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 SwitchStates : int { **HV_Module_OFF** , **HV_Module_MANUAL** , **HV_Module_REMOTE** , **HV_↩ Module_INVALID** }

  enum for different SwitchStates of HVModule

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 SwitchStates : int {
  Main_Switch_OFF , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** ,
  PUMP_ON , **PUMP_OFF** }

  Enum to represent the states of the main switch and pump.
- 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.

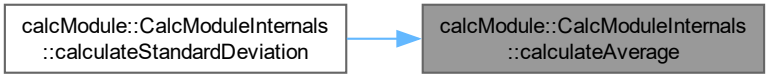
Parameters

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:



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.

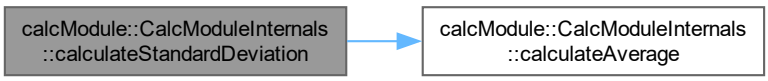
Parameters

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:



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:

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:

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()

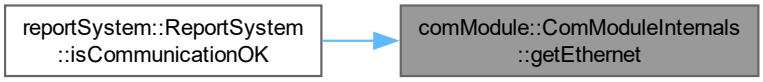
```
EthernetCommunication & ComModuleInternals::getEthernet ( )
```

Get the Ethernet object.

Returns

EthernetCommunication&

Here is the caller graph for this function:



6.3.1.2 getI2C()

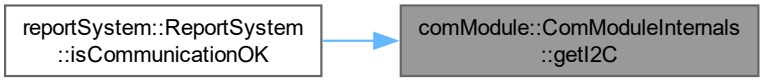
```
I2CCommunication & ComModuleInternals::getI2C ( )
```

Get the I2C object.

Returns

I2CCommunication&

Here is the caller graph for this function:



6.3.1.3 getSerial()

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

Get the Serial object.

Returns

SerialCommunication&

6.3.1.4 getSPI()

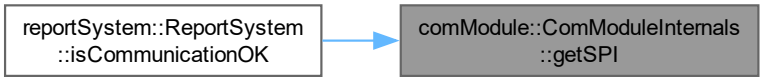
```
SPICommunication & ComModuleInternals::getSPI ( )
```

Get the SPI object.

Returns

SPICommunication&

Here is the caller graph for this function:



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 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.

6.6.1 Detailed Description

Class to handle Ethernet communication.

6.6.2 Member Function Documentation

6.6.2.1 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

6.6.2.2 getClient()

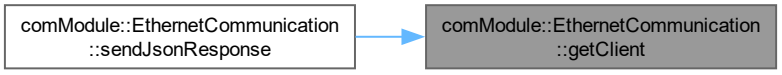
```
EthernetClient & EthernetCommunication::getClient ( )
```

Get the currently active Ethernet client.

Returns

EthernetClient& , Reference to the active Ethernet client

Here is the caller graph for this function:



6.6.2.3 getCompound() [1/3]

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

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

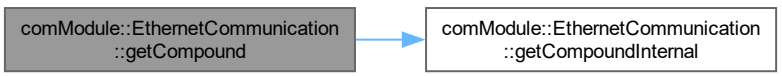
Parameters

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

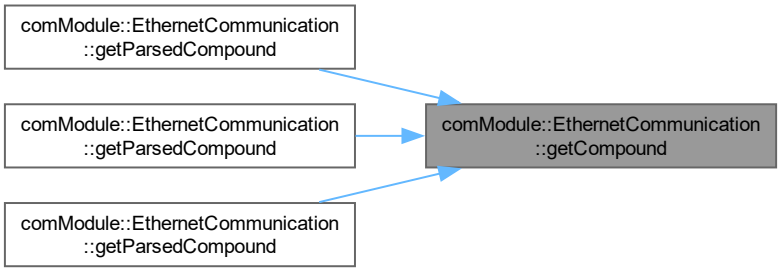
Returns

String -> Response from the valve uC slave

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.4 getCompound() [2/3]

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

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

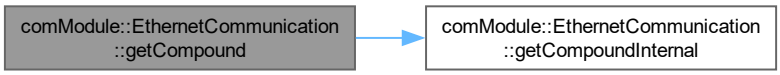
Parameters

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:



6.6.2.5 getCompound() [3/3]

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

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

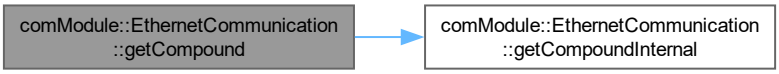
Parameters

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:



6.6.2.6 getCompoundInternal()

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

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

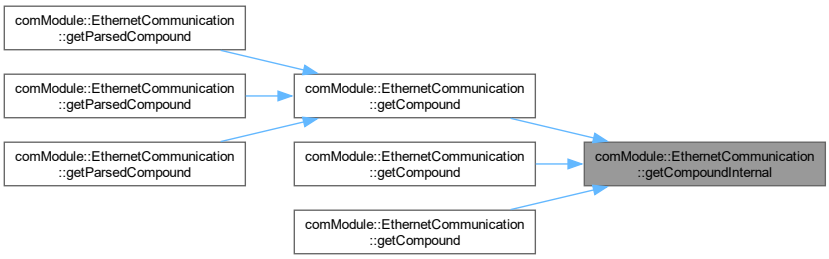
Parameters

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:



6.6.2.7 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.

6.6.2.8 getParsedCompound() [1/3]

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

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

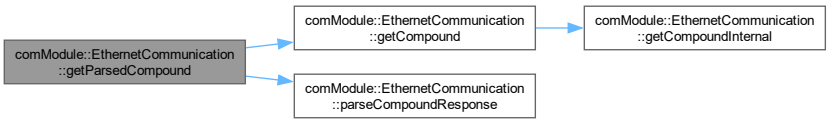
Parameters

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:



6.6.2.9 getParsedCompound() [2/3]

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

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

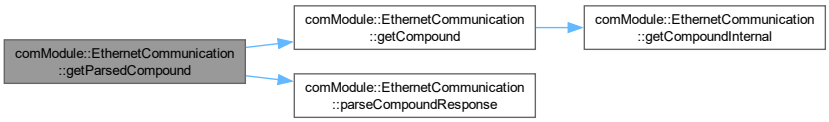
Parameters

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:



6.6.2.10 getParsedCompound() [3/3]

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

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

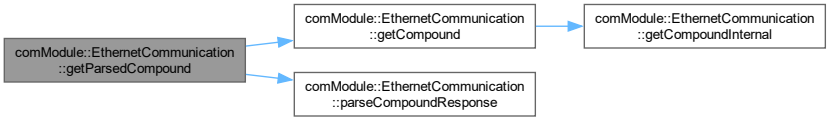
Parameters

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:



6.6.2.11 getRequestedEndpoint()

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

Function to get the requested endpoint.

Returns

String -> The requested endpoint

6.6.2.12 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

6.6.2.13 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

6.6.2.14 isInitialized()

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

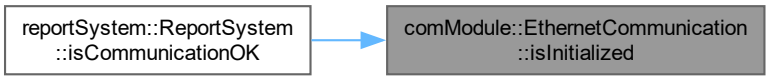
Function to check if the Ethernet communication is initialized.

Returns

true -> if the Ethernet communication is initialized

false -> if the Ethernet communication is not initialized

Here is the caller graph for this function:



6.6.2.15 parseCompoundResponse()

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

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

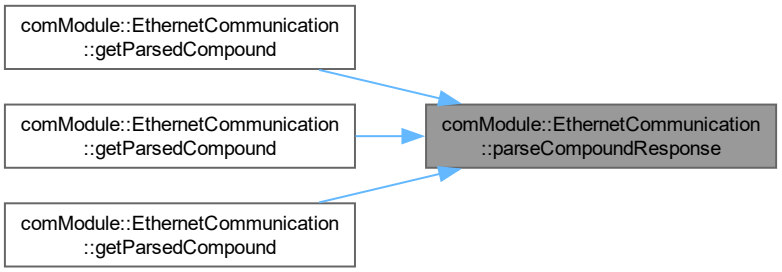
Parameters

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

Returns

Vector<float> -> containing parsed float values.

Here is the caller graph for this function:



6.6.2.16 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

6.6.2.17 sendCommand()

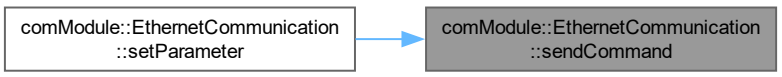
```
void EthernetCommunication::sendCommand (
            String command )
```

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:



6.6.2.18 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

6.6.2.19 sendJsonResponse()

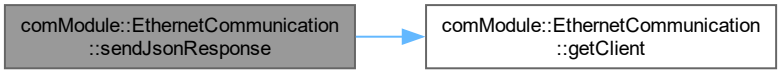
```
void EthernetCommunication::sendJsonResponse (
            const String & jsonBody )
```

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:



6.6.2.20 setCompound() [1/3]

```
void EthernetCommunication::setCompound (
            Compound1 id,
```

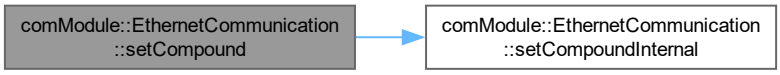
```
            int index,
            String value )
```

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:



6.6.2.21 setCompound() [2/3]

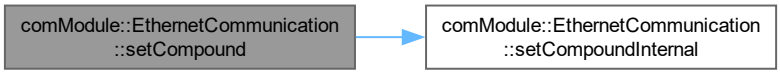
```
void EthernetCommunication::setCompound (
            Compound2 id,
            int index,
            String value )
```

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:



6.6.2.22 setCompound() [3/3]

```
void EthernetCommunication::setCompound (
            Compound3 id,
```

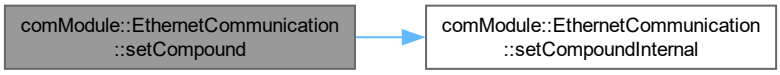
```
            int index,
            String value )
```

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:



6.6.2.23 setCompoundInternal()

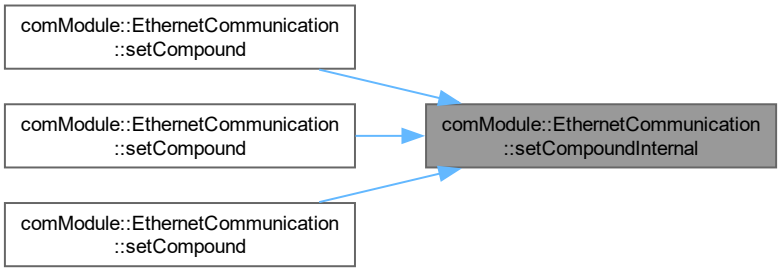
```
void EthernetCommunication::setCompoundInternal (
            String compoundType,
            unsigned long id,
            int index,
            String value )
```

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:



6.6.2.24 setParameter()

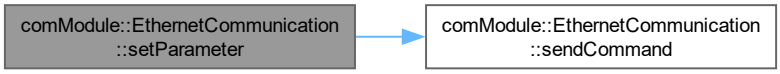
```
void EthernetCommunication::setParameter (
            Compound2 id,
            String value )
```

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:



6.6.2.25 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

6.7 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.
- SwitchStates getSwitchState ()

  Get the state of the Main-Switch.
- 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** (bool state)
- bool **getExternPsu** ()
- 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.

6.7.1 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.

6.7.2 Member Function Documentation

6.7.2.1 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.

6.7.2.2 getSwitchState()

```
SwitchStates Flyback::getSwitchState ( )
```

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")

6.7.2.3 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.

6.7.2.4 getTimerState()

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

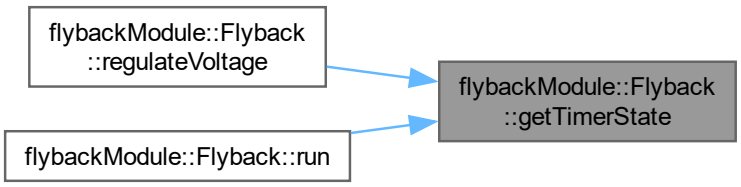
Returns the state of the timer.

Returns

true -> if the timer is initialized

false -> if the timer is not initialized

Here is the caller graph for this function:



6.7.2.5 isInitialized()

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

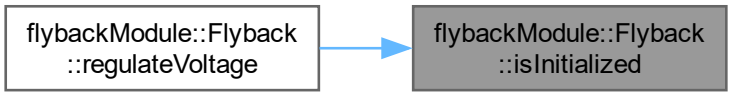
Get the state of the Flyback system.

Returns

true -> Flyback is initialized

false -> Flyback is not initialized

Here is the caller graph for this function:



6.7.2.6 measure()

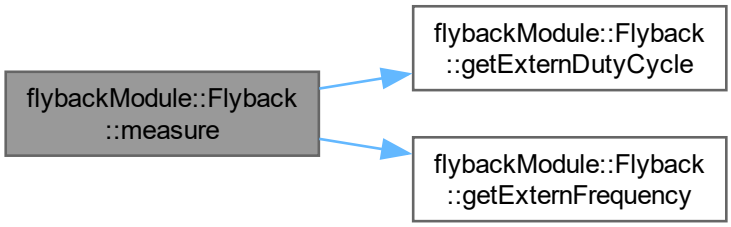
```
Measurement Flyback::measure ( )
```

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

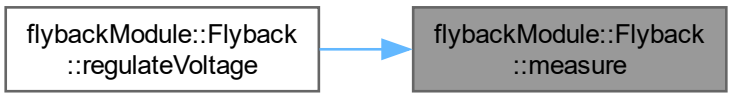
Returns

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.2.7 regulateVoltage()

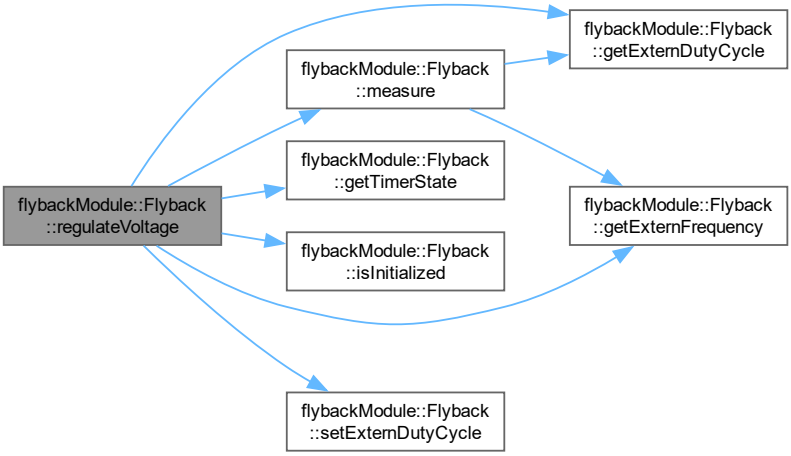
```
void Flyback::regulateVoltage (
            float targetVoltage,
            float hysteresis )
```

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:



6.7.2.8 setExternDutyCycle()

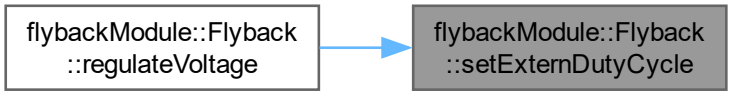
```
void Flyback::setExternDutyCycle (
            int dutyCycle )
```

Function to get the desired DutyCycle from HAS.

Parameters

dutyCycle	-> the dutyCycle to change

Here is the caller graph for this function:



6.7.2.9 setExternFrequency()

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

Function to get the desired Frequency from HAS.

Parameters

frequency	-> the frequency to change

6.7.2.10 setHysteresis()

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

Setter Function for the HAS to regulate the hysteresis.

Parameters

hysteresis	-> The desired hysteresis.

6.7.2.11 setTargetVoltage()

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

Setter Function for the HAS to regulate the target Voltage.

Parameters

voltage	-> The desired Voltage to reach.

6.7.2.12 setTimerState()

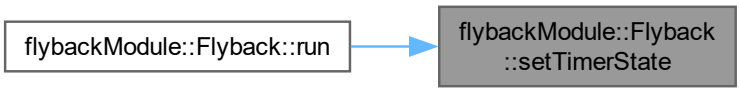
```
void Flyback::setTimerState (
            bool state )
```

Sets the state of the timer.

Parameters

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

Here is the caller graph for this function:



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

6.8 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.

6.8.1 Detailed Description

Class to handle I2C communication.

6.8.2 Member Function Documentation

6.8.2.1 beginI2C()

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

Function to initialize the I2C communication.

Parameters

address	-> The address of the I2C device

6.8.2.2 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

6.8.2.3 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

6.8.2.4 isInitialized()

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

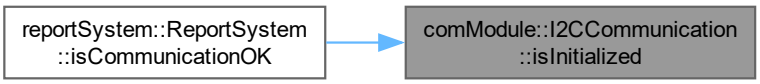
Function to check if the I2C communication is initialized.

Returns

true -> if the I2C communication is initialized

false -> if the I2C communication is not initialized

Here is the caller graph for this function:



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

6.9 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**

6.9.1 Detailed Description

Class for the JSON module internals.

6.9.2 Member Function Documentation

6.9.2.1 createJson()

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

Add a key-value pair to the Json document.

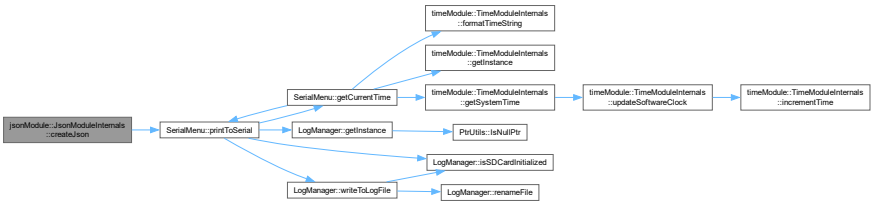
Template Parameters

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:



6.9.2.2 getJsonString()

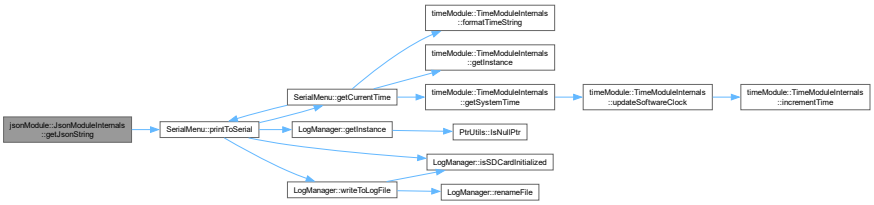
```
String JsonModuleInternals::getJsonString ( ) const
```

Get the Json String object.

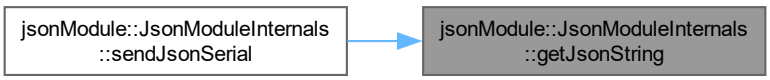
Returns

String -> The Json String object.

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.2.3 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

6.10 LockerBase Class Reference

Base class for the locker system.

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

Public Member Functions

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

6.10.1 Detailed Description

Base class for the locker system.

6.10.2 Member Function Documentation

6.10.2.1 lockEthernetScoped()

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

Returns

locker::ScopedLock

6.10.2.2 lockSerialScoped()

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

Returns

locker::ScopedLock

6.10.2.3 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

6.11 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.

6.11.1 Member Function Documentation

6.11.1.1 getCurrentTime()

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

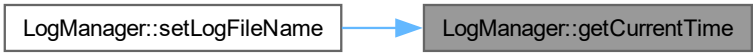
Getter for the current time.

Returns

The current time as a String

Here is the call graph for this function:

Here is the caller graph for this function:



6.11.1.2 getInstance()

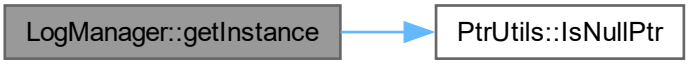
```
LogManager * LogManager::getInstance ( )  [static]
```

Get the Instance object.

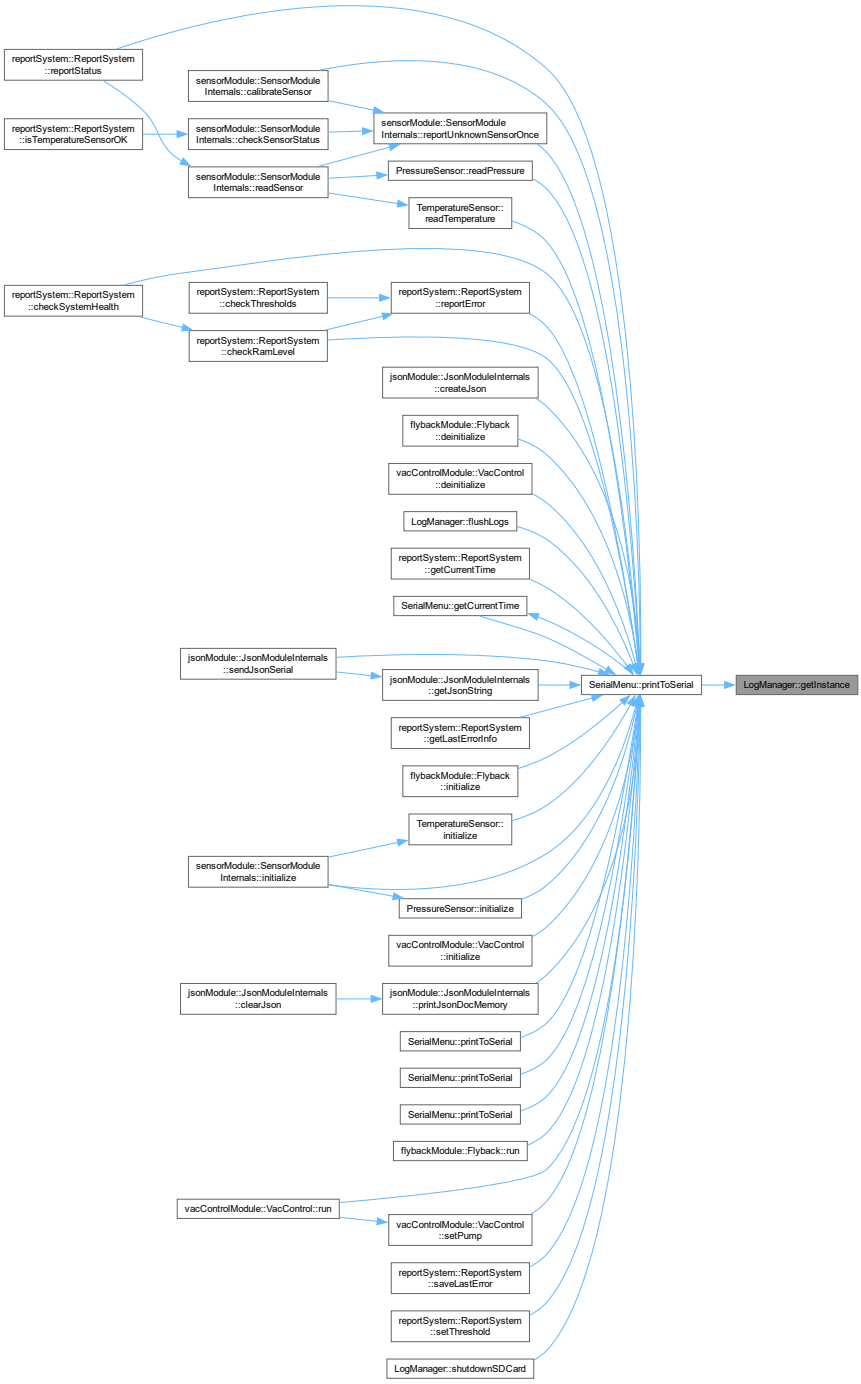
Returns

LogManager*

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.1.3 initSDCard()

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

Function to initialize the SD card.

Parameters

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

6.11.1.4 isSDCardInitialized()

```
bool LogManager::isSDCardInitialized ( ) const
```

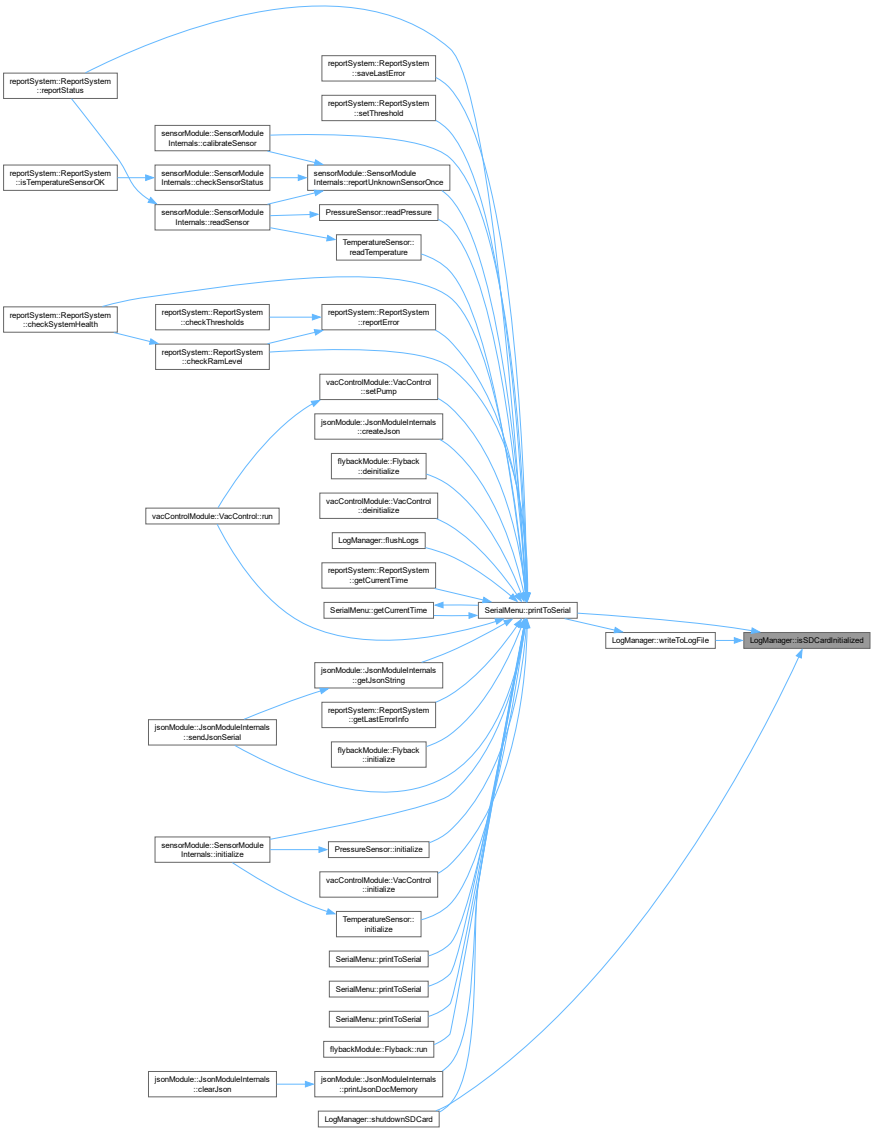
Function to check if the SD card is initialized.

Returns

true -> if the SD card is initialized

false -> if the SD card is not initialized

Here is the caller graph for this function:



6.11.1.5 renameFile()

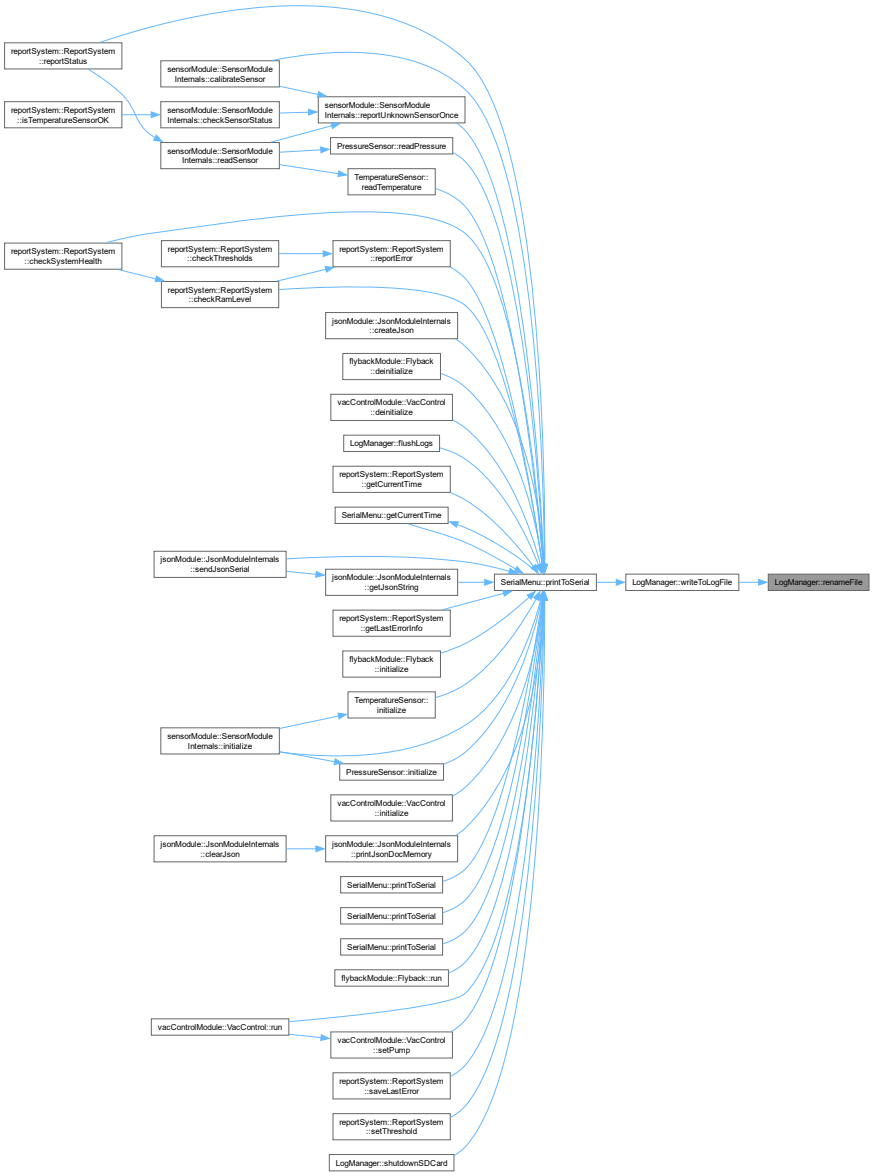
```
void LogManager::renameFile (
            const String & oldName,
            const String & newName )
```

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:



6.11.1.6 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:

6.11.1.7 writeToLogFile()

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

Function to write a log message to the log file.

Parameters

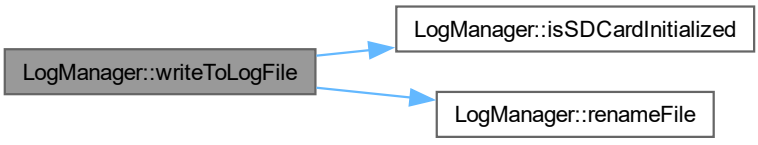
logMessage	-> The log message to write to the log file.

Returns

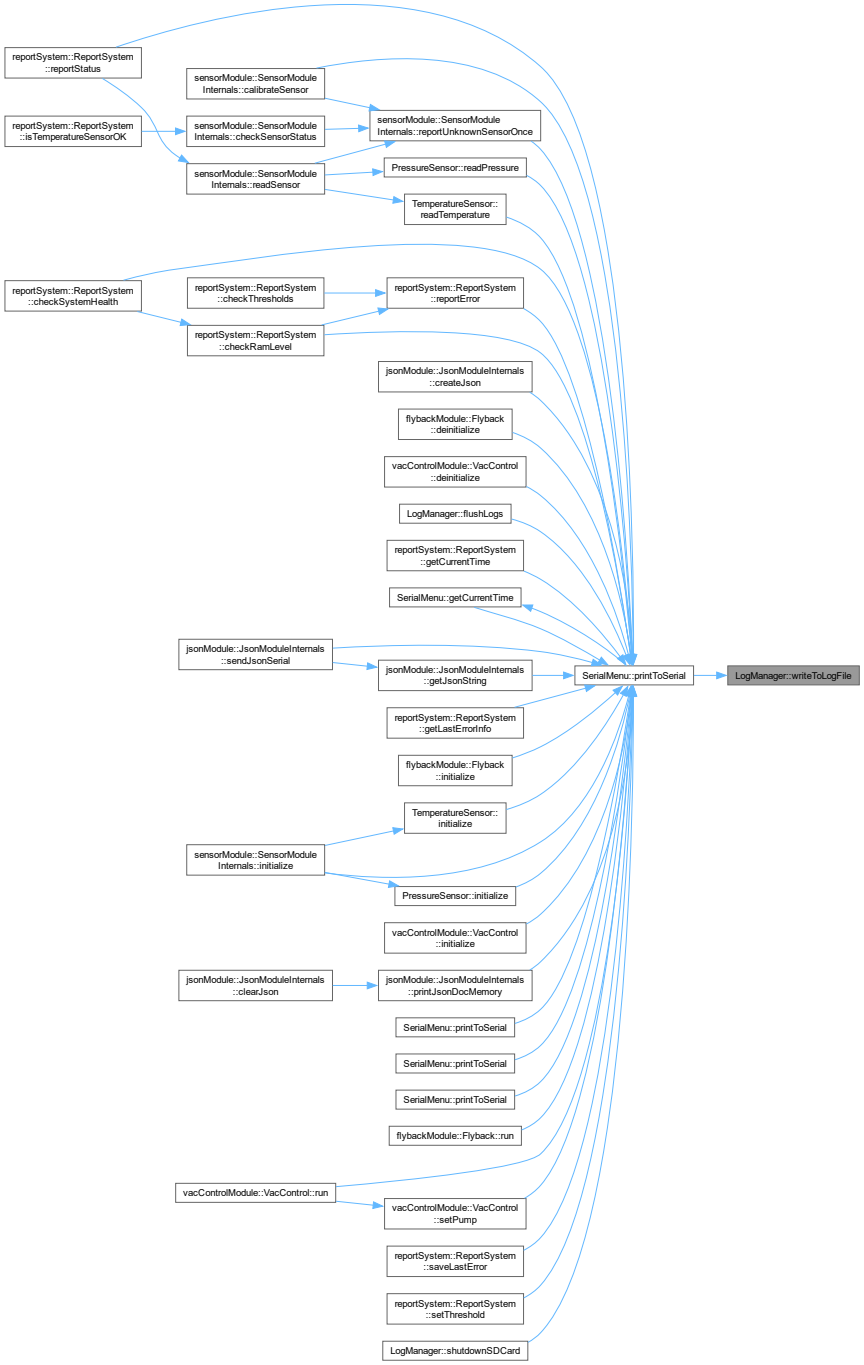
true -> if the log message was written successfully

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:



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

6.12 LogMapper Class Reference

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

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

6.12.1 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

6.13 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**

6.13.1 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

6.14 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>
```

6.14.1 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

6.15 MenuItem Struct Reference

Serial menu structure.

`#include <serialMenu.h>`

Public Attributes

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

6.15.1 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

6.16 Outputlevel Class Reference

Enum Class for the differnet Outputlevels.

`#include <serialMenu.h>`

6.16.1 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

6.17 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.

6.17.1 Detailed Description

template<typename T>
class PointerWrapper< T >

Tempalte class for wrapping a pointer.

Template Parameters

T	

6.17.2 Member Function Documentation

6.17.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.17.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.17.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.17.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.17.2.5 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

6.18 vacControlModule::Pressure Struct Reference

Public Attributes

- float **pressure**

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

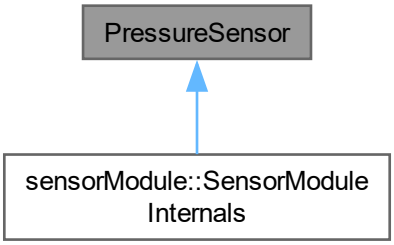
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h

6.19 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.

6.19.1 Detailed Description

Pressure sensor class.

6.19.2 Member Function Documentation

6.19.2.1 isInitialized()

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

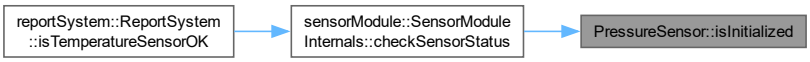
Function to check if the pressure sensor is initialized.

Returns

true -> if the pressure sensor is initialized

false -> if the pressure sensor is not initialized

Here is the caller graph for this function:



6.19.2.2 readPressure()

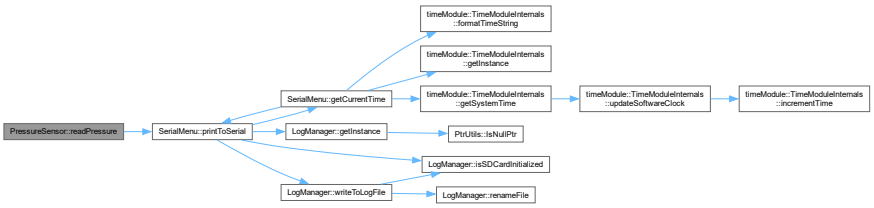
```
float PressureSensor::readPressure ( )
```

Function to read the pressure from the sensor.

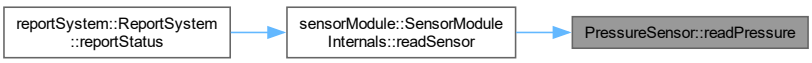
Returns

float -> The pressure value.

Here is the call graph for this function:



Here is the caller graph for this function:



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

6.20 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)

6.20.1 Detailed Description

Utility class for pointer operations.

6.20.2 Member Function Documentation

6.20.2.1 IsNullPtr()

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

Check if a pointer is nullptr.

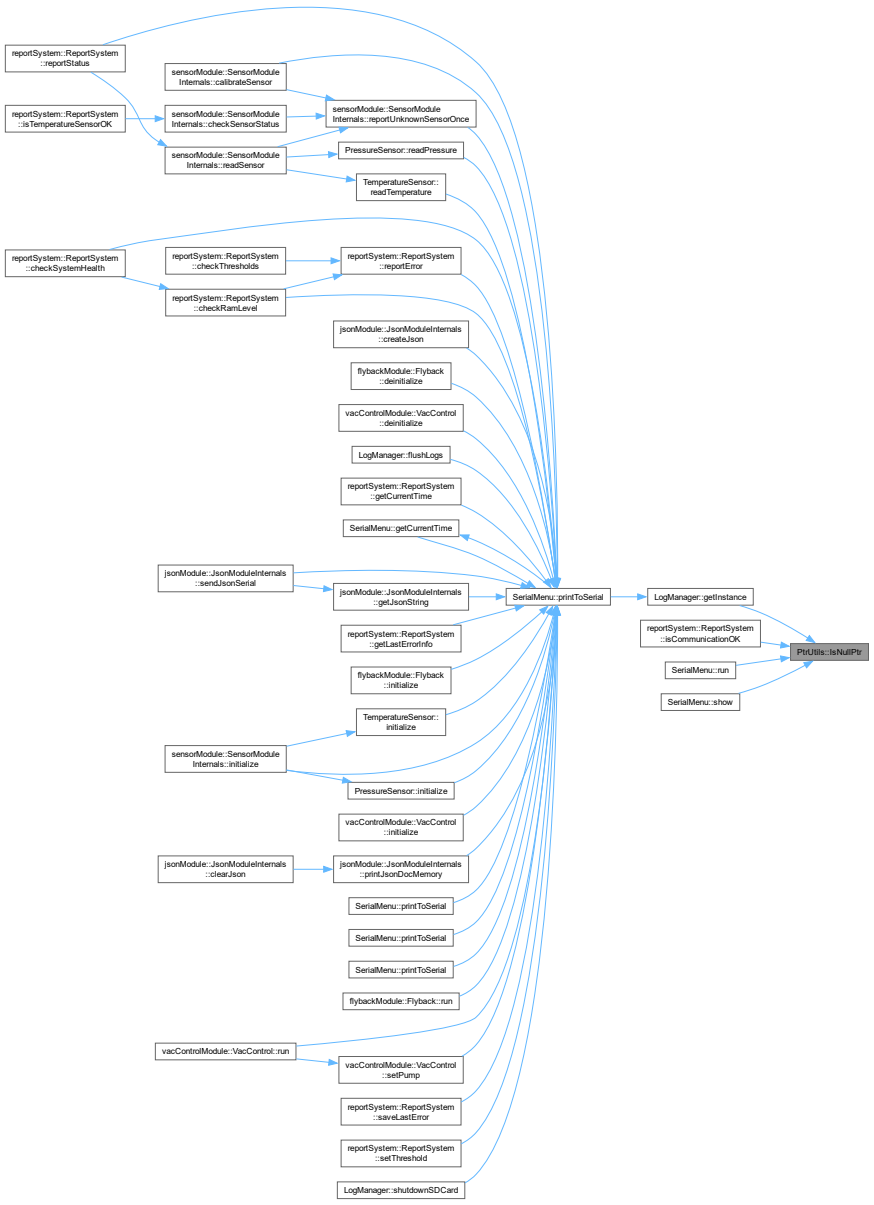
Parameters

ptr	Pointer to check.

Returns

true if the pointer is nullptr, false otherwise.

Here is the caller graph for this function:



6.20.2.2 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

6.21 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.
- String 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.

6.21.1 Detailed Description

Class for the report system.

6.21.2 Member Function Documentation

6.21.2.1 checkRamLevel()

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

Function to check the SRAM level on the hostsystem.

Parameters

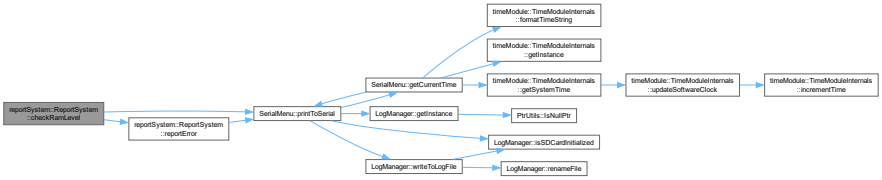
warningThreshold	-> first warning to get
criticalThreshold	-> last warning to get

Returns

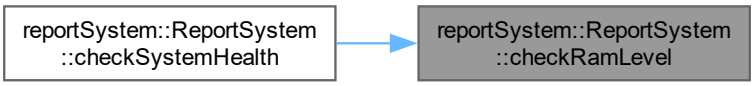
true -> if the level exceeded

false -> if the levels are withing the thresholds

Here is the call graph for this function:



Here is the caller graph for this function:



6.21.2.2 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

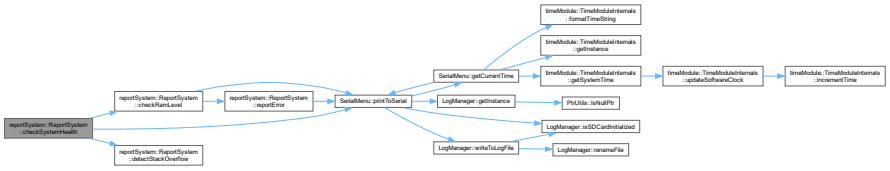
memoryThreshold	-> The memory threshold to check
checkEth	-> Check the Ethernet connection
checkSpi	-> Check the SPI connection
checkI2c	-> Check the I2C connection
checkTemp	-> Check the temperature sensor
checkPress	-> Check the pressure sensor

Returns

true -> if the system is healthy

false -> if the system is not healthy

Here is the call graph for this function:



6.21.2.3 checkThresholds()

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

Check the thresholds for the temperature and pressure sensors.

Parameters

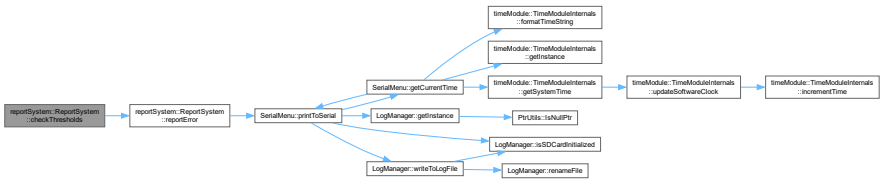
currentTemp	-> The current temperature
currentPressure	-> The current pressure

Returns

true -> if the thresholds are met

false -> if the thresholds are not met

Here is the call graph for this function:



6.21.2.4 detectStackOverflow()

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

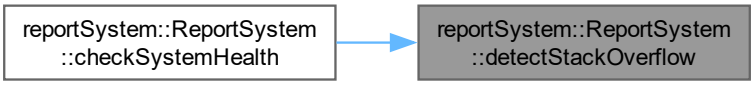
Detect Stack Overflow.

Returns

true -> if the stack has overflowed

false -> if the stack has not overflowed

Here is the caller graph for this function:



6.21.2.5 getCPULoad()

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

Getter for the CPU Load.

Returns

float -> The CPU Load

6.21.2.6 getCurrentTime()

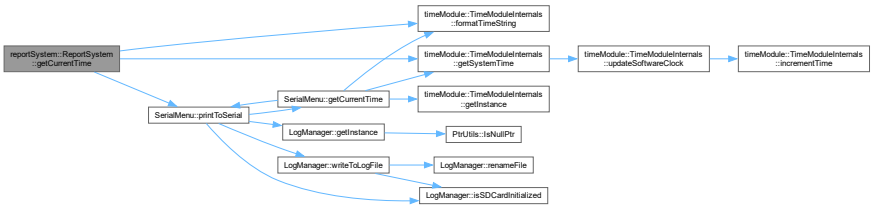
```
String ReportSystem::getCurrentTime ( )
```

Get the Current Time of the system.

Returns

String -> The current time

Here is the call graph for this function:



6.21.2.7 getLastError()

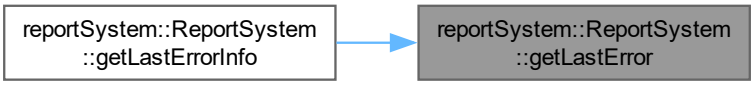
```
String ReportSystem::getLastError ( )
```

Get the Last Error message from EEPROM.

Returns

String -> The last error message

Here is the caller graph for this function:



6.21.2.8 getLastErrorInfo()

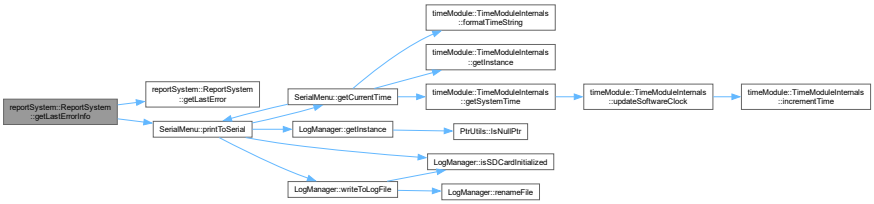
```
bool ReportSystem::getLastErrorInfo ( )
```

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:



6.21.2.9 getMemoryStatus()

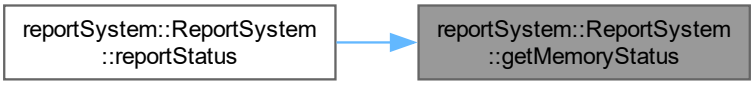
```
String ReportSystem::getMemoryStatus ( )
```

Get the Memory Status of the system.

Returns

String -> The memory status

Here is the caller graph for this function:



6.21.2.10 getStackDump()

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

Get the Stack Dump of the system.

Returns

String -> The stack dump

6.21.2.11 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

6.21.2.12 isCommunicationOK()

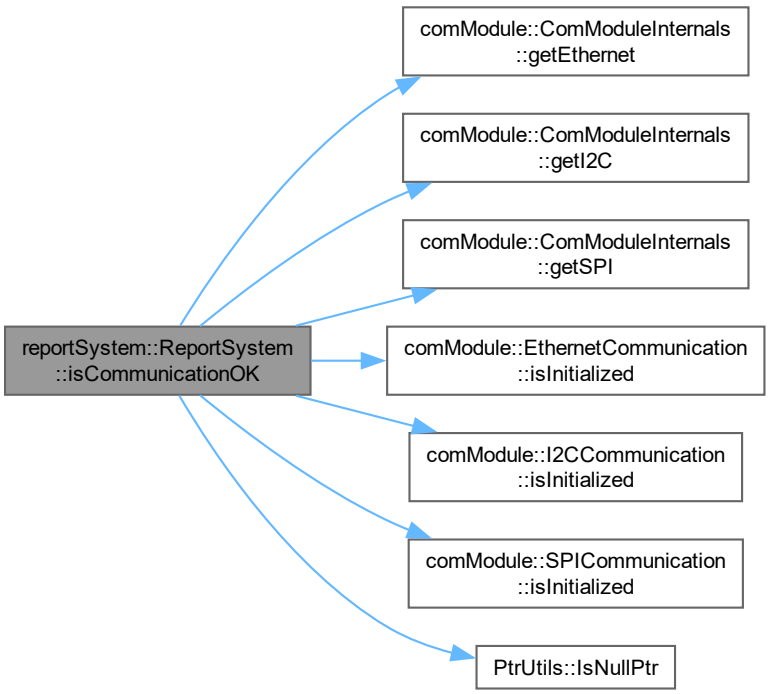
```
bool ReportSystem::isCommunicationOK ( ) const
```

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:



6.21.2.13 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

6.21.2.14 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

6.21.2.15 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

6.21.2.16 isTemperatureSensorOK()

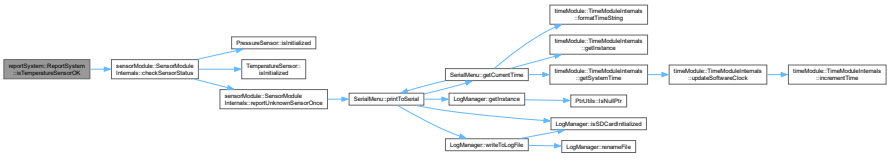
```
bool ReportSystem::isTemperatureSensorOK ( ) const
```

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:



6.21.2.17 reportError()

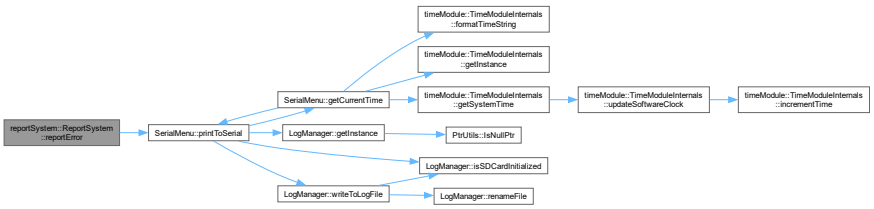
```
void ReportSystem::reportError (
            const char * errorMessage )
```

Function to log an error message.

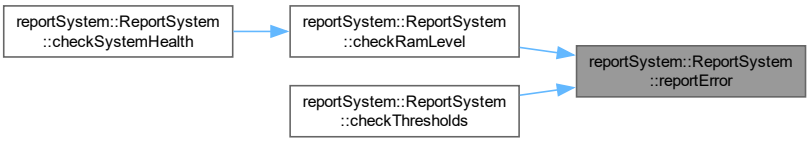
Parameters

errorMessage	-> The error message to log

Here is the call graph for this function:



Here is the caller graph for this function:



6.21.2.18 reportStatus()

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

Function to report the status of the system.

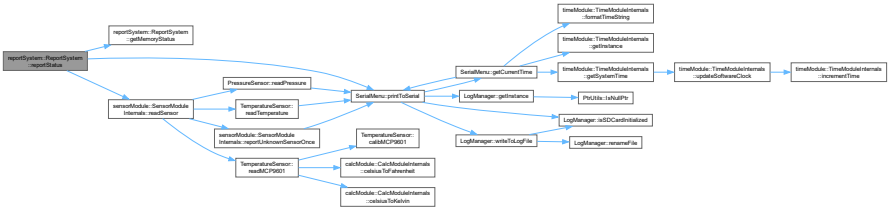
Parameters

active	-> The status of the system

Returns

String -> The status of the system

Here is the call graph for this function:



6.21.2.19 saveLastError()

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

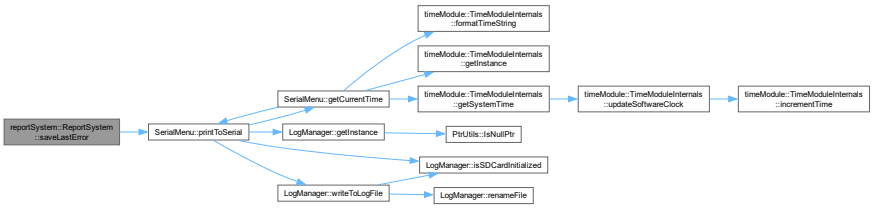
Saves last error message to EEPROM.

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:



6.21.2.20 setThreshold()

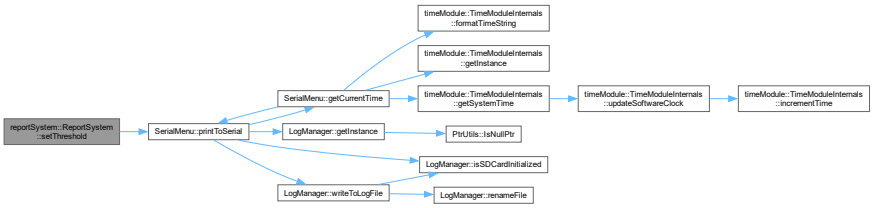
```
void ReportSystem::setThreshold (
            float tempThreshold,
            float pressureThreshold )
```

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:



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

6.22 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

6.22.1 Detailed Description

Scoped lock class for mutexes.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 ScopedLock() [1/2]

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

Construct a new Scoped Lock object.

Parameters

mutex	-> The mutex to lock

6.22.2.2 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

6.23 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.

6.23.1 Detailed Description

template<typename T>
class ScopedPointer< T >

Template class for a Scoped Pointer.

Template Parameters

T	-> The type of the pointer.

6.23.2 Member Function Documentation

6.23.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.23.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.23.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.23.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.23.2.5 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:

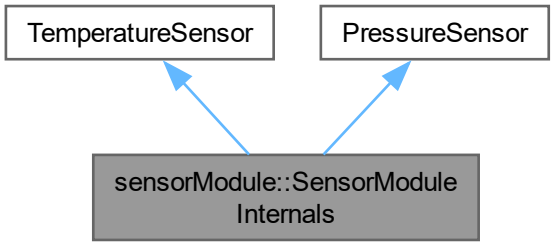
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.24 sensorModule::SensorModuleInternals Class Reference

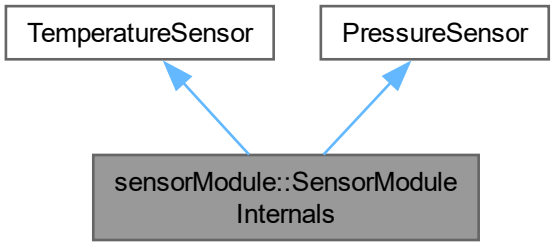
Class for the sensor module internals.

`#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.

6.24.1 Detailed Description

Class for the sensor module internals.

6.24.2 Member Function Documentation

6.24.2.1 calibrateSensor()

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

Function to calibrate the sensor.

Parameters

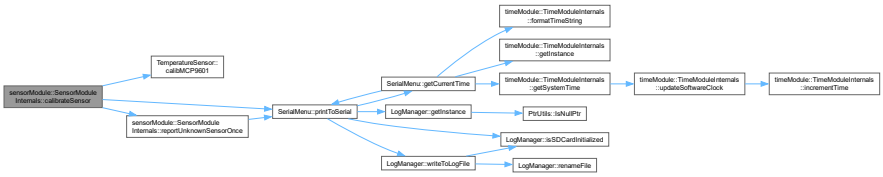
type	-> The type of the sensor to calibrate.

Returns

true -> if the sensor was calibrated successfully

false -> if the sensor was not calibrated successfully

Here is the call graph for this function:



6.24.2.2 checkSensorStatus()

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

Function to check the status of the sensor.

Parameters

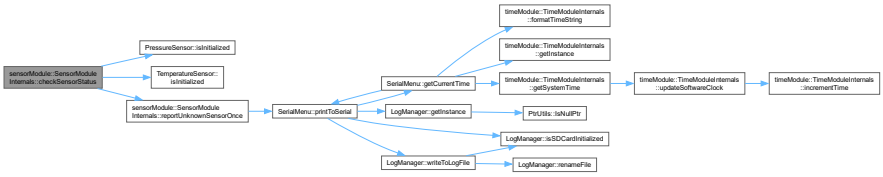
type	-> The type of the sensor to check.

Returns

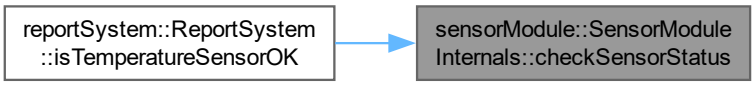
true -> if the sensor is healthy

false -> if the sensor is not healthy

Here is the call graph for this function:



Here is the caller graph for this function:



6.24.2.3 readSensor()

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

Function to read the sensor.

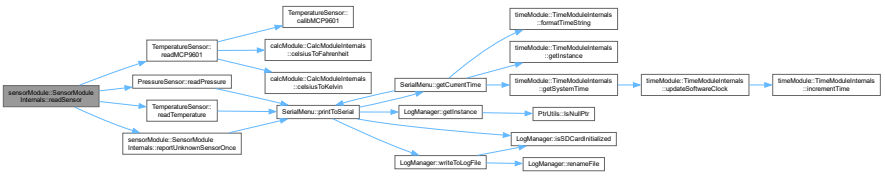
Parameters

type	-> The type of the sensor to read.

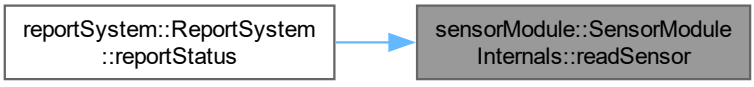
Returns

float -> The value of the sensor.

Here is the call graph for this function:



Here is the caller graph for this function:



6.24.2.4 reportUnknownSensorOnce()

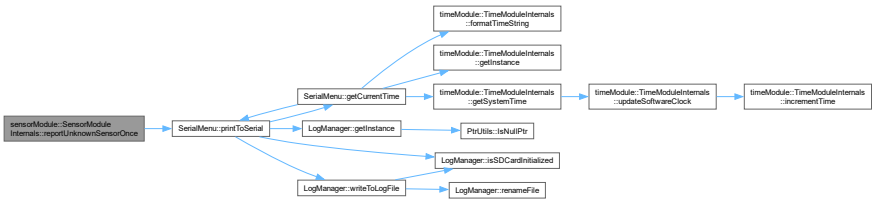
```
void SensorModuleInternals::reportUnknownSensorOnce (
            SensorType type,
            const __FlashStringHelper * context )
```

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

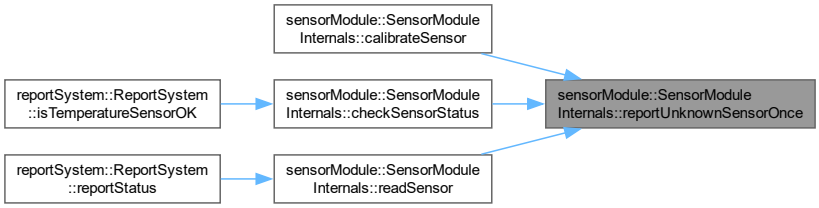
Parameters

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:



Here is the caller graph for this function:



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

6.25 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.

6.25.1 Detailed Description

Class to handle Serial communication.

6.25.2 Member Function Documentation

6.25.2.1 beginSerial()

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

Function to start the serial communication.

Parameters

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

6.25.2.2 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

6.25.2.3 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

6.25.2.4 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

6.26 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.

6.26.1 Detailed Description

Class for the serial menu.

6.26.2 Member Function Documentation

6.26.2.1 getCurrentTime()

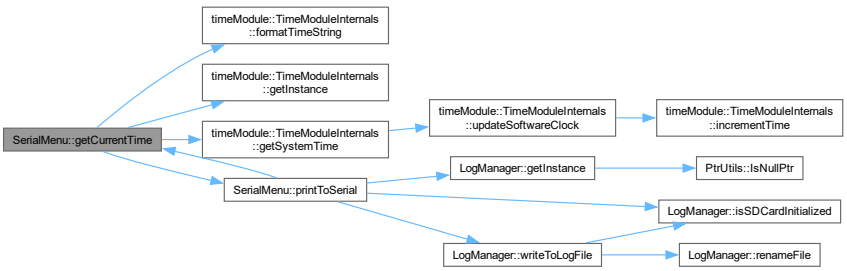
```
String SerialMenu::getCurrentTime ( )  [static]
```

Getter for the current time.

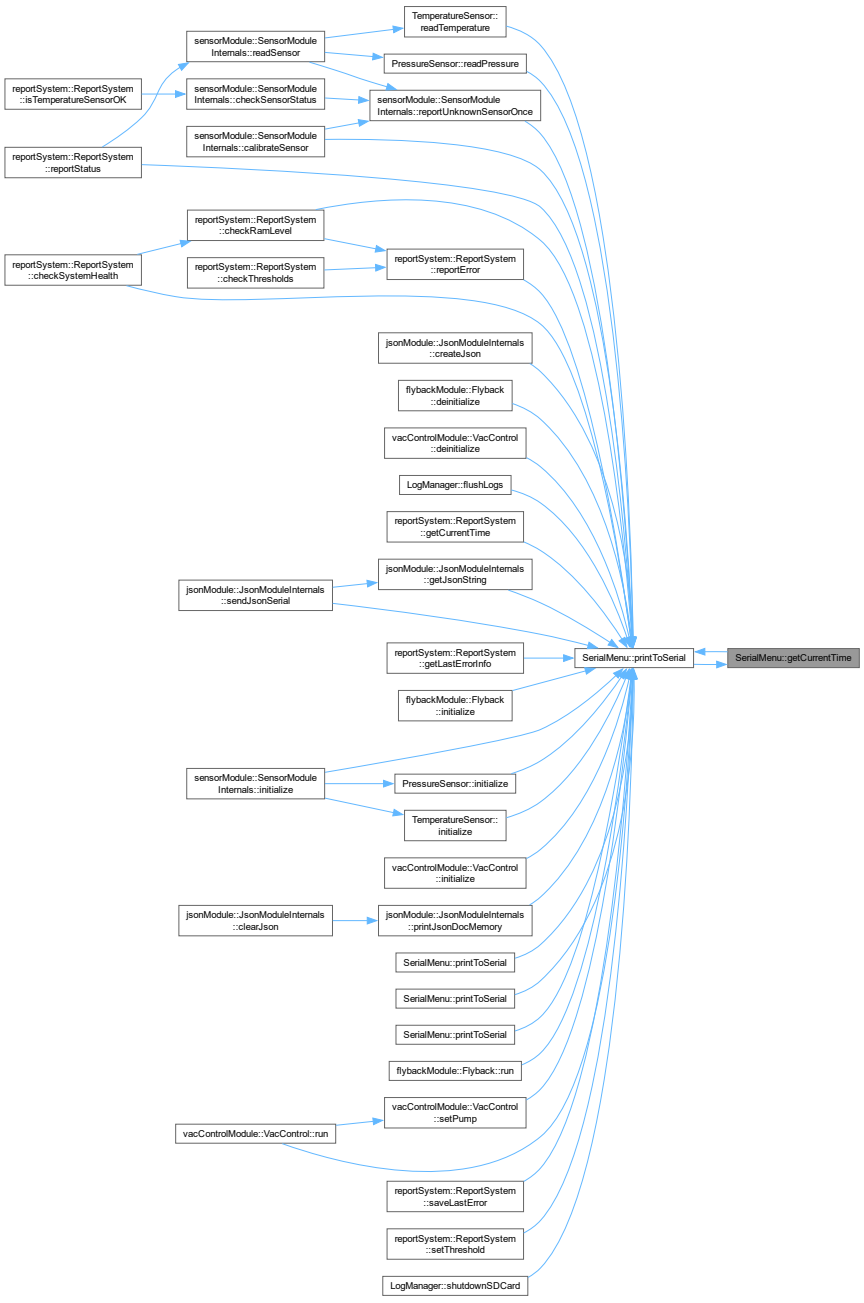
Returns

The current time as a String

Here is the call graph for this function:



Here is the caller graph for this function:



6.26.2.2 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.

6.26.2.3 printToSerial() [1/4]

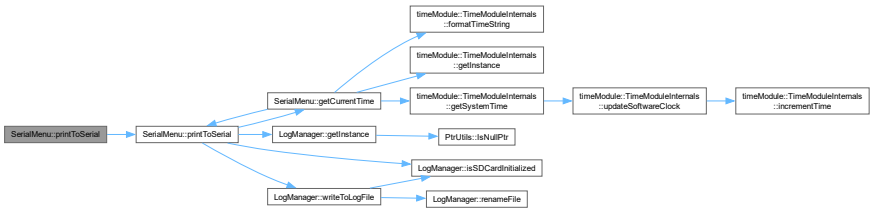
```
void SerialMenu::printToSerial (
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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:



6.26.2.4 printToSerial() [2/4]

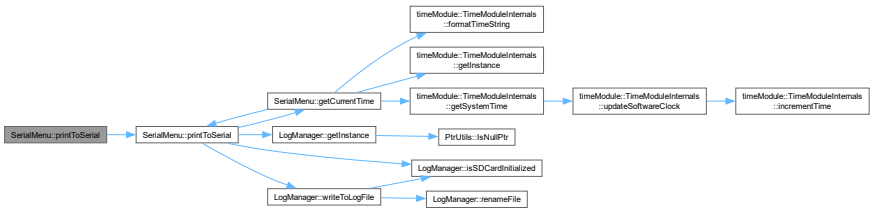
```
void SerialMenu::printToSerial (
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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:



6.26.2.5 printToSerial() [3/4]

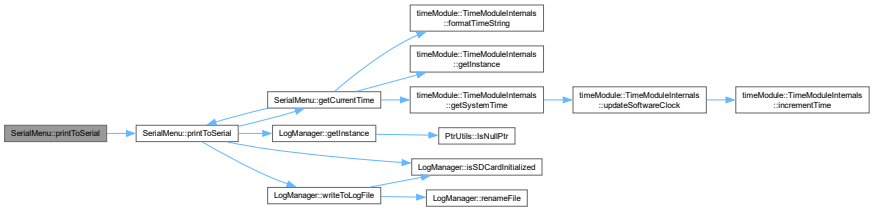
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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:



6.26.2.6 printToSerial() [4/4]

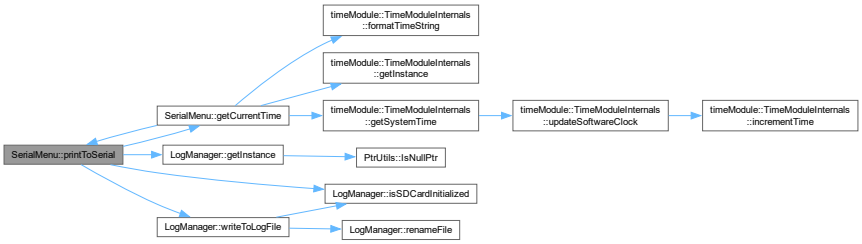
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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

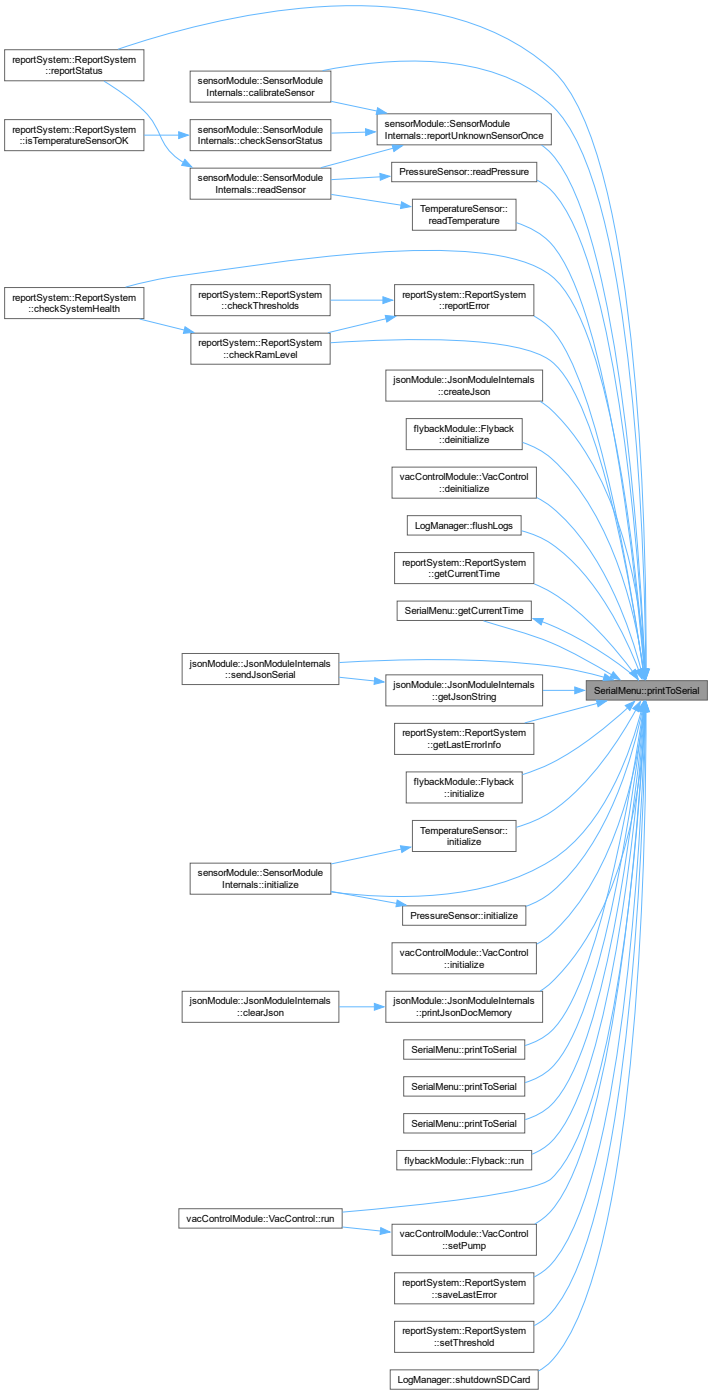
Parameters

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:



Here is the caller graph for this function:



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

6.27 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.

6.27.1 Detailed Description

Class to handle SPI communication.

6.27.2 Member Function Documentation

6.27.2.1 isInitialized()

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

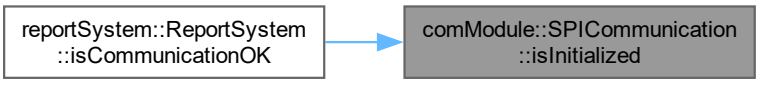
Function to check if the SPI communication is initialized.

Returns

true -> if the SPI communication is initialized

false -> if the SPI communication is not initialized

Here is the caller graph for this function:



6.27.2.2 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

6.27.2.3 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:

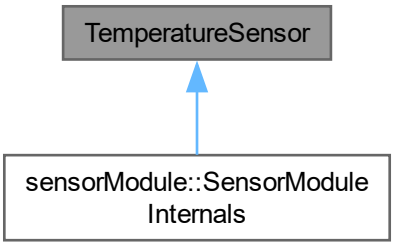
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.cpp

6.28 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.

6.28.1 Detailed Description

Temperature sensor class.

6.28.2 Member Function Documentation

6.28.2.1 calibMCP9601()

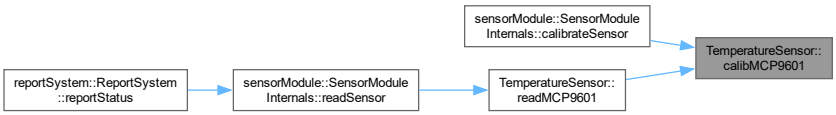
```
uint8_t TemperatureSensor::calibMCP9601 (
            SensorID sensor )
```

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:



6.28.2.2 isInitialized()

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

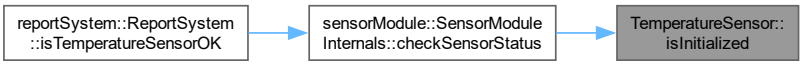
Check if the temperature sensor is initialized.

Returns

true -> if the temperature sensor is initialized

false -> if the temperature sensor is not initialized

Here is the caller graph for this function:



6.28.2.3 readMCP9601()

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

Function to read form specific sensor MCP9601.

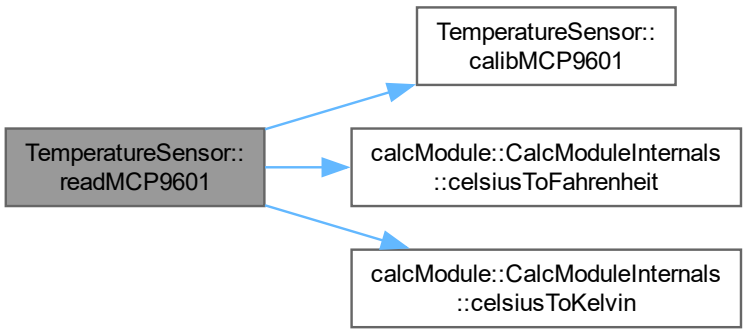
Parameters

unit	-> Choose the unit ℃, F, K

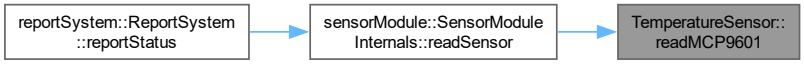
Returns

-> The temperature value

Here is the call graph for this function:



Here is the caller graph for this function:



6.28.2.4 readTemperature()

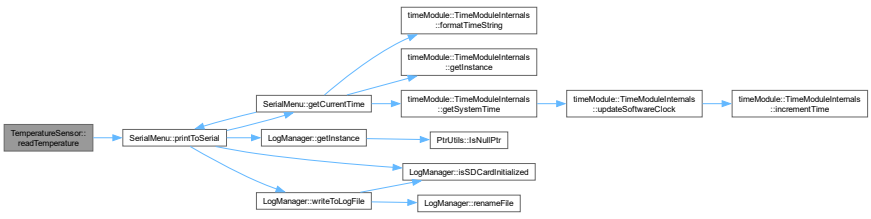
```
float TemperatureSensor::readTemperature ( )
```

Function to read the temperature from the sensor.

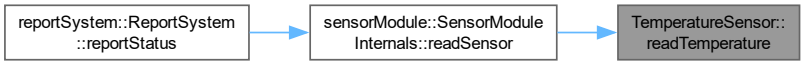
Returns

float -> The temperature value.

Here is the call graph for this function:



Here is the caller graph for this function:



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

6.29 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.

6.29.1 Detailed Description

Class to handle Systemtime.

6.29.2 Member Function Documentation

6.29.2.1 formatTimeString()

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

Function to format the time to a string.

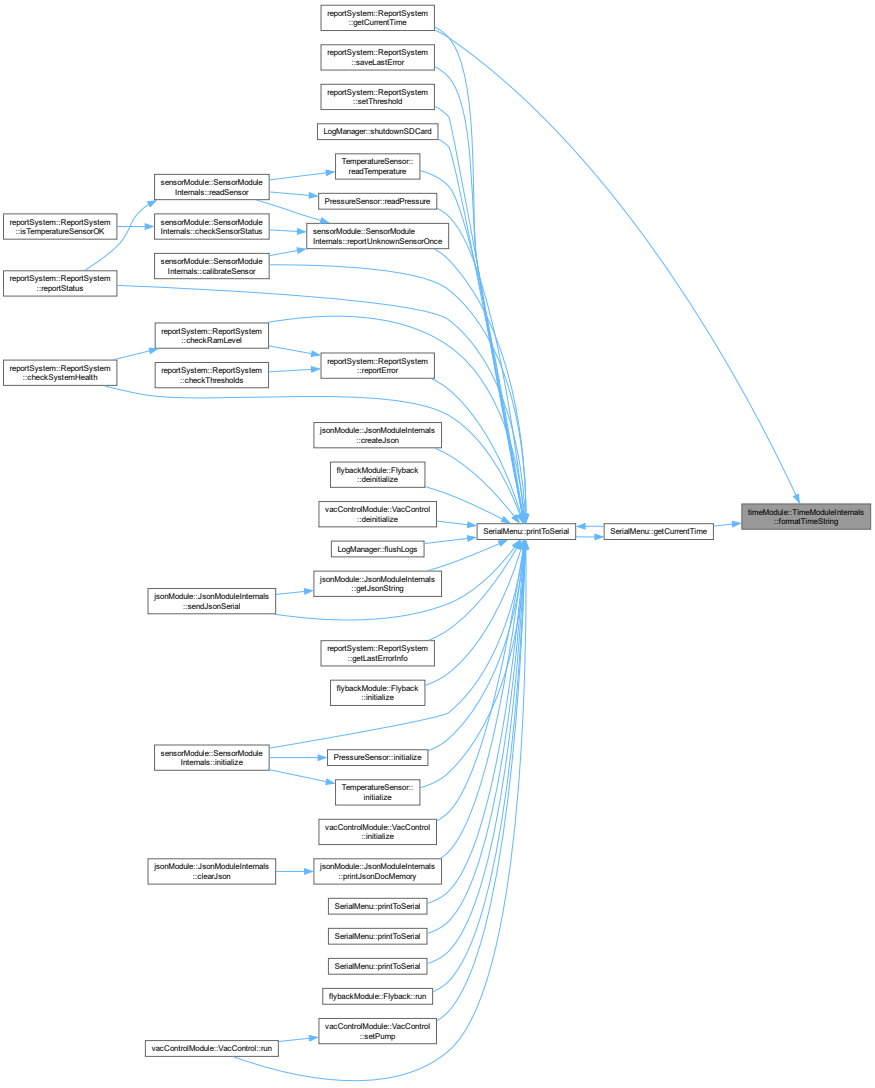
Parameters

dt	-> DateTimeStruct to format

Returns

String -> The formatted time.

Here is the caller graph for this function:



6.29.2.2 getInstance()

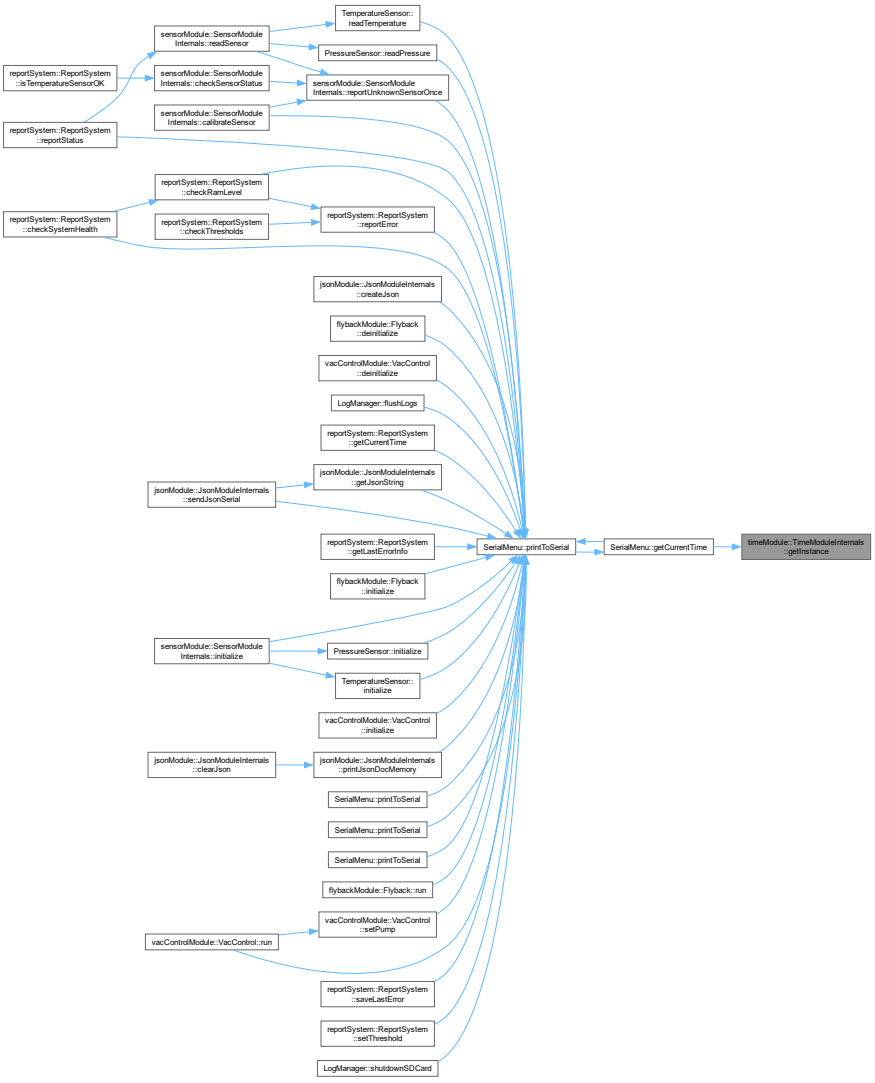
```
TimeModuleInternals * TimeModuleInternals::getInstance ( )  [static]
```

Get the Instance object, Singleton pattern.

Returns

TimeModuleInternals* -> The instance of the TimeModuleInternals.

Here is the caller graph for this function:



6.29.2.3 getSystemTime()

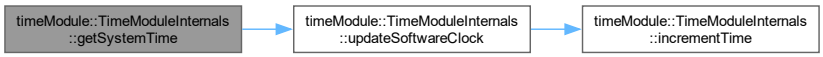
```
DateTimeStruct TimeModuleInternals::getSystemTime ( )
```

Get the System Time object.

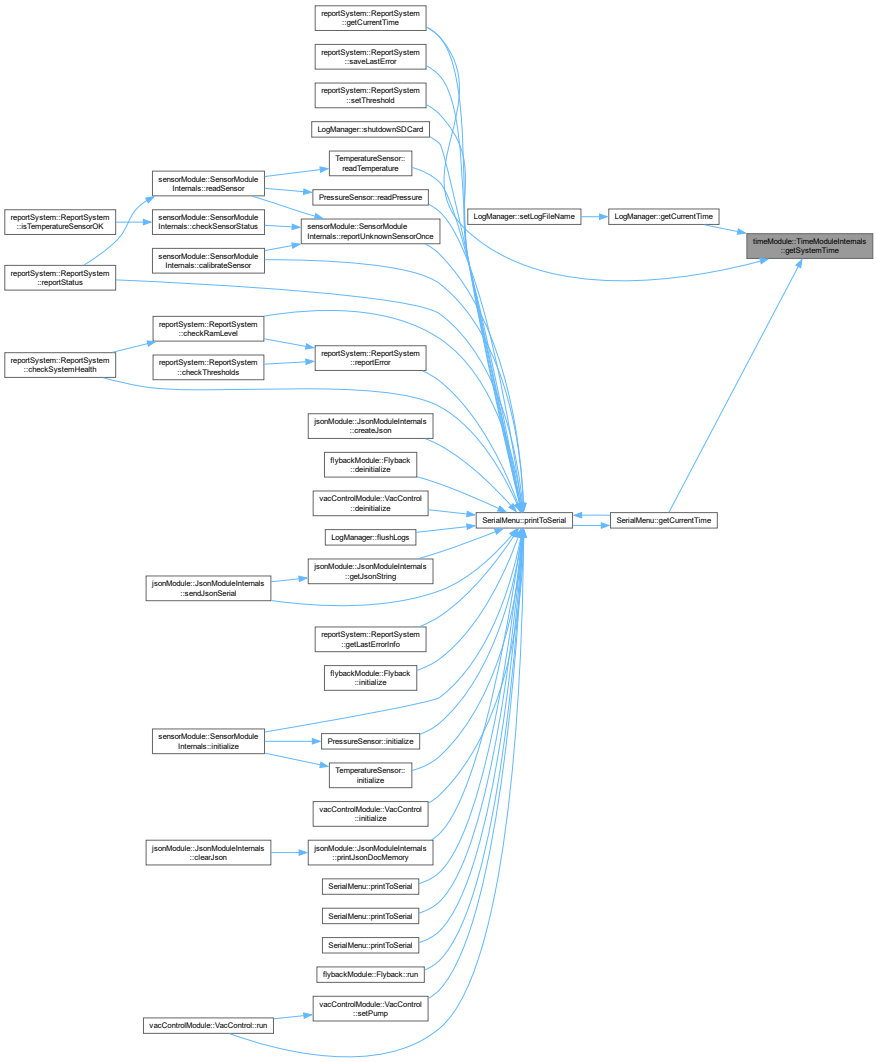
Returns

DateTimeStruct -> The system time.

Here is the call graph for this function:



Here is the caller graph for this function:



6.29.2.4 incrementTime()

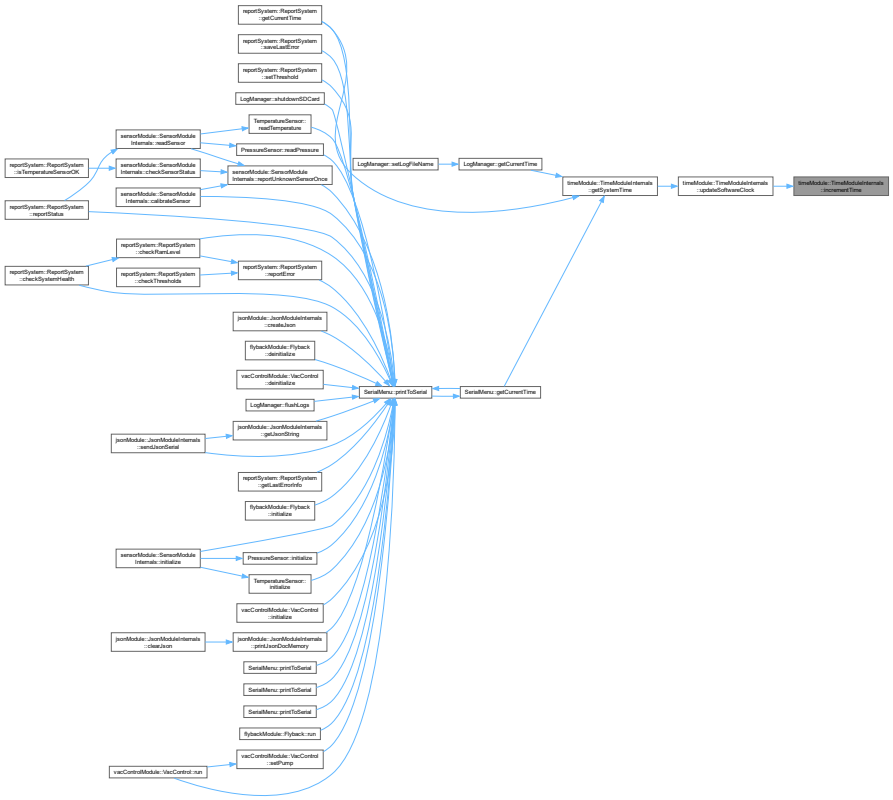
```
void TimeModuleInternals::incrementTime (
            DateTimeStruct * dt )  [static]
```

Function to increment the time of the system.

Parameters

dt	-> DateTimeStruct to increment time

Here is the caller graph for this function:



6.29.2.5 setSystemTime()

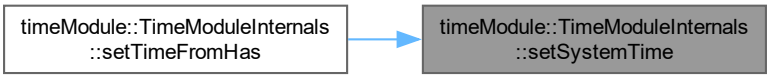
```
void TimeModuleInternals::setSystemTime (
            const DateTimeStruct & dt )
```

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:



6.29.2.6 setTimeFromHas()

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

Set the Time From Has object to the system time.

Parameters

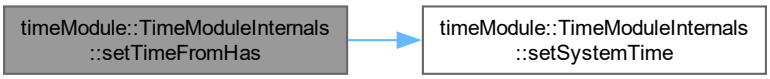
timeString	-> The time string to set the system time to.

Returns

true -> if the time was set successfully

false -> if the time was not set successfully

Here is the call graph for this function:



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

6.30 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.
- SwitchStates getSwitchState ()

  Returns the state of the main switch.
- 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 externProcess ()

  Process external data for scenarios (currently unused)
- void setExternPressure (float pressure)

  Sets the external pressure value.
- float getExternPressure ()

  Gets the external pressure value.

6.30.1 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.

6.30.2 Member Function Documentation

6.30.2.1 externProcess()

```
void vacControlModule::VacControl::externProcess ( )
```

Process external data for scenarios (currently unused)

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

6.30.2.2 getExternPressure()

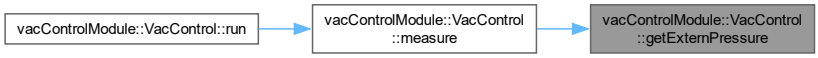
```
float VacControl::getExternPressure ( )
```

Gets the external pressure value.

Returns

The current external pressure value

Here is the caller graph for this function:



6.30.2.3 getExternScenario()

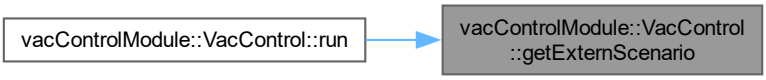
```
int VacControl::getExternScenario ( )
```

Getter function to retrieve the current external scenario state.

Returns

The current external scenario state (integer)

Here is the caller graph for this function:



6.30.2.4 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.

6.30.2.5 getScenarioFromPotValue()

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

Determines the scenario based on the potentiometer value.

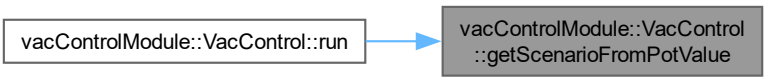
Parameters

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

Returns

The corresponding scenario based on the potentiometer value

Here is the caller graph for this function:



6.30.2.6 getSwitchState()

```
SwitchStates VacControl::getSwitchState ( )
```

Returns the state of the main switch.

Returns

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

6.30.2.7 isInitialized()

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

Get the state of the VacControl system.

Returns

true -> VacControl is initialized and ready

false -> VacControl is not initialized

6.30.2.8 measure()

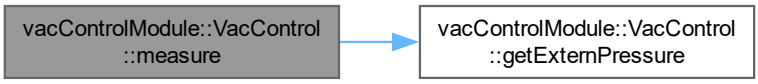
```
Pressure VacControl::measure ( )
```

Measures the actual pressure of the system.

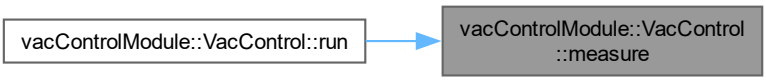
Returns

Measurement -> A Measurement object containing the current pressure

Here is the call graph for this function:



Here is the caller graph for this function:

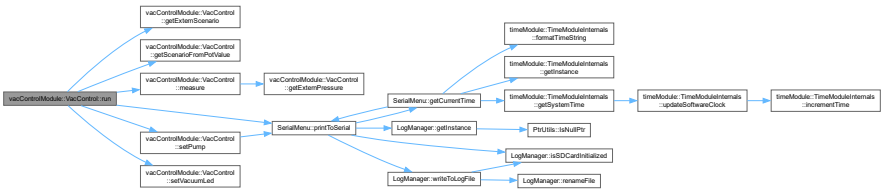


6.30.2.9 run()

```
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:



6.30.2.10 setExternPressure()

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

Sets the external pressure value.

Parameters

pressure	The external pressure value to set

6.30.2.11 setExternScenario()

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

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

Parameters

pressure	The external scenario pressure value

6.30.2.12 setPump()

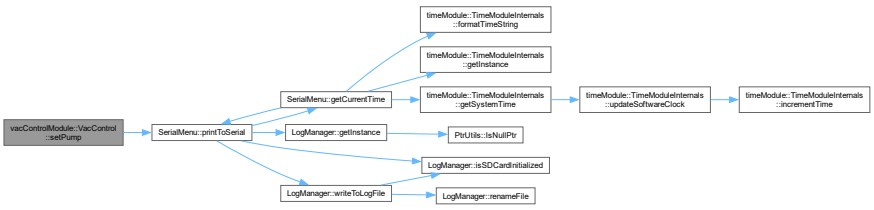
```
void VacControl::setPump (
            bool flag )
```

Set the Pump flag.

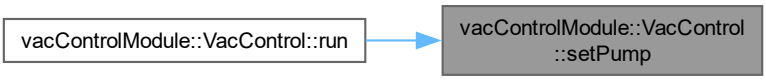
Parameters

flag	This is the boolean flag to set

Here is the call graph for this function:



Here is the caller graph for this function:



6.30.2.13 setVacuumLed()

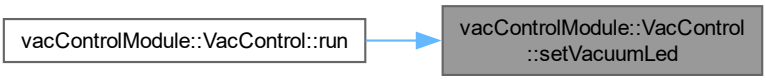
```
void VacControl::setVacuumLed (
            float pressure,
            float targetPressure )
```

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

Here is the caller graph for this function:



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

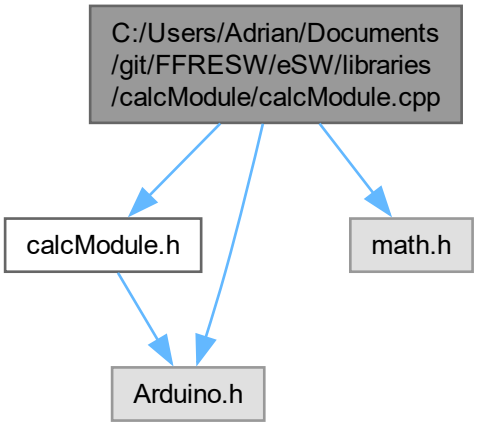
File Documentation

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:



7.1.1 Detailed Description

Implementation of the calcModule class.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

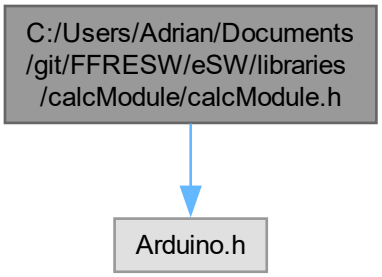
Copyright (c) 2024

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

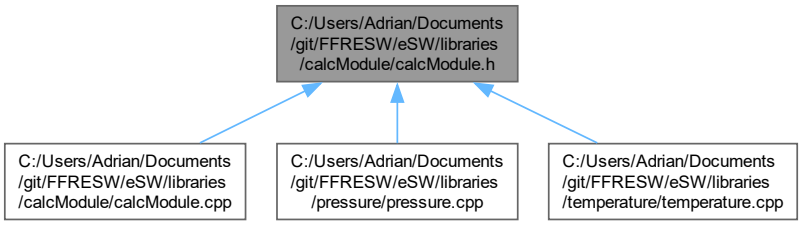
Header file for the calculation module handling sensor data.

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

Include dependency graph for calcModule.h:



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



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

7.3 calcModule.h

Go to the documentation of this file.

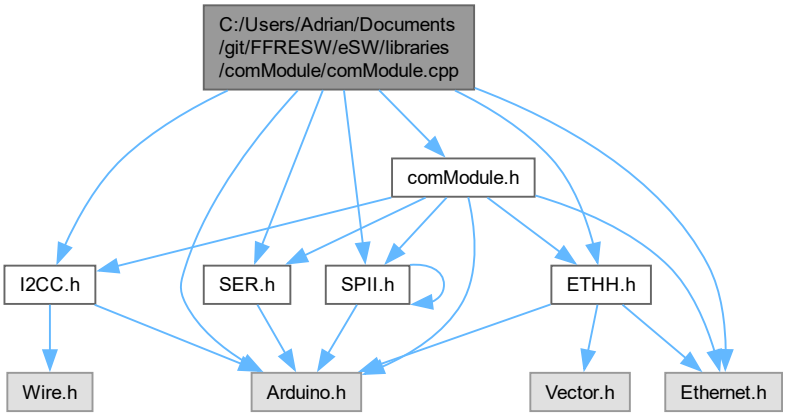
```
00001
00011 #ifndef CALCMODULE_H
00012 #define CALCMODULE_H
00013
00014 #include <Arduino.h>
00015
00017 namespace calcModule
00018 {
00020     enum Type
00021     {
00022         General,
00023         Pressure,
00024         Position
00025     };
00026
00028     enum class PressureUnit
00029     {
00030         Pascal,
00031         Atmosphere,
00032         Psi,
00033         Bar
00034     };
00035
00037     class CalcModuleInternals
00038     {
00039     public:
00040         CalcModuleInternals();
00041         ~CalcModuleInternals();
00042
00050         static float calculateAverage(const float* data, int length);
00051
00059         static float findMaximum(const float* data, int length);
00060
00068         static float findMinimum(const float* data, int length);
00069
00077         static float calculateStandardDeviation(const float* data, int length);
00078
00086         static float findMedian(float* data, int length);
00087
00094         static float celsiusToFahrenheit(float celsius);
00095
00102         static float fahrenheitToCelsius(float fahrenheit);
00103
00110         static float celsiusToKelvin(float celsius);
00111
00118         static float kelvinToCelsius(float kelvin);
00119
00126         static float pascalToAtm(float pascal);
00127
00134         static float atmToPascal(float atm);
00135
00142         static float pascalToPsi(float pascal);
00143
00150         static float psiToPascal(float psi);
00151
00159         static float calculatePower(float voltage, float current);
00160
00168         static float calculateCurrent(float voltage, float resistance);
00169
00177         static float calculateResistance(float voltage, float current);
00178
00192         static float extractFloat(String response, int id);
00193
00202         static float extractFloatFromResponse(const String& response, Type type);
00203
00204         static float calculatePressureFromSensor(int sensorValue, PressureUnit unit =
      PressureUnit::Pascal);
00205
00206     private:
00207
00214         static void sortArray(float* data, int length);
00215
00222         static float roundToPrecision(float value, int precision);
00223     };
00224 }
00225
00226 #endif // CALCMODULE_H
```

7.4 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/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:



7.4.1 Detailed Description

Implementation of the comModule class that utilizes various communication protocols.

7.5 comModule.h

```
00001 #ifndef COMMODULE_H
00002 #define COMMODULE_H
00003
00004 #include <Arduino.h>
00005 #include <Ethernet.h>
00006 #include "ETHH.h"
00007 #include "I2CC.h"
00008 #include "SER.h"
00009 #include "SPII.h"
00010
00012 namespace comModule
00013 {
00015     class ComModuleInternals
00016     {
00017     public:
00018         ComModuleInternals();
00019         ~ComModuleInternals();
00020
00026         EthernetCommunication& getEthernet();
00027
00033         I2CCommunication& getI2C();
```

```
00034
00040         SPICommunication& getSPI();
00041
00047         SerialCommunication& getSerial();
00048
00049     private:
00050         EthernetCommunication eth;
00051         I2CCommunication i2c;
00052         SPICommunication spi;
00053         SerialCommunication ser;
00054     };
00055 }
00056
00057 #endif // COMMODULE_H
```

7.6 ETHH.h

```
00001
00008 #ifndef ETHERNET_COMMUNICATION_H
00009 #define ETHERNET_COMMUNICATION_H
00010
00011 #include <Arduino.h>
00012 #include <Ethernet.h>
00013 #include <Vector.h>
00014
00016 namespace comModule
00017 {
00019     enum class Service : uint8_t
00020     {
00021         SET = 0x01,
00022         GET = 0x0B,
00023         SET_COMPOUND = 0x28,
00024         GET_COMPOUND = 0x29,
00025         SETGET = 0x30
00026     };
00027
00029     enum class Compound1 : uint32_t
00030     {
00031         CONTROL_MODE = 0x0F020000,
00032         TARGET_POSITION = 0x11020000,
00033         TARGET_PRESSURE = 0x07020000,
00034         NOT_USED = 0x00000000
00035     };
00036
00038     enum class Compound2 : uint32_t
00039     {
00040         ACCESS_MODE = 0x0F0B0000,
00041         CONTROL_MODE = 0x0F020000,
00042         TARGET_POSITION = 0x11020000,
00043         TARGET_PRESSURE = 0x07020000,
00044         ACTUAL_POSITION = 0x10010000,
00045         POSITION_STATE = 0x00100000,
00046         ACTUAL_PRESSURE = 0x07010000,
00047         TARGET_PRESSURE_USED = 0x07030000,
00048         WARNING_BITMAP = 0x0F300100,
00049         NOT_USED = 0x00000000
00050     };
00051
00053     enum class Compound3 : uint32_t
00054     {
00055         CONTROL_MODE = 0x0F020000,
00056         TARGET_POSITION = 0x11020000,
00057         TARGET_PRESSURE = 0x07020000,
00058         SEPARATION = 0x00000000,
00059         ACCESS_MODE = 0x0F0B0000,
00060         ACTUAL_POSITION = 0x10010000,
00061         POSITION_STATE = 0x00100000,
00062         ACTUAL_PRESSURE = 0x07010000,
00063         TARGET_PRESSURE_USED = 0x07030000,
00064         WARNING_BITMAP = 0x0F300100,
00065         NOT_USED = 0x00000000
00066     };
00067
00069     enum class Error_Codes : uint8_t
00070     {
00071         NO_ERROR = 0x00,
00072         WRONG_COMMAND_LENGTH = 0x0C,
00073         VALUE_TOO_LOW = 0x1C,
00074         VALUE_TOO_HIGH = 0x1D,
00075         RESULTING_ZERO_ADJUST_OFFSET = 0x20,
00076         NO_SENSOR_ENABLED = 0x21,
00077         WRONG_ACCESS_MODE = 0x50,
00078         TIMEOUT = 0x51,
```

```
00079         NV_MEMORY_NOT_READY = 0x6D,
00080         WRONG_PARAMETER_ID = 0x6E,
00081         PARAMETER_NOT_SETTABLE = 0x70,
00082         PARAMETER_NOT_READABLE = 0x71,
00083         WRONG_PARAMETER_INDEX = 0x73,
00084         WRONG_VALUE_WITHIN_RANGE = 0x76,
00085         NOT_ALLOWED_IN_THIS_STATE = 0x78,
00086         SETTING_LOCK = 0x79,
00087         WRONG_SERVICE = 0x7A,
00088         PARAMETER_NOT_ACTIVE = 0x7B,
00089         PARAMETER_SYSTEM_ERROR = 0x7C,
00090         COMMUNICATION_ERROR = 0x7D,
00091         UNKNOWN_SERVICE = 0x7E,
00092         UNEXPECTED_CHARACTER = 0x7F,
00093         NO_ACCESS_RIGHTS = 0x80,
00094         NO_ADEQUATE_HARDWARE = 0x81,
00095         WRONG_OBJECT_STATE = 0x82,
00096         NO_SLAVE_COMMAND = 0x84,
00097         COMMAND_TO_UNKNOWN_SLAVE = 0x85,
00098         COMMAND_TO_MASTER_ONLY = 0x87,
00099         ONLY_G_COMMAND_ALLOWED = 0x88,
00100         NOT_SUPPORTED = 0x89,
00101         FUNCTION_DISABLED = 0xA0,
00102         ALREADY_DONE = 0xA1
00103     };
00104
00106     class EthernetCommunication
00107     {
00108     public:
00109         EthernetCommunication();
00110         ~EthernetCommunication();
00111
00118         void beginEthernet(uint8_t* macAddress, IPAddress ip);
00119
00126         void sendEthernetData(const char* endpoint, const char* data);
00127
00134         void receiveEthernetData(char* buffer, size_t length);
00135
00140         void handleEthernetClient();
00141
00147         String getRequestedEndpoint();
00148
00155         String getSpecificEndpoint(const String& jsonBody);
00156
00162         void sendJsonResponse(const String& jsonBody);
00163
00169         EthernetClient& getClient();
00170
00177         bool isInitialized() const;
00178
00185         bool getSendDataFlag() const;
00186
00192         void setSendDataFlag(bool flag);
00193
00201         void setCompound(Compound1 id, int index, String value);
00202
00210         void setCompound(Compound2 id, int index, String value);
00211
00219         void setCompound(Compound3 id, int index, String value);
00220
00229         void setCompoundInternal(String compoundType, unsigned long id, int index, String value);
00230
00238         String getCompound(Compound1 id, int index);
00239
00247         String getCompound(Compound2 id, int index);
00248
00256         String getCompound(Compound3 id, int index);
00257
00266         String getCompoundInternal(String compoundType, unsigned long id, int index);
00267
00275         Vector<float> getParsedCompound(Compound1 id, int index);
00276
00284         Vector<float> getParsedCompound(Compound2 id, int index);
00285
00293         Vector<float> getParsedCompound(Compound3 id, int index);
00294
00301         Vector<float> parseCompoundResponse(String response);
00302
00309         void setParameter(Compound2 id, String value);
00310
00317         String getParameter(Compound2 id);
00318
00324         void sendCommand(String command);
00325
00326     private:
00327         EthernetServer server;
00328         EthernetClient client;
```

```
00329         bool ethernetInitialized = false;
00330         bool sendDataFlag = false;
00331 
00338         String floatToIEEE754(float value);
00339 
00346         Vector<float> parseResponse(String response);
00347 
00348     };
00349 }
00350 
00351 #endif // ETHERNET_COMMUNICATION_H
00352 
00353 
```

7.7 I2CC.h

```
00001 
00008 #ifndef I2C_COMMUNICATION_H
00009 #define I2C_COMMUNICATION_H
00010 
00011 #include <Arduino.h>
00012 #include <Wire.h>
00013 
00014 namespace comModule
00015 {
00017     class I2CCommunication
00018     {
00019     public:
00020         I2CCommunication();
00021         ~I2CCommunication();
00022 
00028         void beginI2C(uint8_t address);
00029 
00030 
00035         void beginI2CGlobal();
00036 
00041         void endI2C();
00042 
00050         void i2cWrite(uint8_t deviceAddress, uint8_t* data, size_t length);
00051 
00060         size_t i2cRead(uint8_t deviceAddress, uint8_t* buffer, size_t length);
00061 
00068         bool isInitialized() const;
00069 
00070     private:
00071         bool i2cInitialized = false;
00072     };
00073 }
00074 
00075 #endif // I2C_COMMUNICATION_H
```

7.8 SER.h

```
00001 
00008  #ifndef SERIAL_COMMUNICATION_H
00009  #define SERIAL_COMMUNICATION_H
00010 
00011  #include <Arduino.h>
00012 
00013  namespace comModule
00014  {
00016      class SerialCommunication
00017      {
00018      public:
00019 
00025          void beginSerial(long baudRate);
00026 
00031          void endSerial();
00032 
00038          void sendSerialData(const char* data);
00039 
00046          void receiveSerialData(char* buffer, size_t length);
00047 
00054          bool isInitialized() const;
00055 
00056      private:
00057          bool serInitialized = false;
00058      };
00059  }
00060 
00061  #endif // SERIAL_COMMUNICATION_H
```

7.9 SPII.h

```
00001
00008 #ifndef SPI_COMMUNICATION_H
00009 #define SPI_COMMUNICATION_H
00010
00011 #include "SPII.h"
00012
00013 #include <Arduino.h>
00014
00016 namespace comModule
00017 {
00019     class SPICommunication
00020     {
00021     public:
00022         SPICommunication();
00023         ~SPICommunication();
00024
00029         void beginSPI();
00030
00035         void endSPI();
00036
00043         void spiWrite(uint8_t* data, size_t length);
00044
00051         void spiRead(uint8_t* buffer, size_t length);
00052
00059         bool isInitialized() const;
00060
00061     private:
00062         bool spiInitialized = false;
00063     };
00064 }
00065
00066 #endif // SPI_COMMUNICATION_H
```

7.10 config.h

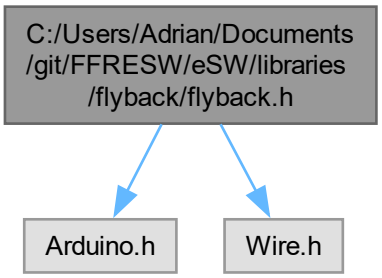
```
00001 // config.h
00002 // FreeROTS Kernel Configuration
00003
00004 #ifndef CONFIG_H
00005 #define CONFIG_H
00006
00007 // Enable Arduino C++ Interface
00008 // This allows the HeliOS kernel to interact with the Arduino API
00009 #define CONFIG_ENABLE_ARDUINO_CPP_INTERFACE
00010
00011 // Enable System Assertions (optional, for debugging purposes)
00012 #define CONFIG_ENABLE_SYSTEM_ASSERT
00013 #define CONFIG_SYSTEM_ASSERT_BEHAVIOR(file, line) __ArduinoAssert__(file, line)
00014
00015 // Message Queue Configuration
00016 #define CONFIG_MESSAGE_VALUE_BYTES 0x8u  // Message queue message value size in bytes
00017
00018 // Task Notification Configuration
00019 #define CONFIG_NOTIFICATION_VALUE_BYTES 0x8u  // Task notification value size in bytes
00020
00021 // Task Name Configuration
00022 #define CONFIG_TASK_NAME_BYTES 0x8u  // Length of task names in bytes
00023
00024 // Memory Region Configuration
00025 #define CONFIG_MEMORY_REGION_SIZE_IN_BLOCKS 0x10u  // Number of memory blocks (16 blocks)
00026 #define CONFIG_MEMORY_REGION_BLOCK_SIZE 0x20u  // Memory block size in bytes (32 bytes)
00027
00028 // Queue Configuration
00029 #define CONFIG_QUEUE_MINIMUM_LIMIT 0x5u  // Minimum queue size limit (5 items)
00030
00031 // Stream Buffer Configuration
00032 #define CONFIG_STREAM_BUFFER_BYTES 0x20u  // Stream buffer length (32 bytes)
00033
00034 // Task Watchdog Timer
00035 #define CONFIG_TASK_WD_TIMER_ENABLE  // Enable watchdog timer for tasks
00036
00037 // Device Name Configuration
00038 #define CONFIG_DEVICE_NAME_BYTES 0x8u  // Device name length (8 bytes)
00039
00040 #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:



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::SwitchStates : int { **HV_Module_OFF** , **HV_Module_MANUAL** , **HV_Module_↩ REMOTE** , **HV_Module_INVALID** }

  enum for different SwitchStates of HVModule

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.

```
00001
00010 #ifndef FLYBACK_H
00011 #define FLYBACK_H
00012
00013 #include <Arduino.h>
00014 #include <Wire.h>
00015
00017 namespace flybackModule
00018 {
00020     enum class SwitchStates : int
00021     {
00022         HV_Module_OFF,
00023         HV_Module_MANUAL,
00024         HV_Module_REMOTE,
00025         HV_Module_INVALID
00026     };
00027
00030     typedef struct Measurement
00031     {
00032         float voltage;
00033         float current;
00034         float power;
00035         int digitalFreqValue;
00036         int digitalDutyValue;
00037         int dutyCycle;
00038         uint32_t frequency;
00039     } meas;
00040
00044     class Flyback
00045     {
00046     public:
00047         Flyback();
00048         ~Flyback();
00049
00054         void initialize();
00055
00060         void deinitialize();
00061
00068         bool isInitialized() const;
00069
00076         bool getTimerState();
00077
00083         void setTimerState(bool state);
00084
00091         SwitchStates getSwitchState();
```

```
00092
00098         Measurement measure();
00099
00104         void run();
00105
00111         void setExternFrequency(uint32_t frequency);
00112
00117         uint32_t getExternFrequency();
00118
00124         void setExternDutyCycle(int dutyCycle);
00125
00130         int getExternDutyCycle();
00131
00132         // TODO MERGE STUFF FROM FRADOM01 into here!
00133         void setExternPsu(bool state);
00134
00135         // TODO MERGE STUFF FROM FRADOM01 into here!
00136         bool getExternPsu();
00137
00143         void regulateVoltage(float targetVoltage, float hysteresis);
00144
00149         void setTargetVoltage(float voltage);
00150
00155         float getTargetVoltage() const;
00156
00157
00162         void setHysteresis(float hysteresis);
00163
00168         float getHysteresis() const;
00169
00170
00171     private:
00172         Measurement meas;
00173
00174         //Define Pins -->Signaltable
00175         static const int Main_Switch_OFF = 27;
00176         static const int Main_Switch_MANUAL = 28;
00177         static const int Main_Switch_REMOTE = 29;
00178         static const int Measure_ADC = A0;          //ADC PIN for Voltage Measurement
00179         static const int PWM_OUT = 11;
00180         static const int PWM_INV = 12;
00181
00182         //Variables for calculating HV
00183         const float R1 = 100000000;
00184         const float R2 = 10000;
00185         const float ADC_Max_Value = 1023.0;
00186         const float Vcc = 5.0;
00187
00188         //Define Pins --> Signaltable
00189         static const int HV_Module_ON = 37;         // Input
00190         static const int HV_Module_Working = 35;    // Output
00191         static const int PWM_Frequency = A1;
00192         static const int PWM_DutyCycle = A2;
00193
00194         // PSU - Power Supply Unit
00195         static const int PSU = 36;      // Output
00196
00197         bool _flybackInitialized;
00198         bool _timerInitialized;
00199
00200         // States
00201         static SwitchStates lastState;
00202         static bool lastTimerState;
00203         static int lastPWMFrequency;
00204         static int lastPWMDutyCycle;
00205
00206         // Default targetVoltage and hysteresisVoltage for to prevent swinging
00207         float _targetVoltage = 0.0f;
00208         float _hysteresis = 0.0f;
00209
00210         // Rate limiting
00211         unsigned long _lastRegulationTime = 0;
00212         const unsigned long _regulationInterval = 100;
00213
00214         // PID vars
00215         float _integral = 0.0f;
00216         float _lastError = 0.0f;
00217         const float _Kp = 1.5f;
00218         const float _Ki = 0.05f;
00219         const float _Kd = 0.1f;
00220
00221         float _integralMin = -10.0f;
00222         float _integralMax = 10.0f;
00223
00224         // Soft start
00225         bool _softStartActive = true;
00226         int _softStartDuty = 1;
```

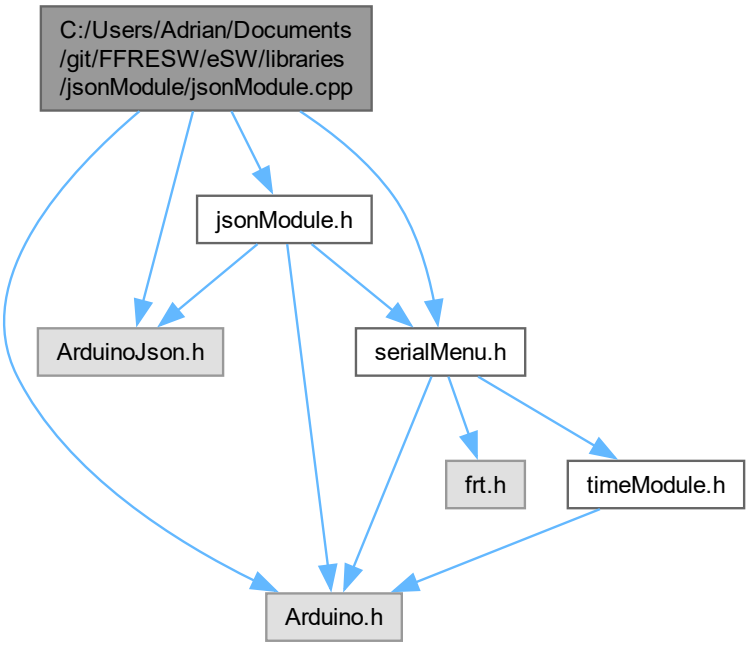
```
00227
00232         void timerConfig();
00233
00240         void setPWMFrequency(uint32_t frequency, int dutyCycle);
00241     };
00242 }
00243
00244 #endif //FLYBACK_H
```

7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/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:



7.13.1 Detailed Description

Implementation of the jsonModule class.

Version

0.2

Date

2025-05-18

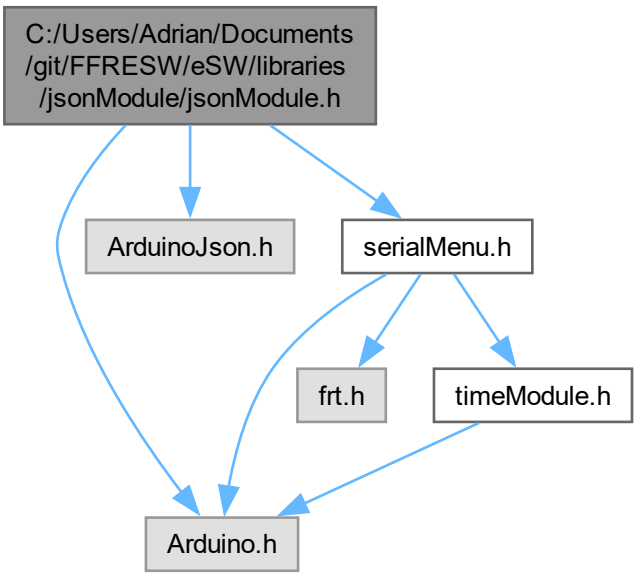
Copyright

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

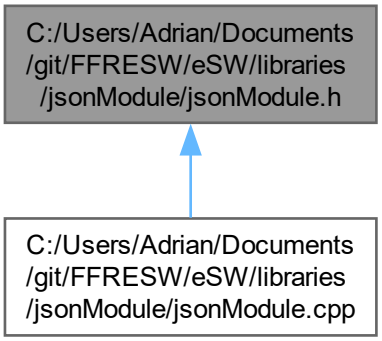
Header for the JsonModuleInternals class.

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

Include dependency graph for jsonModule.h:



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



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.

```
00001
00010 #ifndef JSONMODULE_H
00011 #define JSONMODULE_H
00012
00013 #include <Arduino.h>
00014 #include <ArduinoJson.h>
00015 #include <serialMenu.h>
00016
00018 namespace jsonModule
00019 {
00021     class JsonModuleInternals
00022     {
00023     public:
00024         JsonModuleInternals();
00025         ~JsonModuleInternals();
00026
00034         template<typename T>
00035         void createJson(const char* key, T value)
00036         {
00037             if (!jsonDoc[key].set(value))
00038             {
00039                 SerialMenu::printToSerial(SerialMenu::OutputLevel::ERROR, F("Failed to set JSON
      key."));
00040             }
00041         }
00042
00046         void sendJsonSerial();
00047
00053         String getJsonString() const;
00054
00058         void clearJson();
00059
00063         void printJsonDocMemory();
00064
00072         bool hasCapacityFor(size_t additionalSize) const;
00073
00074         size_t jsonBufferSize;
00075
00076     private:
00077         StaticJsonDocument<512> jsonDoc;
00078     };
00079 }
00080
00081 #endif // JSONMODULE_H
```

7.16 lockerBase.h

```
00001 #ifndef LOCKER_BASE_H
00002 #define LOCKER_BASE_H
00003
00004 #include <frt.h>
00005 #include <Arduino.h>
00006 #include <scopedLock.h>
00007 #include <logManager.h>
00008
00010 class LockerBase
00011 {
00012 public:
00013     LockerBase()
00014     {
00015         if (_logger->isSDCardInitialized())
00016         {
00017             _logger->setLogFileName("log_LockerBase.txt");
00018         }
00019     }
00020
00021     ~LockerBase() {}
00022
00028     locker::ScopedLock lockEthernetScoped()
00029     {
00030         ethernetConnected = true;
00031         logState("Ethernet connected", ethernetConnected);
00032         return locker::ScopedLock(ethernetMutex);
00033     }
00034
00040     locker::ScopedLock lockTemperatureScoped()
00041     {
00042         temperatureReading = true;
```

```
00043         logState("Temperature reading", temperatureReading);
00044         return locker::ScopedLock(temperaturQueueMutex);
00045     }
00046 
00052     locker::ScopedLock lockSerialScoped()
00053     {
00054         serialReading = true;
00055         logState("Serial reading", serialReading);
00056         return locker::ScopedLock(serialMutex);
00057     }
00058 
00059 private:
00060 
00061     // Mutexes for the different resources
00062     frt::Mutex temperaturQueueMutex;
00063     frt::Mutex ethernetMutex;
00064     frt::Mutex serialMutex;
00065 
00066     // Semaphores for the different resources
00067     frt::Semaphore ethernetSemaphore;
00068     frt::Semaphore temperatureSemaphore;
00069     frt::Semaphore serialSemaphore;
00070 
00071     // Flags for the different resources
00072     bool ethernetConnected = false;
00073     bool temperatureReading = false;
00074     bool serialReading = false;
00075 
00076     // Logger instance
00077     LogManager* _logger = LogManager::getInstance();
00078 
00085     void logState(const char* label, bool state)
00086     {
00087         char buffer[64];
00088         snprintf(buffer, sizeof(buffer), "[LockerBase] %s: %s", label, state ? "true" : "false");
00089         _logger->writeToLogFile(buffer);
00090     }
00091 };
00092 
00093 #endif // LOCKER_BASE_H
```

7.17 scopedLock.h

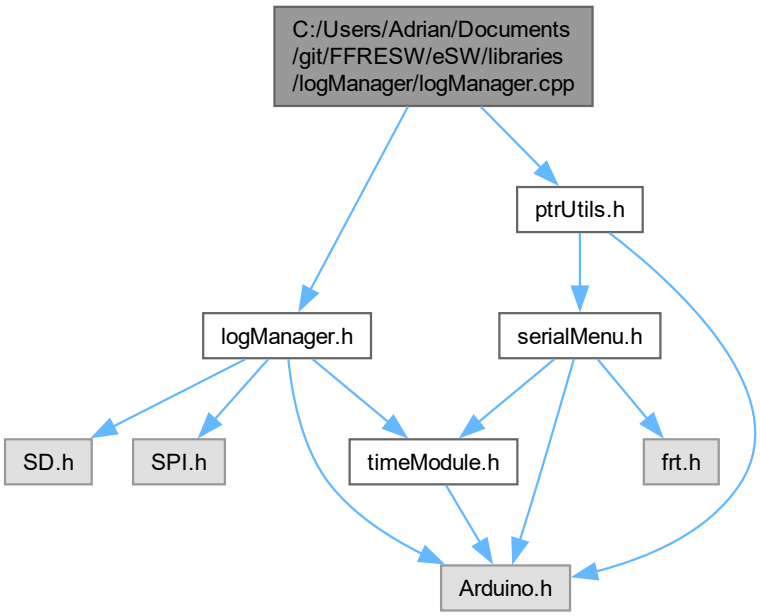
```
00001 #ifndef LOCKER_BASE_SCOPEDLOCK_H
00002 #define LOCKER_BASE_SCOPEDLOCK_H
00003 
00004 #include <frt.h>
00005 
00007 namespace locker
00008 {
00010     class ScopedLock
00011     {
00012     public:
00013 
00019         explicit ScopedLock(frt::Mutex& mutex) : m_mutex(mutex), m_locked(true)
00020         {
00021             m_mutex.lock();
00022         }
00023 
00024         ScopedLock(const ScopedLock&) = delete;
00025         ScopedLock& operator=(const ScopedLock&) = delete;
00026 
00032         ScopedLock(ScopedLock&& other) noexcept : m_mutex(other.m_mutex), m_locked(other.m_locked)
00033         {
00034             other.m_locked = false;
00035         }
00036 
00037         // Deleted: assignment to a reference is illegal in C++
00038         ScopedLock& operator=(ScopedLock&&) = delete;
00039 
00040         ~ScopedLock()
00041         {
00042             if (m_locked)
00043                 m_mutex.unlock();
00044         }
00045 
00046     private:
00047         frt::Mutex& m_mutex;
00048         bool m_locked;
00049     };
00050 }
00051 
00052 #endif // LOCKER_BASE_SCOPEDLOCK_H
```

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

Implementation of the logManager class.

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

Include dependency graph for logManager.cpp:



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

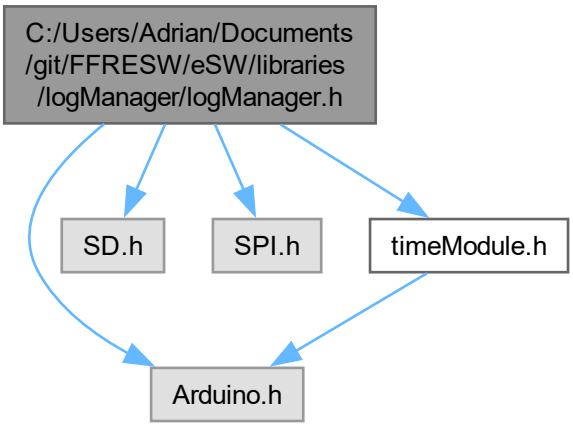
Copyright (c) 2024

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

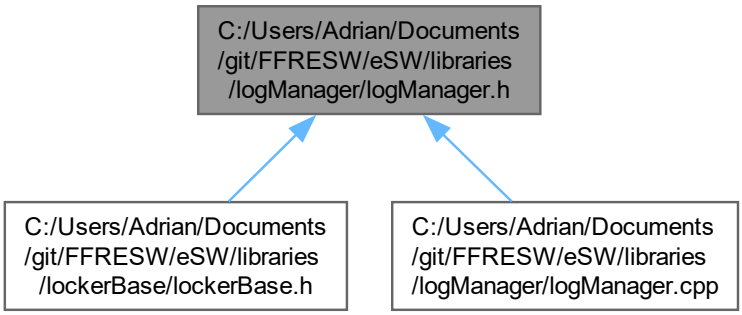
header file for the logManager.

```
#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:



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.

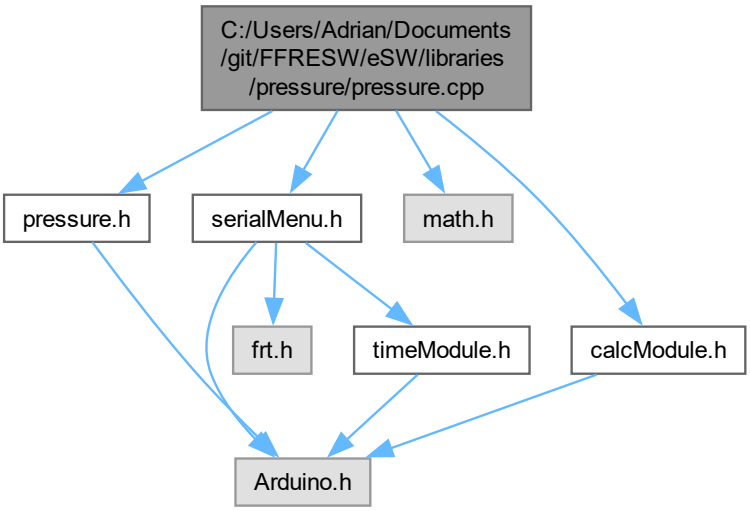
```
00001
00011 #ifndef LOGMANAGER_H
00012 #define LOGMANAGER_H
00013
00014 #include <Arduino.h>
00015 #include <SD.h>
00016 #include <SPI.h>
00017 #include <timeModule.h>
00018
00020 class LogManager
00021 {
00022 public:
00023
00029     static LogManager* getInstance();
00030
00036     void initSDCard(int cs);
00037
00041     void shutdownSDCard();
00042
00046     void flushLogs();
00047
00054     bool isSDCardInitialized() const;
00055
00060     static String getCurrentTime();
00061
00067     void setLogFileName(const String& fileName);
00068
00076     bool writeToLogFile(const String& logMessage);
00077
00084     void renameFile(const String& oldName, const String& newName);
00085
00086 private:
00087     LogManager();
00088     ~LogManager();
00089
00090     static LogManager* _instance;
00091     File logFile;
00092     bool sdCardInitialized = false;
00093     String logFileName;
00094     String baseLogFileName;
00095
00096     static const int chipSelectPinEth = 10; // Default CS pin for SD card
00097     static const long maxLogFileSize = 104857600L; // 100MB Logfile size
00098
00099     LogManager(const LogManager&) = delete;
00100     LogManager& operator=(const LogManager&) = delete;
00101 };
00102
00103
00104 #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

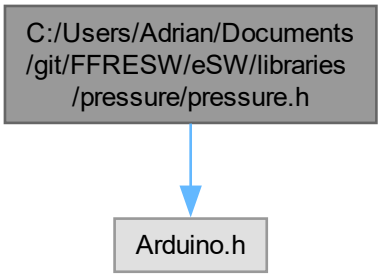
Copyright

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

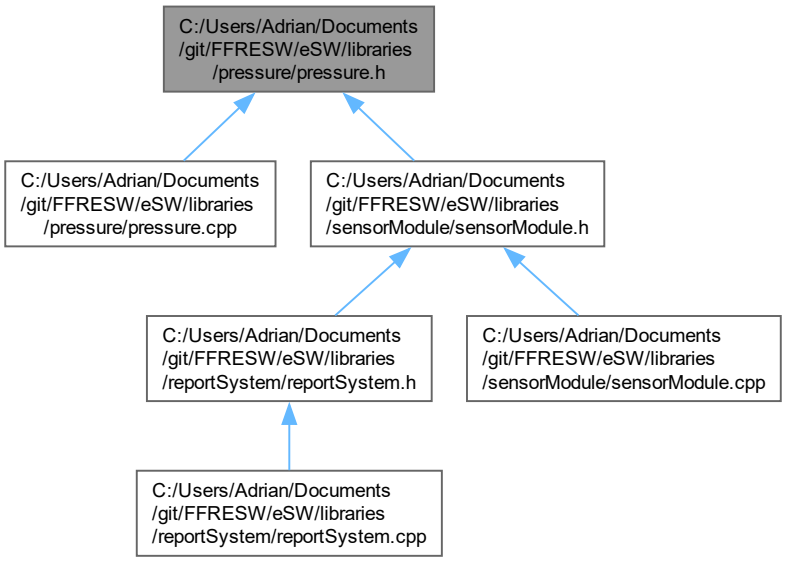
Header file for the pressure library.

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

Include dependency graph for pressure.h:



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



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

7.23 pressure.h

Go to the documentation of this file.

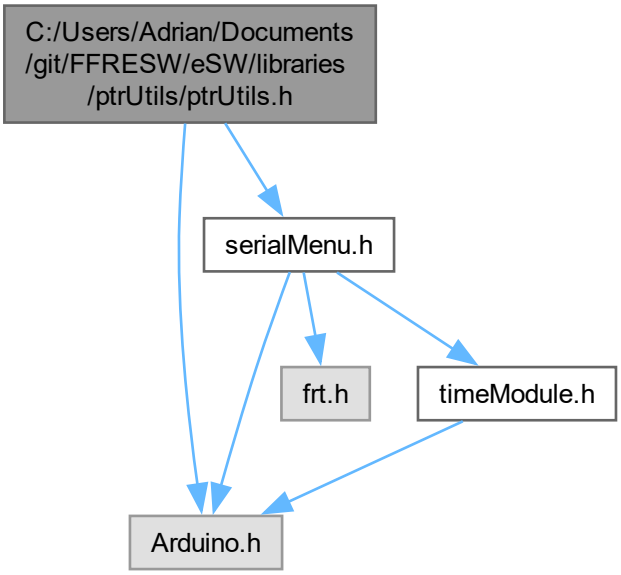
```
00001
00011 #ifndef PRESSURESENSOR_H
00012 #define PRESSURESENSOR_H
00013
00014 #include <Arduino.h>
00015
00017 class PressureSensor
00018 {
00019 public:
00020     PressureSensor();
00021     ~PressureSensor();
00022
00027     void initialize();
00028
00034     float readPressure();
00035
00042     bool isInitialized() const;
00043
00044 private:
00045     bool _pressureSensorInitialized;
00046     static const int PRESSURE_SENSOR_PIN = 0;
00047
00054     float readAnalogSensor(uint8_t pin);
00055
00056 };
00057
00058 #endif // PRESSURESENSOR_H
```

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

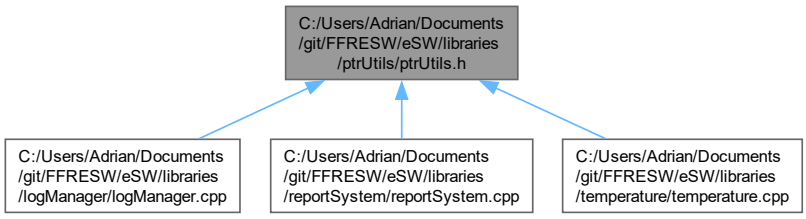
Header only library implementation.

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

Include dependency graph for ptrUtils.h:



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



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.

Macros

- #define tryDeletePtr(ptr)

  Macro to safely delete a pointer and verify it is nullptr.

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 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.

```
00001
00011 #ifndef PTRUTILS_H
00012 #define PTRUTILS_H
00013
00014 #include <Arduino.h>
00015 #include <serialMenu.h>
00016
00023 template <typename T>
00024 static inline void SafeDelete(T*& ptr)
00025 {
00026     if (ptr != nullptr)
00027     {
00028         delete ptr;
00029         ptr = nullptr;
00030     }
00031 }
00032
00039 template <typename T>
00040 static inline void SafeDeleteArray(T*& ptr)
00041 {
00042     if (ptr != nullptr)
00043     {
00044         delete[] ptr;
00045         ptr = nullptr;
00046     }
00047 }
00048
00057 template <typename T>
00058 static inline void Verify(const T& value, const T& expected, const char* errorMsg = nullptr)
00059 {
00060     if (value != expected)
00061     {
00062         if (errorMsg)
00063         {
00064             SerialMenu::printToSerial(errorMsg);
00065         }
00066         else
00067         {
00068             String errStr;
00069             errStr += "[ERROR] Verification failed: Value (";
00070             errStr += value;
00071             errStr += ") does not match expected (";
00072             errStr += expected;
00073             errStr += ").";
00074             SerialMenu::printToSerial(errStr);
00075         }
00076         while (true); // Halt execution
00077     }
00078 }
00079
00088 template <typename T>
00089 static inline void Verify(T* value, T* expected, const char* errorMsg = nullptr)
00090 {
00091     if (value != expected)
00092     {
00093         if (errorMsg)
00094         {
00095             SerialMenu::printToSerial(errorMsg);
00096         }
00097         else
00098         {
00099             String errStr;
00100             errStr += "[ERROR] Verification failed: Pointer (";
00101             errStr += (unsigned long)value, HEX;
00102             errStr += ") does not match expected pointer (";
00103             errStr += (unsigned long)expected, HEX;
00104             errStr += ").";
00105             SerialMenu::printToSerial(errStr);
00106         }
00107         while (true); // Halt execution
00108     }
00109 }
00110
00118 template <typename T>
00119 static inline void Verify(T* value, const char* errorMsg = nullptr)
00120 {
00121     if (value != nullptr)  // Directly compare with nullptr (no std::nullptr_t)
00122     {
00123         if (errorMsg)
00124         {
00125             SerialMenu::printToSerial(errorMsg);
00126         }
```

```
00127         else
00128         {
00129             String errStr;
00130             errStr += "[ERROR] Verification failed: Pointer (";
00131             errStr += (unsigned long)value, HEX;
00132             errStr += ") is not null.";
00133             SerialMenu::printToSerial(errStr);
00134         }
00135         while (true); // Halt execution
00136     }
00137 }
00138 
00139 
00144 #define tryDeletePtr(ptr)                                       \
00145     if (PtrUtils::IsValidPtr(ptr))                              \
00146         SafeDelete(ptr);                                        \
00147 
00148 
00149 
00151 class PtrUtils
00152 {
00153 public:
00160     template <typename T>
00161     static inline bool IsNullPtr(T* ptr)
00162     {
00163         return ptr == nullptr;
00164     }
00165 
00172     template <typename T>
00173     static inline bool IsValidPtr(T* ptr)
00174     {
00175         return ptr != nullptr;
00176     }
00177 };
00178 
00186 template <typename T>
00187 static inline void ClearArray(T* array, size_t size)
00188 {
00189     for (size_t i = 0; i < size; ++i)
00190     {
00191         array[i] = T();
00192     }
00193 }
00194 
00202 template <typename T>
00203 static inline void PrintPtrInfo(T* ptr, const char* ptrName = "Pointer")
00204 {
00205     if (ptr == nullptr)
00206     {
00207         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String("is nullptr"));
00208     }
00209     else
00210     {
00211         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String(" points to address: 0x") +
    (uintptr_t)ptr, HEX);
00212     }
00213 }
00214 
00220 template <typename T>
00221 class ScopedPointer
00222 {
00223 private:
00224     T* ptr;
00225 
00226 public:
00227     explicit ScopedPointer(T* p = nullptr) : ptr(p) {}
00228     ~ScopedPointer() { SafeDelete(ptr); }
00229 
00235     T* get() const { return ptr; }
00236 
00242     T* release()
00243     {
00244         T* temp = ptr;
00245         ptr = nullptr;
00246         return temp;
00247     }
00248 
00254     void reset(T* p = nullptr)
00255     {
00256         SafeDelete(ptr);
00257         ptr = p;
00258     }
00259 
00265     T& operator*() const { return *ptr; }
00266 
00272     T* operator->() const { return ptr; }
00273 };
```

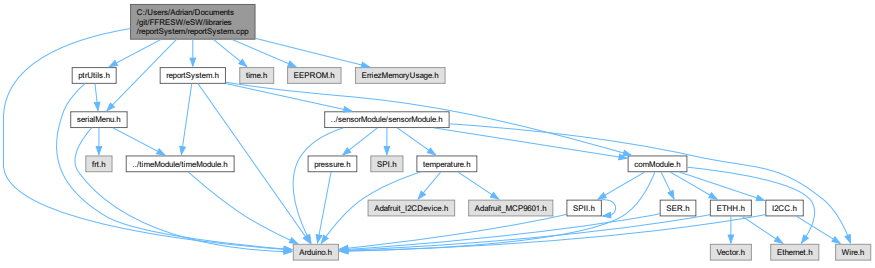
```
00274
00280 template <typename T>
00281 class PointerWrapper
00282 {
00283 private:
00284     T* ptr;
00285
00286 public:
00287     explicit PointerWrapper(T* p = nullptr) : ptr(p) {}
00288     ~PointerWrapper() { SafeDelete(ptr); }
00289
00295     T* get() const { return ptr; }
00296
00302     T* release()
00303     {
00304         T* temp = ptr;
00305         ptr = nullptr;
00306         return temp;
00307     }
00308
00314     void reset(T* p = nullptr)
00315     {
00316         SafeDelete(ptr);
00317         ptr = p;
00318     }
00319
00325     T& operator*() { return *ptr; }
00326
00332     T* operator->() { return ptr; }
00333 };
00334
00335 #endif // PTRUTILS_H
```

7.26 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/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

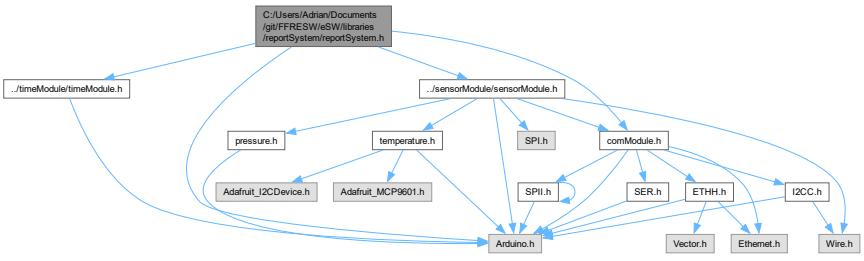
Copyright (c) 2024

7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h File Reference

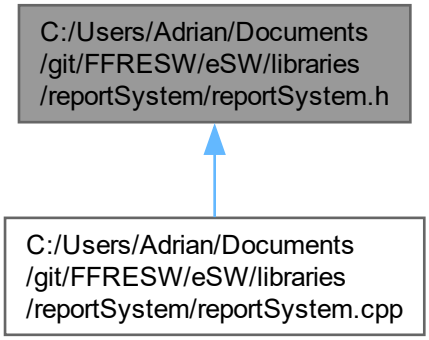
Header file for the ReportSystem.

```
#include <Arduino.h>
#include "../sensorModule/sensorModule.h"
#include "../comModule/comModule.h"
#include "../timeModule/timeModule.h"
```

Include dependency graph for reportSystem.h:



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



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

7.28 reportSystem.h

Go to the documentation of this file.

```
00001
00011 #ifndef REPORTSYSTEM_H
00012 #define REPORTSYSTEM_H
00013
00014 #include <Arduino.h>
00015 #include "../sensorModule/sensorModule.h"
00016 #include "../comModule/comModule.h"
00017 #include "../timeModule/timeModule.h"
00018
00019 #define STACK_GUARD 0xDEAD // Stack guard value
00020 extern volatile uint16_t stackCheck; // Stack check variable
00021
00022 #define EEPROM_ERROR_ADDR 0
00023
00025 namespace reportSystem
00026 {
00028     class ReportSystem
00029     {
00030     public:
00031         ReportSystem();
00032         ~ReportSystem();
00033
00039         void reportError(const char* errorMessage);
00040
00053         bool checkSystemHealth(size_t memoryThreshold, bool checkEth,
00054                                bool checkSpi, bool checkI2c,
00055                                bool checkTemp, bool checkPress);
00056
00063         String reportStatus(bool active);
00064
00071         void setThreshold(float tempThreshold, float pressureThreshold);
00072
00081         bool checkThresholds(float currentTemp, float currentPressure);
00082
00088         String getCurrentTime();
00089
00095         String getMemoryStatus();
00096
00102         String getStackDump();
00103
00108         void startBusyTime();
00109
00114         void startIdleTime();
00115
00121         float getCPULoad();
00122
00127         void resetUsage();
00128
00133         static void initStackGuard();
00134
00141         static bool detectStackOverflow();
00142
00149         void saveLastError(const char* error);
00150
00156         String getLastError();
00157
00163         bool getLastErrorInfo();
00164
00174         bool checkRamLevel(unsigned int warningThreshold, unsigned int criticalThreshold);
00175
00181         bool isTemperatureSensorOK() const;
00182
00188         bool isCommunicationOK() const;
00189
00195         bool isMemoryOK() const;
00196
00202         bool isRamOK() const;
00203
00209         bool isStackSafe() const;
00210
00216         bool hasNoSavedErrors() const;
00217
00218     private:
00219         float tempThreshold;
00220         float pressureThreshold;
00221         unsigned long lastHealthCheck;
00222         const unsigned long healthCheckInterval = 10000; // 10 seconds
00223         unsigned long busyTime = 0;
00224         unsigned long idleTime = 0;
00225         unsigned long lastTimestamp = 0;
00226
00235         bool checkSensors(bool checkTemp, bool checkPress);
```

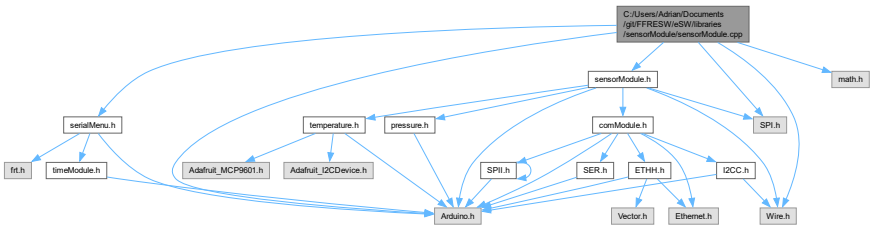
```
00236
00246         bool checkCommunication(bool checkEth, bool checkSpi, bool checkI2c);
00247
00255         bool checkMemory(unsigned int threshold);
00256
00257         sensorModule::SensorModuleInternals* _sens;
00258         comModule::ComModuleInternals* _com;
00259         timeModule::TimeModuleInternals* _time;
00260     };
00261 }
00262
00263 #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

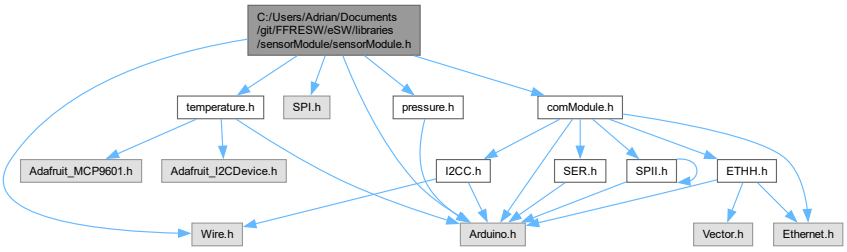
Copyright

7.30 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensor↩ Module/sensorModule.h File Reference

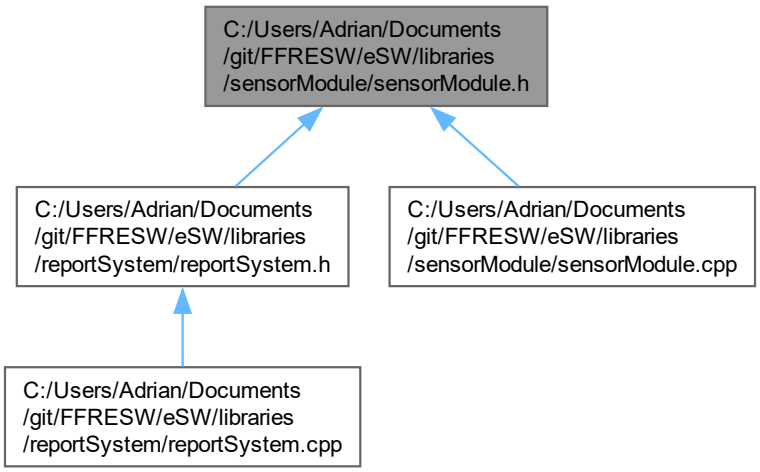
header file for the sensorModule.

```
#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:



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

7.31 sensorModule.h

Go to the documentation of this file.

```
00001
00011 #ifndef SENSORMODULE_H
00012 #define SENSORMODULE_H
00013
00014 #include <Arduino.h>
00015 #include <Wire.h>
00016 #include <SPI.h>
00017 #include <pressure.h>
00018 #include <temperature.h>
00019 #include <comModule.h>
00020
00021
00023 namespace sensorModule
00024 {
00026     enum class SensorType
00027     {
00028         TEMPERATURE,
00029         OBJECTTEMPERATURE,
00030         AMBIENTTEMPERATURE,
00031         PRESSURE,
```

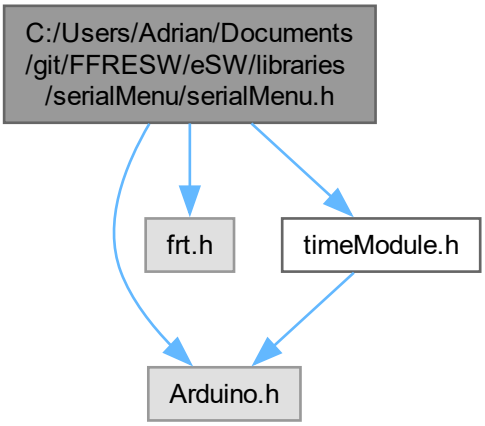
```
00032         DHT11,
00033         MCP9601_Celsius_Indoor,
00034         MCP9601_Fahrenheit_Indoor,
00035         MCP9601_Kelvin_Indoor,
00036         MCP9601_Celsius_Outdoor,
00037         MCP9601_Fahrenheit_Outdoor,
00038         MCP9601_Kelvin_Outdoor,
00039         UNKNOWN
00040     };
00041
00043     class SensorModuleInternals : public TemperatureSensor, public PressureSensor
00044     {
00045     public:
00046         SensorModuleInternals();
00047         ~SensorModuleInternals();
00048
00053         void initialize();
00054
00061         float readSensor(SensorType type);
00062
00070         bool calibrateSensor(SensorType type);
00071
00079         bool checkSensorStatus(SensorType type);
00080
00087         void reportUnknownSensorOnce(SensorType type, const __FlashStringHelper* context);
00088
00089     private:
00090         TemperatureSensor _temperatureSensor;
00091         PressureSensor _pressureSensor;
00092
00093         bool _i2cSensorInitialized;
00094         bool _spiSensorInitialized;
00095
00096         SensorType _lastUnknownSensorType;
00097         bool _unknownSensorReported;
00098     };
00099 }
00100
00101 #endif // SENSORMODULE_H
```

7.32 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h File Reference

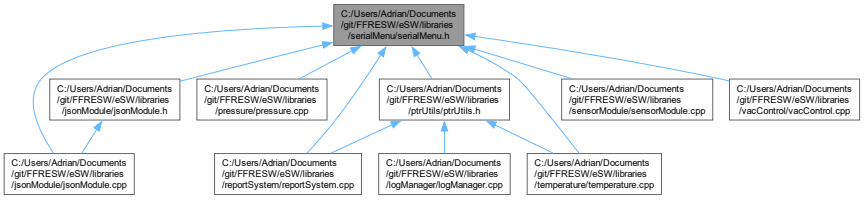
Header file for the serial menu handling serial menu interaction, logging...

```
#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:



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

7.33 serialMenu.h

Go to the documentation of this file.

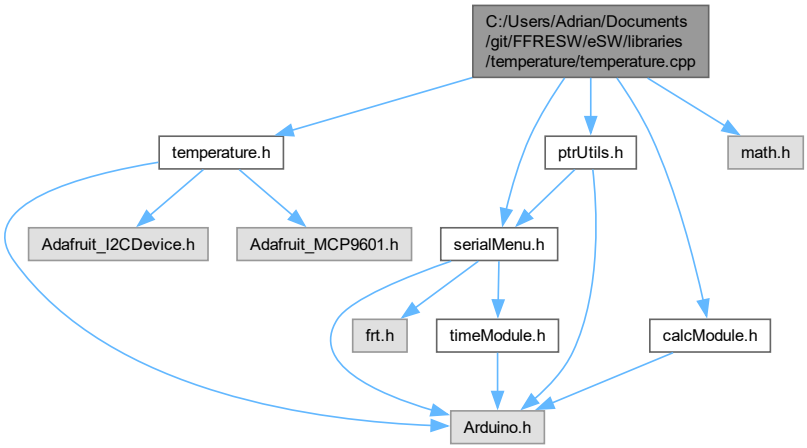
```
00001
00011 #ifndef SERIAL_MENU_H
00012 #define SERIAL_MENU_H
00013
00014 #include <Arduino.h>
00015 #include <frt.h>
00016 #include <timeModule.h>
00017
00018
00020 struct MenuItem
00021 {
00022     const char* label;
00023     char key;
00024     void (*callback)();
00025 };
00026
00028 class SerialMenu
00029 {
00030 public:
00031
00033     enum class OutputLevel
00034     {
00035         DEBUG,
00036         INFO,
00037         WARNING,
00038         ERROR,
00039         CRITICAL,
00040         STATUS,
00041         PLAIN
00042     };
00043
00044     SerialMenu();
00045     ~SerialMenu();
00046
00053     void load(MenuItem* items, size_t size);
00054
00059     void show();
00060
00065     void run();
00066
00074     static void printToSerial(OutputLevel level, const String& message, bool newLine = true, bool
      logMessage = false);
00075
00083     static void printToSerial(OutputLevel level, const __FlashStringHelper* message, bool newLine =
      true, bool logMessage = false);
00084
00091     static void printToSerial(const String& message, bool newLine = true, bool logMessage = false);
00092
00099     static void printToSerial(const __FlashStringHelper* message, bool newLine = true, bool logMessage
      = false);
00100
00105     static String getCurrentTime();
00106
00107 private:
00108     MenuItem* currentMenu;
00109     size_t menuSize;
00110 };
00111
00112 #endif // SERIAL_MENU_H
```

7.34 C:/Users/Adrian/Documents/git/FFRESW/eSW/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

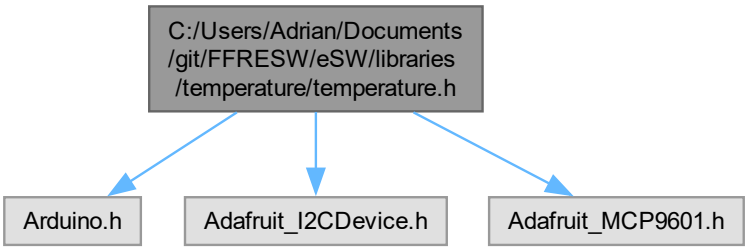
Copyright

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

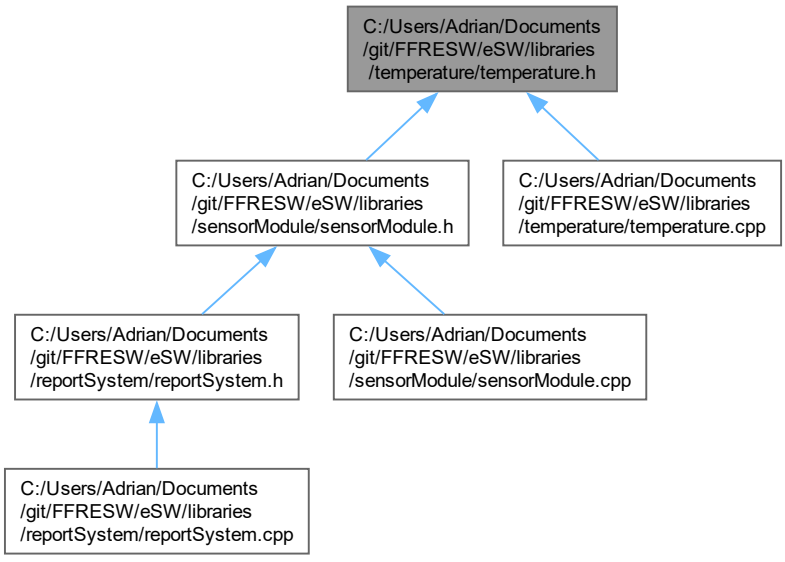
Header file for the temperature library.

```
#include <Arduino.h>
#include <Adafruit_I2CDevice.h>
#include "Adafruit_MCP9601.h"
```

Include dependency graph for temperature.h:



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



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

7.36 temperature.h

Go to the documentation of this file.

```
00001
00011 #ifndef TEMPERATURESENSOR_H
00012 #define TEMPERATURESENSOR_H
00013
00014 #include <Arduino.h>
00015 #include <Adafruit_I2CDevice.h>
00016 #include "Adafruit_MCP9601.h"
00017
00019 enum Units
00020 {
00021     Celsius,
00022     Kelvin,
00023     Fahrenheit
00024 };
00025
00027 enum MCP9601_Status : uint8_t
00028 {
00029     MCP9601_OPENCIRCUIT = 0x10,
00030     MCP9601_SHORTCIRCUIT = 0x20
00031 };
00032
00033 // @brief Enum for the Different Sensors in different Environments \enum SensorID
00034 enum SensorID
00035 {
00036     INDOOR, // Black-Color Patch on Cable
00037     OUTDOOR // Yellow-Color Patch on Cable
00038 };
00039
00041 class TemperatureSensor
00042 {
```

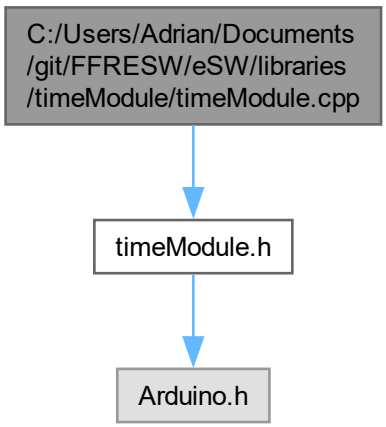
```
00043 public:
00044     TemperatureSensor();
00045     ~TemperatureSensor();
00046 
00051     void initialize();
00052 
00058     float readTemperature();
00059 
00066     float readMCP9601(Units unit, SensorID sensor);
00067 
00074     bool isInitialized() const;
00075 
00076 
00082     uint8_t calibMCP9601(SensorID sensor);
00083 
00084 private:
00085     bool _temperatureSensorInitialized;
00086     static const int TEMP_SENSOR_PIN = A0;
00087     static const int TEMP_SENSOR_PIN_DIG = 4;
00088     static const int DHT11_PIN = 7;
00089 
00090     static const uint8_t MLX90614 = 0x5A;
00091     static const uint8_t AMBIENT_TEMP = 0x06;
00092     static const uint8_t OBJECT_TEMP = 0x07;
00093 
00094     // Settings for the MCP9601 Sensor board
00095     Adafruit_MCP9601 _mcp1;
00096     Adafruit_MCP9601 _mcp2;
00097     Ambient_Resolution _ambientREs = RES_ZERO_POINT_0625;
00098     static const uint8_t MCP9601_I2C = 0x67;
00099 
00100 
00107     float readAnalogSensor(uint8_t pin);
00108 
00115     float readDigitalSensor(uint8_t pin);
00116 };
00117 
00118 #endif // TEMPERATURESENSOR_H
```

7.37 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp File Reference

Implementation of the timeModule class.

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

Include dependency graph for timeModule.cpp:



7.37.1 Detailed Description

Implementation of the timeModule class.

Version

0.1

Date

2024-01-26

Copyright

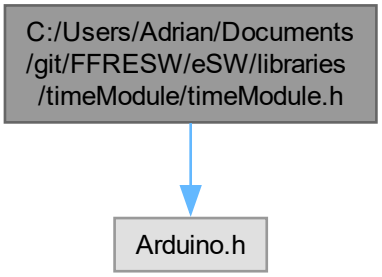
Copyright (c) 2024

7.38 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h File Reference

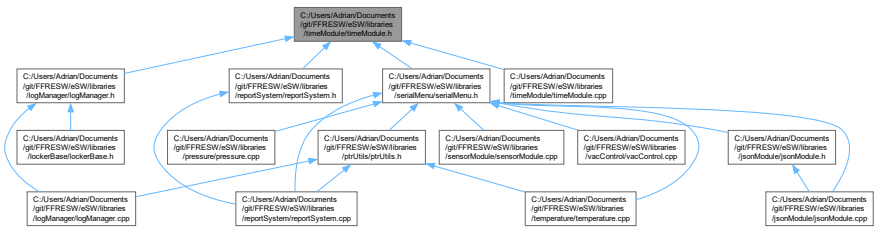
Header file for the time module handling systemtime for logging, api ...

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

Include dependency graph for timeModule.h:



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



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

7.39 timeModule.h

Go to the documentation of this file.

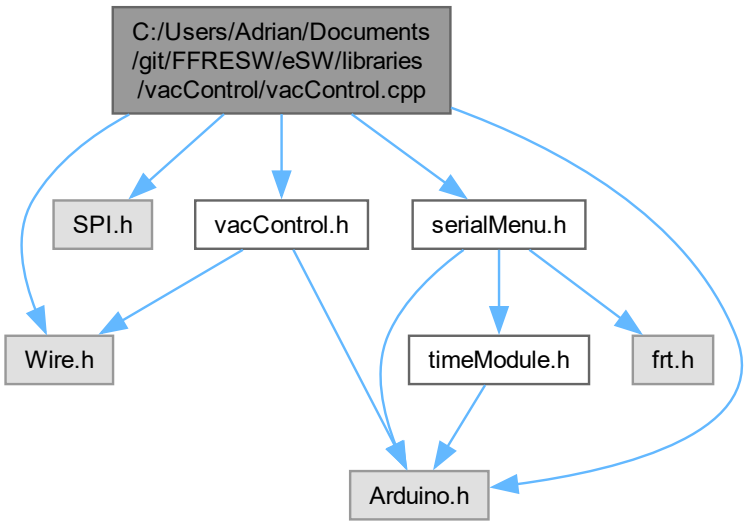
```
00001
00011 #ifndef TIMEMODULE_H
00012 #define TIMEMODULE_H
00013
00014 #include <Arduino.h>
00015
00017 namespace timeModule
00018 {
00020     typedef struct DateTimeStruct
00021     {
00022         int year;
00023         int month;
00024         int day;
00025         int hour;
00026         int minute;
00027         int second;
00028     } DateTimeStruct;
00029
00031     class TimeModuleInternals
00032     {
00033     public:
00034         TimeModuleInternals();
00035         ~TimeModuleInternals();
00036
00042         static void incrementTime(DateTimeStruct *dt);
00043
00050         static String formatTimeString(const DateTimeStruct &dt);
00051
00059         bool setTimeFromHas(const String& timeString);
00060
00066         void setSystemTime(const DateTimeStruct& dt);
00067
00072         void updateSoftwareClock();
00073
00079         DateTimeStruct getSystemTime();
00080
00086         static TimeModuleInternals* getInstance();
00087
00088     private:
00089         DateTimeStruct dt;
00090         unsigned long startMillis = 0;
00091     };
00092 }
00093
00094 #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:



7.40.1 Detailed Description

Implementation of the vacControl class.

Version

0.1

Date

2024-01-26

Copyright

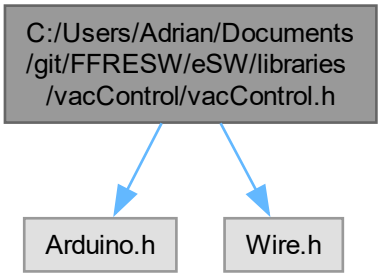
Copyright (c) 2024

7.41 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vac↩ Control/vacControl.h File Reference

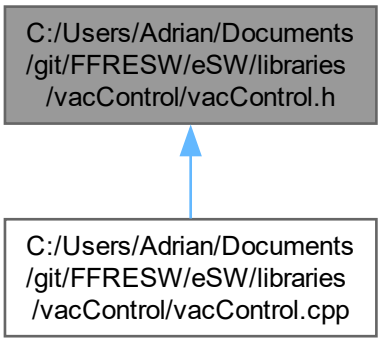
Header for the vacControl class.

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

Include dependency graph for vacControl.h:



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



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::SwitchStates : int {
  Main_Switch_OFF , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** ,
  PUMP_ON , **PUMP_OFF** }

  Enum to represent the states of the main switch and pump.
- 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.

```
00001
00010 #ifndef VACCONTROL_H
00011 #define VACCONTROL_H
00012
00013 #include <Arduino.h>
00014 #include <Wire.h>
00015
00016
00018 namespace vacControlModule
00019 {
00021     enum class SwitchStates : int
00022     {
00023         Main_Switch_OFF,
00024         Main_Switch_MANUAL,
00025         Main_Switch_REMOTE,
00026         Main_switch_INVALID,
00027         PUMP_ON,
00028         PUMP_OFF
```

```
00029     };
00032     typedef struct Pressure
00033     {
00034         float pressure;
00035
00036     } meas;
00037
00039     enum Scenarios
00040     {
00041         Scenario_1 = 0,
00042         Scenario_2 = 1,
00043         Scenario_3 = 2,
00044         Scenario_4 = 3,
00045         Scenario_5 = 4,
00046         Invalid_Scenario = -1
00047     };
00048
00052     class VacControl
00053     {
00054     public:
00055
00056         VacControl();
00057         ~VacControl();
00058
00063         void initialize();
00064
00069         void deinitialize();
00070
00077         bool isInitialized() const;
00078
00084         SwitchStates getSwitchState();
00085
00091         Scenarios getScenario();
00092
00098         Pressure measure();
00099
00106         void setVacuumLed(float pressure, float targetPressure);
00107
00114         int getScenarioFromPotValue(int potValue);
00115
00121         void setPump(bool flag);
00122
00128         void run();
00129
00130
00136         void setExternScenario(int pressure);
00137
00143         int getExternScenario();
00144
00150         void externProcess();
00151
00157         void setExternPressure(float pressure);
00158
00164         float getExternPressure();
00165
00166
00167     private:
00168         Pressure meas;
00169
00170         //Define Pins --> Main_Switch
00171         static const int Main_Switch_OFF = 27;       //Main_Switch OFF Mode 27
00172         static const int Main_Switch_MANUAL = 28;    //Main_Switch Manual Mode 28
00173         static const int Main_Switch_REMOTE = 29;    //Main_Switch Remote Mode 29
00174
00175         //Define Pins --> Vacuum Logic
00176         static const int Switch_Pump_ON = 23;        //Button to turn Pump ON 37
00177         static const int Pump_Status_LED = 24;       //OUTPUT to see State off Pump
00178         static const int Pump_Relay = 25;            //OUTPUT to turn on/off Relais
00179         static const int targetVacuumLED = 26;       //OUTPUT to see Vacuum reached
00180         static const int targetPressure = A4;    //Potentiometer for Regulation
00181
00182         //Variables to save Values
00183         int currentScenario = -1;
00184
00185         //Variables for TargetPressure
00186         static const float TARGET_PRESSURE_1 = 1;
00187         static const float TARGET_PRESSURE_2 = 0.8f;
00188         static const float TARGET_PRESSURE_3 = 0.5;
00189         static const float TARGET_PRESSURE_4 = 0.01f;
00190
00191         // TODO NEW ADDED-> CHECK FUNCTIONALTY WITH FRANIC
00192         static int lastState;
00193         static int lastPumpState;
00194
00195
00196         bool _vacControlInitialized;
00197     };
```

```
00198 }
00199
00200 #endif //VACCONTROL_H
```

Index