,
            size_t size )
```

Function to load the menu items.

Parameters

items	-> The menu items.
size	-> The size of the menu items.

6.28.2.3 printToSerial() `[1/4]`

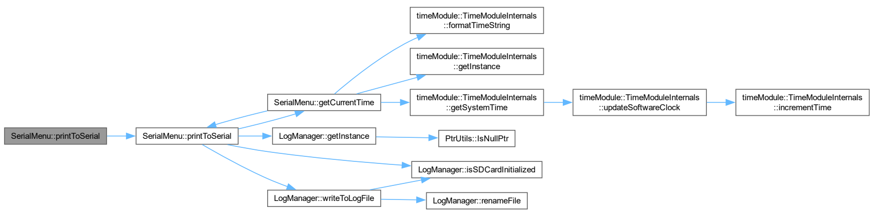
```
void SerialMenu::printToSerial (
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

Function to print a message to the serial port, using mutexes, output level and new line options.

Parameters

message	-> The message to print, a __FlashStringHelper pointer.
newLine	-> Whether to add a new line at the end of the message.

Here is the call graph for this function:



timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
SerialMenu::printToSerial
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

6.28.2.4 printToSerial() `[2/4]`

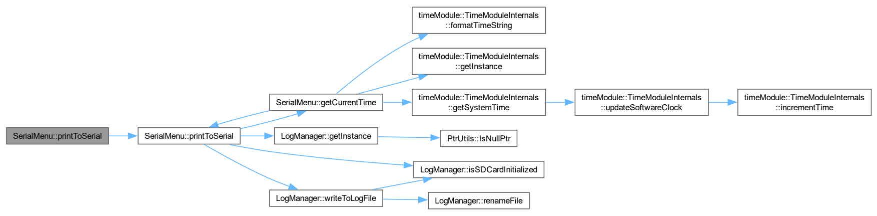
```
void SerialMenu::printToSerial (
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

Funtion to print a message to the serial port, using mutexes, output level and new line options.

Parameters

message	-> The message to print, a String object.
newLine	-> Whether to add a new line at the end of the message.

Here is the call graph for this function:


SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.28.2.5 printToSerial() `[3/4]`

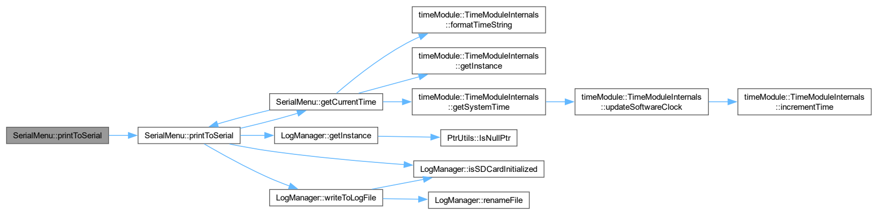
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

Function to print a message to the serial port, using mutexes, output level and new line options.

Parameters

level	-> The output level of the message.
message	-> The message to print, a __FlashStringHelper pointer.
newLine	-> Whether to add a new line at the end of the message.

Here is the call graph for this function:


SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
timeModule::TimeModuleInternals ::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

6.28.2.6 printToSerial() `[4/4]`

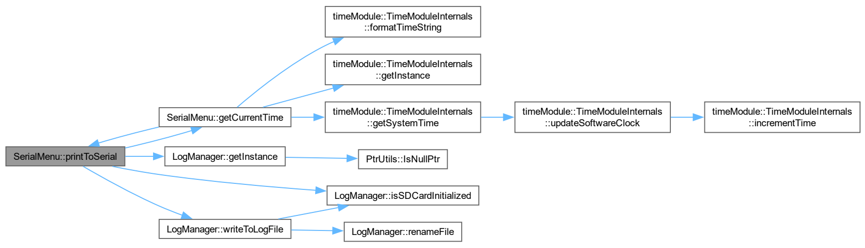
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

Function to print a message to the serial port, using mutexes, output level and new line options.

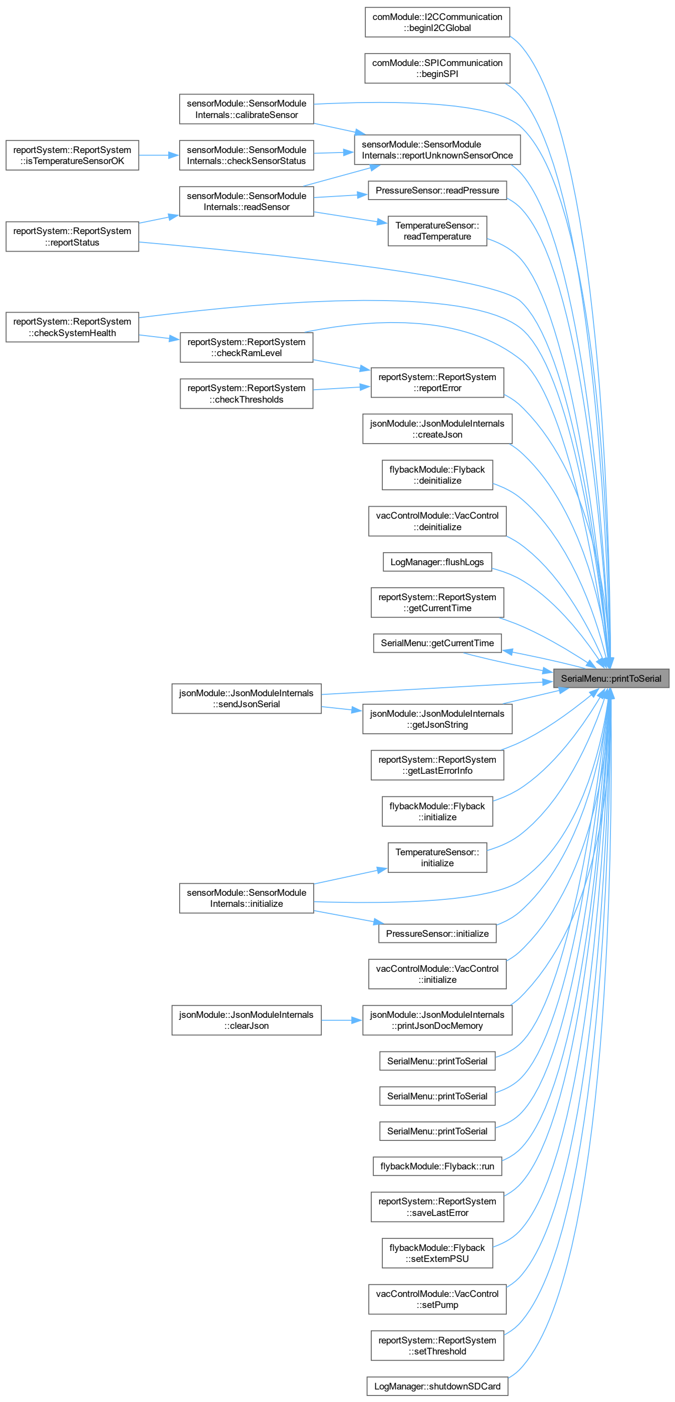
Parameters

level	-> The output level of the message.
message	-> The message to print, a String object.
newLine	-> Whether to add a new line at the end of the message.

Here is the call graph for this function:


timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
SerialMenu::getCurrentTime
SerialMenu::printToSerial
jsonModule::JsonModuleInternals ::sendJsonSerial
jsonModule::JsonModuleInternals ::getJsonString
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
TemperatureSensor:: initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.cpp

6.29 comModule::SPICommunication Class Reference

Class to handle SPI communication.

`#include <SPII.h>`

Public Member Functions

- void **beginSPI** ()

  Function to initialize the SPI communication.
- void **endSPI** ()

  Function to end the SPI communication.
- void spiWrite (uint8_t ∗data, size_t length)

  Function to write data over SPI.
- void spiRead (uint8_t ∗buffer, size_t length)

  Function to read data over SPI.
- bool isInitialized () const

  Function to check if the SPI communication is initialized.

6.29.1 Detailed Description

Class to handle SPI communication.

6.29.2 Member Function Documentation

6.29.2.1 isInitialized()

```
bool SPICommunication::isInitialized ( ) const
```

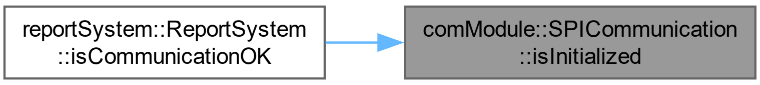
Function to check if the SPI communication is initialized.

Returns

true -> if the SPI communication is initialized

false -> if the SPI communication is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isCommunicationOK
comModule::SPICommunication ::isInitialized

6.29.2.2 spiRead()

```
void SPICommunication::spiRead (
            uint8_t ∗ buffer,
            size_t length )
```

Function to read data over SPI.

Parameters

buffer	-> The buffer to read the data into
length	-> The length of the data to read

6.29.2.3 spiWrite()

```
void SPICommunication::spiWrite (
             uint8_t * data,
             size_t length )
```

Function to write data over SPI.

Parameters

data	-> The data to write
length	-> The length of the data

The documentation for this class was generated from the following files:

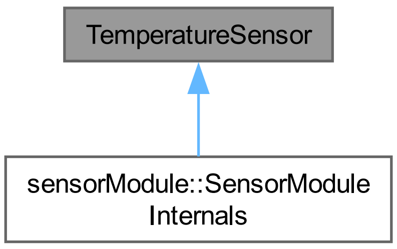
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.cpp

6.30 TemperatureSensor Class Reference

Temperature sensor class.

`#include <temperature.h>`

Inheritance diagram for TemperatureSensor:



Public Member Functions

- void **initialize** ()

  Function to initialize the temperature sensor.
- float readTemperature ()

  Function to read the temperature from the sensor.
- float readMCP9601 (Units unit, SensorID sensor)

  Function to read form specific sensor MCP9601.
- bool isInitialized () const

  Check if the temperature sensor is initialized.
- uint8_t calibMCP9601 (SensorID sensor)

  Method to calibrate the MCP9601 sensor, Indoor and Outdoor Env.

6.30.1 Detailed Description

Temperature sensor class.

6.30.2 Member Function Documentation

6.30.2.1 calibMCP9601()

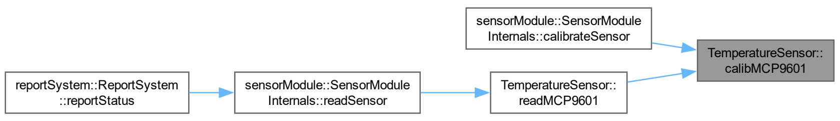
```
uint8_t TemperatureSensor::calibMCP9601 (
            SensorID sensor )
```

Method to calibrate the MCP9601 sensor, Indoor and Outdoor Env.

Returns

uint8_t -> the status of the calibration

Here is the caller graph for this function:


sensorModule::SensorModule Internals::calibrateSensor
TemperatureSensor:: calibMCP9601
reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readMCP9601

6.30.2.2 isInitialized()

```
bool TemperatureSensor::isInitialized ( ) const
```

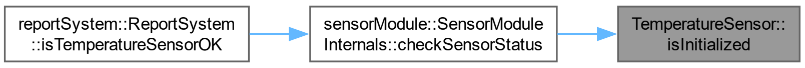
Check if the temperature sensor is initialized.

Returns

true -> if the temperature sensor is initialized

false -> if the temperature sensor is not initialized

Here is the caller graph for this function:


reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
TemperatureSensor:: isInitialized

6.30.2.3 readMCP9601()

```
float TemperatureSensor::readMCP9601 (
            Units unit,
            SensorID sensor )
```

Function to read form specific sensor MCP9601.

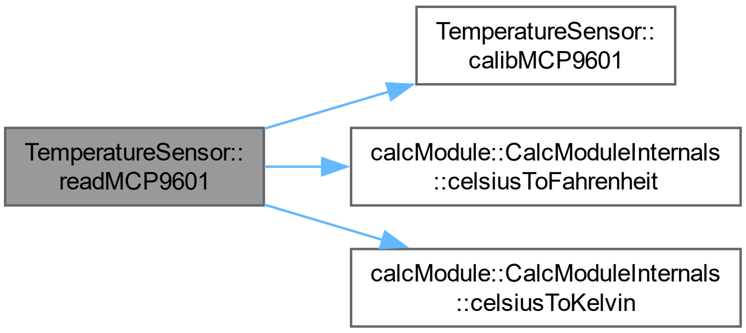
Parameters

unit	-> Choose the unit ℃, F, K

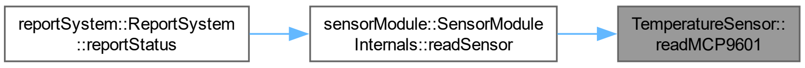
Returns

-> The temperature value

Here is the call graph for this function:



Here is the caller graph for this function:



6.30.2.4 readTemperature()

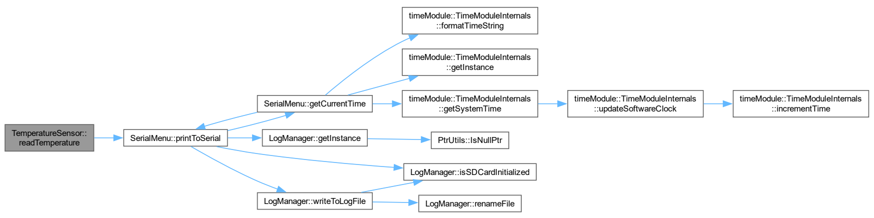
```
float TemperatureSensor::readTemperature ( )
```

Function to read the temperature from the sensor.

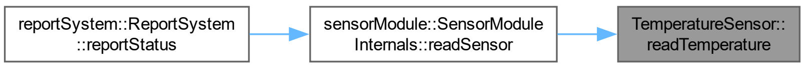
Returns

float -> The temperature value.

Here is the call graph for this function:


TemperatureSensor::
readTemperature
SerialMenu::printToSerial
SerialMenu::getCurrentTime
LogManager::getInstance
LogManager::writeToLogFile
timeModule::TimeModuleInternals
::formatTimeString
timeModule::TimeModuleInternals
::getInstance
timeModule::TimeModuleInternals
::getSystemTime
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::renameFile
timeModule::TimeModuleInternals
::updateSoftwareClock
timeModule::TimeModuleInternals
::incrementTime

Here is the caller graph for this function:


reportSystem::ReportSystem ::reportStatus
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp

6.31 timeModule::TimeModuleInternals Class Reference

Class to handle Systemtime.

```
#include <timeModule.h>
```

Public Member Functions

- bool setTimeFromHas (const String &timeString)

  Set the Time From Has object to the system time.
- void setSystemTime (const DateTimeStruct &dt)

  Set the System Time object of the system.
- void **updateSoftwareClock** ()

  Updates the software clock.
- DateTimeStruct getSystemTime ()

  Get the System Time object.

Static Public Member Functions

- static void incrementTime (DateTimeStruct ∗dt)

  Function to increment the time of the system.
- static String formatTimeString (const DateTimeStruct &dt)

  Function to format the time to a string.
- static TimeModuleInternals ∗ getInstance ()

  Get the Instance object, Singleton pattern.

6.31.1 Detailed Description

Class to handle Systemtime.

6.31.2 Member Function Documentation

6.31.2.1 formatTimeString()

```
String TimeModuleInternals::formatTimeString (
            const DateTimeStruct & dt )  [static]
```

Function to format the time to a string.

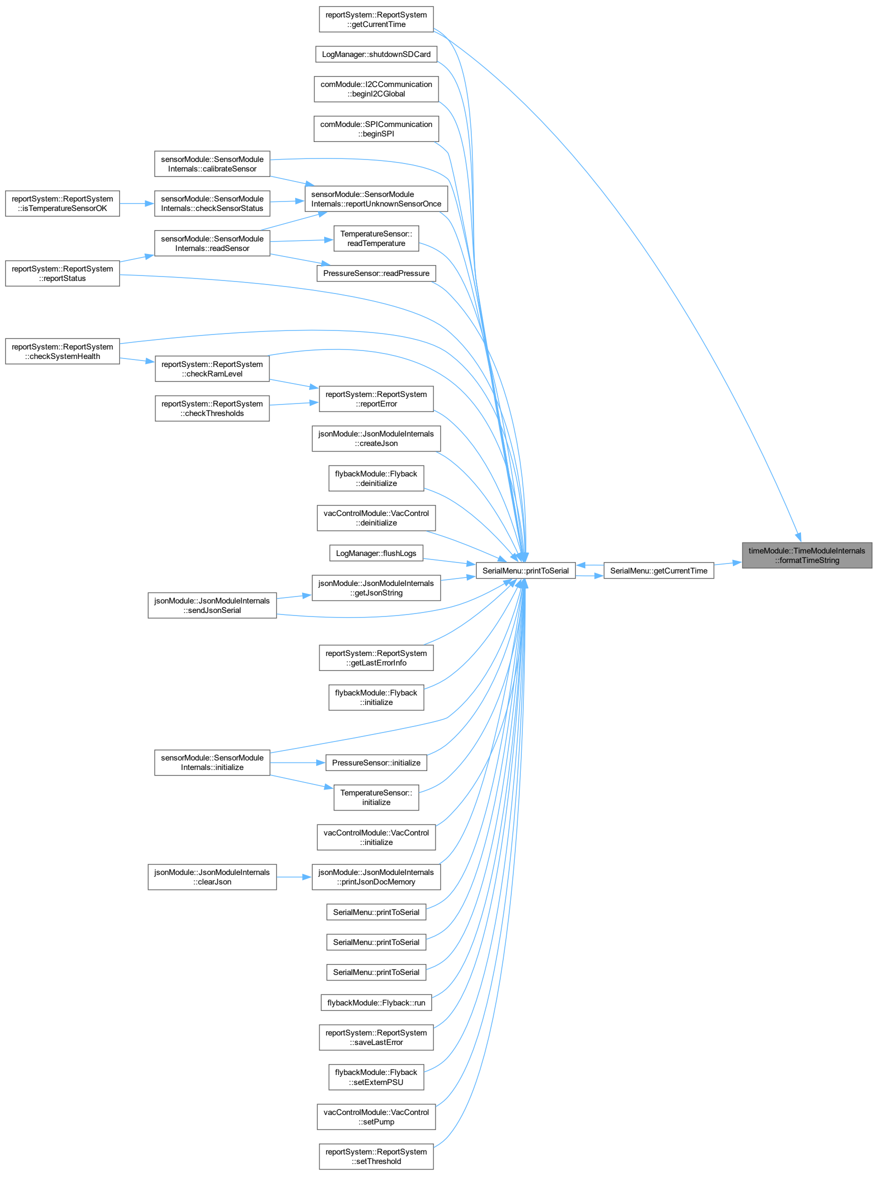
Parameters

dt	-> DateTimeStruct to format

Returns

String -> The formatted time.

Here is the caller graph for this function:


reportSystem::ReportSystem ::getCurrentTime
LogManager::shutdownSDCard
comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::reportStatus
PressureSensor::readPressure
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::reportError
reportSystem::ReportSystem ::checkThresholds
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
SerialMenu::printToSerial
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::formatTimeString
jsonModule::JsonModuleInternals ::getJsonString
jsonModule::JsonModuleInternals ::sendJsonSerial
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
TemperatureSensor:: initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold

6.31.2.2 getInstance()

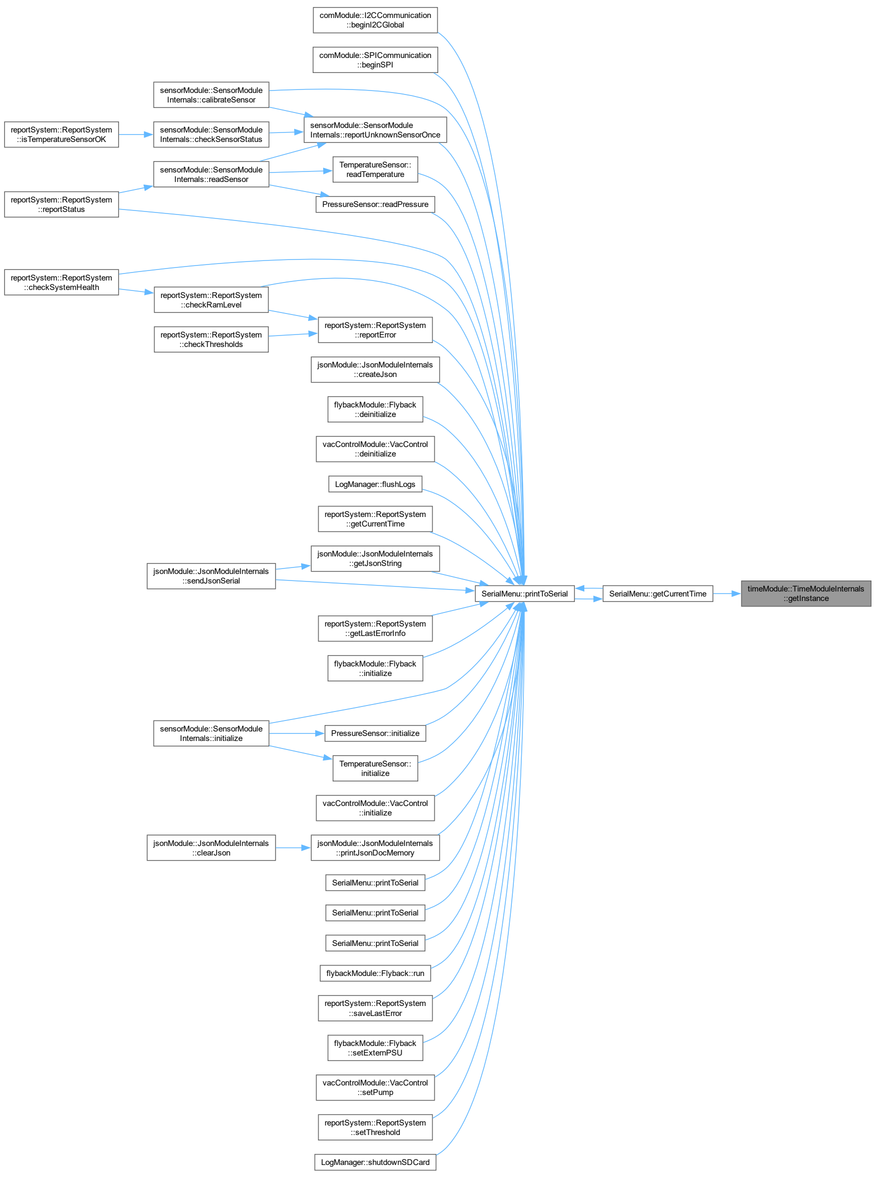
```
TimeModuleInternals * TimeModuleInternals::getInstance ( )  [static]
```

Get the Instance object, Singleton pattern.

Returns

TimeModuleInternals∗ -> The instance of the TimeModuleInternals.

Here is the caller graph for this function:


comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::reportStatus
PressureSensor::readPressure
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::reportError
reportSystem::ReportSystem ::checkThresholds
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
reportSystem::ReportSystem ::getCurrentTime
jsonModule::JsonModuleInternals ::getJsonString
jsonModule::JsonModuleInternals ::sendJsonSerial
SerialMenu::printToSerial
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getInstance
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
TemperatureSensor:: initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold
LogManager::shutdownSDCard

6.31.2.3 getSystemTime()

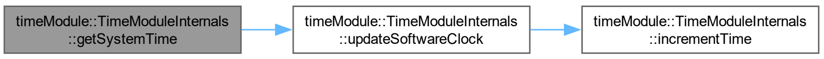
```
DateTimeStruct TimeModuleInternals::getSystemTime ( )
```

Get the System Time object.

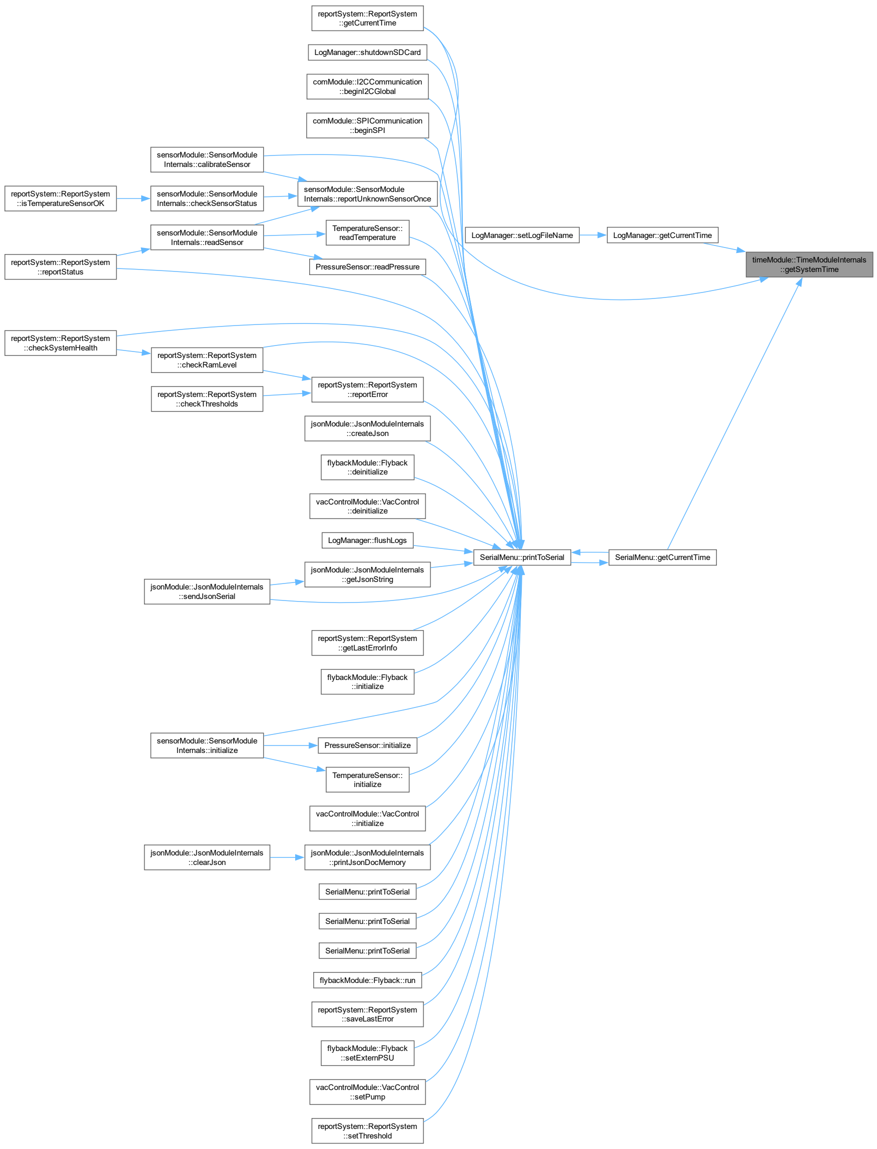
Returns

DateTimeStruct -> The system time.

Here is the call graph for this function:


timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime

Here is the caller graph for this function:


reportSystem::ReportSystem ::getCurrentTime
LogManager::shutdownSDCard
comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature
PressureSensor::readPressure
reportSystem::ReportSystem ::reportStatus
LogManager::setLogFileName
LogManager::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
SerialMenu::printToSerial
SerialMenu::getCurrentTime
jsonModule::JsonModuleInternals ::getJsonString
jsonModule::JsonModuleInternals ::sendJsonSerial
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
TemperatureSensor:: initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold

6.31.2.4 incrementTime()

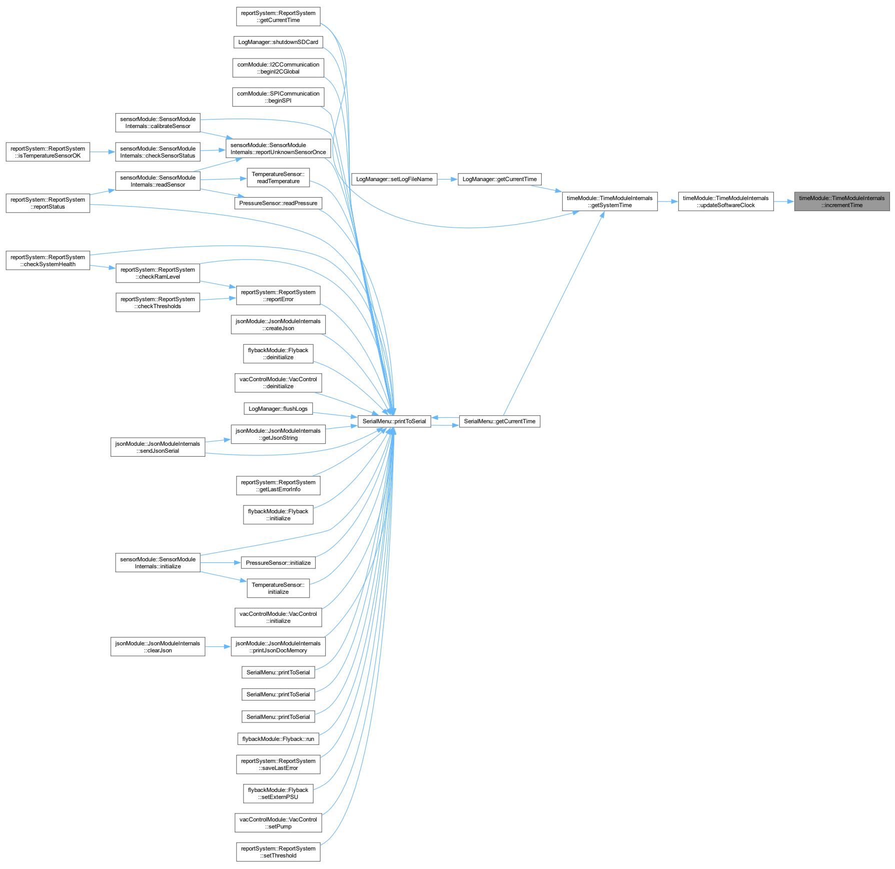
```
void TimeModuleInternals::incrementTime (
            DateTimeStruct * dt ) [static]
```

Function to increment the time of the system.

Parameters

dt	-> DateTimeStruct to increment time

Here is the caller graph for this function:


reportSystem::ReportSystem ::getCurrentTime
LogManager::shutdownSDCard
comModule::I2CCommunication ::beginI2CGlobal
comModule::SPICommunication ::beginSPI
sensorModule::SensorModule Internals::calibrateSensor
reportSystem::ReportSystem ::isTemperatureSensorOK
sensorModule::SensorModule Internals::checkSensorStatus
sensorModule::SensorModule Internals::reportUnknownSensorOnce
sensorModule::SensorModule Internals::readSensor
TemperatureSensor:: readTemperature
reportSystem::ReportSystem ::reportStatus
PressureSensor::readPressure
LogManager::setLogFileName
LogManager::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
reportSystem::ReportSystem ::checkSystemHealth
reportSystem::ReportSystem ::checkRamLevel
reportSystem::ReportSystem ::checkThresholds
reportSystem::ReportSystem ::reportError
jsonModule::JsonModuleInternals ::createJson
flybackModule::Flyback ::deinitialize
vacControlModule::VacControl ::deinitialize
LogManager::flushLogs
SerialMenu::printToSerial
SerialMenu::getCurrentTime
jsonModule::JsonModuleInternals ::getJsonString
jsonModule::JsonModuleInternals ::sendJsonSerial
reportSystem::ReportSystem ::getLastErrorInfo
flybackModule::Flyback ::initialize
sensorModule::SensorModule Internals::initialize
PressureSensor::initialize
TemperatureSensor:: initialize
vacControlModule::VacControl ::initialize
jsonModule::JsonModuleInternals ::clearJson
jsonModule::JsonModuleInternals ::printJsonDocMemory
SerialMenu::printToSerial
SerialMenu::printToSerial
SerialMenu::printToSerial
flybackModule::Flyback::run
reportSystem::ReportSystem ::saveLastError
flybackModule::Flyback ::setExternPSU
vacControlModule::VacControl ::setPump
reportSystem::ReportSystem ::setThreshold

6.31.2.5 setSystemTime()

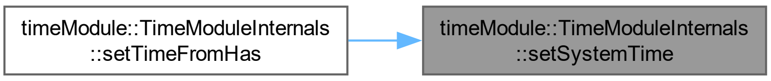
```
void TimeModuleInternals::setSystemTime (
            const DateTimeStruct & dt )
```

Set the System Time object of the system.

Parameters

dt	-> DateTimeStruct to set the system time to.

Here is the caller graph for this function:


timeModule::TimeModuleInternals ::setTimeFromHas
timeModule::TimeModuleInternals ::setSystemTime

6.31.2.6 setTimeFromHas()

```
bool TimeModuleInternals::setTimeFromHas (
            const String & timeString )
```

Set the Time From Has object to the system time.

Parameters

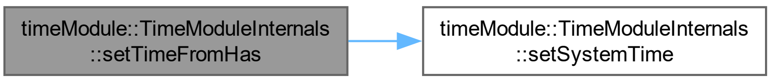
timeString	-> The time string to set the system time to.

Returns

true -> if the time was set successfully

false -> if the time was not set successfully

Here is the call graph for this function:


timeModule::TimeModuleInternals ::setTimeFromHas
timeModule::TimeModuleInternals ::setSystemTime

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp

6.32 vacControlModule::VacControl Class Reference

VacControl class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

```
#include <vacControl.h>
```

Public Member Functions

- void **initialize** ()

  Initialize the VacControl System This method sets up the pins and prepares the system for operation.
- void **deinitialize** ()

  denitialize the VacControl System This method shuts down the pins and prepares graceful restart.
- bool isInitialized () const

  Get the state of the VacControl system.
- MainSwitchStates getMainSwitchState ()

  Returns the state of the main switch.
- PumpSwitchStates getPumpSwitchState ()

  Returns the state of the pump switch.
- PumpState getPumpState ()

  Returns the state of the pump.
- Scenarios getScenario ()

  Executes logic depending on which Main-Switch state is active.
- Pressure measure ()

  Measures the actual pressure of the system.
- void setVacuumLed (float pressure, float targetPressure)

  Controls the vacuum LED based on the current and target pressures.
- int getScenarioFromPotValue (int potValue)

  Determines the scenario based on the potentiometer value.
- void setPump (bool flag)

  Set the Pump flag.
- void run ()

  Runs the main control loop for the VacControl system.
- void setExternScenario (int pressure)

  Function to set an external scenario, typically from remote input.
- int getExternScenario ()

  Getter function to retrieve the current external scenario state.
- void externPump (int pumpState)

  Process external data for scenarios (currently unused)
- void setExternPump (int state)

  Function to get the desired PinMode from HAS.
- int getExternPump ()
- void setExternPressure (float pressure)

  Sets the external pressure value.
- float getExternPressure ()

  Gets the external pressure value.

6.32.1 Detailed Description

VacControl class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

6.32.2 Member Function Documentation

6.32.2.1 externPump()

```
void VacControl::externPump (
             int pumpState )
```

Process external data for scenarios (currently unused)

This function could be expanded to process external scenario commands if needed.

6.32.2.2 getExternPressure()

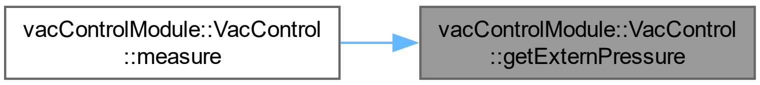
```
float VacControl::getExternPressure ( )
```

Gets the external pressure value.

Returns

The current external pressure value

Here is the caller graph for this function:


vacControlModule::VacControl ::measure
vacControlModule::VacControl ::getExternPressure

6.32.2.3 getExternPump()

```
int VacControl::getExternPump ( )
```

@ brief Getter Function to get the PinMode

6.32.2.4 getExternScenario()

```
int VacControl::getExternScenario ( )
```

Getter function to retrieve the current external scenario state.

Returns

The current external scenario state (integer)

6.32.2.5 getMainSwitchState()

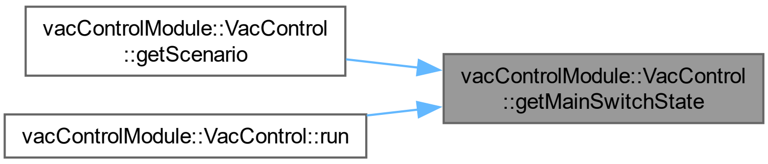
```
MainSwitchStates VacControl::getMainSwitchState ( )
```

Returns the state of the main switch.

Returns

The current state of the switch (Main_Switch_OFF, Main_Switch_MANUAL, etc.)

Here is the caller graph for this function:


vacControlModule::VacControl ::getScenario
vacControlModule::VacControl::run
vacControlModule::VacControl ::getMainSwitchState

6.32.2.6 getPumpState()

```
PumpState VacControl::getPumpState ( )
```

Returns the state of the pump.

Returns

The current state of the output for the pump (pump_ON, pump_OFF)

6.32.2.7 getPumpSwitchState()

```
PumpSwitchStates VacControl::getPumpSwitchState ( )
```

Returns the state of the pump switch.

Returns

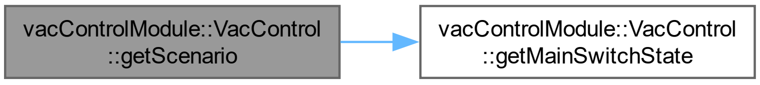
The current state of the switch (Switch_Pump_ON, Switch_PUMP_OFF)

6.32.2.8 getScenario()

```
Scenarios VacControl::getScenario ( )
```

Executes logic depending on which Main-Switch state is active.

This function decides which scenario to run based on the current state of the system. Here is the call graph for this function:


vacControlModule::VacControl ::getScenario
vacControlModule::VacControl ::getMainSwitchState

6.32.2.9 getScenarioFromPotValue()

```
int VacControl::getScenarioFromPotValue (
            int potValue )
```

Determines the scenario based on the potentiometer value.

Parameters

potValue	The value read from the potentiometer (used for pressure regulation)

Returns

The corresponding scenario based on the potentiometer value

6.32.2.10 isInitialized()

```
bool VacControl::isInitialized ( ) const
```

Get the state of the VacControl system.

Returns

true -> VacControl is initialized and ready

false -> VacControl is not initialized

6.32.2.11 measure()

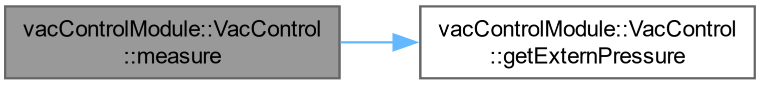
```
Pressure VacControl::measure ( )
```

Measures the actual pressure of the system.

Returns

Measurement -> A Measurement object containing the current pressure

Here is the call graph for this function:

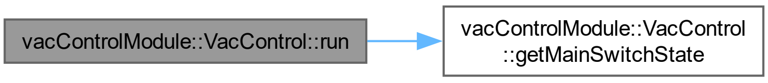

vacControlModule::VacControl ::measure
vacControlModule::VacControl ::getExternPressure

6.32.2.12 run()

```
void VacControl::run ( )
```

Runs the main control loop for the VacControl system.

This function checks the current system state and performs actions accordingly (e.g., switch states, pump control, LED control). Here is the call graph for this function:


vacControlModule::VacControl::run
vacControlModule::VacControl ::getMainSwitchState

6.32.2.13 setExternPressure()

```
void VacControl::setExternPressure (
             float pressure )
```

Sets the external pressure value.

Parameters

pressure	The external pressure value to set

6.32.2.14 setExternPump()

```
void VacControl::setExternPump (
             int state )
```

Function to get the desired PinMode from HAS.

Parameters

state	-> the state to change the PinMode

6.32.2.15 setExternScenario()

```
void VacControl::setExternScenario (
             int pressure )
```

Function to set an external scenario, typically from remote input.

Parameters

pressure	The external scenario pressure value

6.32.2.16 setPump()

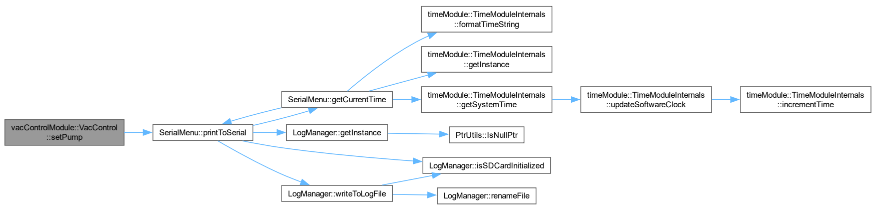
```
void VacControl::setPump (
             bool flag )
```

Set the Pump flag.

Parameters

flag	This is the boolean flag to set

Here is the call graph for this function:


timeModule::TimeModuleInternals ::formatTimeString
timeModule::TimeModuleInternals ::getInstance
SerialMenu::getCurrentTime
timeModule::TimeModuleInternals ::getSystemTime
timeModule::TimeModuleInternals ::updateSoftwareClock
timeModule::TimeModuleInternals ::incrementTime
vacControlModule::VacControl ::setPump
SerialMenu::printToSerial
LogManager::getInstance
PtrUtils::IsNullPtr
LogManager::isSDCardInitialized
LogManager::writeToLogFile
LogManager::renameFile

6.32.2.17 setVacuumLed()

```
void VacControl::setVacuumLed (
            float pressure,
            float targetPressure )
```

Controls the vacuum LED based on the current and target pressures.

Parameters

pressure	The current pressure in the system
targetPressure	The target pressure to reach

The documentation for this class was generated from the following files:

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.cpp

Chapter 7

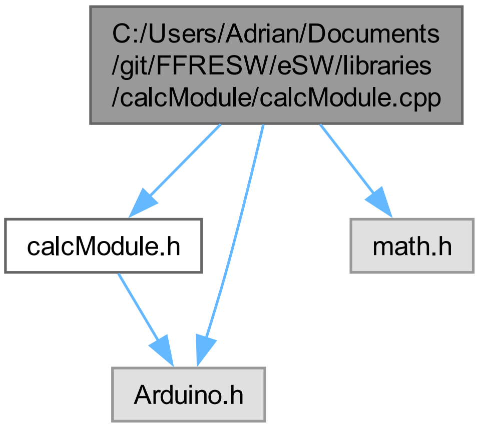
File Documentation

7.1 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calc↵ Module/calcModule.cpp File Reference

Implementation of the calcModule class.

```
#include "calcModule.h"
#include <Arduino.h>
#include <math.h>
```

Include dependency graph for calcModule.cpp:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /calcModule/calcModule.cpp
calcModule.h
math.h
Arduino.h

7.1.1 Detailed Description

Implementation of the calcModule class.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

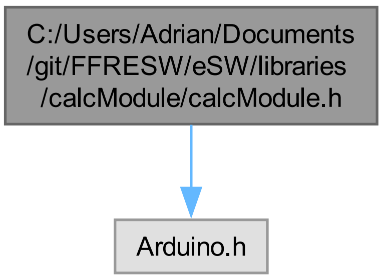
Copyright

Copyright (c) 2024

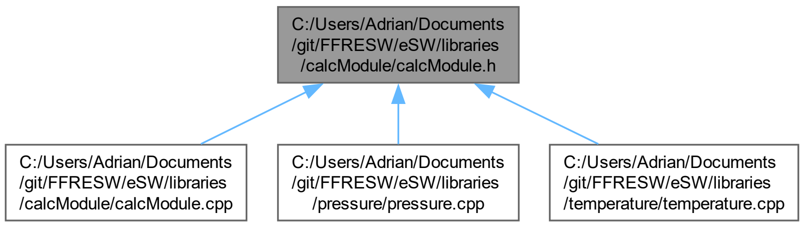
7.2 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calc↩ Module/calcModule.h File Reference

Header file for the calculation module handling sensor data.

`#include <Arduino.h>`
Include dependency graph for calcModule.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/calcModule/calcModule.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/calcModule/calcModule.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/pressure/pressure.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/temperature/temperature.cpp

Classes

- class calcModule::CalcModuleInternals

Namespaces

- namespace calcModule

  Namespace for the calculation module.

Enumerations

- enum calcModule::Type { **General** , **Pressure** , **Position** }

  Enum for the different Types we want to extract from a response.
- enum class calcModule::PressureUnit { **Pascal** , **Atmosphere** , **Psi** , **Bar** }

  Enum class for different Pressure Units aviable.

7.2.1 Detailed Description

Header file for the calculation module handling sensor data.

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.3 calcModule.h

Go to the documentation of this file.

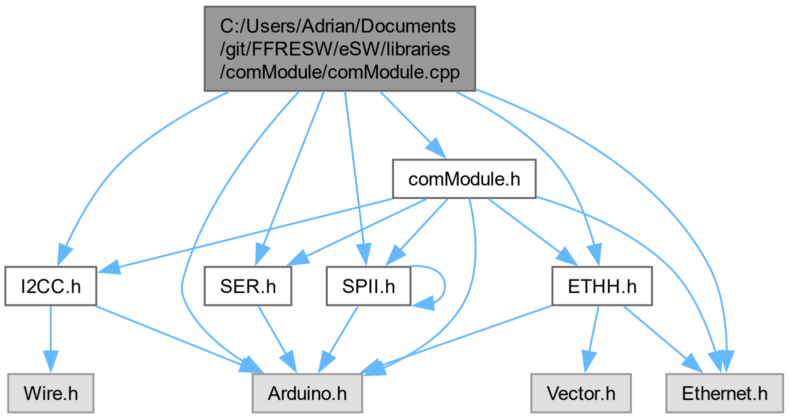
```

#ifndef CALCMODULE_H
#define CALCMODULE_H

#include <Arduino.h>

namespace calcModule
{
    enum Type
    {
        General,
        Pressure,
        Position
    };

    enum class PressureUnit
    {
        Pascal,
        Atmosphere,
        Psi,
        Bar
    };

    class CalcModuleInternals
    {
    public:
        CalcModuleInternals();
        ~CalcModuleInternals();

        static float calculateAverage(const float* data, int length);

        static float findMaximum(const float* data, int length);

        static float findMinimum(const float* data, int length);

        static float calculateStandardDeviation(const float* data, int length);

        static float findMedian(float* data, int length);

        static float celsiusToFahrenheit(float celsius);

        static float fahrenheitToCelsius(float fahrenheit);

        static float celsiusToKelvin(float celsius);

        static float kelvinToCelsius(float kelvin);

        static float pascalToAtm(float pascal);

        static float atmToPascal(float atm);

        static float pascalToPsi(float pascal);

        static float psiToPascal(float psi);

        static float calculatePower(float voltage, float current);

        static float calculateCurrent(float voltage, float resistance);

        static float calculateResistance(float voltage, float current);

        static float extractFloat(String response, int id);

        static float extractFloatFromResponse(const String& response, Type type);

        static float calculatePressureFromSensor(int sensorValue, PressureUnit unit =
PressureUnit::Pascal);

    private:

        static void sortArray(float* data, int length);

        static float roundToPrecision(float value, int precision);
    };
}

#endif // CALCMODULE_H
```

7.4 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/com←Module/comModule.cpp File Reference

Implementation of the comModule class that utilizes various communication protocols.

```
#include <Arduino.h>
#include <Ethernet.h>
#include <comModule.h>
#include "ETHH.h"
#include "I2CC.h"
#include "SER.h"
#include "SPII.h"
```

Include dependency graph for comModule.cpp:


C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp
comModule.h
I2CC.h
SER.h
SPII.h
ETHH.h
Wire.h
Arduino.h
Vector.h
Ethernet.h

7.4.1 Detailed Description

Implementation of the comModule class that utilizes various communication protocols.

7.5 comModule.h

```
#ifndef COMMODULE_H
#define COMMODULE_H

#include <Arduino.h>
#include <Ethernet.h>
#include "ETHH.h"
#include "I2CC.h"
#include "SER.h"
#include "SPII.h"

namespace comModule
{
    class ComModuleInternals
    {
    public:
        ComModuleInternals();
        ~ComModuleInternals();

        EthernetCommunication& getEthernet();

        I2CCommunication& getI2C();
        SPICommunication& getSPI();

        SerialCommunication& getSerial();

    private:
        EthernetCommunication eth;
        I2CCommunication i2c;
        SPICommunication spi;
        SerialCommunication ser;
    };
}

#endif // COMMODULE_H
```

7.6 ETHH.h

```
#ifndef ETHERNET_COMMUNICATION_H
#define ETHERNET_COMMUNICATION_H

#include <Arduino.h>
#include <Ethernet.h>
#include <Vector.h>

namespace comModule
{
    enum class Service : uint8_t
    {
        SET = 0x01,
        GET = 0x0B,
        SET_COMPOUND = 0x28,
        GET_COMPOUND = 0x29,
        SETGET = 0x30
    };

    enum class Compound1 : uint32_t
    {
        CONTROL_MODE = 0x0F020000,
        TARGET_POSITION = 0x11020000,
        TARGET_PRESSURE = 0x07020000,
        NOT_USED = 0x00000000
    };

    enum class Compound2 : uint32_t
    {
        ACCESS_MODE = 0x0F0B0000,
        CONTROL_MODE = 0x0F020000,
        TARGET_POSITION = 0x11020000,
        TARGET_PRESSURE = 0x07020000,
        ACTUAL_POSITION = 0x10010000,
        POSITION_STATE = 0x00100000,
        ACTUAL_PRESSURE = 0x07010000,
        TARGET_PRESSURE_USED = 0x07030000,
        WARNING_BITMAP = 0x0F300100,
        NOT_USED = 0x00000000
    };

    enum class Compound3 : uint32_t
    {
        CONTROL_MODE = 0x0F020000,
        TARGET_POSITION = 0x11020000,
        TARGET_PRESSURE = 0x07020000,
        SEPARATION = 0x00000000,
        ACCESS_MODE = 0x0F0B0000,
        ACTUAL_POSITION = 0x10010000,
        POSITION_STATE = 0x00100000,
        ACTUAL_PRESSURE = 0x07010000,
        TARGET_PRESSURE_USED = 0x07030000,
        WARNING_BITMAP = 0x0F300100,
        NOT_USED = 0x00000000
    };

    enum class Error_Codes : uint8_t
    {
        NO_ERROR = 0x00,
        WRONG_COMMAND_LENGTH = 0x0C,
        VALUE_TOO_LOW = 0x1C,
        VALUE_TOO_HIGH = 0x1D,
        RESULTING_ZERO_ADJUST_OFFSET = 0x20,
        NO_SENSOR_ENABLED = 0x21,
        WRONG_ACCESS_MODE = 0x50,
        TIMEOUT = 0x51,
        NV_MEMORY_NOT_READY = 0x6D,
        WRONG_PARAMETER_ID = 0x6E,
        PARAMETER_NOT_SETTABLE = 0x70,
        PARAMETER_NOT_READABLE = 0x71,
        WRONG_PARAMETER_INDEX = 0x73,
        WRONG_VALUE_WITHIN_RANGE = 0x76,
        NOT_ALLOWED_IN_THIS_STATE = 0x78,
        SETTING_LOCK = 0x79,
        WRONG_SERVICE = 0x7A,
        PARAMETER_NOT_ACTIVE = 0x7B,
        PARAMETER_SYSTEM_ERROR = 0x7C,
        COMMUNICATION_ERROR = 0x7D,
        UNKNOWN_SERVICE = 0x7E,
        UNEXPECTED_CHARACTER = 0x7F,
        NO_ACCESS_RIGHTS = 0x80,
        NO_ADEQUATE_HARDWARE = 0x81,
        WRONG_OBJECT_STATE = 0x82,
        NO_SLAVE_COMMAND = 0x84,
        COMMAND_TO_UNKNOWN_SLAVE = 0x85,
        COMMAND_TO_MASTER_ONLY = 0x87,
        ONLY_G_COMMAND_ALLOWED = 0x88,
        NOT_SUPPORTED = 0x89,
        FUNCTION_DISABLED = 0xA0,
        ALREADY_DONE = 0xA1
    };

    class EthernetCommunication
    {
    public:
        EthernetCommunication();
        ~EthernetCommunication();

        void beginEthernet(uint8_t* macAddress, IPAddress ip);

        void sendEthernetData(const char* endpoint, const char* data);

        void receiveEthernetData(char* buffer, size_t length);

        void handleEthernetClient();

        String getRequestedEndpoint();

        String getSpecificEndpoint(const String& jsonBody);

        void sendJsonResponse(const String& jsonBody);

        EthernetClient& getClient();

        bool isInitialized() const;

        bool getSendDataFlag() const;

        void setSendDataFlag(bool flag);

        void setCompound(Compound1 id, int index, String value);

        void setCompound(Compound2 id, int index, String value);

        void setCompound(Compound3 id, int index, String value);

        void setCompoundInternal(String compoundType, unsigned long id, int index, String value);

        String getCompound(Compound1 id, int index);

        String getCompound(Compound2 id, int index);

        String getCompound(Compound3 id, int index);

        String getCompoundInternal(String compoundType, unsigned long id, int index);

        Vector<float> getParsedCompound(Compound1 id, int index);

        Vector<float> getParsedCompound(Compound2 id, int index);

        Vector<float> getParsedCompound(Compound3 id, int index);

        Vector<float> parseCompoundResponse(String response);

        void setParameter(Compound2 id, String value);

        String getParameter(Compound2 id);

        void sendCommand(String command);

    private:
        EthernetServer server;
        EthernetClient client;
        bool ethernetInitialized = false;
        bool sendDataFlag = false;

        String floatToIEEE754(float value);

        Vector<float> parseResponse(String response);

    };
}

#endif // ETHERNET_COMMUNICATION_H


```

7.7 I2CC.h

```

#ifndef I2C_COMMUNICATION_H
#define I2C_COMMUNICATION_H

#include <Arduino.h>
#include <Wire.h>

namespace comModule
{
    class I2CCommunication
    {
    public:
        I2CCommunication();
        ~I2CCommunication();

        void beginI2C(uint8_t address);


        void beginI2CGlobal();

        void endI2C();

        void i2cWrite(uint8_t deviceAddress, uint8_t* data, size_t length);

        size_t i2cRead(uint8_t deviceAddress, uint8_t* buffer, size_t length);

        bool isInitialized() const;

    private:
        bool i2cInitialized = false;
    };
}

#endif // I2C_COMMUNICATION_H
```

7.8 SER.h

```

 #ifndef SERIAL_COMMUNICATION_H
 #define SERIAL_COMMUNICATION_H

 #include <Arduino.h>

 namespace comModule
 {
     class SerialCommunication
     {
     public:
        SerialCommunication();
        ~SerialCommunication();

         void beginSerial(long baudRate);

         void endSerial();

         void sendSerialData(const char* data);

         void receiveSerialData(char* buffer, size_t length);

         bool isInitialized() const;

     private:
         bool serInitialized = false;
```

```
    };
}

#endif // SERIAL_COMMUNICATION_H
```

7.9 SPII.h

```

#ifndef SPI_COMMUNICATION_H
#define SPI_COMMUNICATION_H

#include "SPII.h"

#include <Arduino.h>

namespace comModule
{
    class SPICommunication
    {
    public:
        SPICommunication();
        ~SPICommunication();

        void beginSPI();

        void endSPI();

        void spiWrite(uint8_t* data, size_t length);

        void spiRead(uint8_t* buffer, size_t length);

        bool isInitialized() const;

    private:
        bool spiInitialized = false;
    };
}

#endif // SPI_COMMUNICATION_H
```

7.10 config.h

```
// config.h
// FreeROTS Kernel Configuration

#ifndef CONFIG_H
#define CONFIG_H

// Enable Arduino C++ Interface
// This allows the HeliOS kernel to interact with the Arduino API
#define CONFIG_ENABLE_ARDUINO_CPP_INTERFACE

// Enable System Assertions (optional, for debugging purposes)
#define CONFIG_ENABLE_SYSTEM_ASSERT
#define CONFIG_SYSTEM_ASSERT_BEHAVIOR(file, line) __ArduinoAssert__(file, line)

// Message Queue Configuration
#define CONFIG_MESSAGE_VALUE_BYTES 0x8u  // Message queue message value size in bytes

// Task Notification Configuration
#define CONFIG_NOTIFICATION_VALUE_BYTES 0x8u  // Task notification value size in bytes

// Task Name Configuration
#define CONFIG_TASK_NAME_BYTES 0x8u  // Length of task names in bytes

// Memory Region Configuration
#define CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS 0x10u  // Number of memory blocks (16 blocks)
#define CONFIG_MEMORY_REGION_BLOCK_SIZE 0x20u  // Memory block size in bytes (32 bytes)

// Queue Configuration
#define CONFIG_QUEUE_MINIMUM_LIMIT 0x5u  // Minimum queue size limit (5 items)

// Stream Buffer Configuration
#define CONFIG_STREAM_BUFFER_BYTES 0x20u  // Stream buffer length (32 bytes)

// Task Watchdog Timer
#define CONFIG_TASK_WD_TIMER_ENABLE  // Enable watchdog timer for tasks

// Device Name Configuration
```

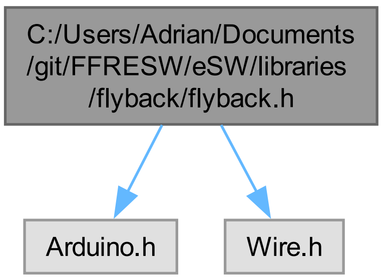
```
#define CONFIG_DEVICE_NAME_BYTES 0x8u  // Device name length (8 bytes)

#endif // CONFIG_H
```

7.11 C:/Users/Adrian/Documents/git/FFRESW/e↩ SW/libraries/flyback/flyback.h File Reference

Header for the flyback class.

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

Include dependency graph for flyback.h:



C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h
Arduino.h
Wire.h

Classes

- struct flybackModule::Measurement

  Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.
- class flybackModule::Flyback

  Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

Namespaces

- namespace flybackModule

  Namespace for the Flyback module.

Typedefs

- typedef struct flybackModule::Measurement **flybackModule::meas**

Enumerations

- enum class flybackModule::MainSwitchStates : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main↩_Switch_REMOTE** , **Main_switch_INVALID** }

  enum for different SwitchStates of HVModule
- enum class flybackModule::HVSwitchStates : int { **HV_Module_OFF** , **HV_Module_ON** }

  enum for SwitchStates of HV_Module on/off
- enum class **HVModule** : int { **powerSupply_OFF** , **powerSupply_ON** }

7.11.1 Detailed Description

Header for the flyback class.

Author

Domin

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.12 flyback.h

Go to the documentation of this file.

```

#ifndef FLYBACK_H
#define FLYBACK_H

#include <Arduino.h>
#include <Wire.h>

namespace flybackModule
{
    enum class MainSwitchStates : int
    {
        Main_Switch_OFF,
        Main_Switch_MANUAL,
        Main_Switch_REMOTE,
        Main_switch_INVALID
    };

    enum class HVSwitchStates : int
    {
        HV_Module_OFF,
        HV_Module_ON
    };

    enum class HVModule : int
    {
        powerSupply_OFF,
        powerSupply_ON
    };

```

```
typedef struct Measurement
{
    float voltage;
    float current;
    float power;
    int digitalFreqValue;
    int digitalDutyValue;
    int dutyCycle;
    uint32_t frequency;
} meas;

class Flyback
{
public:
    Flyback();
    ~Flyback();

    void initialize();

    void deinitialize();

    bool isInitialized() const;

    bool getTimerState();

    void setTimerState(bool state);

    MainSwitchStates getMainSwitchState();

    HVSwitchStates getHVSwitchState();

    HVModule getHVState();

    Measurement measure();

    void run();

    void setExternFrequency(uint32_t frequency);

    uint32_t getExternFrequency();

    void setExternDutyCycle(int dutyCycle);

    int getExternDutyCycle();

    void setExternPSU(int state);

    int getExternPSU();

    void regulateVoltage(float targetVoltage, float hysteresis);

    void setTargetVoltage(float voltage);

    float getTargetVoltage() const;

    void setHysteresis(float hysteresis);

    float getHysteresis() const;


private:
    Measurement meas;

    //Define Pins -->Inputs
    static const int Main_Switch_OFF = 27;
    static const int Main_Switch_MANUAL = 28;
    static const int Main_Switch_REMOTE = 29;
    static const int HV_Module_ON = 37;
    static const int Measure_ADC = A0;
    static const int PWM_Frequency = A1;
    static const int PWM_DutyCycle = A2;

    //Define Pins -> Outputs
    static const int PWM_OUT = 11;
    static const int PWM_INV = 12;
    static const int PSU = 36;  //Power Supply ON OFF
    static const int HV_Module_Working = 35; //Signal LED

    //Variables for calculating HV
    const float R1 = 100000000;
    const float R2 = 10000;
    const float ADC_Max_Value = 1023.0;
    const float Vcc = 5.0;

    bool _flybackInitialized;
    bool _timerInitialized;

```

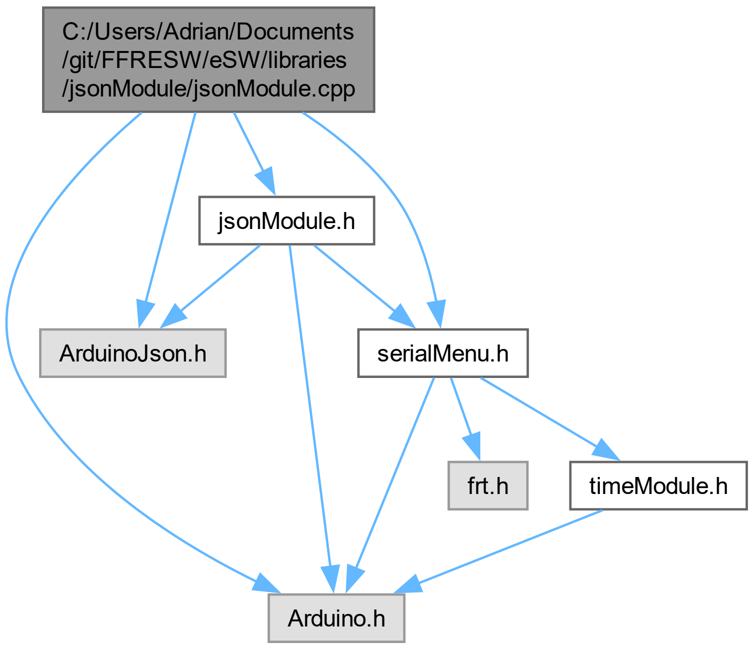
```
        // States
        static MainSwitchStates lastState;
        static bool lastTimerState;
        static int lastPWMFrequency;
        static int lastPWMDutyCycle;
        int currentPsuState = -1;

        // Default targetVoltage and hysteresisVoltage for to prevent swinging
        float _targetVoltage = 0.0f;
        float _hysteresis = 0.0f;

        // Rate limiting
        unsigned long _lastRegulationTime = 0;
        const unsigned long _regulationInterval = 100;

        // PID vars
        float _integral = 0.0f;
        float _lastError = 0.0f;
        const float _Kp = 1.5f;
        const float _Ki = 0.05f;
        const float _Kd = 0.1f;

        float _integralMin = -10.0f;
        float _integralMax = 10.0f;

        // Soft start
        bool _softStartActive = true;
        int _softStartDuty = 1;

        void timerConfig();

        void setPWMFrequency(uint32_t frequency, int dutyCycle);

        void setHVandPSU(int hvLedState, int powerSupplyState);

        void handleState(MainSwitchStates state);

        void handleOffState();

        void handleManualState();

        void handleRemoteState();

        void handleInvalidState();
    };
}

#endif //FLYBACK_H
```

7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/json↩ Module/jsonModule.cpp File Reference

Implementation of the jsonModule class.

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <jsonModule.h>
#include <serialMenu.h>
```

Include dependency graph for jsonModule.cpp:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/jsonModule/jsonModule.cpp
jsonModule.h
ArduinoJson.h
serialMenu.h
frt.h
timeModule.h
Arduino.h

7.13.1 Detailed Description

Implementation of the jsonModule class.

Version

0.2

Date

2025-05-18

Copyright

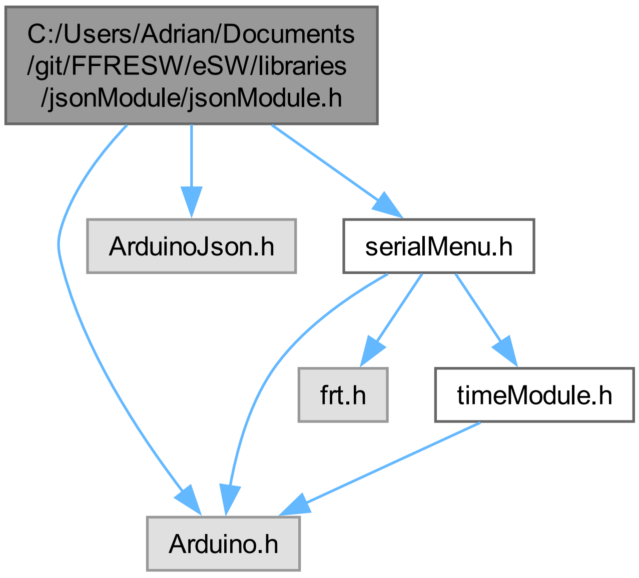
Copyright (c) 2025

7.14 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/json↩ Module/jsonModule.h File Reference

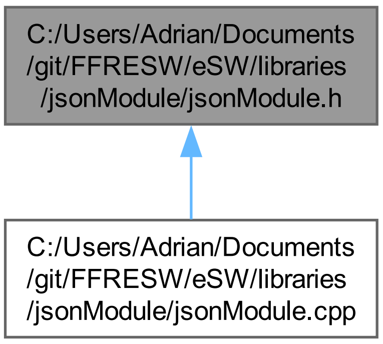
Header for the JsonModuleInternals class.

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

Include dependency graph for jsonModule.h:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /jsonModule/jsonModule.h
ArduinoJson.h
serialMenu.h
frt.h
timeModule.h
Arduino.h

This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /jsonModule/jsonModule.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /jsonModule/jsonModule.cpp

Classes

- class jsonModule::JsonModuleInternals

  Class for the JSON module internals.

Namespaces

- namespace jsonModule

  Namespace for the JSON module.

7.14.1 Detailed Description

Header for the JsonModuleInternals class.

Author

Adrian Goessl

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.15 jsonModule.h

Go to the documentation of this file.

```
#ifndef JSONMODULE_H
#define JSONMODULE_H

#include <Arduino.h>
#include <ArduinoJson.h>
#include <serialMenu.h>

namespace jsonModule
{
    class JsonModuleInternals
    {
    public:
        JsonModuleInternals();
        ~JsonModuleInternals();

        template<typename T>
        void createJson(const char* key, T value)
        {
            if (!jsonDoc[key].set(value))
            {
                SerialMenu::printToSerial(SerialMenu::OutputLevel::ERROR, F("Failed to set JSON
key."));
            }
        }

        void sendJsonSerial();

        String getJsonString() const;

        void clearJson();

        void printJsonDocMemory();

        bool hasCapacityFor(size_t additionalSize) const;

        size_t jsonBufferSize;

    private:
        StaticJsonDocument<512> jsonDoc;
    };
}

#endif // JSONMODULE_H
```

7.16 lockerBase.h

```
#ifndef LOCKER_BASE_H
#define LOCKER_BASE_H

#include <frt.h>
#include <Arduino.h>
#include <scopedLock.h>
#include <logManager.h>

class LockerBase
{
public:
    LockerBase()
    {
        if (_logger->isSDCardInitialized())
        {
            _logger->setLogFileName("log_LockerBase.txt");
        }
    }

    ~LockerBase() {}

    locker::ScopedLock lockEthernetScoped()
    {
        ethernetConnected = true;
        logState("Ethernet connected", ethernetConnected);
        return locker::ScopedLock(ethernetMutex);
    }

    locker::ScopedLock lockTemperatureScoped()
    {
        temperatureReading = true;
        logState("Temperature reading", temperatureReading);
        return locker::ScopedLock(temperaturQueueMutex);
    }

    locker::ScopedLock lockSerialScoped()
    {
        serialReading = true;
        logState("Serial reading", serialReading);
        return locker::ScopedLock(serialMutex);
    }

private:

    // Mutexes for the different resources
    frt::Mutex temperaturQueueMutex;
    frt::Mutex ethernetMutex;
    frt::Mutex serialMutex;

    // Semaphores for the different resources
    frt::Semaphore ethernetSemaphore;
    frt::Semaphore temperatureSemaphore;
    frt::Semaphore serialSemaphore;

    // Flags for the different resources
    bool ethernetConnected = false;
    bool temperatureReading = false;
    bool serialReading = false;

    // Logger instance
    LogManager* _logger = LogManager::getInstance();

    void logState(const char* label, bool state)
    {
        char buffer[64];
        snprintf(buffer, sizeof(buffer), "[LockerBase] %s: %s", label, state ? "true" : "false");
        _logger->writeToLogFile(buffer);
    }
};

#endif // LOCKER_BASE_H
```

7.17 scopedLock.h

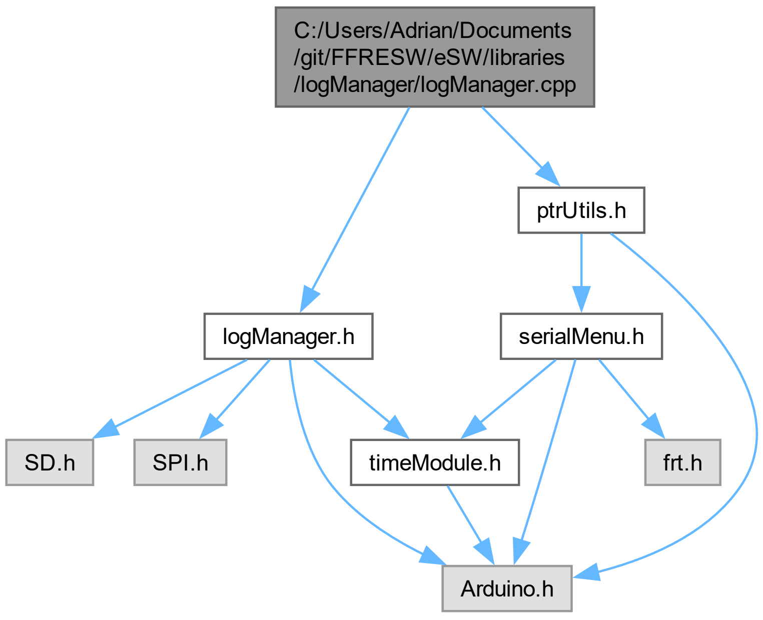
```
#ifndef LOCKER_BASE_SCOPEDLOCK_H
#define LOCKER_BASE_SCOPEDLOCK_H

#include <frt.h>

namespace locker
{
    class ScopedLock
    {
    public:
        explicit ScopedLock(frt::Mutex& mutex) : m_mutex(mutex), m_locked(true)
        {
            m_mutex.lock();
        }

        ScopedLock(const ScopedLock&) = delete;
        ScopedLock& operator=(const ScopedLock&) = delete;

        ScopedLock(ScopedLock&& other) noexcept : m_mutex(other.m_mutex), m_locked(other.m_locked)
        {
            other.m_locked = false;
        }

        // Deleted: assignment to a reference is illegal in C++
        ScopedLock& operator=(ScopedLock&&) = delete;

        ~ScopedLock()
        {
            if (m_locked)
                m_mutex.unlock();
        }

    private:
        frt::Mutex& m_mutex;
        bool m_locked;
    };
}

#endif // LOCKER_BASE_SCOPEDLOCK_H
```

7.18 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/log↩ Manager/logManager.cpp File Reference

Implementation of the logManager class.

```
#include <logManager.h>
#include <ptrUtils.h>
```

Include dependency graph for logManager.cpp:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /logManager/logManager.cpp
ptrUtils.h
logManager.h
serialMenu.h
SD.h
SPI.h
timeModule.h
frt.h
Arduino.h

Variables

- Sd2Card **card**
- SdVolume **volume**
- SdFile **root**

7.18.1 Detailed Description

Implementation of the logManager class.

Version

0.1

Date

2024-01-26

Copyright

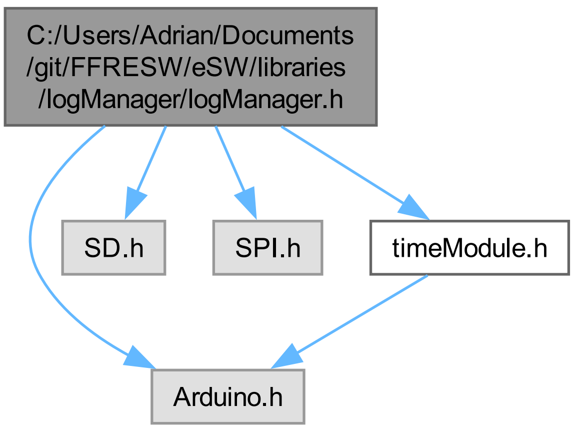
Copyright (c) 2024

7.19 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/log↩ Manager/logManager.h File Reference

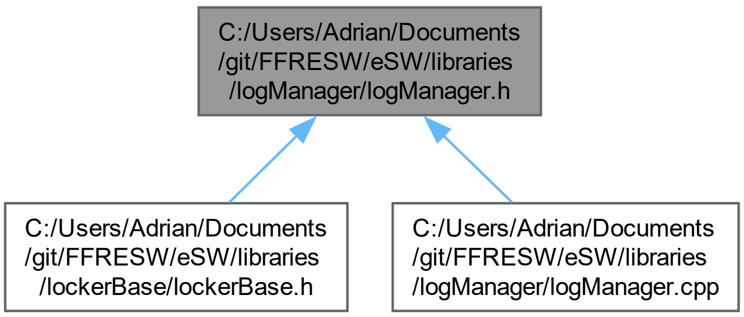
header file for the logManager.

```
#include <Arduino.h>
#include <SD.h>
#include <SPI.h>
#include <timeModule.h>
```

Include dependency graph for logManager.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/logManager/logManager.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/lockerBase/lockerBase.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/logManager/logManager.cpp

Classes

- class LogManager

7.19.1 Detailed Description

header file for the logManager.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.20 logManager.h

Go to the documentation of this file.

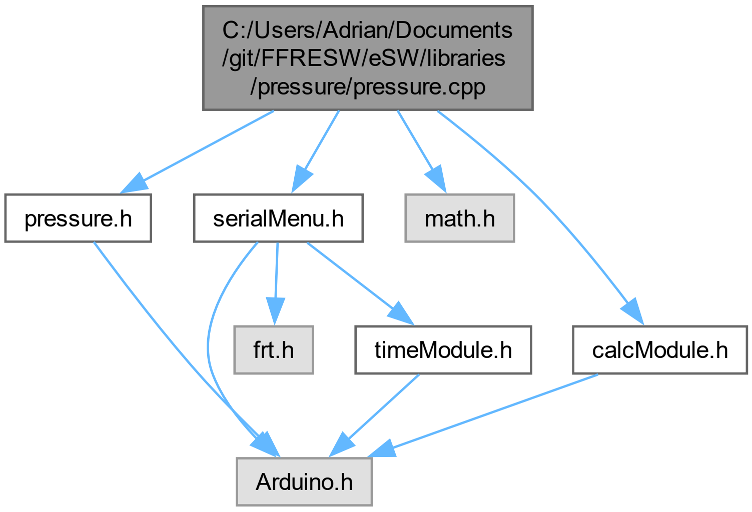
```

#ifndef LOGMANAGER_H
#define LOGMANAGER_H

#include <Arduino.h>
#include <SD.h>
#include <SPI.h>
#include <timeModule.h>

class LogManager
{
public:

    static LogManager* getInstance();

    void initSDCard(int cs);

    void shutdownSDCard();

    void flushLogs();

    bool isSDCardInitialized() const;

    static String getCurrentTime();

    void setLogFileName(const String& fileName);

    bool writeToLogFile(const String& logMessage);

    void renameFile(const String& oldName, const String& newName);

private:
    LogManager();
    ~LogManager();

    static LogManager* _instance;
    File logFile;
    bool sdCardInitialized = false;
    String logFileName;
    String baseLogFileName;

    static const int chipSelectPinEth = 10; // Default CS pin for SD card
    static const long maxLogFileSize = 104857600L; // 100MB Logfile size

    LogManager(const LogManager&) = delete;
    LogManager& operator=(const LogManager&) = delete;
};


#endif // LOGMANAGER_H
```

7.21 C:/Users/Adrian/Documents/git/FFRESW/e↩ SW/libraries/pressure/pressure.cpp File Reference

Implementation of the pressure class.

```
#include "pressure.h"
#include <serialMenu.h>
#include <math.h>
#include <calcModule.h>
```

Include dependency graph for pressure.cpp:



7.21.1 Detailed Description

Implementation of the pressure class.

Version

0.1

Date

2024-01-26

Copyright

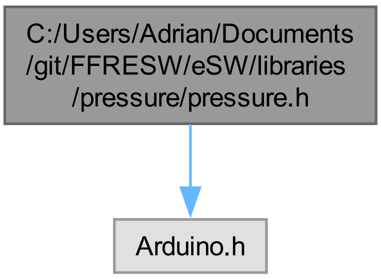
Copyright (c) 2024

7.22 C:/Users/Adrian/Documents/git/FFRESW/e↵ SW/libraries/pressure/pressure.h File Reference

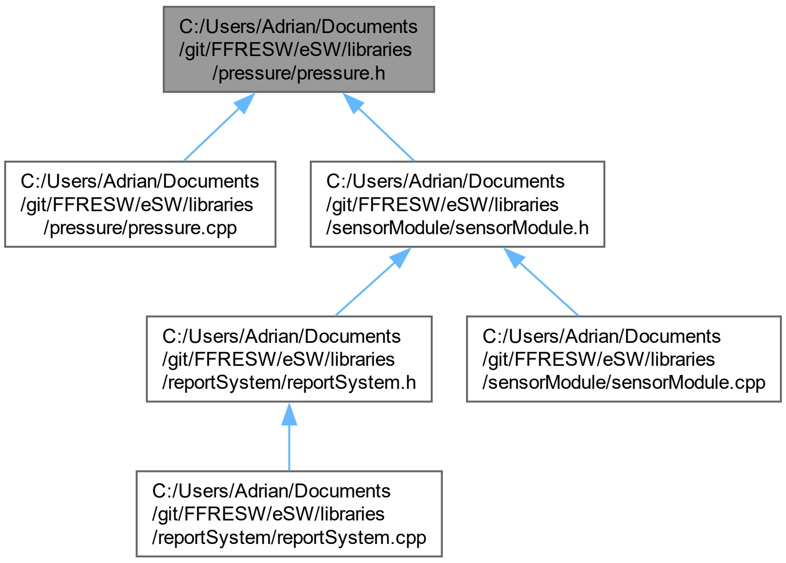
Header file for the pressure library.

`#include <Arduino.h>`

Include dependency graph for pressure.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/pressure/pressure.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/pressure/pressure.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/sensorModule/sensorModule.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/reportSystem/reportSystem.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/sensorModule/sensorModule.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/reportSystem/reportSystem.cpp

Classes

- class PressureSensor

    Pressure sensor class.

7.22.1 Detailed Description

Header file for the pressure library.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.23 pressure.h

Go to the documentation of this file.

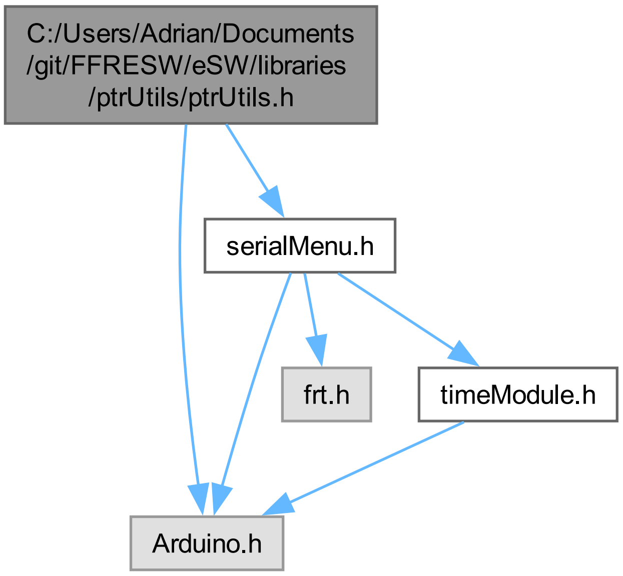
```

#ifndef PRESSURESENSOR_H
#define PRESSURESENSOR_H

#include <Arduino.h>

class PressureSensor
{
public:
    PressureSensor();
    ~PressureSensor();

    void initialize();

    float readPressure();

    bool isInitialized() const;

private:
    bool _pressureSensorInitialized;
    static const int PRESSURE_SENSOR_PIN = 0;

    float readAnalogSensor(uint8_t pin);

};

#endif // PRESSURESENSOR_H
```

7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptr↩ Utils.h File Reference

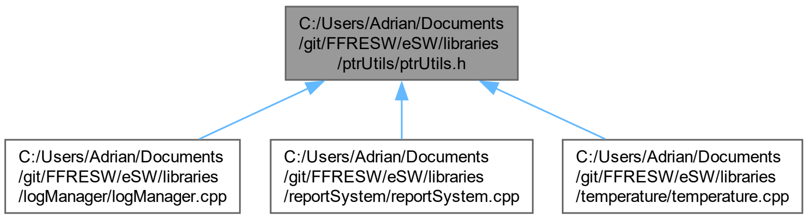
Header only library implementation.

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

Include dependency graph for ptrUtils.h:


C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h
serialMenu.h
frt.h
timeModule.h
Arduino.h

This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.cpp
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp

Classes

- class PtrUtils

  Utility class for pointer operations.
- class ScopedPointer< T >

  Template class for a Scoped Pointer.
- class PointerWrapper< T >

  Tempalte class for wrapping a pointer.
- struct Error

Macros

- #define tryDeletePtr(ptr)

    Macro to safely delete a pointer and verify it is nullptr.
- #define **TRACE_DEPTH** 4
- #define **UNIQUE_NAME**(base) __UNIQUE_NAME(base, __LINE__)

    Various macros for errorhandling.
- #define **__UNIQUE_NAME**(base, line) base##line
- #define TRY
- #define **CATCH** else if (__SAFE_ERR_PTR__ && __SAFE_ERR_PTR__->hasError())
- #define **FINALLY** else
- #define THROW(code, msg)
- #define THROW_SOFT(code, msg)
- #define RETHROW(code, msg)
- #define THROW_IF_ERR(err)
- #define THROW_IF_NULL(ptr, code, msg)
- #define THROW_IF_FALSE(cond, code, msg)
- #define RETURN_IF_ERR(expr)

Enumerations

- enum ErrorSeverity { **ERROR_NONE** = 0 , **ERROR_SOFT** = 1 , **ERROR_FATAL** = 2 }

    Enum to declare the severity of the error.

7.24.1 Detailed Description

Header only library implementation.

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.24.2 Macro Definition Documentation

7.24.2.1 RETHROW

```
#define RETHROW(
                code,
                msg )
```

Value:

```
    do {                                                    \
        if (__SAFE_ERR_PTR__)                               \
        {                                                   \
            __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__); \
            __SAFE_ERR_PTR__->code = code;                  \
            __SAFE_ERR_PTR__->msg = msg;                    \
            break;                                          \
        }                                                   \
    } while (0)
```

7.24.2.2 RETURN_IF_ERR

```
#define RETURN_IF_ERR(
                expr )
```

Value:

```
    do {                                \
        Error _e = (expr);             \
        if (_e.hasError()) return _e;   \
    } while (0)
```

7.24.2.3 THROW

```
#define THROW(
                code,
                msg )
```

Value:

```
    do {                                                                                \
        if (__SAFE_ERR_PTR__)                                                           \
        {                                                                               \
            *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_FATAL, __FILE__, __LINE__); \
            break;                                                                      \
        }                                                                               \
    } while (0)
```

7.24.2.4 THROW_IF_ERR

```
#define THROW_IF_ERR(
                err )
```

Value:

```
    do {                                                    \
        if ((err).hasError() && __SAFE_ERR_PTR__)           \
        {                                                   \
            *__SAFE_ERR_PTR__ = err;                        \
            __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__); \
            break;                                          \
        }                                                   \
    } while (0)
```

7.24.2.5 THROW_IF_FALSE

```
#define THROW_IF_FALSE(
                cond,
                code,
                msg )
```

Value:
```
    do {                                        \
        if (!(cond)) { THROW(code, msg); }      \
    } while (0)
```

7.24.2.6 THROW_IF_NULL

```
#define THROW_IF_NULL(
                ptr,
                code,
                msg )
```

Value:
```
    do {                                                  \
        if ((ptr) == nullptr) { THROW(code, msg); } \
    } while (0)
```

7.24.2.7 THROW_SOFT

```
#define THROW_SOFT(
                code,
                msg )
```

Value:
```
    do {                                                                        \
        if (__SAFE_ERR_PTR__)                                                   \
        {                                                                       \
            *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_SOFT, __FILE__, __LINE__);  \
            break;                                                              \
        }                                                                       \
    } while (0)
```

7.24.2.8 TRY

```
#define TRY
```

Value:
```
    for (Error UNIQUE_NAME(_safetry_err_) = {}, *prev = __SAFE_ERR_PTR__, *err =
      &UNIQUE_NAME(_safetry_err_);    \
         __SAFE_ERR_PTR__ = err, err != nullptr;
      \
         __SAFE_ERR_PTR__ = prev, err = nullptr)
```

7.24.2.9 tryDeletePtr

```
#define tryDeletePtr(
                ptr )
```

Value:
```
    if (PtrUtils::IsValidPtr(ptr))                      \
        SafeDelete(ptr);                                \
```

Macro to safely delete a pointer and verify it is nullptr.

7.25 ptrUtils.h

Go to the documentation of this file.

```
#ifndef PTRUTILS_H
#define PTRUTILS_H

#include <Arduino.h>
#include <serialMenu.h>

template <typename T>
static inline void SafeDelete(T*& ptr)
{
    if (ptr != nullptr)
    {
        delete ptr;
        ptr = nullptr;
    }
}

template <typename T>
static inline void SafeDeleteArray(T*& ptr)
{
    if (ptr != nullptr)
    {
        delete[] ptr;
        ptr = nullptr;
    }
}

template <typename T>
static inline void Verify(const T& value, const T& expected, const char* errorMsg = nullptr)
{
    if (value != expected)
    {
        if (errorMsg)
        {
            SerialMenu::printToSerial(errorMsg);
        }
        else
        {
            String errStr;
            errStr += "[ERROR] Verification failed: Value (";
            errStr += value;
            errStr += ") does not match expected (";
            errStr += expected;
            errStr += ").";
            SerialMenu::printToSerial(errStr);
        }
        while (true); // Halt execution
    }
}

template <typename T>
static inline void Verify(T* value, T* expected, const char* errorMsg = nullptr)
{
    if (value != expected)
    {
        if (errorMsg)
        {
            SerialMenu::printToSerial(errorMsg);
        }
        else
        {
            String errStr;
            errStr += "[ERROR] Verification failed: Pointer (";
            errStr += (unsigned long)value, HEX;
            errStr += ") does not match expected pointer (";
            errStr += (unsigned long)expected, HEX;
            errStr += ").";
            SerialMenu::printToSerial(errStr);
        }
        while (true); // Halt execution
    }
}

template <typename T>
static inline void Verify(T* value, const char* errorMsg = nullptr)
{
    if (value != nullptr)  // Directly compare with nullptr (no std::nullptr_t)
    {
        if (errorMsg)
        {
            SerialMenu::printToSerial(errorMsg);
        }
        else
        {
            String errStr;
            errStr += "[ERROR] Verification failed: Pointer (";
            errStr += (unsigned long)value, HEX;
            errStr += ") is not null.";
            SerialMenu::printToSerial(errStr);
        }
        while (true); // Halt execution
    }
}


#define tryDeletePtr(ptr)                                               \
    if (PtrUtils::IsValidPtr(ptr))                                      \
        SafeDelete(ptr);                                                \



class PtrUtils
{
public:
    template <typename T>
    static inline bool IsNullPtr(T* ptr)
    {
        return ptr == nullptr;
    }

    template <typename T>
    static inline bool IsValidPtr(T* ptr)
    {
        return ptr != nullptr;
    }
};

template <typename T>
static inline void ClearArray(T* array, size_t size)
{
    for (size_t i = 0; i < size; ++i)
    {
        array[i] = T();
    }
}

template <typename T>
static inline void PrintPtrInfo(T* ptr, const char* ptrName = "Pointer")
{
    if (ptr == nullptr)
    {
        SerialMenu::printToSerial("[INFO] " + String(ptrName) + String("is nullptr"));
    }
    else
    {
        SerialMenu::printToSerial("[INFO] " + String(ptrName) + String(" points to address: 0x") + (uintptr_t)ptr, HEX);
    }
}

template <typename T>
class ScopedPointer
{
private:
    T* ptr;

public:
    explicit ScopedPointer(T* p = nullptr) : ptr(p) {}
    ~ScopedPointer() { SafeDelete(ptr); }

    T* get() const { return ptr; }

    T* release()
    {
        T* temp = ptr;
        ptr = nullptr;
        return temp;
    }

    void reset(T* p = nullptr)
    {
        SafeDelete(ptr);
        ptr = p;
    }

    T& operator*() const { return *ptr; }

    T* operator->() const { return ptr; }
};
```

```
template <typename T>
class PointerWrapper
{
private:
    T* ptr;

public:
    explicit PointerWrapper(T* p = nullptr) : ptr(p) {}
    ~PointerWrapper() { SafeDelete(ptr); }

    T* get() const { return ptr; }

    T* release()
    {
        T* temp = ptr;
        ptr = nullptr;
        return temp;
    }

    void reset(T* p = nullptr)
    {
        SafeDelete(ptr);
        ptr = p;
    }

    T& operator*() { return *ptr; }

    T* operator->() { return ptr; }
};

#define TRACE_DEPTH 4

enum ErrorSeverity
{
    ERROR_NONE = 0,
    ERROR_SOFT = 1,
    ERROR_FATAL = 2
};

struct Error
{
    int code;
    ErrorSeverity severity;
    const char* msg;
    const char* traceFiles[TRACE_DEPTH];
    int traceLines[TRACE_DEPTH];
    int traceDepth;

    Error()
        : code(0)
        , severity(ERROR_NONE)
        , msg("")
        , traceDepth(0)
    {

    }

    Error(int c, const char* m, ErrorSeverity s, const char* file, int line)
        : code(c)
        , severity(s)
        , msg(m)
        , traceDepth(0)
    {
        addTrace(file, line);
    }

    void addTrace(const char* file, int line)
    {
        if (traceDepth < TRACE_DEPTH)
        {
            traceFiles[traceDepth] = file;
            traceLines[traceDepth] = line;
            traceDepth++;
        }
    }

    bool hasError() const
    {
        return code != 0;
    }

    bool isFatal() const
    {
        return severity == ERROR_FATAL;
    }

    void log() const
    {
        Serial.print(F("[ERROR] "));
        Serial.print(code);
        Serial.print(F(": "));
        Serial.print(msg);
        Serial.print(F(" | Severity: "));
        Serial.println(severity == ERROR_FATAL ? F("FATAL") : F("SOFT"));

        for (int i = 0; i < traceDepth; ++i)
        {
            Serial.print(F(" -> at "));
            Serial.print(traceFiles[i]);
            Serial.print(F(":"));
            Serial.println(traceLines[i]);
        }
    }
};

#define UNIQUE_NAME(base) __UNIQUE_NAME(base, __LINE__)
#define __UNIQUE_NAME(base, line) base##line

static Error* __SAFE_ERR_PTR__ = nullptr;

#define TRY
\
    for (Error UNIQUE_NAME(_safetry_err_) = {}, *prev = __SAFE_ERR_PTR__, *err =
&UNIQUE_NAME(_safetry_err_);   \
         __SAFE_ERR_PTR__ = err, err != nullptr;
\
         __SAFE_ERR_PTR__ = prev, err = nullptr)

#define CATCH \
    else if (__SAFE_ERR_PTR__ && __SAFE_ERR_PTR__->hasError())

#define FINALLY \
    else

#define THROW(code, msg)                                                            \
    do {                                                                            \
        if (__SAFE_ERR_PTR__)                                                       \
        {                                                                           \
            *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_FATAL, __FILE__, __LINE__);  \
            break;                                                                  \
        }                                                                           \
    } while (0)

#define THROW_SOFT(code, msg)                                                       \
    do {                                                                            \
        if (__SAFE_ERR_PTR__)                                                       \
        {                                                                           \
            *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_SOFT, __FILE__, __LINE__);   \
            break;                                                                  \
        }                                                                           \
    } while (0)

#define RETHROW(code, msg)                                          \
    do {                                                            \
        if (__SAFE_ERR_PTR__)                                       \
        {                                                           \
            __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__);         \
            __SAFE_ERR_PTR__->code = code;                          \
            __SAFE_ERR_PTR__->msg = msg;                            \
            break;                                                  \
        }                                                           \
    } while (0)

#define THROW_IF_ERR(err)                                           \
    do {                                                            \
        if ((err).hasError() && __SAFE_ERR_PTR__)                   \
        {                                                           \
            *__SAFE_ERR_PTR__ = err;                                \
            __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__);         \
            break;                                                  \
        }                                                           \
    } while (0)

#define THROW_IF_NULL(ptr, code, msg)                       \
    do {                                                    \
        if ((ptr) == nullptr) { THROW(code, msg); }         \
    } while (0)

#define THROW_IF_FALSE(cond, code, msg)             \
    do {                                            \
        if (!(cond)) { THROW(code, msg); }          \
    } while (0)

```

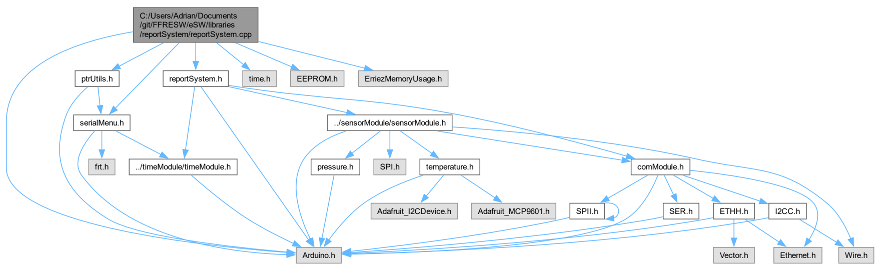
```
#define RETURN_IF_ERR(expr)                  \
    do {                                     \
        Error _e = (expr);                   \
        if (_e.hasError()) return _e;        \
    } while (0)


#endif // PTRUTILS_H
```

7.26 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/report↵System/reportSystem.cpp File Reference

Unified system health and error reporting module.

```
#include "reportSystem.h"
#include "ptrUtils.h"
#include <Arduino.h>
#include <time.h>
#include <EEPROM.h>
#include <ErriezMemoryUsage.h>
#include <serialMenu.h>
```

Include dependency graph for reportSystem.cpp:



Functions

- volatile uint16_t stackCheck **__attribute__** ((section(".noinit")))

7.26.1 Detailed Description

Unified system health and error reporting module.

Author

Adrian Goessl

Version

0.3

Date

2024-09-28

Copyright

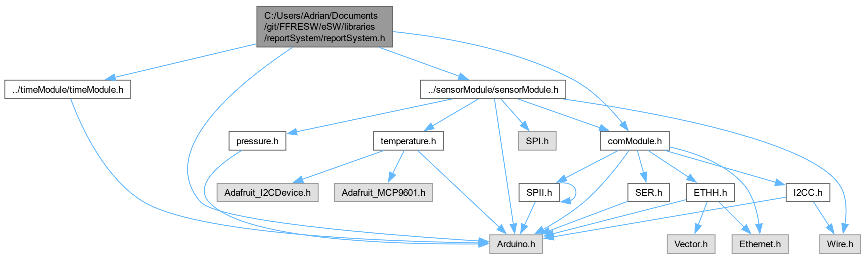
Copyright (c) 2024

7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/report↩ System/reportSystem.h File Reference

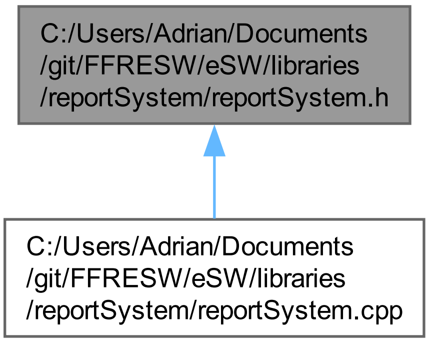
Header file for the ReportSystem.

```
#include <Arduino.h>
#include "../sensorModule/sensorModule.h"
#include "../comModule/comModule.h"
#include "../timeModule/timeModule.h"
```

Include dependency graph for reportSystem.h:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.h
../timeModule/timeModule.h
../sensorModule/sensorModule.h
pressure.h
temperature.h
SPI.h
comModule.h
Adafruit_I2CDevice.h
Adafruit_MCP9601.h
SPII.h
SER.h
ETHH.h
I2CC.h
Arduino.h
Vector.h
Ethernet.h
Wire.h

This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.cpp

Classes

- class reportSystem::ReportSystem

  Class for the report system.

Namespaces

- namespace reportSystem

  Namespace for the report system.

Macros

- #define **STACK_GUARD** 0xDEAD
- #define **EEPROM_ERROR_ADDR** 0

Variables

- volatile uint16_t **stackCheck**

7.27.1 Detailed Description

Header file for the ReportSystem.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.28 reportSystem.h

Go to the documentation of this file.

```

#ifndef REPORTSYSTEM_H
#define REPORTSYSTEM_H

#include <Arduino.h>
#include "../sensorModule/sensorModule.h"
#include "../comModule/comModule.h"
#include "../timeModule/timeModule.h"

#define STACK_GUARD 0xDEAD // Stack guard value
extern volatile uint16_t stackCheck; // Stack check variable

#define EEPROM_ERROR_ADDR 0

namespace reportSystem
{
    class ReportSystem
    {
    public:
        ReportSystem();
        ~ReportSystem();

        void reportError(const char* errorMessage);

        bool checkSystemHealth(size_t memoryThreshold, bool checkEth,
                               bool checkSpi, bool checkI2c,
                               bool checkTemp, bool checkPress);

        bool reportStatus(bool active);

        void setThreshold(float tempThreshold, float pressureThreshold);

        bool checkThresholds(float currentTemp, float currentPressure);

        String getCurrentTime();

        String getMemoryStatus();

        String getStackDump();

        void startBusyTime();

        void startIdleTime();

        float getCPULoad();

        void resetUsage();

        static void initStackGuard();

        static bool detectStackOverflow();

        void saveLastError(const char* error);

        String getLastError();

        bool getLastErrorInfo();

        bool checkRamLevel(unsigned int warningThreshold, unsigned int criticalThreshold);

        bool isTemperatureSensorOK() const;

        bool isCommunicationOK() const;

        bool isMemoryOK() const;

        bool isRamOK() const;

        bool isStackSafe() const;

        bool hasNoSavedErrors() const;

    private:
        float tempThreshold;
        float pressureThreshold;
        unsigned long lastHealthCheck;
        const unsigned long healthCheckInterval = 10000; // 10 seconds
        unsigned long busyTime = 0;
        unsigned long idleTime = 0;
        unsigned long lastTimestamp = 0;
        bool lastHealthCheckPassed = true;
        bool firstHealthCheckDone = false;

        bool checkSensors(bool checkTemp, bool checkPress);

        bool checkCommunication(bool checkEth, bool checkSpi, bool checkI2c);

        bool checkMemory(unsigned int threshold);

        sensorModule::SensorModuleInternals* _sens;
        comModule::ComModuleInternals* _com;
        timeModule::TimeModuleInternals* _time;
    };
}

#endif // REPORTSYSTEM_H
```

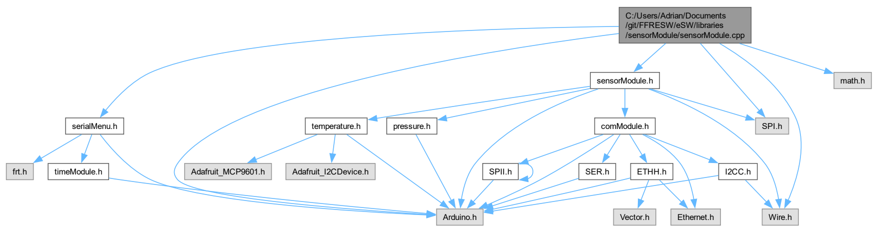
7.29 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.cpp File Reference

Implementation of the sensorModule class.

```
#include "sensorModule.h"
#include <math.h>
#include <Wire.h>
#include <SPI.h>
```

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

Include dependency graph for sensorModule.cpp:



7.29.1 Detailed Description

Implementation of the sensorModule class.

Version

0.1

Date

2024-01-26

Copyright

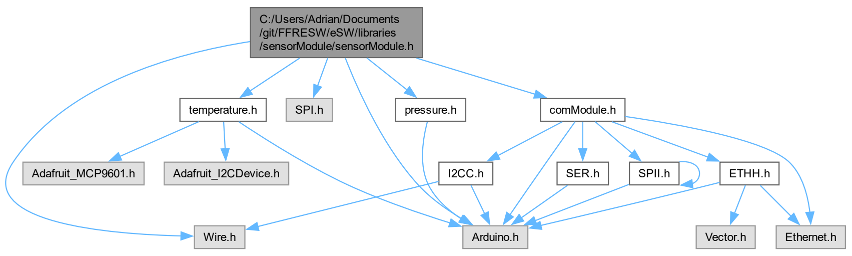
Copyright (c) 2024

7.30 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensor↩ Module/sensorModule.h File Reference

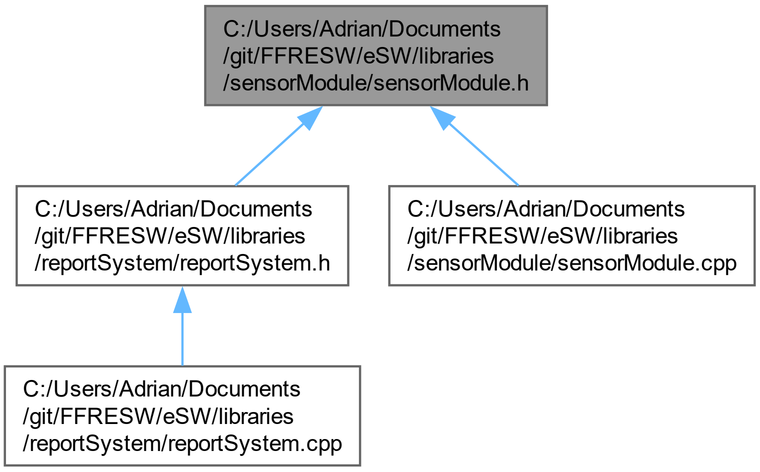
header file for the sensorModule.

```
#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <pressure.h>
#include <temperature.h>
#include <comModule.h>
```

Include dependency graph for sensorModule.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /sensorModule/sensorModule.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /sensorModule/sensorModule.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.cpp

Classes

- class sensorModule::SensorModuleInternals

  Class for the sensor module internals.

Namespaces

- namespace sensorModule

  Namespace for the sensor module.

Enumerations

- enum class sensorModule::SensorType {
  TEMPERATURE , **OBJECTTEMPERATURE** , **AMBIENTTEMPERATURE** , **PRESSURE** ,
  DHT11 , **MCP9601_Celsius_Indoor** , **MCP9601_Fahrenheit_Indoor** , **MCP9601_Kelvin_Indoor** ,
  MCP9601_Celsius_Outdoor , **MCP9601_Fahrenheit_Outdoor** , **MCP9601_Kelvin_Outdoor** , **UNKNOWN**
  }

  Enum class for the sensor types.

7.30.1 Detailed Description

header file for the sensorModule.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.31 sensorModule.h

Go to the documentation of this file.

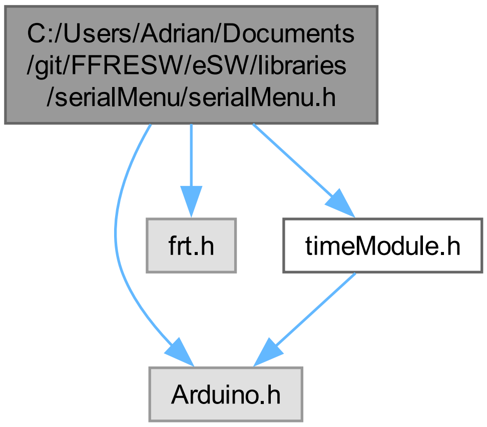
```
#ifndef SENSORMODULE_H
#define SENSORMODULE_H

#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <pressure.h>
#include <temperature.h>
#include <comModule.h>


namespace sensorModule
{
    enum class SensorType
    {
        TEMPERATURE,
        OBJECTTEMPERATURE,
        AMBIENTTEMPERATURE,
        PRESSURE,
        DHT11,
        MCP9601_Celsius_Indoor,
        MCP9601_Fahrenheit_Indoor,
        MCP9601_Kelvin_Indoor,
        MCP9601_Celsius_Outdoor,
        MCP9601_Fahrenheit_Outdoor,
        MCP9601_Kelvin_Outdoor,
        UNKNOWN
    };

    class SensorModuleInternals : public TemperatureSensor, public PressureSensor
    {
    public:
        SensorModuleInternals();
        ~SensorModuleInternals();

        void initialize();

        float readSensor(SensorType type);

        bool calibrateSensor(SensorType type);

        bool checkSensorStatus(SensorType type);

        void reportUnknownSensorOnce(SensorType type, const __FlashStringHelper* context);

    private:
        TemperatureSensor _temperatureSensor;
        PressureSensor _pressureSensor;

        SensorType _lastUnknownSensorType;
        bool _unknownSensorReported;
    };
}

#endif // SENSORMODULE_H
```

7.32 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serial↩ Menu/serialMenu.h File Reference

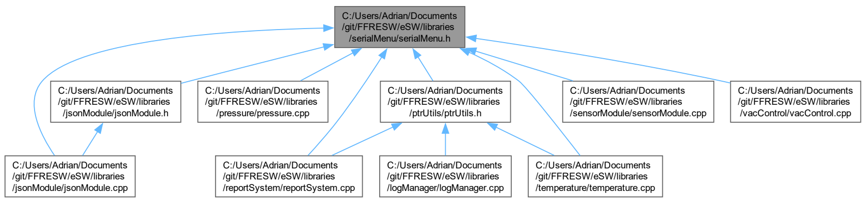
Header file for the serial menu handling serial menu interaction, logging...

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

Include dependency graph for serialMenu.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/serialMenu/serialMenu.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/jsonModule/jsonModule.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/pressure/pressure.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/ptrUtils/ptrUtils.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/sensorModule/sensorModule.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/vacControl/vacControl.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/jsonModule/jsonModule.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/reportSystem/reportSystem.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/logManager/logManager.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/temperature/temperature.cpp

Classes

- struct MenuItem

  Serial menu structure.
- class SerialMenu

  Class for the serial menu.

7.32.1 Detailed Description

Header file for the serial menu handling serial menu interaction, logging...

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.33 serialMenu.h

Go to the documentation of this file.

```
#ifndef SERIAL_MENU_H
#define SERIAL_MENU_H

#include <Arduino.h>
#include <frt.h>
#include <timeModule.h>


struct MenuItem
{
    const char* label;
    char key;
    void (*callback)();
};

class SerialMenu
{
public:

    enum class OutputLevel
    {
        DEBUG,
        INFO,
        WARNING,
        ERROR,
        CRITICAL,
        STATUS,
        PLAIN
    };

    SerialMenu();
    ~SerialMenu();

    void load(MenuItem* items, size_t size);

    void show();

    void run();

    static void printToSerial(OutputLevel level, const String& message, bool newLine = true, bool
logMessage = false);

    static void printToSerial(OutputLevel level, const __FlashStringHelper* message, bool newLine =
true, bool logMessage = false);
```

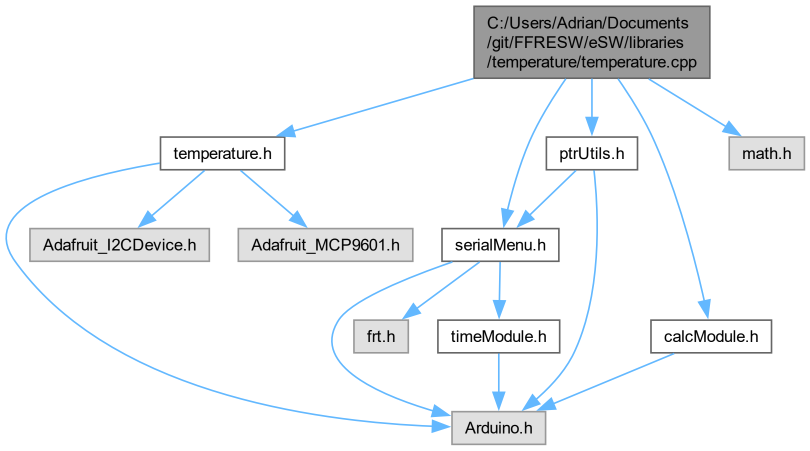
```
    static void printToSerial(const String& message, bool newLine = true, bool logMessage = false);

    static void printToSerial(const __FlashStringHelper* message, bool newLine = true, bool logMessage
= false);

    static String getCurrentTime();

private:
    MenuItem* currentMenu;
    size_t menuSize;
};

#endif // SERIAL_MENU_H
```

7.34 C:/Users/Adrian/Documents/git/FFRESW/e↩ SW/libraries/temperature/temperature.cpp File Reference

Implementation of the temperature class.

```
#include <temperature.h>
#include <serialMenu.h>
#include <ptrUtils.h>
#include <calcModule.h>
#include <math.h>
```

Include dependency graph for temperature.cpp:



7.34.1 Detailed Description

Implementation of the temperature class.

Version

0.1

Date

2024-01-26

Copyright

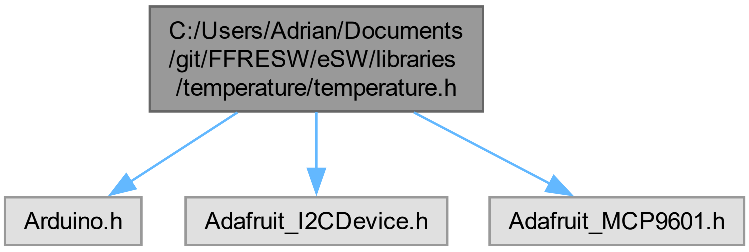
Copyright (c) 2024

7.35 C:/Users/Adrian/Documents/git/FFRESW/e↩ SW/libraries/temperature/temperature.h File Reference

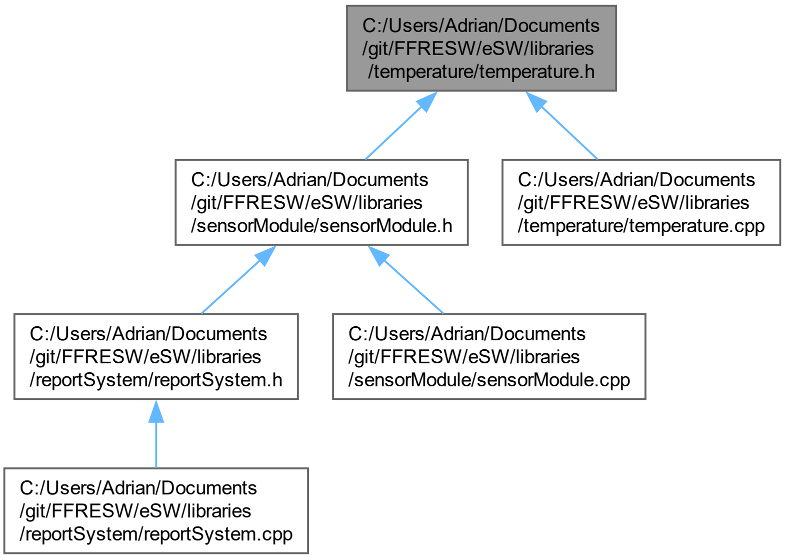
Header file for the temperature library.

```
#include <Arduino.h>
#include <Adafruit_I2CDevice.h>
#include "Adafruit_MCP9601.h"
```

Include dependency graph for temperature.h:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/temperature/temperature.h
Arduino.h
Adafruit_I2CDevice.h
Adafruit_MCP9601.h

This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/temperature/temperature.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/sensorModule/sensorModule.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/temperature/temperature.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/reportSystem/reportSystem.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/sensorModule/sensorModule.cpp
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/reportSystem/reportSystem.cpp

Classes

- class TemperatureSensor

    Temperature sensor class.

Enumerations

- enum Units { **Celsius** , **Kelvin** , **Fahrenheit** }

  Enum for different units used by mehtods as paramters.
- enum MCP9601_Status : uint8_t { **MCP9601_OPENCIRCUIT** = 0x10 , **MCP9601_SHORTCIRCUIT** = 0x20 }

  Enum for the different status codes of the MCP9601 sensor.
- enum **SensorID** { **INDOOR** , **OUTDOOR** }

7.35.1 Detailed Description

Header file for the temperature library.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.36 temperature.h

Go to the documentation of this file.

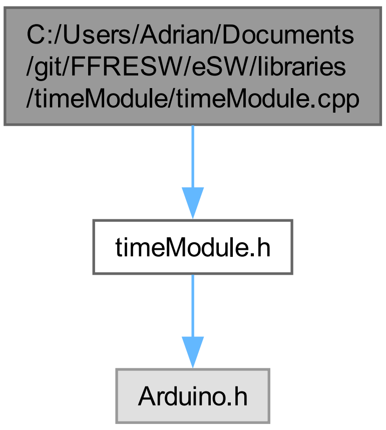
```
#ifndef TEMPERATURESENSOR_H
#define TEMPERATURESENSOR_H

#include <Arduino.h>
#include <Adafruit_I2CDevice.h>
#include "Adafruit_MCP9601.h"

enum Units
{
    Celsius,
    Kelvin,
    Fahrenheit
};

enum MCP9601_Status : uint8_t
{
    MCP9601_OPENCIRCUIT = 0x10,
    MCP9601_SHORTCIRCUIT = 0x20
};

// @brief Enum for the Different Sensors in different Environments \enum SensorID
enum SensorID
{
    INDOOR, // Green Cable
    OUTDOOR // Silver Cable
};

class TemperatureSensor
{
```

```
public:
    TemperatureSensor();
    ~TemperatureSensor();

    void initialize();

    float readTemperature();

    float readMCP9601(Units unit, SensorID sensor);

    bool isInitialized() const;


    uint8_t calibMCP9601(SensorID sensor);

private:
    bool _temperatureSensorInitialized;
    static const int TEMP_SENSOR_PIN = A0;
    static const int TEMP_SENSOR_PIN_DIG = 4;
    static const int DHT11_PIN = 7;

    static const uint8_t MLX90614 = 0x5A;
    static const uint8_t AMBIENT_TEMP = 0x06;
    static const uint8_t OBJECT_TEMP = 0x07;

    // Settings for the MCP9601 Sensor board
    Adafruit_MCP9601 _mcp1;
    Adafruit_MCP9601 _mcp2;
    Ambient_Resolution _ambientREs = RES_ZERO_POINT_0625;
    static const uint8_t MCP9601_I2C = 0x67;


    float readAnalogSensor(uint8_t pin);

    float readDigitalSensor(uint8_t pin);
};

#endif // TEMPERATURESENSOR_H
```

7.37 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/time↩ Module/timeModule.cpp File Reference

Implementation of the timeModule class.

```
#include <timeModule.h>
```
Include dependency graph for timeModule.cpp:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/timeModule/timeModule.cpp
timeModule.h
Arduino.h

7.37.1 Detailed Description

Implementation of the timeModule class.

Version

0.1

Date

2024-01-26

Copyright

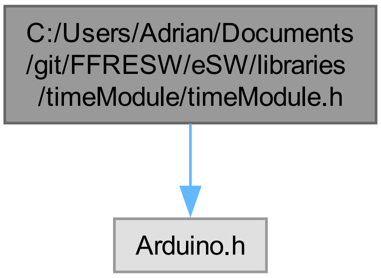
Copyright (c) 2024

7.38 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/time↩ Module/timeModule.h File Reference

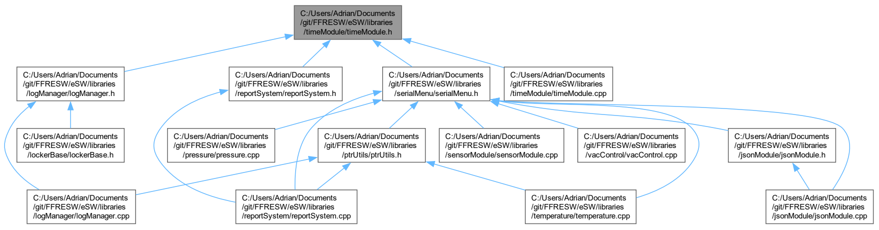
Header file for the time module handling systemtime for logging, api ...

`#include <Arduino.h>`

Include dependency graph for timeModule.h:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /timeModule/timeModule.h
Arduino.h

This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /timeModule/timeModule.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /logManager/logManager.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /serialMenu/serialMenu.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /timeModule/timeModule.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /lockerBase/lockerBase.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /pressure/pressure.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /ptrUtils/ptrUtils.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /sensorModule/sensorModule.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /vacControl/vacControl.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /jsonModule/jsonModule.h
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /logManager/logManager.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /reportSystem/reportSystem.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /temperature/temperature.cpp
C:/Users/Adrian/Documents /git/FFRESW/eSW/libraries /jsonModule/jsonModule.cpp

Classes

- struct timeModule::DateTimeStruct

  Struct to hold the date and time.
- class timeModule::TimeModuleInternals

  Class to handle Systemtime.

Namespaces

- namespace timeModule

  namespace for the timeModule

Typedefs

- typedef struct timeModule::DateTimeStruct **timeModule::DateTimeStruct**

7.38.1 Detailed Description

Header file for the time module handling systemtime for logging, api ...

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.39 timeModule.h

Go to the documentation of this file.

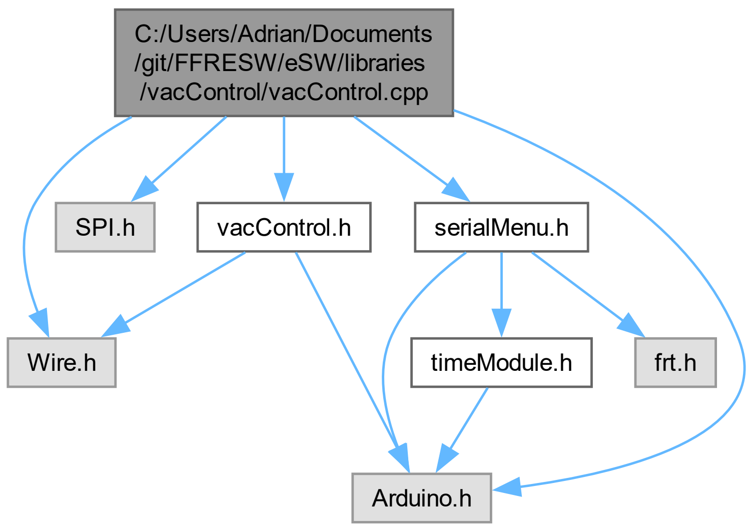
```

#ifndef TIMEMODULE_H
#define TIMEMODULE_H

#include <Arduino.h>

namespace timeModule
{
    typedef struct DateTimeStruct
    {
        int year;
        int month;
        int day;
        int hour;
        int minute;
        int second;
    } DateTimeStruct;

    class TimeModuleInternals
    {
    public:
        TimeModuleInternals();
        ~TimeModuleInternals();

        static void incrementTime(DateTimeStruct *dt);

        static String formatTimeString(const DateTimeStruct &dt);

        bool setTimeFromHas(const String& timeString);

        void setSystemTime(const DateTimeStruct& dt);

        void updateSoftwareClock();

        DateTimeStruct getSystemTime();

        static TimeModuleInternals* getInstance();

    private:
        DateTimeStruct dt;
        unsigned long startMillis = 0;
    };
}

#endif // TIMEMODULE
```

7.40 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vac←Control/vacControl.cpp File Reference

Implementation of the vacControl class.

```
#include <Wire.h>
#include <SPI.h>
#include <Arduino.h>
#include <vacControl.h>
#include <serialMenu.h>
```

Include dependency graph for vacControl.cpp:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/vacControl/vacControl.cpp
SPI.h
vacControl.h
serialMenu.h
Wire.h
timeModule.h
frt.h
Arduino.h

7.40.1 Detailed Description

Implementation of the vacControl class.

Version

0.1

Date

2024-01-26

Copyright

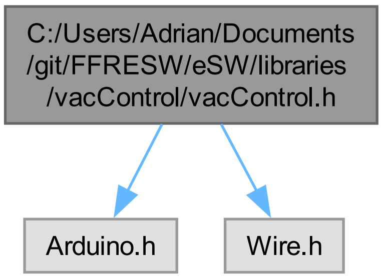
Copyright (c) 2024

7.41 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vac←Control/vacControl.h File Reference

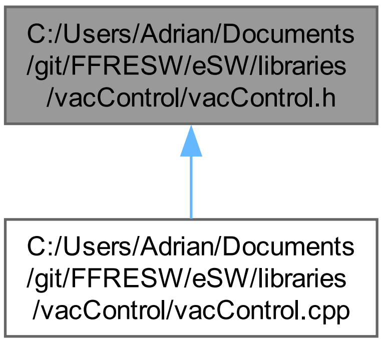
Header for the vacControl class.

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

Include dependency graph for vacControl.h:



This graph shows which files directly or indirectly include this file:


C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/vacControl/vacControl.h
C:/Users/Adrian/Documents
/git/FFRESW/eSW/libraries
/vacControl/vacControl.cpp

Classes

- struct vacControlModule::Pressure
- class vacControlModule::VacControl

  VacControl class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

Namespaces

- namespace vacControlModule

  Namespace for the VacControl module.

Typedefs

- typedef struct vacControlModule::Pressure **vacControlModule::meas**

Enumerations

- enum class vacControlModule::MainSwitchStates : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }

  Enum to represent the states of the main switch and pump.
- enum class vacControlModule::PumpSwitchStates : int { **Switch_Pump_OFF** , **Switch_Pump_ON** }

  Enum for extern Setup for pinMode for the PumpState.
- enum class vacControlModule::PumpState : int { **pump_OFF** , **pump_ON** }

  Enum for extern Setup for pinMode for the PumpState.
- enum vacControlModule::Scenarios {
  Scenario_1 = 0 , **Scenario_2** = 1 , **Scenario_3** = 2 , **Scenario_4** = 3 ,
  Scenario_5 = 4 , **Invalid_Scenario** = -1 }

  Enum to represent the different operating scenarios of the VacControl system.

7.41.1 Detailed Description

Header for the vacControl class.

Author

Domin

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.42 vacControl.h

Go to the documentation of this file.

```

#ifndef VACCONTROL_H
#define VACCONTROL_H

#include <Arduino.h>
#include <Wire.h>


namespace vacControlModule
{
    enum class MainSwitchStates : int
    {
        Main_Switch_OFF,
        Main_Switch_MANUAL,
        Main_Switch_REMOTE,
        Main_switch_INVALID
    };

    enum class PumpSwitchStates : int
    {
        Switch_Pump_OFF,
        Switch_Pump_ON
    };

    enum class PumpState : int
    {
        pump_OFF,
        pump_ON
    };

    typedef struct Pressure
    {
        float pressure;

    } meas;

    enum Scenarios
    {
        Scenario_1 = 0,
        Scenario_2 = 1,
        Scenario_3 = 2,
        Scenario_4 = 3,
        Scenario_5 = 4,
        Invalid_Scenario = -1
    };

    class VacControl
    {
    public:

        VacControl();
        ~VacControl();

        void initialize();

        void deinitialize();

        bool isInitialized() const;

        MainSwitchStates getMainSwitchState();

        PumpSwitchStates getPumpSwitchState();

        PumpState getPumpState();

        Scenarios getScenario();

        Pressure measure();

        void setVacuumLed(float pressure, float targetPressure);

        int getScenarioFromPotValue(int potValue);

        void setPump(bool flag);

        void run();

        void setExternScenario(int pressure);

        int getExternScenario();

        void externPump(int pumpState);

        void setExternPump(int state);

        int getExternPump();

        void setExternPressure(float pressure);

        float getExternPressure();


    private:
        Pressure meas;

        //Define Pins --> Input
        static const int Main_Switch_OFF = 27;      //Main_Switch OFF Mode 27
        static const int Main_Switch_MANUAL = 28;   //Main_Switch Manual Mode 28
        static const int Main_Switch_REMOTE = 29;   //Main_Switch Remote Mode 29
        static const int Switch_Pump_ON = 23;       //Button to turn Pump ON 23
        static const int targetPressure = A3;       //Potentiometer for Regulation

        //Define Pins --> Output
        static const int Pump_Status_LED = 24;      //OUTPUT to see State off Pump
        static const int Pump_Relay = 25;           //OUTPUT to turn on/off Relais
        static const int targetVacuumLED = 26;      //OUTPUT to see Vacuum reached

        //Variables to save Values
        int currentScenario = -1;
        int currentPumpState = -1;

        //Variables for TargetPressure
        static const float TARGET_PRESSURE_1 = 1;
        static const float TARGET_PRESSURE_2 = 0.8f;
        static const float TARGET_PRESSURE_3 = 0.5;
        static const float TARGET_PRESSURE_4 = 0.01f;

        static int lastState;
        static int lastPumpState;

        bool _vacControlInitialized;

        void handleOffMode();

        void handleManualMode();

        void handleRemoteMode();

        void handleInvalidMode();
    };
}

#endif //VACCONTROL_H
```

Index