

FFRESW

Generated by Doxygen 1.9.8

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 calcModule Namespace Reference	9
5.1.1 Detailed Description	9
5.2 comModule Namespace Reference	9
5.2.1 Detailed Description	10
5.3 flybackModule Namespace Reference	10
5.3.1 Detailed Description	11
5.4 jsonModule Namespace Reference	11
5.4.1 Detailed Description	11
5.5 locker Namespace Reference	11
5.5.1 Detailed Description	12
5.6 reportSystem Namespace Reference	12
5.6.1 Detailed Description	12
5.7 sensorModule Namespace Reference	12
5.7.1 Detailed Description	12
5.8 timeModule Namespace Reference	12
5.8.1 Detailed Description	13
5.9 vacControlModule Namespace Reference	13
5.9.1 Detailed Description	13
6 Class Documentation	15
6.1 calcModule::CalcModuleInternals Class Reference	15
6.1.1 Member Function Documentation	16
6.1.1.1 atmToPascal()	16
6.1.1.2 calculateAverage()	16
6.1.1.3 calculateCurrent()	16
6.1.1.4 calculatePower()	17
6.1.1.5 calculateResistance()	17
6.1.1.6 calculateStandardDeviation()	17
6.1.1.7 celsiusToFahrenheit()	18
6.1.1.8 celsiusToKelvin()	18
6.1.1.9 extractFloat()	19

6.1.1.10 fahrenheitToCelsius()	19
6.1.1.11 findMaximum()	19
6.1.1.12 findMedian()	20
6.1.1.13 findMinimum()	20
6.1.1.14 kelvinToCelsius()	20
6.1.1.15 pascalToAtm()	21
6.1.1.16 pascalToPsi()	21
6.1.1.17 psiToPascal()	21
6.2 calcModuleInternals Class Reference	22
6.2.1 Detailed Description	22
6.3 comModule::ComModuleInternals Class Reference	22
6.3.1 Member Function Documentation	23
6.3.1.1 getEthernet()	23
6.3.1.2 getI2C()	23
6.3.1.3 getSerial()	23
6.3.1.4 getSPI()	23
6.4 comModuleInternals Class Reference	24
6.4.1 Detailed Description	24
6.5 timeModule::DateTimeStruct Struct Reference	24
6.5.1 Detailed Description	24
6.6 comModule::EthernetCommunication Class Reference	24
6.6.1 Detailed Description	26
6.6.2 Member Function Documentation	26
6.6.2.1 beginEthernet()	26
6.6.2.2 getClient()	26
6.6.2.3 getCompound() [1/3]	26
6.6.2.4 getCompound() [2/3]	27
6.6.2.5 getCompound() [3/3]	28
6.6.2.6 getCompoundInternal()	28
6.6.2.7 getParameter()	29
6.6.2.8 getParsedCompound() [1/3]	29
6.6.2.9 getParsedCompound() [2/3]	30
6.6.2.10 getParsedCompound() [3/3]	31
6.6.2.11 getRequestedEndpoint()	31
6.6.2.12 getSendDataFlag()	31
6.6.2.13 getSpecificEndpoint()	31
6.6.2.14 isInitialized()	32
6.6.2.15 parseCompoundResponse()	32
6.6.2.16 receiveEthernetData()	33
6.6.2.17 sendCommand()	33
6.6.2.18 sendEthernetData()	33
6.6.2.19 sendJsonResponse()	34

6.6.2.20 setCompound() [1/3]	34
6.6.2.21 setCompound() [2/3]	34
6.6.2.22 setCompound() [3/3]	35
6.6.2.23 setCompoundInternal()	35
6.6.2.24 setParameter()	36
6.6.2.25 setSendDataFlag()	37
6.7 flybackModule::Flyback Class Reference	37
6.7.1 Detailed Description	38
6.7.2 Member Function Documentation	38
6.7.2.1 getSwitchState()	38
6.7.2.2 getTimerState()	38
6.7.2.3 isInitialized()	39
6.7.2.4 measure()	39
6.7.2.5 setExternDutyCycle()	39
6.7.2.6 setExternFrequency()	40
6.7.2.7 setTimerState()	40
6.8 comModule::I2CCommunication Class Reference	40
6.8.1 Detailed Description	41
6.8.2 Member Function Documentation	41
6.8.2.1 beginI2C()	41
6.8.2.2 i2cRead()	41
6.8.2.3 i2cWrite()	42
6.8.2.4 isInitialized()	42
6.9 jsonModule::JsonModuleInternals Class Reference	42
6.9.1 Member Function Documentation	43
6.9.1.1 createJson()	43
6.9.1.2 createJsonFloat()	43
6.9.1.3 createJsonInt()	44
6.9.1.4 createJsonString()	44
6.9.1.5 createJsonStringConst()	44
6.9.1.6 getJsonString()	44
6.9.1.7 mapJsonToDoubles()	45
6.9.1.8 sendJsonEthernet()	46
6.10 jsonModuleInternals Class Reference	46
6.10.1 Detailed Description	46
6.11 LockerBase Class Reference	47
6.11.1 Detailed Description	47
6.11.2 Member Function Documentation	47
6.11.2.1 lockEthernetScoped()	47
6.11.2.2 lockSerialScoped()	47
6.11.2.3 lockTemperatureScoped()	47
6.12 LogManager Class Reference	48

6.12.1 Member Function Documentation	48
6.12.1.1 getCurrentTime()	48
6.12.1.2 getInstance()	49
6.12.1.3 initSDCard()	50
6.12.1.4 isSDCardInitialized()	51
6.12.1.5 renameFile()	52
6.12.1.6 setLogFileName()	53
6.12.1.7 writeToLogFile()	53
6.13 LogMapper Class Reference	55
6.13.1 Detailed Description	55
6.14 flybackModule::Measurement Struct Reference	55
6.14.1 Detailed Description	55
6.15 Measurement Struct Reference	55
6.15.1 Detailed Description	56
6.16 MenuItem Struct Reference	56
6.16.1 Detailed Description	56
6.17 Outputlevel Class Reference	56
6.17.1 Detailed Description	56
6.18 PointerWrapper< T > Class Template Reference	57
6.18.1 Detailed Description	57
6.18.2 Member Function Documentation	57
6.18.2.1 get()	57
6.18.2.2 operator*()	58
6.18.2.3 operator->()	58
6.18.2.4 release()	58
6.18.2.5 reset()	58
6.19 vacControlModule::Pressure Struct Reference	59
6.20 PressureSensor Class Reference	59
6.20.1 Detailed Description	59
6.20.2 Member Function Documentation	60
6.20.2.1 isInitialized()	60
6.20.2.2 readPressure()	60
6.21 PtrUtils Class Reference	61
6.21.1 Detailed Description	61
6.21.2 Member Function Documentation	61
6.21.2.1 IsNullPtr()	61
6.21.2.2 IsValidPtr()	62
6.22 reportSystem::ReportSystem Class Reference	63
6.22.1 Detailed Description	64
6.22.2 Member Function Documentation	64
6.22.2.1 checkRamLevel()	64
6.22.2.2 checkSystemHealth()	65

6.22.2.3 checkThresholds()	66
6.22.2.4 detectStackOverflow()	66
6.22.2.5 getCPULoad()	67
6.22.2.6 getCurrentTime()	67
6.22.2.7 getLastError()	67
6.22.2.8 getLastErrorInfo()	68
6.22.2.9 getMemoryStatus()	68
6.22.2.10 getStackDump()	68
6.22.2.11 reportError()	68
6.22.2.12 reportStatus()	69
6.22.2.13 saveLastError()	70
6.22.2.14 setThreshold()	70
6.23 locker::ScopedLock Class Reference	71
6.23.1 Detailed Description	71
6.23.2 Constructor & Destructor Documentation	71
6.23.2.1 ScopedLock() [1/2]	71
6.23.2.2 ScopedLock() [2/2]	72
6.24 ScopedPointer< T > Class Template Reference	72
6.24.1 Detailed Description	72
6.24.2 Member Function Documentation	73
6.24.2.1 get()	73
6.24.2.2 operator*()	73
6.24.2.3 operator->()	73
6.24.2.4 release()	73
6.24.2.5 reset()	73
6.25 sensorModule::SensorModuleInternals Class Reference	74
6.25.1 Detailed Description	75
6.25.2 Member Function Documentation	75
6.25.2.1 calibrateSensor()	75
6.25.2.2 checkSensorStatus()	76
6.25.2.3 initialize()	77
6.25.2.4 readSensor()	77
6.26 comModule::SerialCommunication Class Reference	78
6.26.1 Detailed Description	78
6.26.2 Member Function Documentation	78
6.26.2.1 beginSerial()	78
6.26.2.2 isInitialized()	78
6.26.2.3 receiveSerialData()	79
6.26.2.4 sendSerialData()	79
6.27 SerialMenu Class Reference	79
6.27.1 Detailed Description	80
6.27.2 Member Function Documentation	80

6.27.2.1 <code>getCurrentTime()</code>	80
6.27.2.2 <code>load()</code>	82
6.27.2.3 <code>printToSerial()</code> [1/4]	83
6.27.2.4 <code>printToSerial()</code> [2/4]	83
6.27.2.5 <code>printToSerial()</code> [3/4]	84
6.27.2.6 <code>printToSerial()</code> [4/4]	84
6.28 <code>comModule::SPICommunication</code> Class Reference	87
6.28.1 Detailed Description	87
6.28.2 Member Function Documentation	87
6.28.2.1 <code>isInitialized()</code>	87
6.28.2.2 <code>spiRead()</code>	87
6.28.2.3 <code>spiWrite()</code>	88
6.29 <code>TemperatureSensor</code> Class Reference	88
6.29.1 Detailed Description	89
6.29.2 Member Function Documentation	89
6.29.2.1 <code>isInitialized()</code>	89
6.29.2.2 <code>readDht11()</code>	89
6.29.2.3 <code>readMLX90614()</code>	90
6.29.2.4 <code>readTemperature()</code>	91
6.30 <code>timeModule::TimeModuleInternals</code> Class Reference	91
6.30.1 Detailed Description	92
6.30.2 Member Function Documentation	92
6.30.2.1 <code>formatTimeString()</code>	92
6.30.2.2 <code>getInstance()</code>	93
6.30.2.3 <code>getSystemTime()</code>	94
6.30.2.4 <code>incrementTime()</code>	95
6.30.2.5 <code>setSystemTime()</code>	96
6.30.2.6 <code>setTimeFromHas()</code>	97
6.31 <code>vacControlModule::VacControl</code> Class Reference	97
6.31.1 Detailed Description	98
6.31.2 Member Function Documentation	98
6.31.2.1 <code>externProcess()</code>	98
6.31.2.2 <code>getExternPressure()</code>	99
6.31.2.3 <code>getExternScenario()</code>	99
6.31.2.4 <code>getScenario()</code>	99
6.31.2.5 <code>getScenarioFromPotValue()</code>	99
6.31.2.6 <code>getSwitchState()</code>	100
6.31.2.7 <code>isInitialized()</code>	100
6.31.2.8 <code>measure()</code>	101
6.31.2.9 <code>run()</code>	101
6.31.2.10 <code>setExternPressure()</code>	101
6.31.2.11 <code>setExternScenario()</code>	102

6.31.2.12 setPump()	102
6.31.2.13 setVacuumLed()	103
7 File Documentation	105
7.1 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp File Reference	105
7.1.1 Detailed Description	105
7.2 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h File Reference	106
7.2.1 Detailed Description	107
7.3 calcModule.h	107
7.4 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp File Reference	108
7.4.1 Detailed Description	109
7.5 comModule.h	109
7.6 ETHH.h	109
7.7 I2CC.h	111
7.8 SER.h	112
7.9 SPII.h	112
7.10 config.h	113
7.11 flyback.h	113
7.12 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp File Reference	115
7.12.1 Detailed Description	115
7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h File Reference	115
7.13.1 Detailed Description	117
7.14 jsonModule.h	117
7.15 lockerBase.h	118
7.16 scopedLock.h	118
7.17 logManager.h	119
7.18 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp File Reference	120
7.18.1 Detailed Description	120
7.19 pressure.h	121
7.20 ptrUtils.h	121
7.21 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp File Reference	123
7.21.1 Detailed Description	124
7.22 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h File Reference	124
7.22.1 Detailed Description	126
7.23 reportSystem.h	126
7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h File Reference	127
7.24.1 Detailed Description	128
7.25 sensorModule.h	129
7.26 serialMenu.h	130
7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp File Reference	130
7.27.1 Detailed Description	131
7.28 temperature.h	131

7.29 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp File Reference	132
7.29.1 Detailed Description	133
7.30 timeModule.h	133
7.31 vacControl.h	134
Index	137

Chapter 1

Namespace Index

1.1 Namespace List

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

calcModule	Namespace for the calculation module	9
comModule	Namespace for the communication module	9
flybackModule	Namespace for the Flyback module	10
jsonModule	Namespace for the JSON module	11
locker	Namespace for the locker system	11
reportSystem	Namespace for the report system	12
sensorModule	Namespace for the sensor module	12
timeModule	Namespace for the timeModule	12
vacControlModule	Namespace for the VacControl module	13

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

calcModule::CalcModuleInternals	15
calcModuleInternals	22
comModule::ComModuleInternals	22
comModuleInternals	24
timeModule::DateTimeStruct	24
comModule::EthernetCommunication	24
flybackModule::Flyback	37
comModule::I2CCommunication	40
jsonModule::JsonModuleInternals	42
jsonModuleInternals	46
LockerBase	47
LogManager	48
LogMapper	55
flybackModule::Measurement	55
Measurement	55
MenuItem	56
OutputLevel	56
PointerWrapper< T >	57
vacControlModule::Pressure	59
PressureSensor	59
sensorModule::SensorModuleInternals	74
PtrUtils	61
reportSystem::ReportSystem	63
locker::ScopedLock	71
ScopedPointer< T >	72
comModule::SerialCommunication	78
SerialMenu	79
comModule::SPICommunication	87
TemperatureSensor	88
sensorModule::SensorModuleInternals	74
timeModule::TimeModuleInternals	91
vacControlModule::VacControl	97

Chapter 3

Class Index

3.1 Class List

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

calcModule::CalcModuleInternals	15
calcModuleInternals	
Class for the calculation module internals	22
comModule::ComModuleInternals	22
comModuleInternals	
Class for the communication module internals	24
timeModule::DateTimeStruct	
Struct to hold the date and time	24
comModule::EthernetCommunication	
Class to handle Ethernet communication	24
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	37
comModule::I2CCommunication	
Class to handle I2C communication	40
jsonModule::JsonModuleInternals	42
jsonModuleInternals	
Class for the JSON module internals	46
LockerBase	
Base class for the locker system	47
LogManager	48
LogMapper	
Class which handle the printed log messages, maps aka parses them and saves them to the SD card	55
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	55
Measurement	
Structure to store the measured values of the system This structure holds the pressure values measured from the system	55
MenuItem	
Serial menu structure	56
Outputlevel	
Enum Class for the differnet Outputlevels	56

PointerWrapper< T >	
Template class for wrapping a pointer	57
vacControlModule::Pressure	59
PressureSensor	
Pressure sensor class	59
PtrUtils	
Utility class for pointer operations	61
reportSystem::ReportSystem	
Class for the report system	63
locker::ScopedLock	
Scoped lock class for mutexes	71
ScopedPointer< T >	
Template class for a Scoped Pointer	72
sensorModule::SensorModuleInternals	
Class for the sensor module internals	74
comModule::SerialCommunication	
Class to handle Serial communication	78
SerialMenu	
Class for the serial menu	79
comModule::SPICommunication	
Class to handle SPI communication	87
TemperatureSensor	
Temperature sensor class	88
timeModule::TimeModuleInternals	
Class to handle Systemtime	91
vacControlModule::VacControl	
VacControl class to manage the vacuum control system This class provides methods for initial- izing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes	97

Chapter 4

File Index

4.1 File List

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

C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp	105
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h	
Header file for the calculation module handling sensor data	106
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp	
Implementation of the comModule class that utilizes various communication protocols	108
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h	109
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH/ETHH.h	109
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC/I2CC.h	111
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER/SER.h	112
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII/SPII.h	112
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h	113
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h	113
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp	
Implementation of the jsonModule class	115
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h	115
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h	118
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h	118
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h	119
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp	
Implementation of the pressure class	120
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h	121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h	121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp	
Unified system health and error reporting module	123
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h	124
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h	127
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h	130
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp	
Implementation of the temperature class	130
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.h	131
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp	
Implementation of the timeModule class	132
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h	133
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h	134

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.

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](#)

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 , MAX6675_Celsius , MAX6675_Fahrenheit , MAX6675_Kelvin ,
 MAX31855_Celsius , MAX31855_Fahrenheit , MAX31855_Kelvin , 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)
Function 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)

6.1.1 Member Function Documentation

6.1.1.1 atmToPascal()

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

Function to convert atm to pascal.

Parameters

<i>atm</i>	-> 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

<i>data</i>	-> The data set to calculate the average from.
<i>length</i>	-> 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

<i>voltage</i>	-> The voltage.
<i>resistance</i>	-> 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

<i>voltage</i>	-> The voltage.
<i>current</i>	-> 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

<i>voltage</i>	-> The voltage.
<i>current</i>	-> 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

<i>data</i>	-> The data set to calculate the standard deviation from.
<i>length</i>	-> 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

<i>celsius</i>	-> The temperature in celsius.
----------------	--------------------------------

Returns

float -> The temperature in fahrenheit.

6.1.1.8 celsiusToKelvin()

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

Function to convert celsius to kelvin.

Parameters

<i>celsius</i>	-> The temperature in celsius.
----------------	--------------------------------

Returns

float -> The temperature in kelvin.

6.1.1.9 extractFloat()

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

Extract the float from a VAT uC eth frame.

Parameters

<i>response</i>	-> The response from the VAT uC.
<i>id</i>	-> 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 fahrenheitToCelsius()

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

Function to convert fahrenheit to celsius.

Parameters

<i>fahrenheit</i>	-> The temperature in fahrenheit.
-------------------	-----------------------------------

Returns

float -> The temperature in celsius.

6.1.1.11 findMaximum()

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

Function to calculate the maximum value of a data set.

Parameters

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

Returns

float -> The maximum value of the data set.

6.1.1.12 findMedian()

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

Function to calculate the median of a data set.

Parameters

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

Returns

float -> The median of the data set.

6.1.1.13 findMinimum()

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

Function to calculate the minimum value of a data set.

Parameters

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

Returns

float -> The minimum value of the data set.

6.1.1.14 kelvinToCelsius()

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

Function to convert kelvin to celsius.

Parameters

<i>kelvin</i>	-> The temperature in kelvin.
---------------	-------------------------------

Returns

float -> The temperature in celsius.

6.1.1.15 pascalToAtm()

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

Function to convert pascal to atm.

Parameters

<i>pascal</i>	-> The pressure in pascal.
---------------	----------------------------

Returns

float -> The pressure in atm.

6.1.1.16 pascalToPsi()

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

Function to convert pascal to psi.

Parameters

<i>pascal</i>	-> The pressure in pascal.
---------------	----------------------------

Returns

float -> The pressure in psi.

6.1.1.17 psiToPascal()

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

Function to convert psi to pascal.

Parameters

<i>psi</i>	-> 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`&

6.3.1.2 getI2C()

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

Get the I2C object.

Returns

`I2CCommunication`&

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`&

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 Internal 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

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

6.6.2.2 getClient()

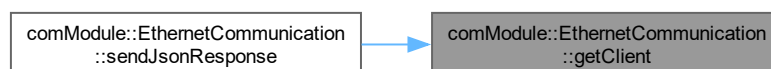
```
EthernetClient & EthernetCommunication::getClient ( )
```

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

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

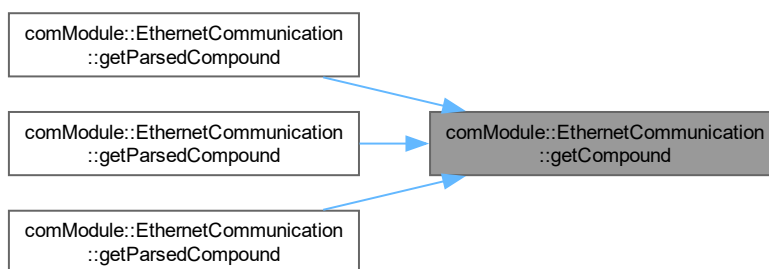
Returns

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

<i>id</i>	-> Enum ID from Compound2
<i>index</i>	-> 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

<i>id</i>	-> Enum ID from Compound3
<i>index</i>	-> 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.

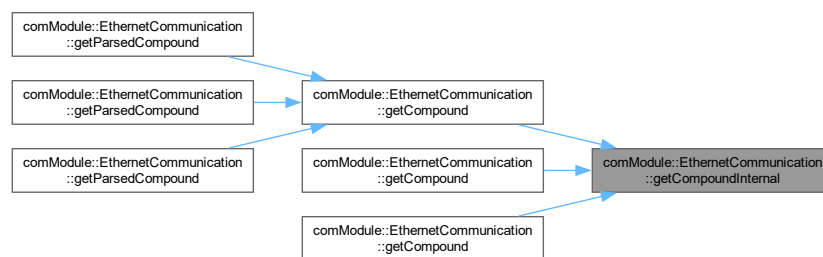
Parameters

<i>compoundType</i>	-> The type of the compound
<i>id</i>	-> The ID of the compound
<i>index</i>	-> 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

<i>id</i>	-> 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)

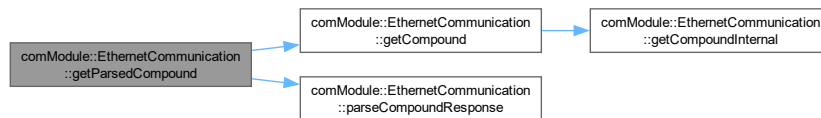
Parameters

<i>id</i>	-> Enum ID from Compound1
<i>index</i>	-> 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)

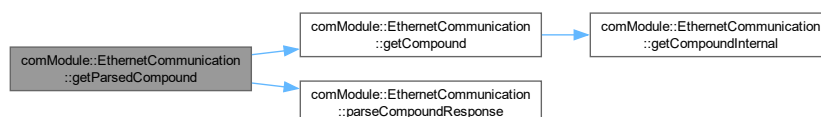
Parameters

<i>id</i>	-> Enum ID from Compound1
<i>index</i>	-> 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)

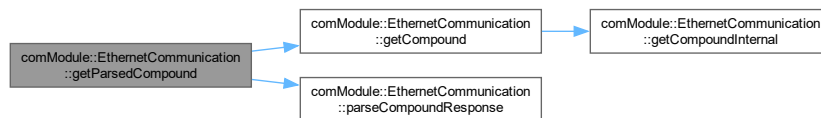
Parameters

<i>id</i>	-> Enum ID from Compound1
<i>index</i>	-> 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

<i>jsonBody</i>	-> 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

6.6.2.15 parseCompoundResponse()

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

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

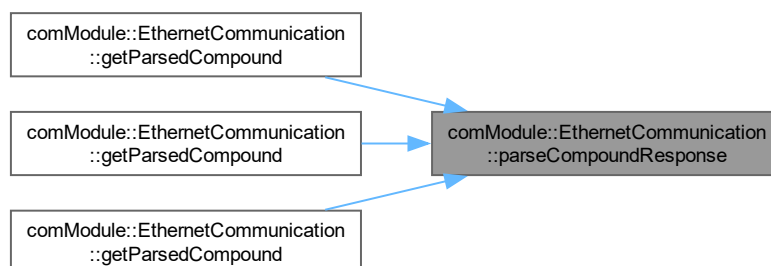
Parameters

<i>response</i>	-> 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

<i>buffer</i>	-> The buffer to read the data into
<i>length</i>	-> 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

<i>command</i>	-> 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

<i>endpoint</i>	-> endpoint to send data to
<i>data</i>	-> 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

<i>jsonBody</i>	-> 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

<i>id</i>	-> Enum ID from Compound1
<i>index</i>	-> Index of the command
<i>value</i>	-> 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

<i>id</i>	-> Enum ID from Compound2
<i>index</i>	-> Index of the command
<i>value</i>	-> 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

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

Here is the call graph for this function:



6.6.2.23 setCompoundInternal()

```
void EthernetCommunication::setCompoundInternal (
```

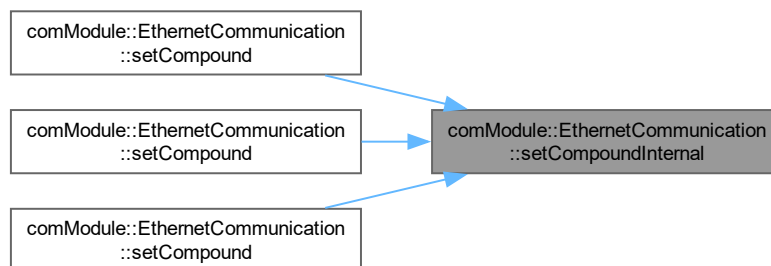
```
String compoundType,
unsigned long id,
int index,
String value )
```

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

Parameters

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

<i>id</i>	-> The ID of the parameter to set
<i>value</i>	-> 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

<i>flag</i>	-> 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/ETHH.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH/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** ()
denitalize 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 *.*

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 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")

6.7.2.2 getTimerState()

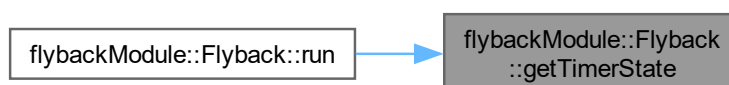
```
bool Flyback::getTimerState ( )
```

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:



6.7.2.3 isInitialized()

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

Get the state of the [Flyback](#) system.

Returns

true -> [Flyback](#) is initialized
false -> [Flyback](#) is not initialized

6.7.2.4 measure()

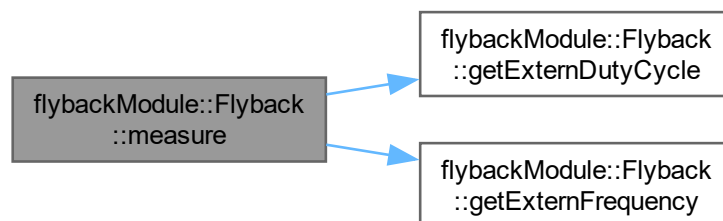
```
Measurement Flyback::measure ( )
```

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:



6.7.2.5 setExternDutyCycle()

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

Function to get the desired DutyCycle from HAS.

Parameters

<i>dutyCycle</i>	-> the dutyCycle to change
------------------	----------------------------

6.7.2.6 setExternFrequency()

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

Function to get the desired Frequency from HAS.

Parameters

<i>frequency</i>	-> the frequency to change
------------------	----------------------------

6.7.2.7 setTimerState()

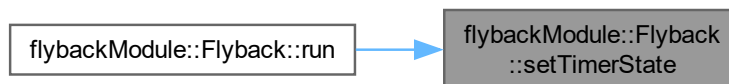
```
void Flyback::setTimerState (
    bool state )
```

Sets the state of the timer.

Parameters

<i>state</i>	-> 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 `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

<i>address</i>	-> 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

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

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

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

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

6.9 jsonModule::JsonModuleInternals Class Reference**Public Member Functions**

- void [createJson](#) (const char *key, const char *value)
Create a Json object with a key and a value.
- void [createJsonFloat](#) (const char *key, float value)
Create a Json Float object.
- void [createJsonInt](#) (const char *key, int value)
Create a Json Int object.
- void [createJsonString](#) (const char *key, String &value)
Create a Json String object.
- void [createJsonStringConst](#) (const char *key, const String &value)

Create a Json String Const object.

- void **sendJsonSerial** ()

Function to send the Json object over the Serial connection.

- void **sendJsonEthernet** (const char *endpoint)

Function to send the Json object over the Ethernet connection.

- String **getJSONString** () const

Get the Json String object.

- std::map< String, float > **mapJsonToDoubles** (const String &rawJson)

Map the Json object to a map of Strings and floats.

- void **clearJson** ()

Clear the Json object.

- void **printJsonDocMemory** ()

Prints information about the Json object.

Public Attributes

- size_t **jsonBuffer**

6.9.1 Member Function Documentation

6.9.1.1 createJson()

```
void JsonModuleInternals::createJson (
    const char * key,
    const char * value )
```

Create a Json object with a key and a value.

Parameters

<i>key</i>	-> The key of the Json object
<i>value</i>	-> The value of the Json object

6.9.1.2 createJsonFloat()

```
void JsonModuleInternals::createJsonFloat (
    const char * key,
    float value )
```

Create a Json Float object.

Parameters

<i>key</i>	-> The key of the Json object
<i>value</i>	-> The value of the Json object

6.9.1.3 createJsonInt()

```
void JsonModuleInternals::createJsonInt (
    const char * key,
    int value )
```

Create a Json Int object.

Parameters

<i>key</i>	-> The key of the Json object
<i>value</i>	-> The value of the Json object

6.9.1.4 createJsonString()

```
void JsonModuleInternals::createJsonString (
    const char * key,
    String & value )
```

Create a Json String object.

Parameters

<i>key</i>	-> The key of the Json object
<i>value</i>	-> The value of the Json object

6.9.1.5 createJsonStringConst()

```
void JsonModuleInternals::createJsonStringConst (
    const char * key,
    const String & value )
```

Create a Json String Const object.

Parameters

<i>key</i>	-> The key of the Json object
<i>value</i>	-> The value of the Json object

6.9.1.6 getJsonString()

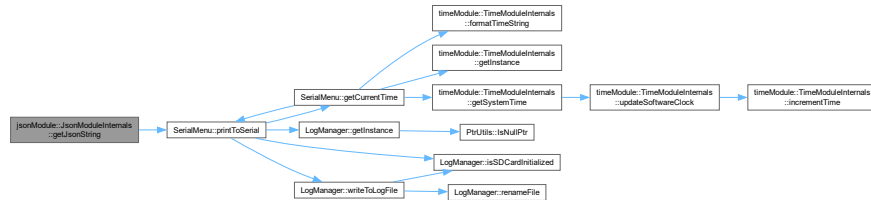
```
String JsonModuleInternals::getJsonString ( ) const
```

Get the Json String object.

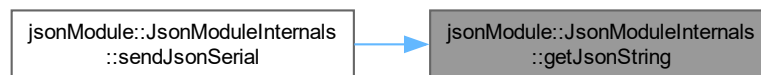
Returns

String -> The Json String object

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.1.7 mapJsonToDoubles()

```
std::map< String, float > JsonModuleInternals::mapJsonToDoubles (
    const String & rawJson )
```

Map the Json object to a map of Strings and floats.

<https://github.com/mike-matera/ArduinoSTL/issues/84>

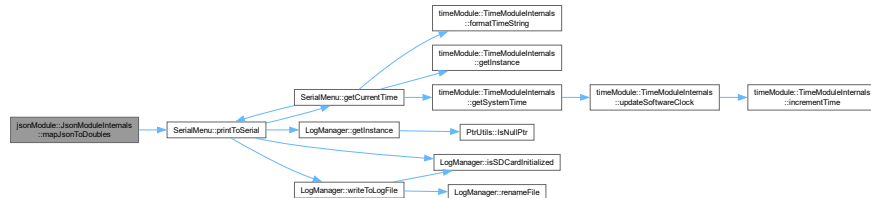
Parameters

<i>rawJson</i>	-> The raw Json object
----------------	------------------------

Returns

`std::map<String, float>` -> The mapped Json object

Here is the call graph for this function:

**6.9.1.8 sendJsonEthernet()**

```
void jsonModule::JsonModuleInternals::sendJsonEthernet (
    const char * endpoint )
```

Function to send the Json object over the Ethernet connection.

Parameters

<i>endpoint</i>	-> The endpoint to send the Json object to
-----------------	--

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 jsonModuleInternals Class Reference

Class for the JSON module internals.

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

6.10.1 Detailed Description

Class for the JSON module internals.

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

- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/[jsonModule.h](#)

6.11 LockerBase Class Reference

Base class for the locker system.

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

Public Member Functions

- [locker::ScopedLock lockEthernetScoped \(\)](#)
- [locker::ScopedLock lockTemperatureScoped \(\)](#)
- [locker::ScopedLock lockSerialScoped \(\)](#)

6.11.1 Detailed Description

Base class for the locker system.

6.11.2 Member Function Documentation

6.11.2.1 lockEthernetScoped()

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

Returns

[locker::ScopedLock](#)

6.11.2.2 lockSerialScoped()

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

Returns

[locker::ScopedLock](#)

6.11.2.3 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

6.12 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` ()
- 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.

6.12.1 Member Function Documentation

6.12.1.1 `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:



6.12.1.2 getInstance()

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

Get the Instance object.

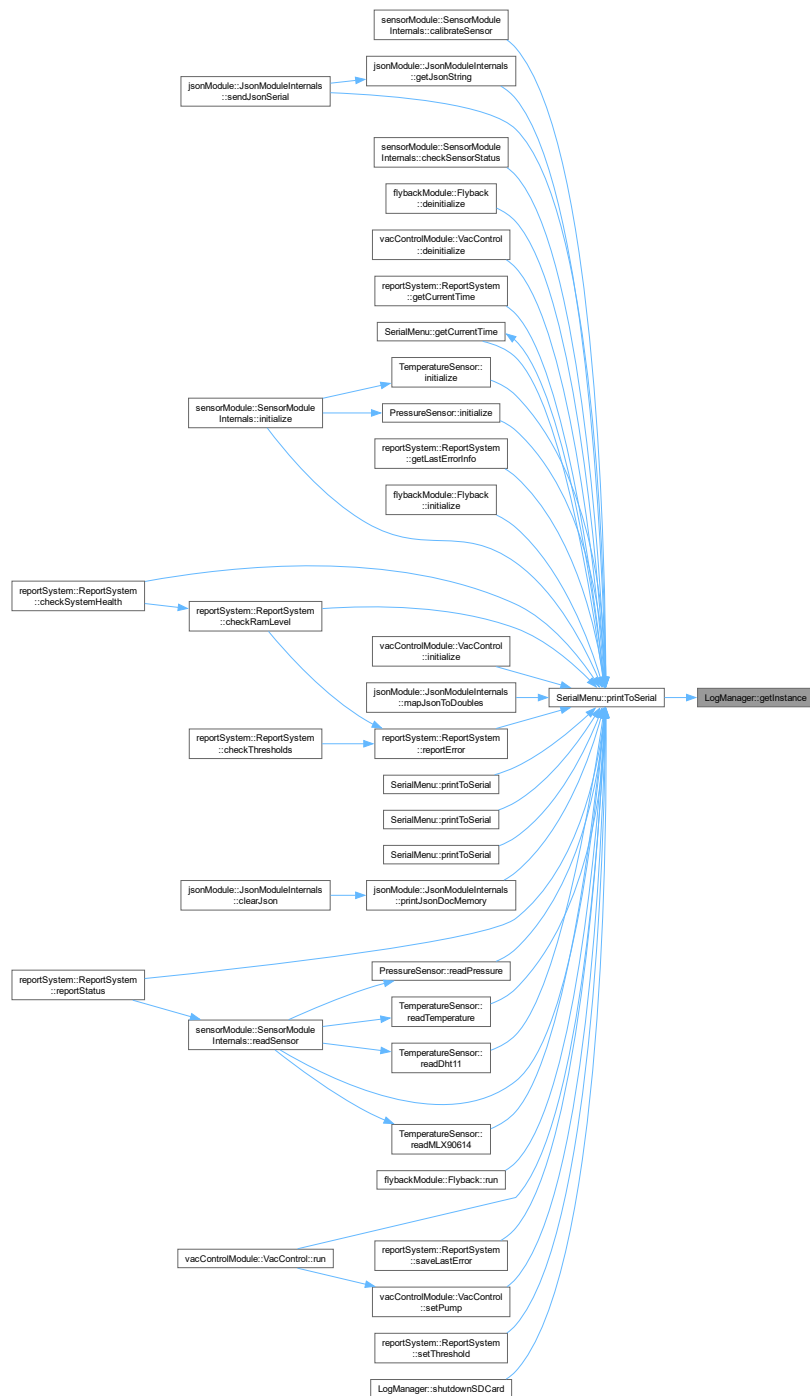
Returns

LogManager*

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.1.3 initSDCard()

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

Function to initialize the SD card.

6.12.1.5 renameFile()

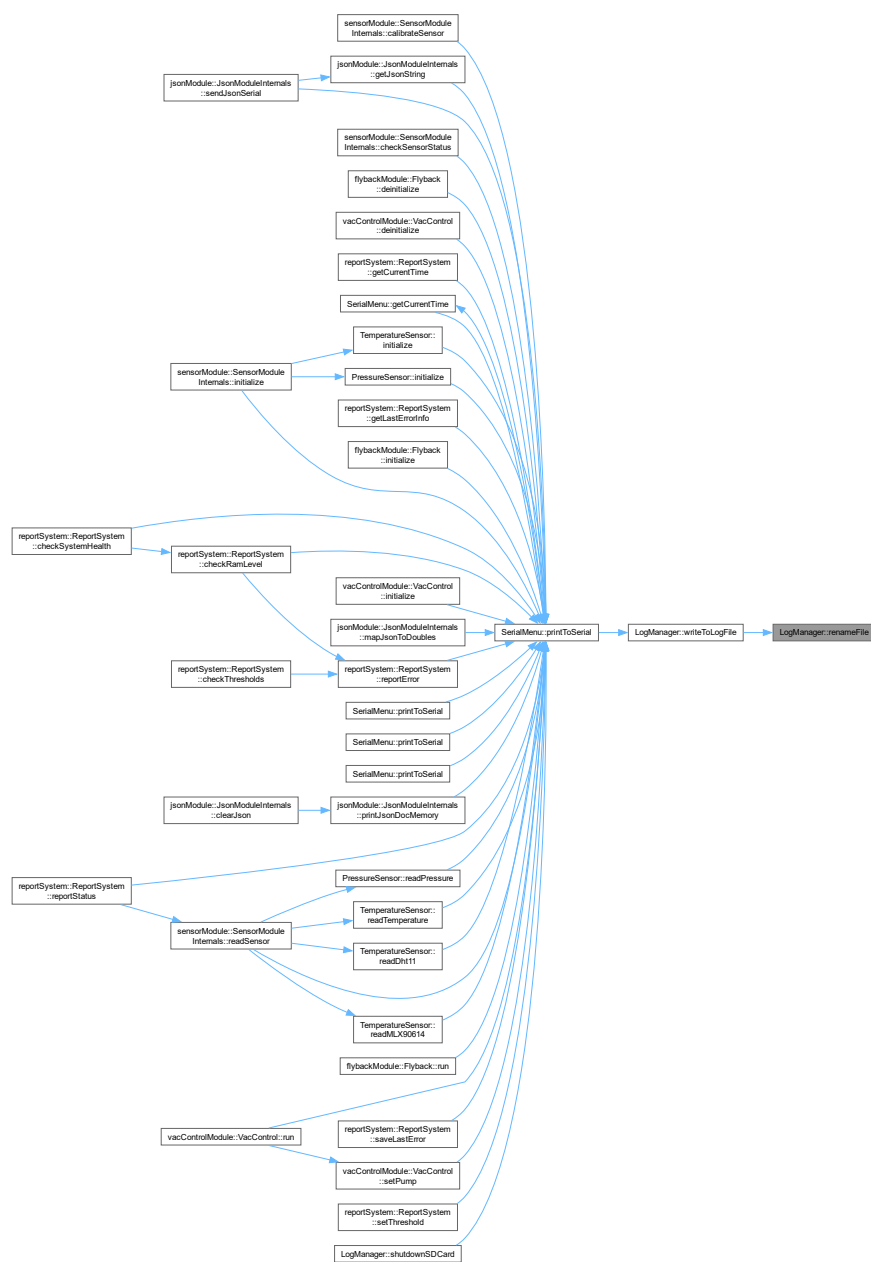
```
void LogManager::renameFile (
    const String & oldName,
    const String & newName )
```

Function to rename the currently written to file.

Parameters

<i>oldName</i>	-> This is the oldName of the file
<i>newName</i>	-> This is the newName of the file

Here is the caller graph for this function:



6.12.1.6 setLogFileName()

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

Set the Log File Name object.

Parameters

<i>fileName</i>	-> The file name to set the log file name to.
-----------------	---

Here is the call graph for this function:



6.12.1.7 writeToLogFile()

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

Function to write a log message to the log file.

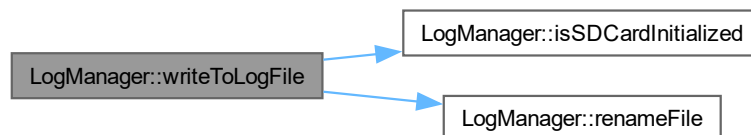
Parameters

<i>logMessage</i>	-> The log message to write to the log file.
-------------------	--

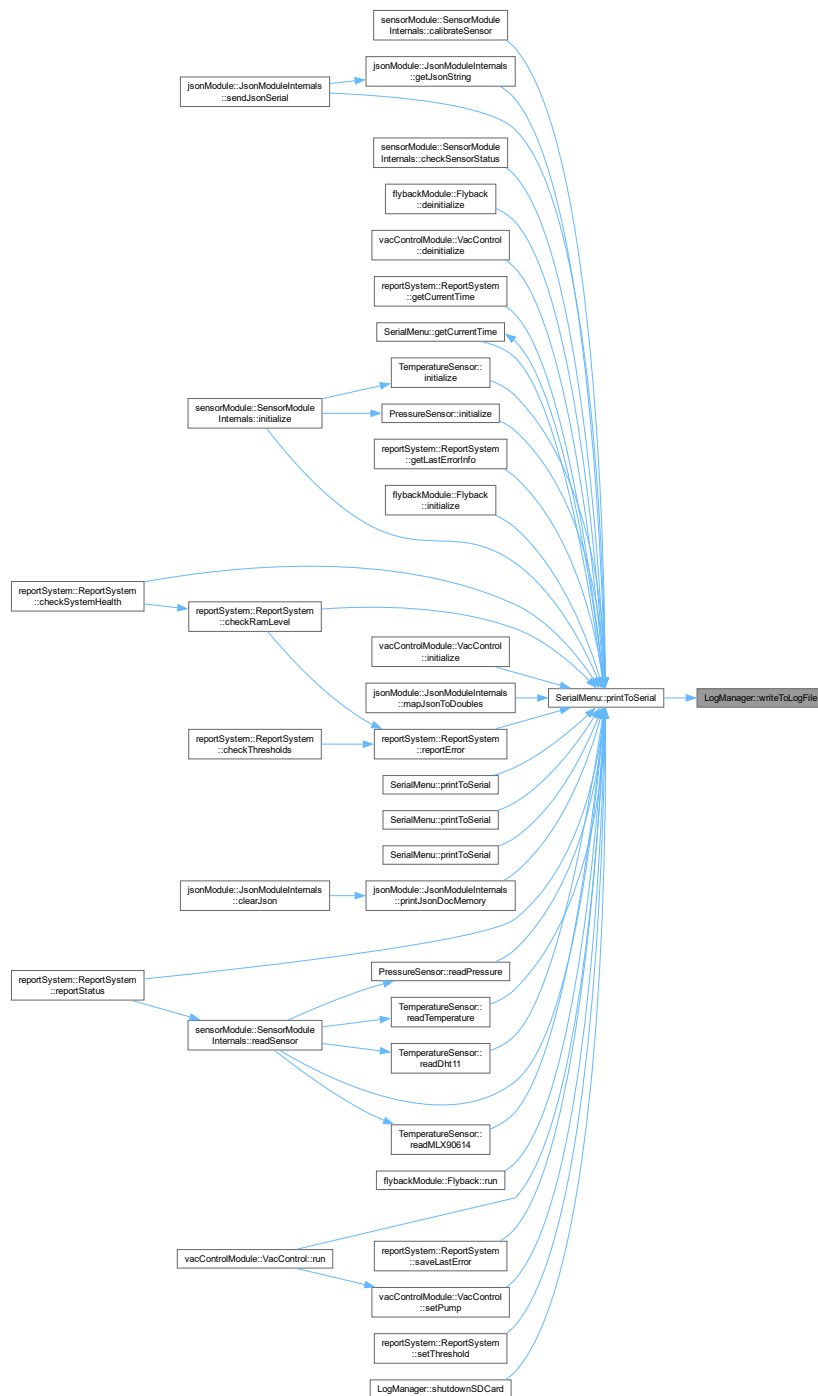
Returns

- 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

6.13 LogMapper Class Reference

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

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

6.13.1 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

6.14 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**

6.14.1 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

6.15 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>
```

6.15.1 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

6.16 MenuItem Struct Reference

Serial menu structure.

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

Public Attributes

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

6.16.1 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

6.17 Outputlevel Class Reference

Enum Class for the differnet Outputlevels.

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

6.17.1 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

6.18 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.

6.18.1 Detailed Description

```
template<typename T>
class PointerWrapper< T >
```

Tempalte class for wrapping a pointer.

Template Parameters

<i>T</i>	
----------	--

6.18.2 Member Function Documentation

6.18.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.18.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.18.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.18.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.18.2.5 reset()

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

Function to reset the pointer.

Parameters

<i>p</i>	-> 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

6.19 vacControlModule::Pressure Struct Reference

Public Attributes

- float **pressure**

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

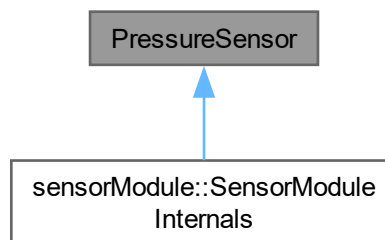
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h

6.20 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.

6.20.1 Detailed Description

Pressure sensor class.

6.20.2 Member Function Documentation

6.20.2.1 isInitialized()

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

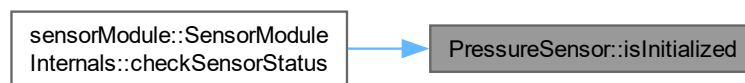
Function to check if the pressure sensor is initialized.

Returns

true -> if the pressure sensor is initialized

false -> if the pressure sensor is not initialized

Here is the caller graph for this function:



6.20.2.2 readPressure()

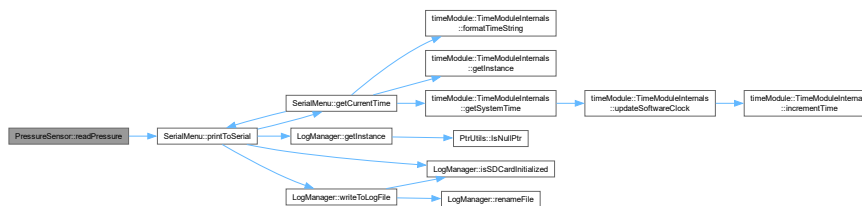
```
float PressureSensor::readPressure ( )
```

Function to read the pressure from the sensor.

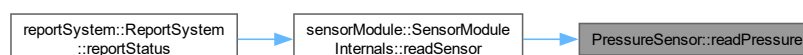
Returns

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](#)

6.21 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)`

6.21.1 Detailed Description

Utility class for pointer operations.

6.21.2 Member Function Documentation

6.21.2.1 IsNullPtr()

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

Check if a pointer is nullptr.

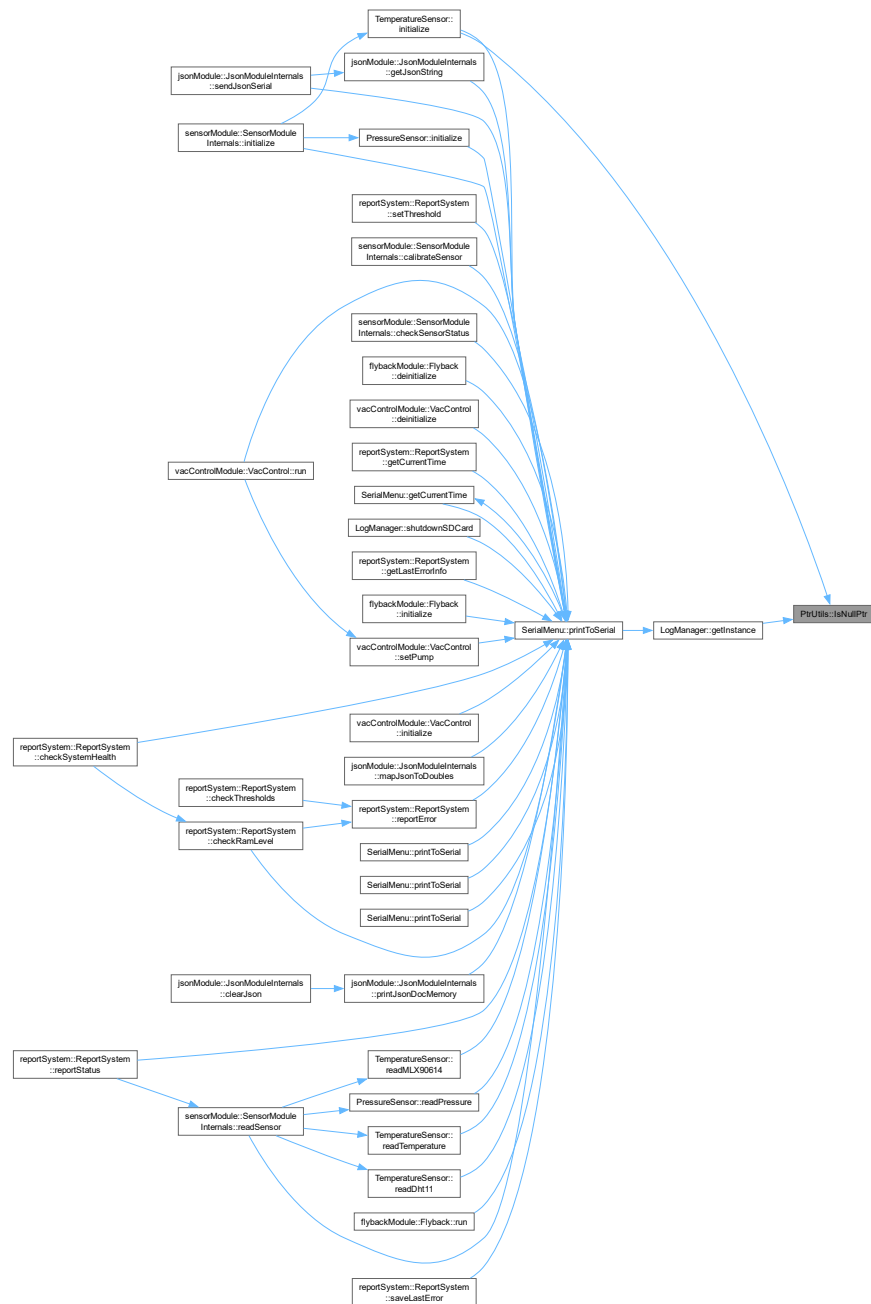
Parameters

<i>ptr</i>	Pointer to check.
------------	-------------------

Returns

true if the pointer is nullptr, false otherwise.

Here is the caller graph for this function:



6.21.2.2 IsValidPtr()

```

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

```

Check if a pointer is valid (not nullptr).

Parameters

<i>ptr</i>	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

6.22 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.

Static Public Member Functions

- static void **initStackGuard** ()
Initialize the Stack Guard.
- static bool **detectStackOverflow** ()
Detect Stack Overflow.

6.22.1 Detailed Description

Class for the report system.

6.22.2 Member Function Documentation

6.22.2.1 checkRamLevel()

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

Function to check the SRAM level on the hostsystem.

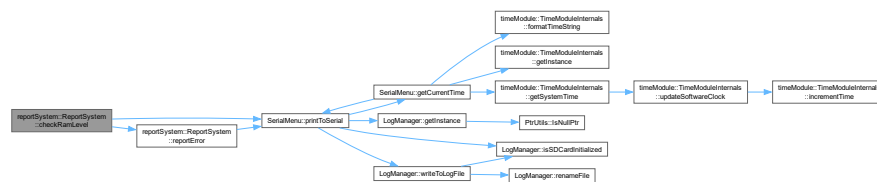
Parameters

<i>warningThreshold</i>	-> first warning to get
<i>criticalThreshold</i>	-> 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:



```
bool ReportSystem::checkSysT
```

Function to check the system health of the uC.

memory
1. 1.5:

false -> if the system is not healthy

6.22.2.3 checkThresholds()

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

Check the thresholds for the temperature and pressure sensors.

Parameters

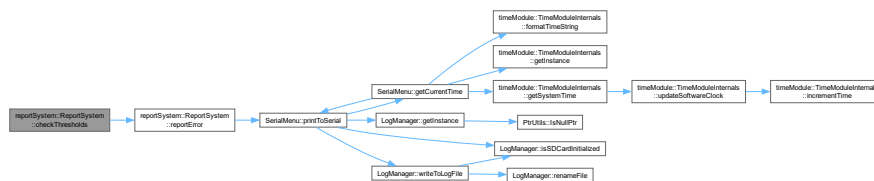
<i>currentTemp</i>	-> The current temperature
<i>currentPressure</i>	-> The current pressure

Returns

true -> if the thresholds are met

false -> if the thresholds are not met

Here is the call graph for this function:



6.22.2.4 detectStackOverflow()

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

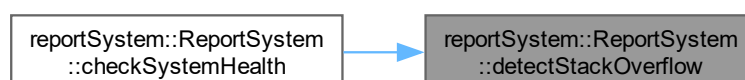
Detect Stack Overflow.

Returns

true -> if the stack has overflowed

false -> if the stack has not overflowed

Here is the caller graph for this function:



6.22.2.5 getCPULoad()

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

Getter for the CPU Load.

Returns

float -> The CPU Load

6.22.2.6 getCurrentTime()

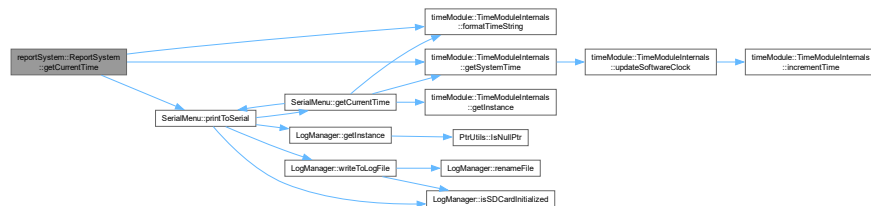
```
String ReportSystem::getCurrentTime ( )
```

Get the Current Time of the system.

Returns

String -> The current time

Here is the call graph for this function:



6.22.2.7 getLastError()

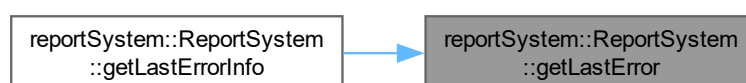
```
String ReportSystem::getLastError ( )
```

Get the Last Error message from EEPROM.

Returns

String -> The last error message

Here is the caller graph for this function:



6.22.2.8 getLastErrorInfo()

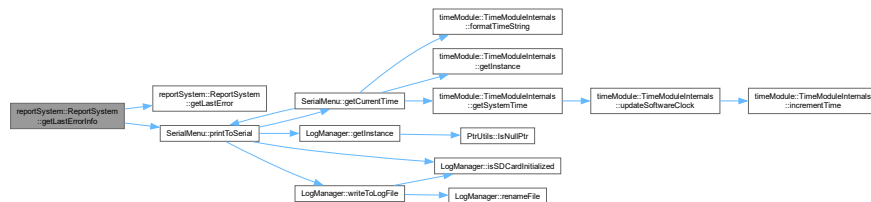
```
bool ReportSystem::getLastErrorInfo ( )
```

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:



6.22.2.9 getMemoryStatus()

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

Get the Memory Status of the system.

Returns

String -> The memory status

Here is the caller graph for this function:



6.22.2.10 getStackDump()

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

Get the Stack Dump of the system.

Returns

String -> The stack dump

6.22.2.11 reportError()

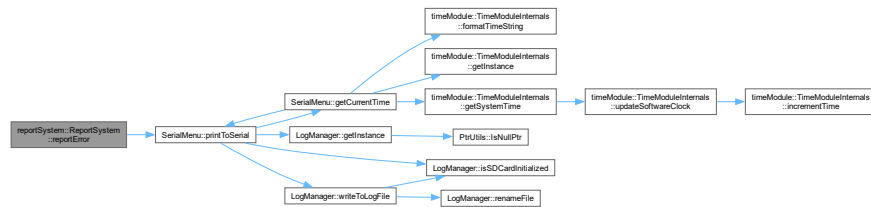
```
void ReportSystem::reportError (
    const char * errorMessage )
```

Function to log an error message.

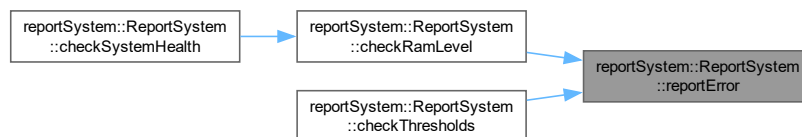
Parameters

<i>errorMessage</i>	-> The error message to log
---------------------	-----------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



6.22.2.12 reportStatus()

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

Function to report the status of the system.

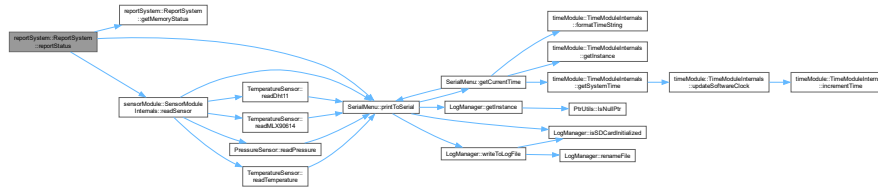
Parameters

<i>active</i>	-> The status of the system
---------------	-----------------------------

Returns

String -> The status of the system

Here is the call graph for this function:



6.22.2.13 saveLastError()

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

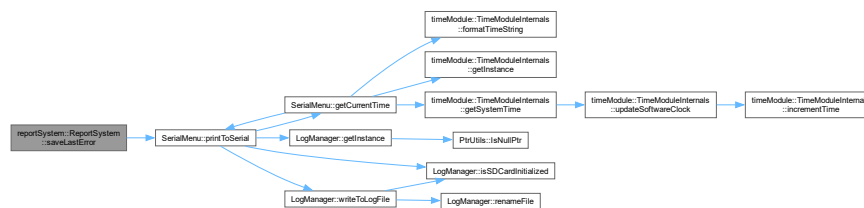
Saves last error message to EEPROM.

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

Parameters

<i>error</i>	-> The error message to save
--------------	------------------------------

Here is the call graph for this function:



6.22.2.14 setThreshold()

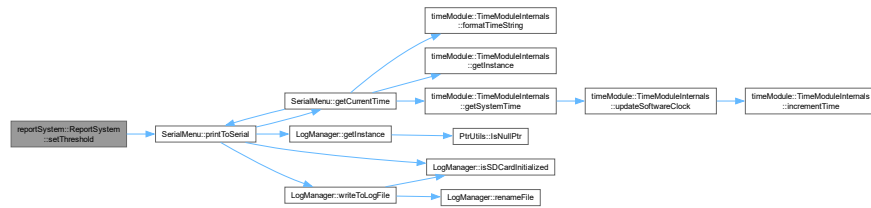
```
void ReportSystem::setThreshold (
    float tempThreshold,
    float pressureThreshold )
```

Set Thresholds for the pressure and temperature sensors.

Parameters

<i>tempThreshold</i>	-> The temperature threshold
<i>pressureThreshold</i>	-> 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](#)

6.23 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

6.23.1 Detailed Description

Scoped lock class for mutexes.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 ScopedLock() [1/2]

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

Construct a new Scoped Lock object.

Parameters

<code>mutex</code>	-> The mutex to lock
--------------------	----------------------

6.23.2.2 ScopedLock() [2/2]

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

Construct a new Scoped Lock object.

Parameters

<code>other</code>	-> 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

6.24 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.

6.24.1 Detailed Description

```
template<typename T>
class ScopedPointer< T >
```

Template class for a Scoped Pointer.

Template Parameters

<i>T</i>	-> The type of the pointer.
----------	-----------------------------

6.24.2 Member Function Documentation

6.24.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.24.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.24.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.24.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.24.2.5 reset()

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

Function to reset the pointer.

Parameters

<i>p</i>	-> The pointer to reset to.
----------	-----------------------------

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

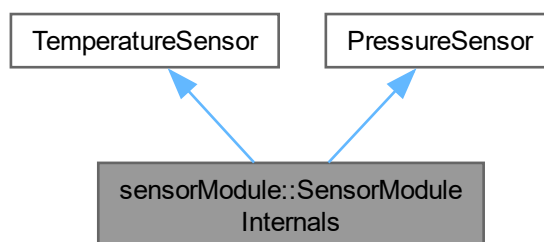
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.25 sensorModule::SensorModuleInternals Class Reference

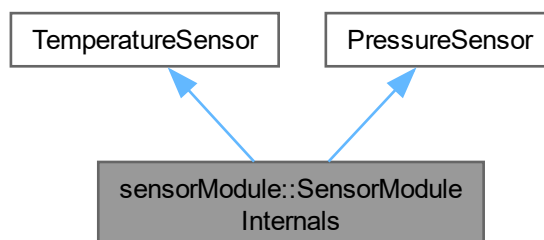
Class for the sensor module internals.

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

Inheritance diagram for sensorModule::SensorModuleInternals:



Collaboration diagram for sensorModule::SensorModuleInternals:



Public Member Functions

- void `initialize` ()
Function to begin the sensor module.
- 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.

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 `readDht11` ()
Function to read from specific sensor DH11.
- float `readMLX90614` (int choice)
Function to read from specific sensor MLX90614.
- float `readMAX6675` (Units unit)
- float `readMAX31855` (Units unit)
- bool `isInitialized` () const
Check if the temperature sensor is initialized.

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.

6.25.1 Detailed Description

Class for the sensor module internals.

6.25.2 Member Function Documentation

6.25.2.1 `calibrateSensor()`

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

Function to calibrate the sensor.

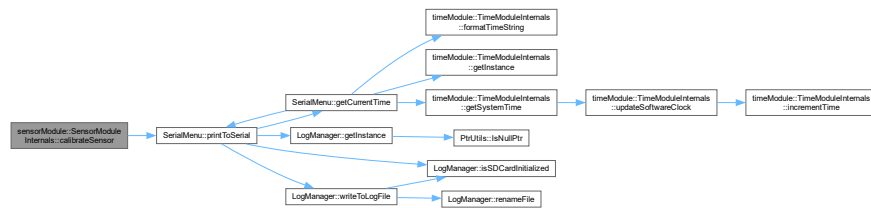
Parameters

<i>type</i>	-> 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:



6.25.2.2 checkSensorStatus()

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

Function to check the status of the sensor.

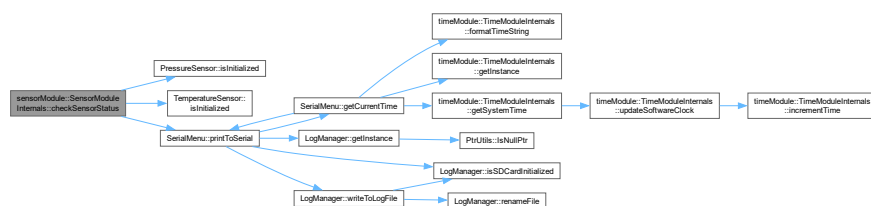
Parameters

<i>type</i>	-> 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:

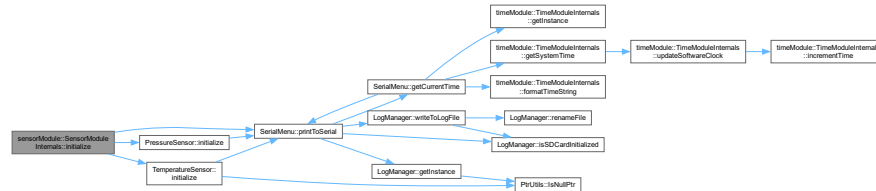


6.25.2.3 initialize()

```
void SensorModuleInternals::initialize ( )
```

Function to begin the sensor module.

Initialize the sensors. Here is the call graph for this function:



6.25.2.4 readSensor()

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

Function to read the sensor.

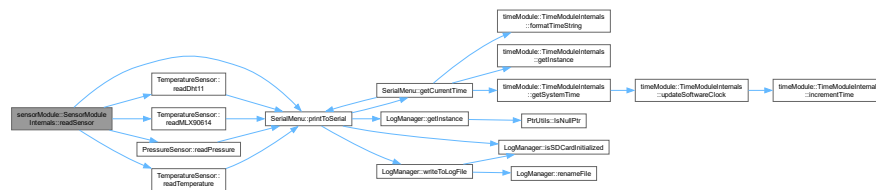
Parameters

<i>type</i>	-> The type of the sensor to read.
-------------	------------------------------------

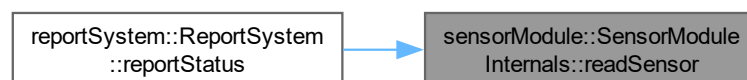
Returns

float -> The value of the sensor.

Here is the call graph for this function:



Here is the caller graph for this function:



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

6.26 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.

6.26.1 Detailed Description

Class to handle Serial communication.

6.26.2 Member Function Documentation

6.26.2.1 beginSerial()

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

Function to start the serial communication.

Parameters

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

6.26.2.2 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

6.26.2.3 receiveSerialData()

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

Function to receive data over serial.

Parameters

<i>buffer</i>	-> The buffer to read the data into
<i>length</i>	-> The length of the data to read

6.26.2.4 sendSerialData()

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

Function to end the serial communication.

Parameters

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

6.27 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 logMessage=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)
Function 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.

6.27.1 Detailed Description

Class for the serial menu.

6.27.2 Member Function Documentation

6.27.2.1 [getCurrentTime\(\)](#)

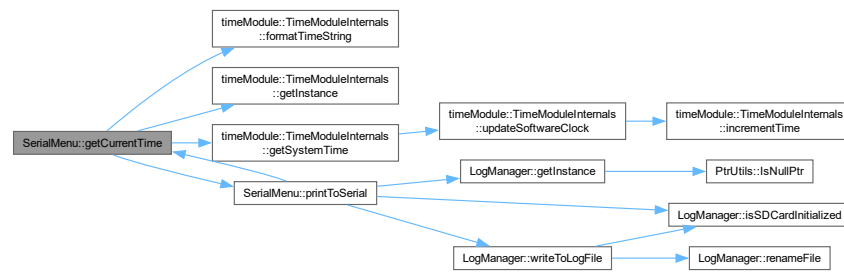
```
String SerialMenu::getCurrentTime ( ) [static]
```

Getter for the current time.

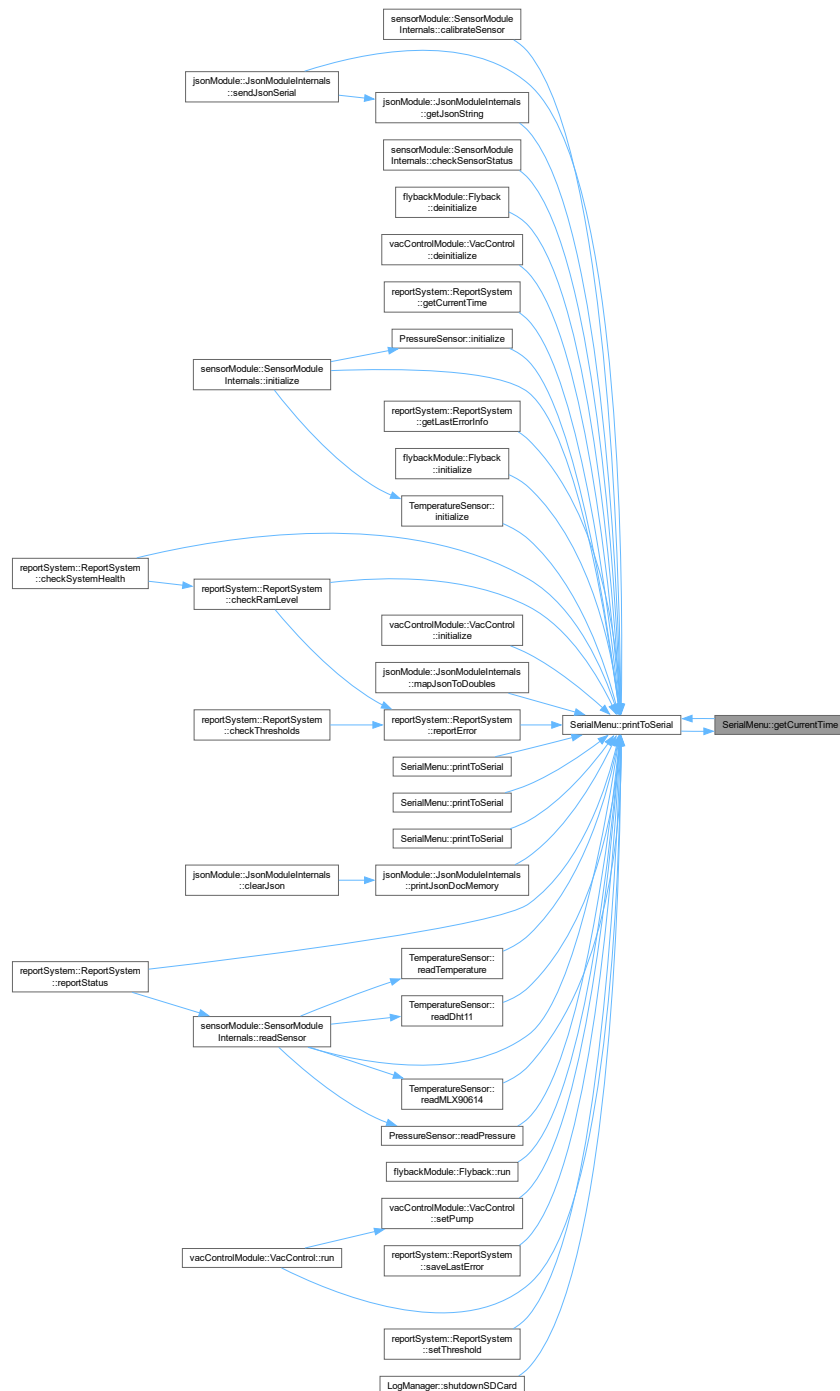
Returns

The current time as a String

Here is the call graph for this function:



Here is the caller graph for this function:



6.27.2.2 load()

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

Function to load the menu items.

Parameters

<i>items</i>	-> The menu items.
<i>size</i>	-> The size of the menu items.

6.27.2.3 printToSerial() [1/4]

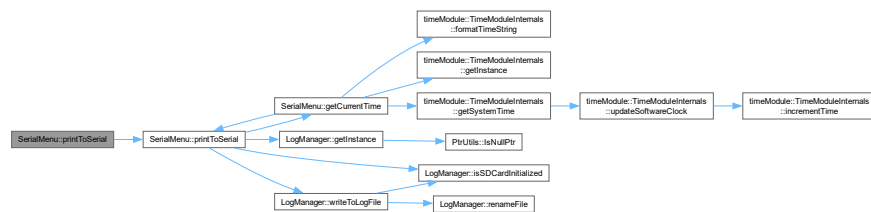
```
void SerialMenu::printToSerial (
    const __FlashStringHelper * message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.4 printToSerial() [2/4]

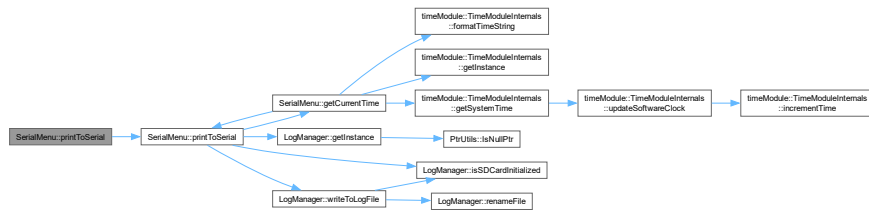
```
void SerialMenu::printToSerial (
    const String & message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.5 printToSerial() [3/4]

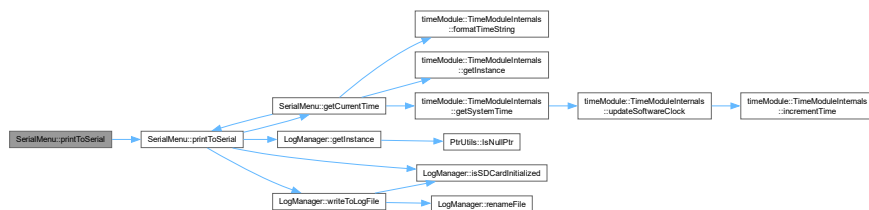
```
void SerialMenu::printToSerial (
    OutputLevel level,
    const __FlashStringHelper * message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.6 printToSerial() [4/4]

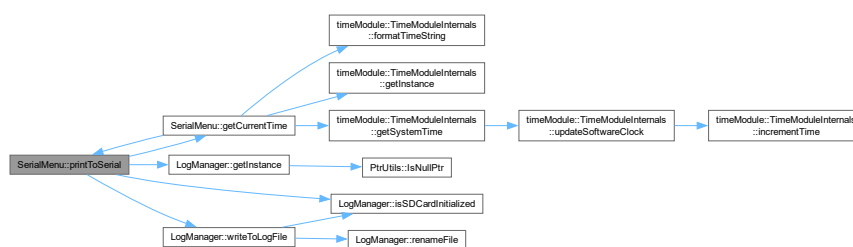
```
void SerialMenu::printToSerial (
    OutputLevel level,
    const String & message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

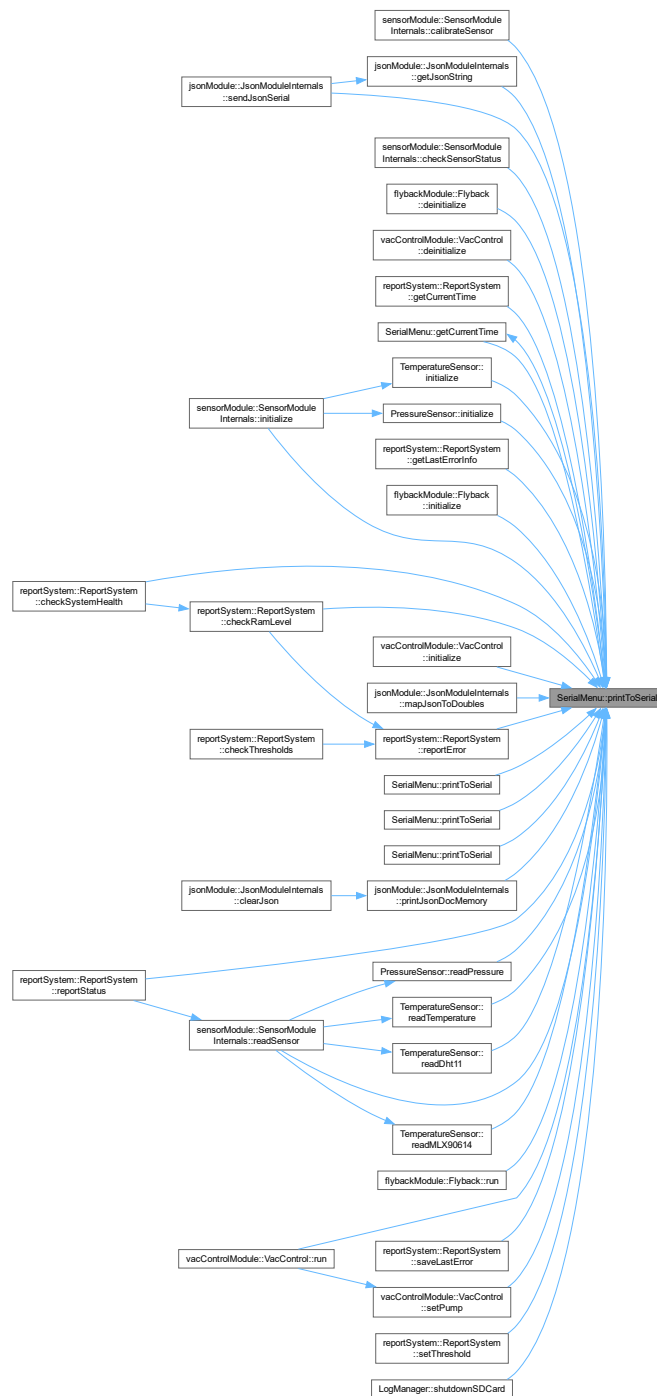
Parameters

<i>level</i>	-> The output level of the message.
<i>message</i>	-> The message to print, a String object.
<i>newLine</i>	-> 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

6.28 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.

6.28.1 Detailed Description

Class to handle SPI communication.

6.28.2 Member Function Documentation

6.28.2.1 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

6.28.2.2 spiRead()

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

Function to read data over SPI.

Parameters

<i>buffer</i>	-> The buffer to read the data into
<i>length</i>	-> The length of the data to read

6.28.2.3 spiWrite()

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

Function to write data over SPI.

Parameters

<i>data</i>	-> The data to write
<i>length</i>	-> The length of the data

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

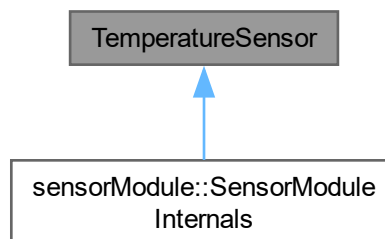
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII/SPII.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII/SPII.cpp

6.29 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 **readDht11** ()
Function to read from specific sensor DH11.
- float **readMLX90614** (int choice)
Function to read from specific sensor MLX90614.
- float **readMAX6675** (Units unit)
- float **readMAX31855** (Units unit)
- bool **isInitialized** () const
Check if the temperature sensor is initialized.

6.29.1 Detailed Description

Temperature sensor class.

6.29.2 Member Function Documentation

6.29.2.1 isInitialized()

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

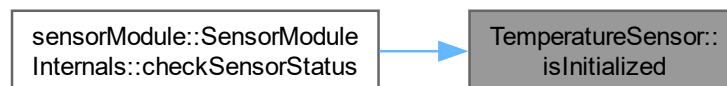
Check if the temperature sensor is initialized.

Returns

true -> if the temperature sensor is initialized

false -> if the temperature sensor is not initialized

Here is the caller graph for this function:



6.29.2.2 readDht11()

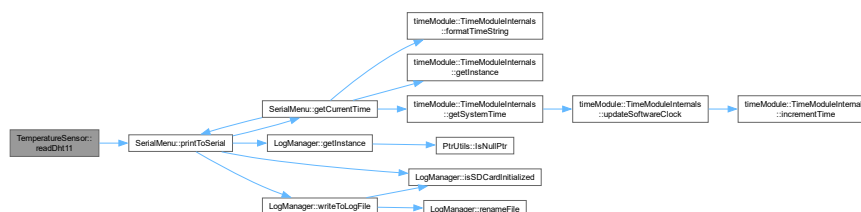
```
float TemperatureSensor::readDht11 ( )
```

Function to read from specific sensor DH11.

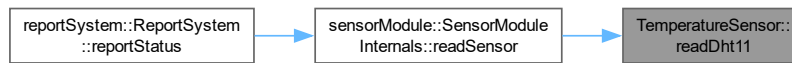
Returns

float -> The temperature value.

Here is the call graph for this function:



Here is the caller graph for this function:



6.29.2.3 readMLX90614()

```
float TemperatureSensor::readMLX90614 (
    int choice )
```

Function to read from specific sensor MLX90614.

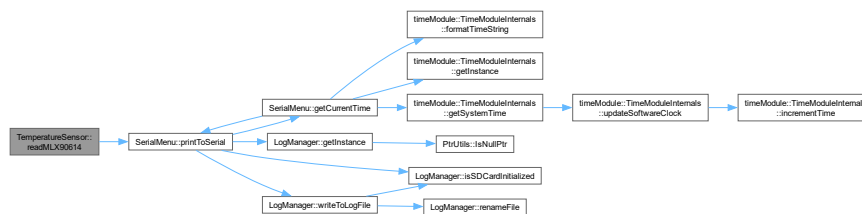
Parameters

<i>choice</i>	-> The choice of the sensor to read from.
---------------	---

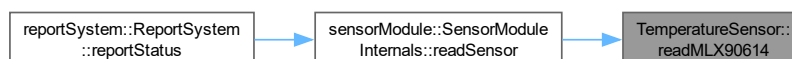
Returns

float -> The temperature value.

Here is the call graph for this function:



Here is the caller graph for this function:



6.29.2.4 readTemperature()

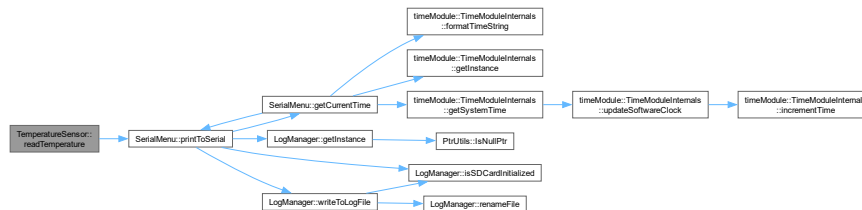
```
float TemperatureSensor::readTemperature ( )
```

Function to read the temperature from the sensor.

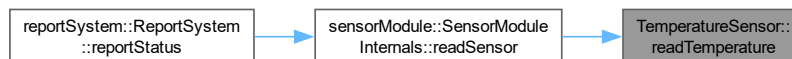
Returns

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

6.30 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.

6.30.1 Detailed Description

Class to handle Systemtime.

6.30.2 Member Function Documentation

6.30.2.1 `formatTimeString()`

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

Function to format the time to a string.

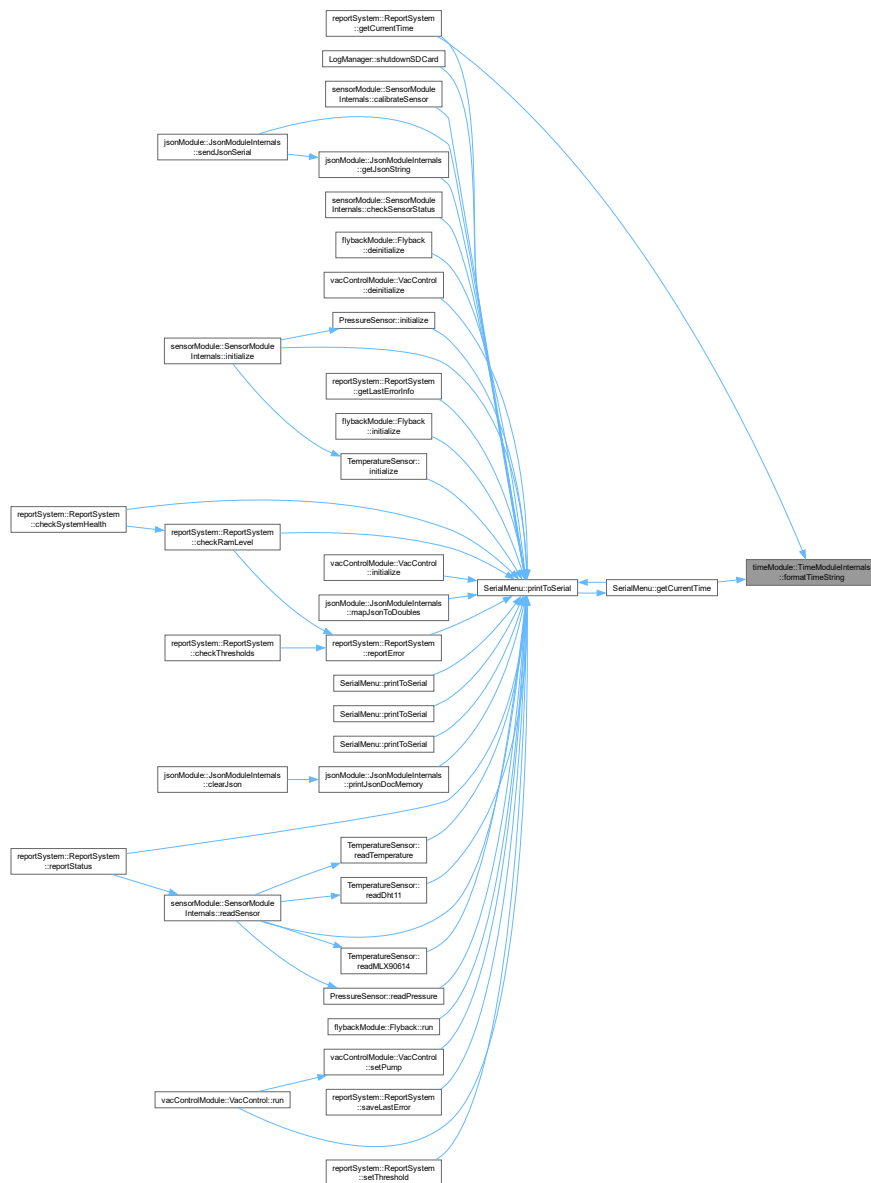
Parameters

<code>dt</code>	-> <code>DateTimeStruct</code> to format
-----------------	--

Returns

String -> The formatted time.

Here is the caller graph for this function:



6.30.2.2 getInstance()

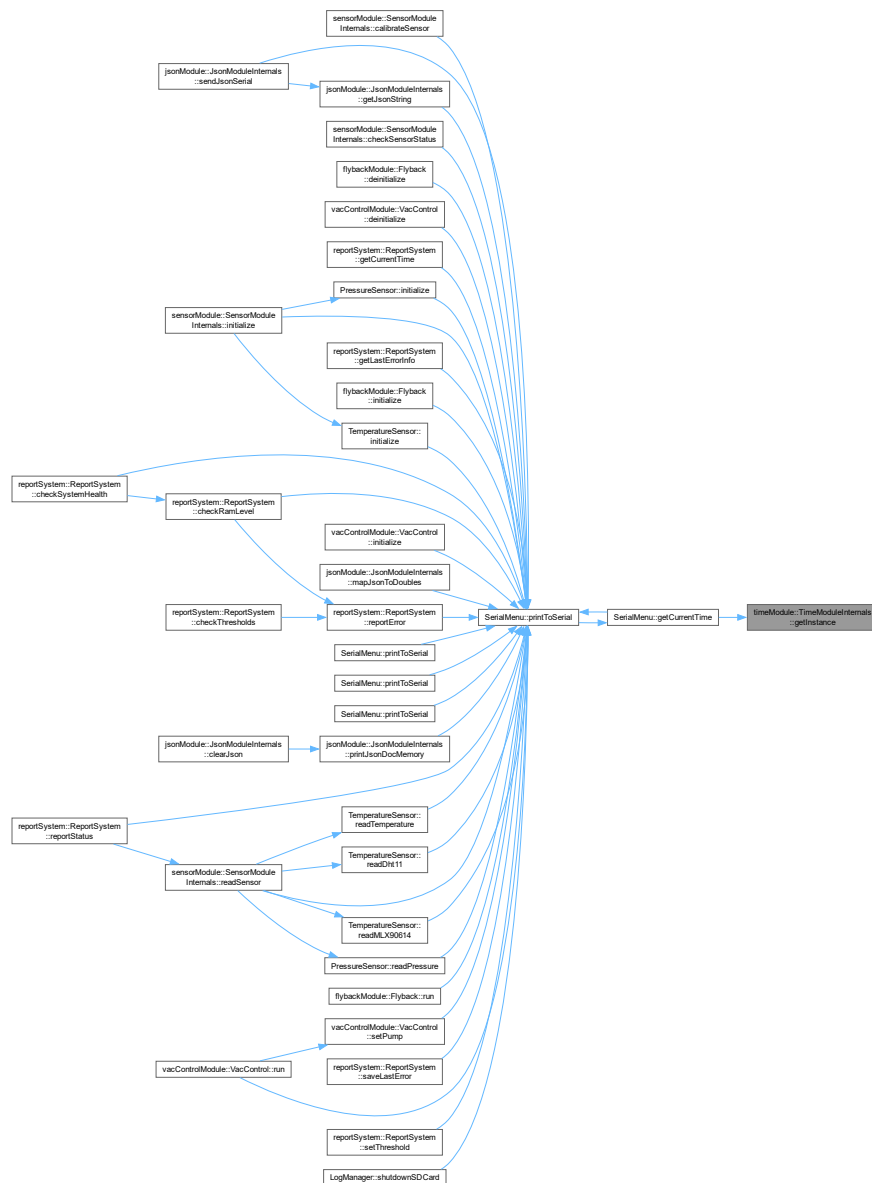
```
TimeModuleInternals * TimeModuleInternals::getInstance ( ) [static]
```

Get the Instance object, Singleton pattern.

Returns

TimeModuleInternals* -> The instance of the [TimeModuleInternals](#).

Here is the caller graph for this function:



6.30.2.3 getTimeSystemTime()

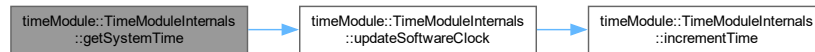
```
DateTimeStruct TimeModuleInternals::getTimeSystemTime ( )
```

Get the System Time object.

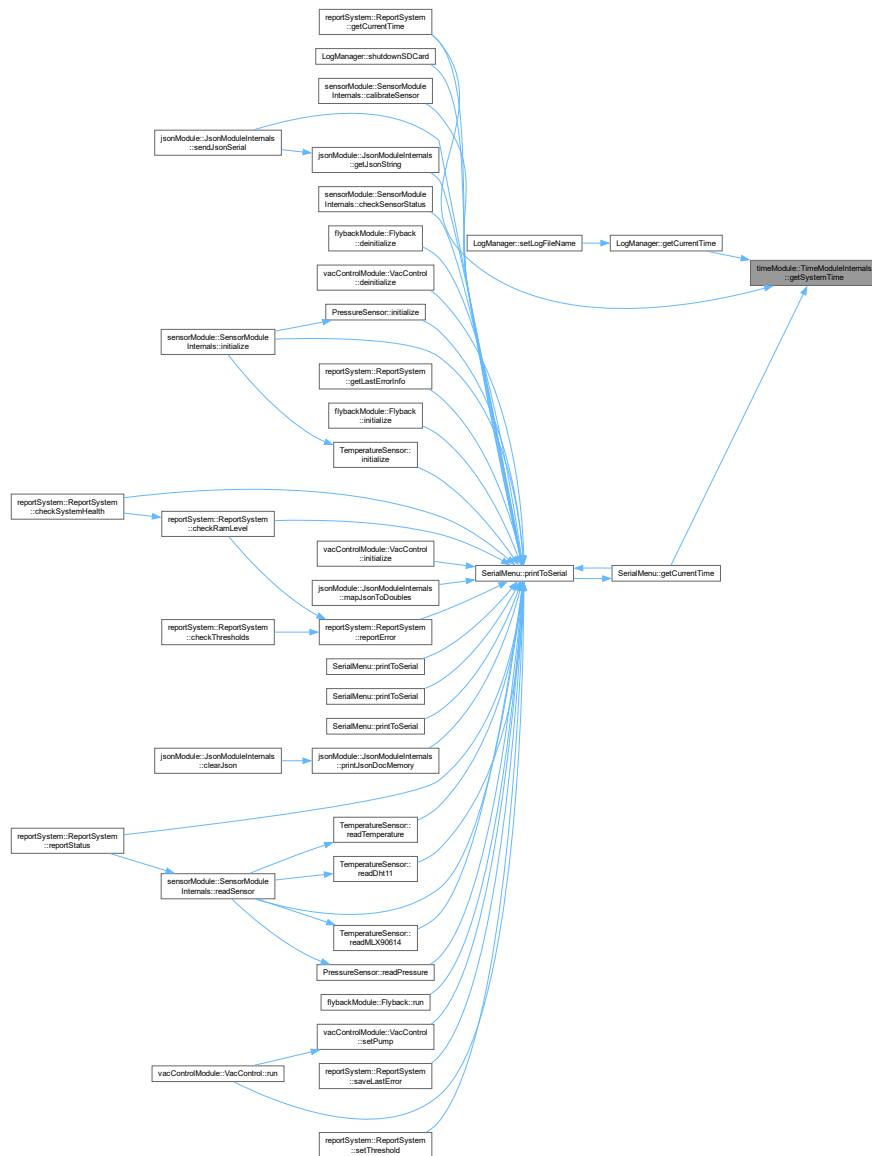
Returns

`DateTimeStruct` -> The system time.

Here is the call graph for this function:



Here is the caller graph for this function:



6.30.2.4 incrementTime()

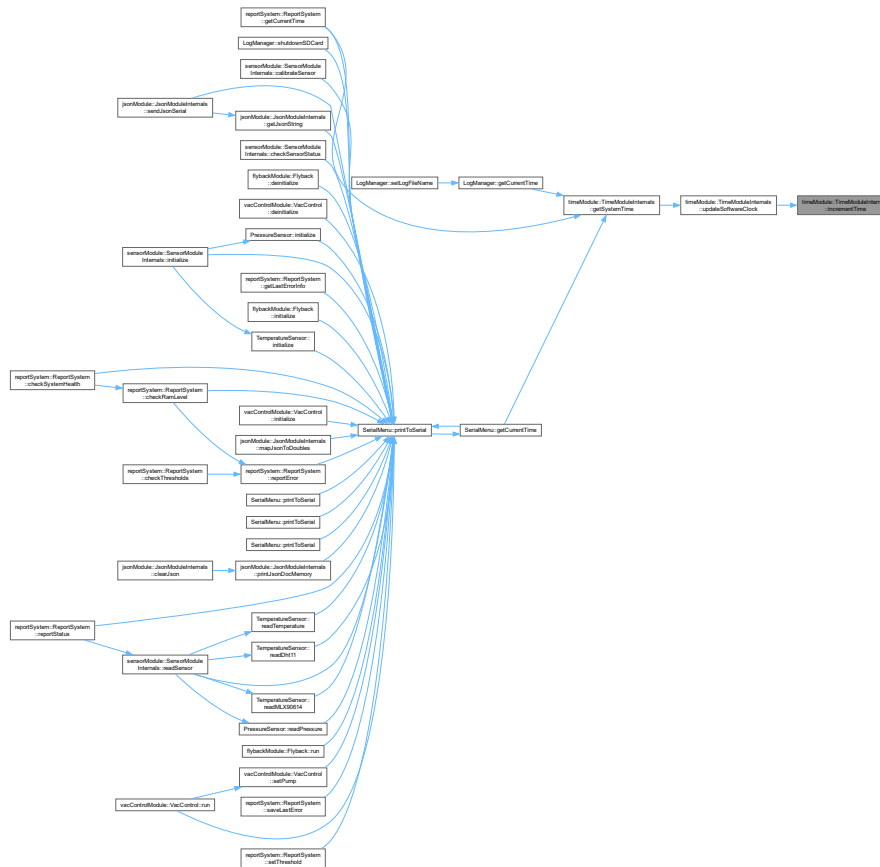
```
void TimeModuleInternals::incrementTime (
    DateTimeStruct * dt ) [static]
```

Function to increment the time of the system.

Parameters

<code>dt</code>	-> DateTimeStruct to increment time
-----------------	---

Here is the caller graph for this function:



6.30.2.5 setSystemTime()

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

Set the System Time object of the system.

Parameters

<code>dt</code>	-> DateTimeStruct to set the system time to.
-----------------	--

Here is the caller graph for this function:



6.30.2.6 setTimeFromHas()

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

Set the Time From Has object to the system time.

Parameters

<i>timeString</i>	-> 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](#)

6.31 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** ()
denitalize 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.

6.31.1 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.

6.31.2 Member Function Documentation

6.31.2.1 externProcess()

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

Process external data for scenarios (currently unused)

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

6.31.2.2 getExternPressure()

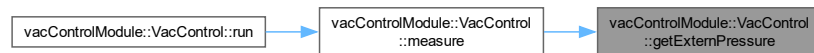
```
float VacControl::getExternPressure ( )
```

Gets the external pressure value.

Returns

The current external pressure value

Here is the caller graph for this function:

**6.31.2.3 getExternScenario()**

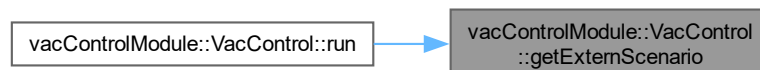
```
int VacControl::getExternScenario ( )
```

Getter function to retrieve the current external scenario state.

Returns

The current external scenario state (integer)

Here is the caller graph for this function:

**6.31.2.4 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.

6.31.2.5 getScenarioFromPotValue()

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

Determines the scenario based on the potentiometer value.

Parameters

<i>potValue</i>	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:

**6.31.2.6 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.)

6.31.2.7 isInitialized()

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

Get the state of the [VacControl](#) system.

Returns

true -> [VacControl](#) is initialized and ready
false -> [VacControl](#) is not initialized

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

Returns

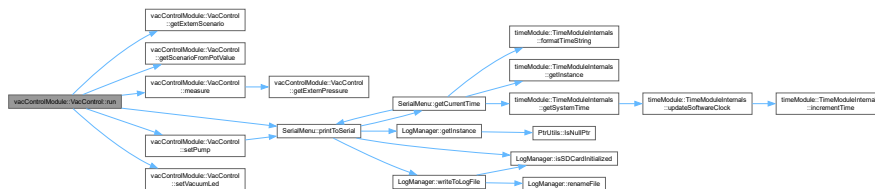
Here is the call graph for this function:



```
graph LR
    A[vacControlModule::VacControl::run] --> B[vacControlModule::VacControl::measure]
```

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

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:



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

Generated by Doxygen

Parameters

<i>pressure</i>	The external pressure value to set
-----------------	------------------------------------

6.31.2.11 setExternScenario()

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

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

Parameters

<i>pressure</i>	The external scenario pressure value
-----------------	--------------------------------------

6.31.2.12 setPump()

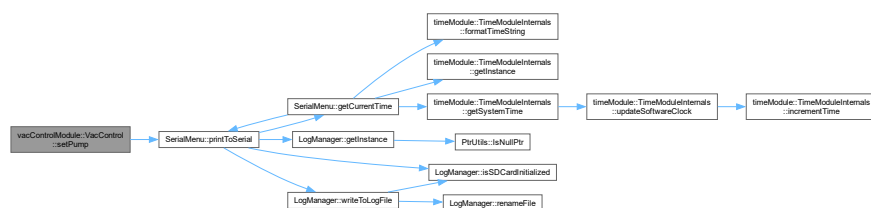
```
void VacControl::setPump (
    bool flag )
```

Set the Pump flag.

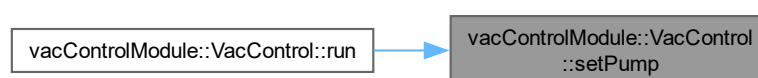
Parameters

<i>flag</i>	This is the boolean flag to set
-------------	---------------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



6.31.2.13 setVacuumLed()

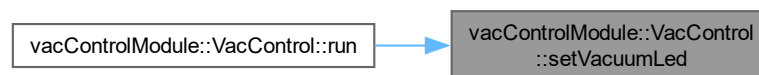
```
void VacControl::setVacuumLed (
    float pressure,
    float targetPressure )
```

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

Parameters

<i>pressure</i>	The current pressure in the system
<i>targetPressure</i>	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

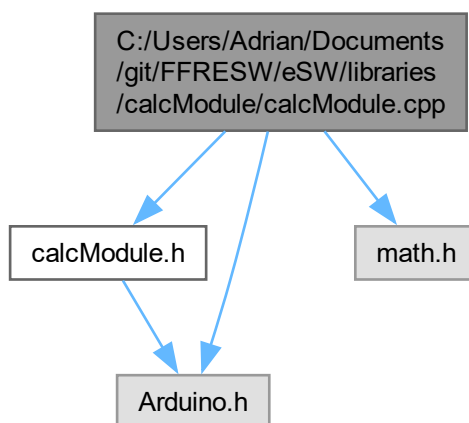
Chapter 7

File Documentation

7.1 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calculatorModule/calculatorModule.cpp File Reference

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

Include dependency graph for calculatorModule.cpp:



7.1.1 Detailed Description

Author

your name (you@domain.com)

Version

0.1

Date

2024-09-28

Copyright

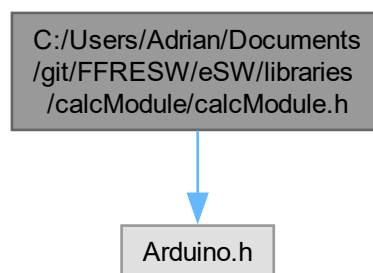
Copyright (c) 2024

7.2 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calculator/Module/calculatorModule.h File Reference

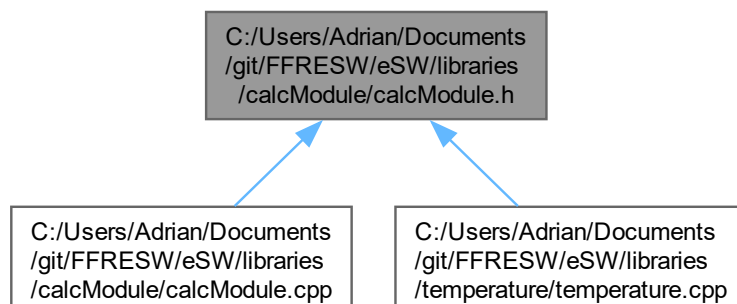
Header file for the calculation module handling sensor data.

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

Include dependency graph for calculatorModule.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.

7.2.1 Detailed Description

Header file for the calculation module handling sensor data.

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.3 calcModule.h

[Go to the documentation of this file.](#)

```
00001
00012 #ifndef CALCMODULE_H
00013 #define CALCMODULE_H
00014
00015 #include <Arduino.h>
00016
00017 namespace calcModule
00018 {
00019     enum Type
00020     {
00021         General,
00022         Pressure,
00023         Position
00024     };
00025
00026     class CalcModuleInternals
00027     {
00028     public:
```

```

00032     CalcModuleInternals();
00033     ~CalcModuleInternals();
00034
00042     static float calculateAverage(const float* data, int length);
00043
00051     static float findMaximum(const float* data, int length);
00052
00060     static float findMinimum(const float* data, int length);
00061
00069     static float calculateStandardDeviation(const float* data, int length);
00070
00078     static float findMedian(float* data, int length);
00079
00086     static float celsiusToFahrenheit(float celsius);
00087
00094     static float fahrenheitToCelsius(float fahrenheit);
00095
00102     static float celsiusToKelvin(float celsius);
00103
00110     static float kelvinToCelsius(float kelvin);
00111
00118     static float pascalToAtm(float pascal);
00119
00126     static float atmToPascal(float atm);
00127
00134     static float pascalToPsi(float pascal);
00135
00142     static float psiToPascal(float psi);
00143
00151     static float calculatePower(float voltage, float current);
00152
00160     static float calculateCurrent(float voltage, float resistance);
00161
00169     static float calculateResistance(float voltage, float current);
00170
00184     static float extractFloat(String response, int id);
00185
00186     static float extractFloatFromResponse(const String& response, Type type);
00187
00188     private:
00189
00196     static void sortArray(float* data, int length);
00197
00204     static float roundToPrecision(float value, int precision);
00205 };
00206 }
00207
00208 #endif // CALCMODULE_H

```

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

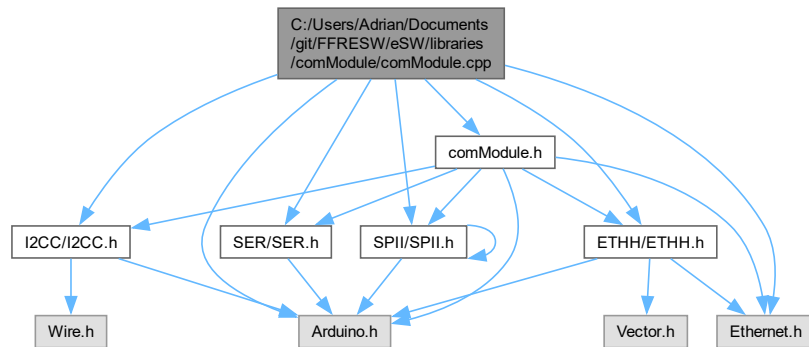
Implementation of the `comModule` class that utilizes various communication protocols.

```

#include <Arduino.h>
#include <Ethernet.h>
#include "comModule.h"
#include "ETHH/ETHH.h"
#include "I2CC/I2CC.h"
#include "SER/SER.h"
#include "SPII/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/ETHH.h"
00007 #include "I2CC/I2CC.h"
00008 #include "SER/SER.h"
00009 #include "SPII/SPII.h"
00010
00012 namespace comModule
00013 {
00015     class ComModuleInternals
00016     {
00017     public:
00018         ComModuleInternals();
00019         ~ComModuleInternals();
00020
00026         EthernetCommunication& getEthernet();
00027
00033         I2CCCommunication& getI2C();
00034
00040         SPICommunication& getSPI();
00041
00047         SerialCommunication& getSerial();
00048
00049     private:
00050         EthernetCommunication eth;
00051         I2CCCommunication 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
00034         void endI2C();
00035
00043         void i2cWrite(uint8_t deviceAddress, uint8_t* data, size_t length);
00044
00053         size_t i2cRead(uint8_t deviceAddress, uint8_t* buffer, size_t length);
00054
00061         bool isInitialized() const;
00062
00063     private:
00064         bool i2cInitialized = false;
00065     };
00066 }
00067
00068 #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 // FreeRTOS 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 flyback.h

```

00001 /*
00002  * flyback.h
00003  *
00004  * Created on: 07.12.2024
00005  * Author: domin
00006  */
00007 #ifndef FLYBACK_H
00008 #define FLYBACK_H
00009
00010 #include <Arduino.h>
00011 #include <Wire.h>
00012
00014 namespace flybackModule
00015 {
00017     enum class SwitchStates : int
00018     {
00019         HV_Module_OFF,
00020         HV_Module_MANUAL,
00021         HV_Module_REMOTE,

```

```

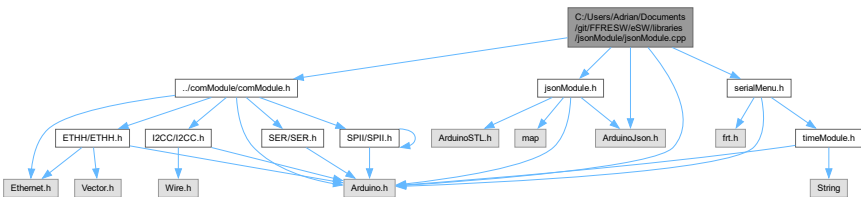
00022         HV_Module_INVALID
00023     };
00024
00027     typedef struct Measurement
00028     {
00029         float voltage;
00030         float current;
00031         float power;
00032         int digitalFreqValue;
00033         int digitalDutyValue;
00034         int dutyCycle;
00035         uint32_t frequency;
00036     } meas;
00037
00041     class Flyback
00042     {
00043     public:
00044         Flyback();
00045         ~Flyback();
00046
00051         void initialize();
00052
00057         void deinitialize();
00058
00065         bool isInitialized() const;
00066
00073         bool getTimerState();
00074
00080         void setTimerState(bool state);
00081
00088         SwitchStates getSwitchState();
00089
00095         Measurement measure();
00096
00101         void run();
00102
00108         void setExternFrequency(uint32_t frequency);
00109
00114         uint32_t getExternFrequency();
00115
00121         void setExternDutyCycle(int dutyCycle);
00122
00127         int getExternDutyCycle();
00128
00129     private:
00130         Measurement meas;
00131
00132         //Define Pins -->Signaltable
00133         static const int Main_Switch_OFF = 27;
00134         static const int Main_Switch_MANUAL = 28;
00135         static const int Main_Switch_REMOTE = 29;
00136         static const int Measure_ADC = A0;           //ADC PIN for Voltage Measurement
00137         static const int PWM_OUT = 11;
00138         static const int PWM_INV = 12;
00139
00140         //Variables for calculating HV
00141         const float R1 = 100000000;
00142         const float R2 = 10000;
00143         const float ADC_Max_Value = 1023.0;
00144         const float Vcc = 5.0;
00145
00146         //Define Pins --> Signaltable
00147         static const int HV_Module_ON = 37;
00148         static const int HV_Module_OFF = 36;
00149         static const int HV_Module_Working = 35;
00150         static const int PWM_Frequency = A1;
00151         static const int PWM_DutyCycle = A2;
00152
00153         bool _flybackInitialized;
00154         bool _timerInitialized;
00155
00156         // States
00157         static SwitchStates lastState;
00158         static bool lastTimerState;
00159         static int lastPWMFrequency;
00160         static int lastPWMDutyCycle;
00161
00162         void timerConfig();
00163
00167         void setPWMFrequency(uint32_t frequency, int dutyCycle);
00168
00175     };
00176 }
00177
00178 #endif //FLYBACK_H

```

7.12 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 "../comModule/comModule.h"
#include <jsonModule.h>
#include <serialMenu.h>
Include dependency graph for jsonModule.cpp:
```



7.12.1 Detailed Description

Implementation of the `jsonModule` class.

Version

0.1

Date

2024-01-26

Copyright

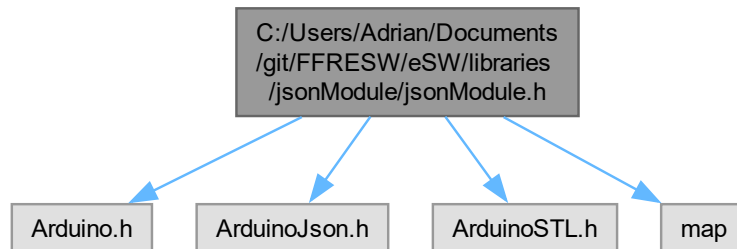
Copyright (c) 2024

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

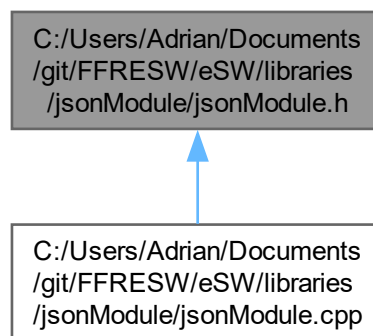
```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <ArduinoSTL.h>
```

```
#include <map>
```

Include dependency graph for jsonModule.h:



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



Classes

- class `jsonModule::JsonModuleInternals`

Namespaces

- namespace `jsonModule`
Namespace for the JSON module.

Macros

- `#define ARDUINO_STL_MEMORY 0`

7.13.1 Detailed Description

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.14 jsonModule.h

[Go to the documentation of this file.](#)

```

00001
00012 #ifndef JSONMODULE_H
00013 #define JSONMODULE_H
00014
00015 #ifndef ARDUINO_STL_MEMORY
00016 #define ARDUINO_STL_MEMORY 0 // Disable STL memory management (new, delete, new_handler)
00017 #endif
00018
00019 #include <Arduino.h>
00020 #include <ArduinoJson.h>
00021 #include <ArduinoSTL.h>
00022 #include <map>
00023
00025 namespace jsonModule
00026 {
00028     class JsonModuleInternals
00029     {
00030     public:
00031         JsonModuleInternals();
00032         ~JsonModuleInternals();
00033
00040         void createJson(const char* key, const char* value);
00041
00048         void createJsonFloat(const char* key, float value);
00049
00056         void createJsonInt(const char* key, int value);
00057
00064         void createJsonString(const char* key, String& value);
00065
00072         void createJsonStringConst(const char* key, const String& value);
00073
00078         void sendJsonSerial();
00079
00085         void sendJsonEthernet(const char* endpoint);
00086
00092         String getJsonString() const;
00093
00094
00103         std::map<String, float> mapJsonToDoubles(const String& rawJson);
00104
00109         void clearJson();
00110
00115         void printJsonDocMemory();
00116
00117         size_t jsonBuffer;
00118
00119     private:
00120         StaticJsonDocument<512> jsonDoc;
00121
00122     };
00123 }
00124
00125 #endif // JSONMODULE_H

```

7.15 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
00023     locker::ScopedLock lockEthernetScoped()
00024     {
00025         ethernetConnected = true;
00026         logState("Ethernet connected", ethernetConnected);
00027         return locker::ScopedLock(ethernetMutex);
00028     }
00029
00030     locker::ScopedLock lockTemperatureScoped()
00031     {
00032         temperatureReading = true;
00033         logState("Temperature reading", temperatureReading);
00034         return locker::ScopedLock(temperaturQueueMutex);
00035     }
00036
00037     locker::ScopedLock lockSerialScoped()
00038     {
00039         serialReading = true;
00040         logState("Serial reading", serialReading);
00041         return locker::ScopedLock(serialMutex);
00042     }
00043
00044 private:
00045     // Mutexes for the different resources
00046     frt::Mutex temperaturQueueMutex;
00047     frt::Mutex ethernetMutex;
00048     frt::Mutex serialMutex;
00049
00050     // Semaphores for the different resources
00051     frt::Semaphore ethernetSemaphore;
00052     frt::Semaphore temperatureSemaphore;
00053     frt::Semaphore serialSemaphore;
00054
00055     // Flags for the different resources
00056     bool ethernetConnected = false;
00057     bool temperatureReading = false;
00058     bool serialReading = false;
00059
00060     // Logger instance
00061     LogManager* _logger = LogManager::getInstance();
00062
00063     void logState(const char* label, bool state)
00064     {
00065         char buffer[64];
00066         snprintf(buffer, sizeof(buffer), "[LockerBase] %s: %s", label, state ? "true" : "false");
00067         _logger->writeToLogFile(buffer);
00068     }
00069 };
00070
00071 #endif // LOCKER_BASE_H

```

7.16 scopedLock.h

```

00001 #ifndef LOCKER_BASE_SCOPEDLOCK_H
00002 #define LOCKER_BASE_SCOPEDLOCK_H
00003
00004 #include <frt.h>
00005
00006 namespace locker
00007 {

```



```

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

```

7.17 logManager.h

```

00001 #ifndef LOGMANAGER_H
00002 #define LOGMANAGER_H
00003
00004 #include <Arduino.h>
00005 #include <SD.h>
00006 #include <SPI.h>
00007 #include <String.h>
00008 #include <timeModule.h>
00009
00011 class LogManager
00012 {
00013 public:
00014
00020     static LogManager* getInstance();
00021
00027     void initSDCard(int cs);
00028
00032     void shutdownSDCard();
00033
00034     void flushLogs();
00035
00042     bool isSDCardInitialized() const;
00043
00048     static String getCurrentTime();
00049
00055     void setLogFileName(const String& fileName);
00056
00064     bool writeToLogFile(const String& logMessage);
00065
00072     void renameFile(const String& oldName, const String& newName);
00073
00074 private:
00075     LogManager();
00076     ~LogManager();
00077
00078     static LogManager* _instance;
00079     File logFile;
00080     bool sdCardInitialized = false;
00081     String logFileName;
00082     String baseLogFileName;
00083
00084     static const int chipSelectPinEth = 10; // Default CS pin for SD card
00085     static const int maxLogFileSize = 1024 * 1024 * 100; // 100 MB
00086
00087     LogManager(const LogManager&) = delete;
00088     LogManager& operator=(const LogManager&) = delete;
00089 };

```

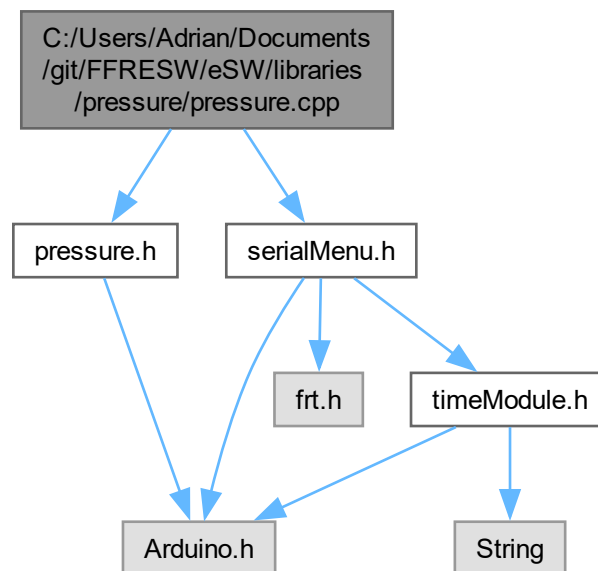
```
00090
00091
00092 #endif // LOGMANAGER_H
```

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

Implementation of the pressure class.

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

Include dependency graph for pressure.cpp:



7.18.1 Detailed Description

Implementation of the pressure class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.19 pressure.h

```

00001 #ifndef PRESSURESENSOR_H
00002 #define PRESSURESENSOR_H
00003
00004 #include <Arduino.h>
00005
00007 class PressureSensor
00008 {
00009 public:
00010     PressureSensor();
00011     ~PressureSensor();
00012
00017     void initialize();
00018
00024     float readPressure();
00025
00032     bool isInitialized() const;
00033
00034 private:
00035     bool _pressureSensorInitialized;
00036     static const int PRESSURE_SENSOR_PIN = 0;
00037
00044     float readAnalogSensor(int pin);
00045
00046 };
00047
00048 #endif // PRESSURESENSOR_H

```

7.20 ptrUtils.h

```

00001 #ifndef PTRUTILS_H
00002 #define PTRUTILS_H
00003
00004 #include <Arduino.h>
00005 #include <serialMenu.h>
00006
00013 template <typename T>
00014 static inline void SafeDelete(T*& ptr)
00015 {
00016     if (ptr != nullptr)
00017     {
00018         delete ptr;
00019         ptr = nullptr;
00020     }
00021 }
00022
00029 template <typename T>
00030 static inline void SafeDeleteArray(T*& ptr)
00031 {
00032     if (ptr != nullptr)
00033     {
00034         delete[] ptr;
00035         ptr = nullptr;
00036     }
00037 }
00038
00047 template <typename T>
00048 static inline void Verify(const T& value, const T& expected, const char* errorMsg = nullptr)
00049 {
00050     if (value != expected)
00051     {
00052         if (errorMsg)
00053         {
00054             SerialMenu::printToSerial(errorMsg);
00055         }
00056         else
00057         {
00058             String errStr;
00059             errStr += "[ERROR] Verification failed: Value ";
00060             errStr += value;
00061             errStr += " does not match expected ";
00062             errStr += expected;
00063             errStr += " .";
00064             SerialMenu::printToSerial(errStr);
00065         }
00066         while (true); // Halt execution
00067     }
00068 }
00069
00078 template <typename T>
00079 static inline void Verify(T* value, T* expected, const char* errorMsg = nullptr)
00080 {

```

```

00081     if (value != expected)
00082     {
00083         if (errorMsg)
00084         {
00085             SerialMenu::printToSerial(errorMsg);
00086         }
00087         else
00088         {
00089             String errStr;
00090             errStr += "[ERROR] Verification failed: Pointer ";
00091             errStr += (unsigned long)value, HEX;
00092             errStr += ") does not match expected pointer ";
00093             errStr += (unsigned long)expected, HEX;
00094             errStr += ").";
00095             SerialMenu::printToSerial(errStr);
00096         }
00097         while (true); // Halt execution
00098     }
00099 }
00100
00101 template <typename T>
00102 static inline void Verify(T* value, const char* errorMsg = nullptr)
00103 {
00104     if (value != nullptr) // Directly compare with nullptr (no std::nullptr_t)
00105     {
00106         if (errorMsg)
00107         {
00108             SerialMenu::printToSerial(errorMsg);
00109         }
00110         else
00111         {
00112             String errStr;
00113             errStr += "[ERROR] Verification failed: Pointer ";
00114             errStr += (unsigned long)value, HEX;
00115             errStr += ") is not null.";
00116             SerialMenu::printToSerial(errStr);
00117         }
00118         while (true); // Halt execution
00119     }
00120 }
00121
00122 #define tryDeletePtr(ptr) \
00123     if (PtrUtils::IsValidPtr(ptr)) \
00124         SafeDelete(ptr); \
00125
00126
00127 class PtrUtils
00128 {
00129 public:
00130     template <typename T>
00131     static inline bool IsNullPtr(T* ptr)
00132     {
00133         return ptr == nullptr;
00134     }
00135
00136     template <typename T>
00137     static inline bool IsValidPtr(T* ptr)
00138     {
00139         return ptr != nullptr;
00140     }
00141 };
00142
00143 template <typename T>
00144 static inline void ClearArray(T* array, size_t size)
00145 {
00146     for (size_t i = 0; i < size; ++i)
00147     {
00148         array[i] = T();
00149     }
00150 }
00151
00152 template <typename T>
00153 static inline void PrintPtrInfo(T* ptr, const char* ptrName = "Pointer")
00154 {
00155     if (ptr == nullptr)
00156     {
00157         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String("is nullptr"));
00158     }
00159     else
00160     {
00161         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String(" points to address: 0x") +
00162             (uintptr_t)ptr, HEX);
00163     }
00164 }
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204

```

```

00210 template <typename T>
00211 class ScopedPointer
00212 {
00213 private:
00214     T* ptr;
00215
00216 public:
00217     explicit ScopedPointer(T* p = nullptr) : ptr(p) {}
00218     ~ScopedPointer() { SafeDelete(ptr); }
00219
00220     T* get() const { return ptr; }
00221
00222     T* release()
00223     {
00224         T* temp = ptr;
00225         ptr = nullptr;
00226         return temp;
00227     }
00228
00229     void reset(T* p = nullptr)
00230     {
00231         SafeDelete(ptr);
00232         ptr = p;
00233     }
00234
00235     T& operator*() const { return *ptr; }
00236
00237     T* operator->() const { return ptr; }
00238 };
00239
00240 template <typename T>
00241 class PointerWrapper
00242 {
00243 private:
00244     T* ptr;
00245
00246 public:
00247     explicit PointerWrapper(T* p = nullptr) : ptr(p) {}
00248     ~PointerWrapper() { SafeDelete(ptr); }
00249
00250     T* get() const { return ptr; }
00251
00252     T* release()
00253     {
00254         T* temp = ptr;
00255         ptr = nullptr;
00256         return temp;
00257     }
00258
00259     void reset(T* p = nullptr)
00260     {
00261         SafeDelete(ptr);
00262         ptr = p;
00263     }
00264
00265     T& operator*() { return *ptr; }
00266
00267     T* operator->() { return ptr; }
00268 };
00269
00270 #endif // PTRUTILS_H

```

7.21 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp File Reference

Unified system health and error reporting module.

```

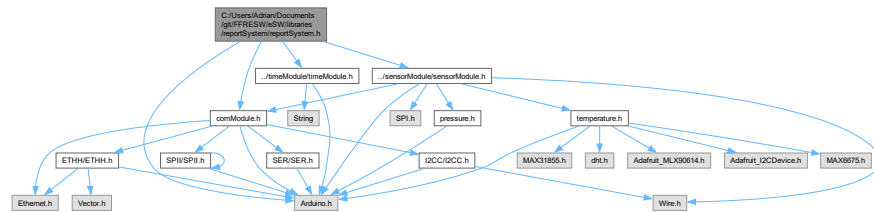
#include "reportSystem.h"
#include "ptrUtils.h"
#include <Arduino.h>
#include <time.h>
#include <EEPROM.h>
#include <ErriezMemoryUsage.h>

```

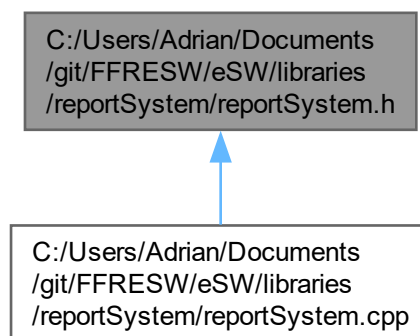


```
#include "../timeModule/timeModule.h"
```

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**

7.22.1 Detailed Description

Author

your name (you@domain.com)

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.23 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
00176     private:
00177         float tempThreshold;
00178         float pressureThreshold;
00179         unsigned long lastHealthCheck;
00180         const unsigned long healthCheckInterval = 10000; // 10 seconds
00181         unsigned long busyTime = 0;
00182         unsigned long idleTime = 0;
00183         unsigned long lastTimestamp = 0;
00184
00193         bool checkSensors(bool checkTemp, bool checkPress);
00194
00204         bool checkCommunication(bool checkEth, bool checkSpi, bool checkI2c);
00205
00213         bool checkMemory(unsigned int threshold);
00214
00215         sensorModule::SensorModuleInternals* _sens;
00216         comModule::ComModuleInternals* _com;
00217         timeModule::TimeModuleInternals* _time;
00218     };
00219 }
00220
00221 #endif // REPORTSYSTEM_H

```

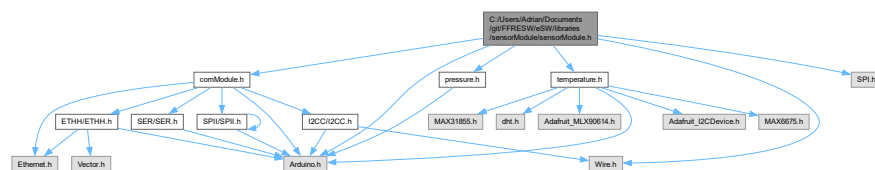
7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h File Reference

```

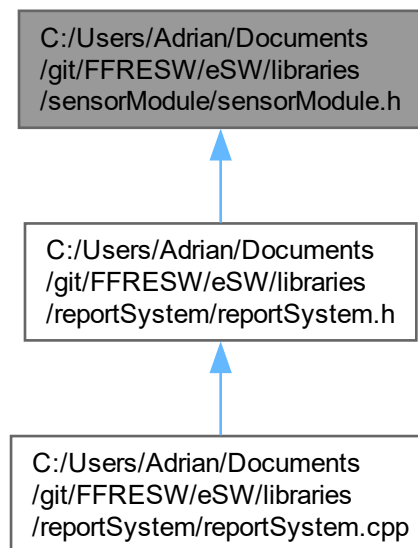
#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <pressure.h>
#include <temperature.h>
#include <comModule.h>

```

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 , MAX6675_Celsius , MAX6675_Fahrenheit , MAX6675_Kelvin ,
 MAX31855_Celsius , MAX31855_Fahrenheit , MAX31855_Kelvin , UNKNOWN }**
Enum class for the sensor types.

7.24.1 Detailed Description

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.25 sensorModule.h

[Go to the documentation of this file.](#)

```

00001
00012 #ifndef SENSORMODULE_H
00013 #define SENSORMODULE_H
00014
00015 #include <Arduino.h>
00016 #include <Wire.h>
00017 #include <SPI.h>
00018 #include <pressure.h>
00019 #include <temperature.h>
00020
00021 #include <comModule.h>
00022
00023
00025 namespace sensorModule
00026 {
00028     enum class SensorType
00029     {
00030         TEMPERATURE,
00031         OBJECTTEMPERATURE,
00032         AMBIENTTEMPERATURE,
00033         PRESSURE,
00034         DHT11,
00035         MAX6675_Celsius,
00036         MAX6675_Fahrenheit,
00037         MAX6675_Kelvin,
00038         MAX31855_Celsius,
00039         MAX31855_Fahrenheit,
00040         MAX31855_Kelvin,
00041         UNKNOWN
00042     };
00043
00045     class SensorModuleInternals : public TemperatureSensor, public PressureSensor
00046     {
00047     public:
00048         SensorModuleInternals();
00049         ~SensorModuleInternals();
00050
00055         //void beginSensor();
00056
00061         void initialize();
00062
00069         float readSensor(SensorType type);
00070
00078         bool calibrateSensor(SensorType type);
00079
00087         bool checkSensorStatus(SensorType type);
00088
00089     private:
00090         TemperatureSensor _temperatureSensor;
00091         PressureSensor _pressureSensor;
00092
00093         bool _i2cSensorInitialized;
00094         bool _spiSensorInitialized;
00095
00096         static const uint8_t I2C_SENSOR_ADDRESS = 0x76;
00097         static const int SPI_CS_PIN = 10;
00098     };
00099 }
00100
00101 #endif // SENSORMODULE_H

```

7.26 serialMenu.h

```

00001 #ifndef SERIAL_MENU_H
00002 #define SERIAL_MENU_H
00003
00004 #include <Arduino.h>
00005 #include <firt.h>
00006 #include <timeModule.h>
00007
00008
00010 struct MenuItem
00011 {
00012     const char* label;
00013     char key;
00014     void (*callback)();
00015 };
00016
00018 class SerialMenu
00019 {
00020 public:
00021
00023     enum class OutputLevel
00024     {
00025         DEBUG,
00026         INFO,
00027         WARNING,
00028         ERROR,
00029         CRITICAL,
00030         STATUS,
00031         PLAIN
00032     };
00033
00034     SerialMenu();
00035     ~SerialMenu();
00036
00043     void load(MenuItem* items, size_t size);
00044
00049     void show();
00050
00055     void run();
00056
00064     static void printToSerial(OutputLevel level, const String& message, bool newLine = true, bool
logMessage = false);
00065
00073     static void printToSerial(OutputLevel level, const __FlashStringHelper* message, bool newLine =
true, bool logMessage = false);
00074
00081     static void printToSerial(const String& message, bool newLine = true, bool logMessage = false);
00082
00089     static void printToSerial(const __FlashStringHelper* message, bool newLine = true, bool logMessage
= false);
00090
00095     static String getCurrentTime();
00096
00097 private:
00098     MenuItem* currentMenu;
00099     size_t menuSize;
00100 };
00101
00102 #endif // SERIAL_MENU_H

```

7.27 C:/Users/Adrian/Documents/git/FFRESW/e↵ SW/libraries/temperature/temperature.cpp File Reference

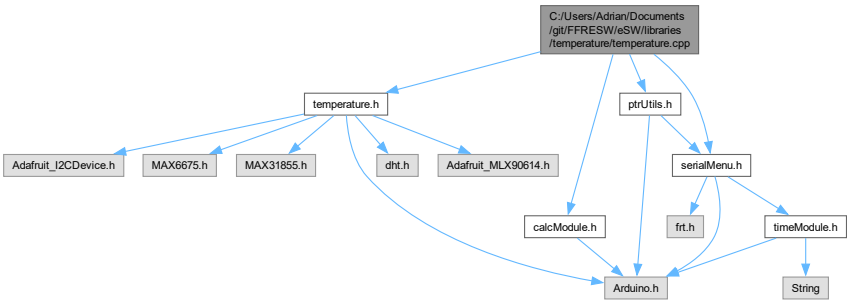
Implementation of the temperature class.

```

#include <temperature.h>
#include <serialMenu.h>
#include <ptrUtils.h>
#include <calcModule.h>

```

Include dependency graph for temperature.cpp:



7.27.1 Detailed Description

Implementation of the temperature class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.28 temperature.h

```
00001 #ifndef TEMPERATURESENSOR_H
00002 #define TEMPERATURESENSOR_H
00003
00004 #include <Arduino.h>
00005 #include <dht.h>
00006 #include <Adafruit_MLX90614.h>
00007 #include <Adafruit_I2CDevice.h>
00008 #include "MAX6675.h"
00009 #include "MAX31855.h"
00010
00011 enum Units
00012 {
00013     Celsius,
00014     Kelvin,
00015     Fahrenheit
00016 };
00017
00019 class TemperatureSensor
00020 {
00021 public:
00022     TemperatureSensor();
00023     ~TemperatureSensor();
00024
00029     void initialize();
00030
00036     float readTemperature();
00037
00043     float readDht11();
00044
```

```

00051     float readMLX90614(int choice);
00052
00053
00054     float readMAX6675(Units unit);
00055
00056     float readMAX31855(Units unit);
00057
00064     bool isInitialized() const;
00065
00066 private:
00067     bool _temperatureSensorInitialized;
00068     static const int TEMP_SENSOR_PIN = A0;
00069     static const int TEMP_SENSOR_PIN_DIG = 4;
00070     static const int DHT11_PIN = 7;
00071
00072     static const uint8_t MLX90614 = 0x5A;
00073     static const uint8_t AMBIENT_TEMP = 0x06;
00074     static const uint8_t OBJECT_TEMP = 0x07;
00075
00076     Adafruit_MLX90614 _mlx = Adafruit_MLX90614();
00077
00078     // Settings for MAX6675 Sensor
00079     MAX6675* _thermocouple6675;
00080     static const uint8_t thermo6675CS = 9;
00081
00082     // Settings for MAX31855 Sensor
00083     MAX31855_Class _thermocouple31855;
00084     static const uint8_t thermo31855CS = 11;
00085
00086
00093     float readAnalogSensor(int pin);
00094
00101     float readDigitalSensor(int pin);
00102
00103     dht DHT;
00104
00105 };
00106
00107 #endif // TEMPERATURESENSOR_H

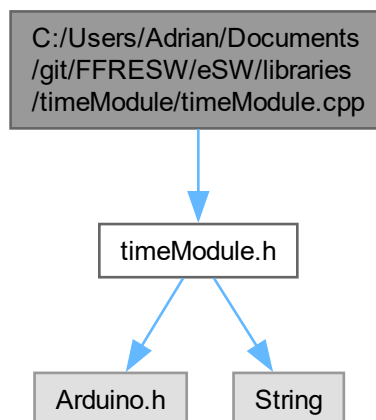
```

7.29 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`:



7.29.1 Detailed Description

Implementation of the `timeModule` class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.30 timeModule.h

```
00001 #ifndef TIMEMODULE_H
00002 #define TIMEMODULE_H
00003
00004 #include <Arduino.h>
00005 #include <String>
00006
00008 namespace timeModule
00009 {
00011     typedef struct DateTimeStruct
00012     {
00013         int year;
00014         int month;
00015         int day;
00016         int hour;
00017         int minute;
00018         int second;
00019     } DateTimeStruct;
00020
00022     class TimeModuleInternals
00023     {
00024     public:
00025         TimeModuleInternals();
00026         ~TimeModuleInternals();
00027
00033         static void incrementTime(DateTimeStruct *dt);
00034
00041         static String formatTimeString(const DateTimeStruct &dt);
00042
00050         bool setTimeFromHas(const String& timeString);
00051
00057         void setSystemTime(const DateTimeStruct& dt);
00058
00063         void updateSoftwareClock();
00064
00070         DateTimeStruct getSystemTime();
00071
00077         static TimeModuleInternals* getInstance();
00078
00079     private:
00080         DateTimeStruct dt;
00081         unsigned long startMillis = 0;
00082     };
00083 }
00084
00085 #endif // TIMEMODULE
```

7.31 vacControl.h

```

00001  /*
00002  * flyback.h
00003  *
00004  * Created on: 28.03.2025
00005  * Author: domin
00006  */
00007  #ifndef VACCONTROL_H
00008  #define VACCONTROL_H
00009
00010  #include <Arduino.h>
00011  #include <Wire.h>
00012
00013
00014  namespace vacControlModule
00015  {
00016      enum class SwitchStates : int
00017      {
00018          Main_Switch_OFF,
00019          Main_Switch_MANUAL,
00020          Main_Switch_REMOTE,
00021          Main_switch_INVALID,
00022          PUMP_ON,
00023          PUMP_OFF
00024      };
00025      typedef struct Pressure
00026      {
00027          float pressure;
00028      } meas;
00029
00030      enum Scenarios
00031      {
00032          Scenario_1 = 0,
00033          Scenario_2 = 1,
00034          Scenario_3 = 2,
00035          Scenario_4 = 3,
00036          Scenario_5 = 4,
00037          Invalid_Scenario = -1
00038      };
00039
00040      class VacControl
00041      {
00042      public:
00043          VacControl();
00044          ~VacControl();
00045
00046          void initialize();
00047
00048          void deinitialize();
00049
00050          bool isInitialized() const;
00051
00052          SwitchStates getSwitchState();
00053
00054          Scenarios getScenario();
00055
00056          Pressure measure();
00057
00058          void setVacuumLed(float pressure, float targetPressure);
00059
00060          int getScenarioFromPotValue(int potValue);
00061
00062          void setPump(bool flag);
00063
00064          void run();
00065
00066          void setExternScenario(int pressure);
00067
00068          int getExternScenario();
00069
00070          void externProcess();
00071
00072          void setExternPressure(float pressure);
00073
00074          float getExternPressure();
00075
00076      private:
00077          Pressure meas;
00078
00079          //Define Pins --> Main_Switch
00080          static const int Main_Switch_OFF = 27; //Main_Switch OFF Mode 27
00081          static const int Main_Switch_MANUAL = 28; //Main_Switch Manual Mode 28

```



```
00170     static const int Main_Switch_REMOTE = 29;    //Main_Switch Remote Mode 29
00171
00172     //Define Pins --> Vacuum Logic
00173     static const int Switch_Pump_ON = 23;        //Button to turn Pump ON 37
00174     static const int Pump_Status_LED = 24;       //OUTPUT to see State off Pump
00175     static const int Pump_Relay = 25;           //OUTPUT to turn on/off Relais
00176     static const int targetVacuumLED = 26;       //OUTPUT to see Vacuum reached
00177     static const int targetPressure = A4;        //Potentiometer for Regulation
00178
00179     //Variables to save Values
00180     int currentScenario = -1;
00181
00182     //Variables for TargetPressure
00183     static const float TARGET_PRESSURE_1 = 1;
00184     static const float TARGET_PRESSURE_2 = 0.8f;
00185     static const float TARGET_PRESSURE_3 = 0.5;
00186     static const float TARGET_PRESSURE_4 = 0.01f;
00187
00188     // TODO NEW ADDED-> CHECK FUNCTIONALTY WITH FRANIC
00189     static int lastState;
00190     static int lastPumpState;
00191
00192
00193     bool _vacControlInitialized;
00194 };
00195 }
00196
00197 #endif //VACCONTROL_H
```


Index

<code>atmToPascal</code>	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule::CalcModuleInternals, 16	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.cpp, 127, 129
<code>beginEthernet</code>	<code>comModule::EthernetCommunication</code> , 26	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h, 130
<code>beginI2C</code>	<code>comModule::I2CCommunication</code> , 41	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h, 130
<code>beginSerial</code>	<code>comModule::SerialCommunication</code> , 78	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h, 131
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.cpp,	105	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h, 132
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h,	106, 107	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/calcModule.h, 133
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp,	108	C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h, 134
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h,	109	<code>calcModule::CalcModuleInternals</code> , 15
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH/ETHH.h,	109	<code>atmToPascal</code> , 16
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CG/I2CG.h,	111	<code>calculateAverage</code> , 16
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER/SER.h,	112	<code>calculateCurrent</code> , 16
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPI/SPI.h,	112	<code>calculatePower</code> , 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h,	113	<code>calculateResistance</code> , 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h,	113	<code>calculateStandardDeviation</code> , 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp,	115	<code>celsiusToFahrenheit</code> , 18
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h,	115, 117	<code>celsiusToKelvin</code> , 18
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h,	118	<code>extractFloat</code> , 19
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h,	118	<code>fahrenheitToCelsius</code> , 19
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp,	120	<code>findMaximum</code> , 19
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h,	121	<code>findMedian</code> , 20
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h,	121	<code>findMinimum</code> , 20
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp,	123	<code>kelvinToCelsius</code> , 20
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h,	124, 126	<code>pascalToAtm</code> , 21

- calcModule::CalcModuleInternals, 18
- checkRamLevel
 - reportSystem::ReportSystem, 64
- checkSensorStatus
 - sensorModule::SensorModuleInternals, 76
- checkSystemHealth
 - reportSystem::ReportSystem, 65
- checkThresholds
 - reportSystem::ReportSystem, 65
- comModule, 9
- comModule::ComModuleInternals, 22
 - getEthernet, 23
 - getI2C, 23
 - getSerial, 23
 - getSPI, 23
- comModule::EthernetCommunication, 24
 - beginEthernet, 26
 - getClient, 26
 - getCompound, 26–28
 - getCompoundInternal, 28
 - getParameter, 29
 - getParsedCompound, 29, 30
 - getRequestedEndpoint, 31
 - getSendDataFlag, 31
 - getSpecificEndpoint, 31
 - isInitialized, 32
 - parseCompoundResponse, 32
 - receiveEthernetData, 32
 - sendCommand, 33
 - sendEthernetData, 33
 - sendJsonResponse, 33
 - setCompound, 34, 35
 - setCompoundInternal, 35
 - setParameter, 36
 - setSendDataFlag, 37
- comModule::I2CCommunication, 40
 - beginI2C, 41
 - i2cRead, 41
 - i2cWrite, 42
 - isInitialized, 42
- comModule::SerialCommunication, 78
 - beginSerial, 78
 - isInitialized, 78
 - receiveSerialData, 79
 - sendSerialData, 79
- comModule::SPICommunication, 87
 - isInitialized, 87
 - spiRead, 87
 - spiWrite, 88
- comModuleInternals, 24
- createJson
 - jsonModule::JsonModuleInternals, 43
- createJsonFloat
 - jsonModule::JsonModuleInternals, 43
- createJsonInt
 - jsonModule::JsonModuleInternals, 43
- createJsonString
 - jsonModule::JsonModuleInternals, 44
- createJsonStringConst
 - jsonModule::JsonModuleInternals, 44
- detectStackOverflow
 - reportSystem::ReportSystem, 66
- externProcess
 - vacControlModule::VacControl, 98
- extractFloat
 - calcModule::CalcModuleInternals, 19
- fahrenheitToCelsius
 - calcModule::CalcModuleInternals, 19
- findMaximum
 - calcModule::CalcModuleInternals, 19
- findMedian
 - calcModule::CalcModuleInternals, 20
- findMinimum
 - calcModule::CalcModuleInternals, 20
- flybackModule, 10
- flybackModule::Flyback, 37
 - getSwitchState, 38
 - getTimerState, 38
 - isInitialized, 38
 - measure, 39
 - setExternDutyCycle, 39
 - setExternFrequency, 39
 - setTimerState, 40
- flybackModule::Measurement, 55
- formatTimeString
 - timeModule::TimeModuleInternals, 92
- get
 - PointerWrapper< T >, 57
 - ScopedPointer< T >, 73
- getClient
 - comModule::EthernetCommunication, 26
- getCompound
 - comModule::EthernetCommunication, 26–28
- getCompoundInternal
 - comModule::EthernetCommunication, 28
- getCPULoad
 - reportSystem::ReportSystem, 66
- getCurrentTime
 - LogManager, 48
 - reportSystem::ReportSystem, 67
 - SerialMenu, 80
- getEthernet
 - comModule::ComModuleInternals, 23
- getExternPressure
 - vacControlModule::VacControl, 98
- getExternScenario
 - vacControlModule::VacControl, 99
- getI2C
 - comModule::ComModuleInternals, 23
- getInstance
 - LogManager, 48
 - timeModule::TimeModuleInternals, 93
- getJsonString

- jsonModule::JsonModuleInternals, 44
- getLastError
 - reportSystem::ReportSystem, 67
- getLastErrorInfo
 - reportSystem::ReportSystem, 67
- getMemoryStatus
 - reportSystem::ReportSystem, 68
- getParameter
 - comModule::EthernetCommunication, 29
- getParsedCompound
 - comModule::EthernetCommunication, 29, 30
- getRequestedEndpoint
 - comModule::EthernetCommunication, 31
- getScenario
 - vacControlModule::VacControl, 99
- getScenarioFromPotValue
 - vacControlModule::VacControl, 99
- getSendDataFlag
 - comModule::EthernetCommunication, 31
- getSerial
 - comModule::ComModuleInternals, 23
- getSpecificEndpoint
 - comModule::EthernetCommunication, 31
- getSPI
 - comModule::ComModuleInternals, 23
- getStackDump
 - reportSystem::ReportSystem, 68
- getSwitchState
 - flybackModule::Flyback, 38
 - vacControlModule::VacControl, 100
- getSystemTime
 - timeModule::TimeModuleInternals, 94
- getTimerState
 - flybackModule::Flyback, 38
- i2cRead
 - comModule::I2CCommunication, 41
- i2cWrite
 - comModule::I2CCommunication, 42
- incrementTime
 - timeModule::TimeModuleInternals, 95
- initialize
 - sensorModule::SensorModuleInternals, 76
- initSDCard
 - LogManager, 50
- isInitialized
 - comModule::EthernetCommunication, 32
 - comModule::I2CCommunication, 42
 - comModule::SerialCommunication, 78
 - comModule::SPICommunication, 87
 - flybackModule::Flyback, 38
 - PressureSensor, 60
 - TemperatureSensor, 89
 - vacControlModule::VacControl, 100
- IsNullPtr
 - PtrUtils, 61
- isSDCardInitialized
 - LogManager, 51
- IsValidPtr
 - PtrUtils, 62
- jsonModule, 11
- jsonModule::JsonModuleInternals, 42
 - createJson, 43
 - createJsonFloat, 43
 - createJsonInt, 43
 - createJsonString, 44
 - createJsonStringConst, 44
 - getJsonString, 44
 - mapJsonToDoubles, 45
 - sendJsonEthernet, 46
- jsonModuleInternals, 46
- kelvinToCelsius
 - calcModule::CalcModuleInternals, 20
- load
 - SerialMenu, 82
- locker, 11
- locker::ScopedLock, 71
 - ScopedLock, 71, 72
- LockerBase, 47
 - lockEthernetScoped, 47
 - lockSerialScoped, 47
 - lockTemperatureScoped, 47
- lockEthernetScoped
 - LockerBase, 47
- lockSerialScoped
 - LockerBase, 47
- lockTemperatureScoped
 - LockerBase, 47
- LogManager, 48
 - getCurrentTime, 48
 - getInstance, 48
 - initSDCard, 50
 - isSDCardInitialized, 51
 - renameFile, 51
 - setLogFileName, 53
 - writeToLogFile, 53
- LogMapper, 55
- mapJsonToDoubles
 - jsonModule::JsonModuleInternals, 45
- measure
 - flybackModule::Flyback, 39
 - vacControlModule::VacControl, 100
- Measurement, 55
- MenuItem, 56
- operator->
 - PointerWrapper< T >, 58
 - ScopedPointer< T >, 73
- operator*
 - PointerWrapper< T >, 57
 - ScopedPointer< T >, 73
- OutputLevel, 56
- parseCompoundResponse
 - comModule::EthernetCommunication, 32

pascalToAtm
 calcModule::CalcModuleInternals, 21
 pascalToPsi
 calcModule::CalcModuleInternals, 21
 PointerWrapper< T >, 57
 get, 57
 operator->, 58
 operator*, 57
 release, 58
 reset, 58
 PressureSensor, 59
 isInitialized, 60
 readPressure, 60
 printToSerial
 SerialMenu, 83, 84
 psiToPascal
 calcModule::CalcModuleInternals, 21
 PtrUtils, 61
 IsNullPtr, 61
 IsValidPtr, 62

 readDht11
 TemperatureSensor, 89
 readMLX90614
 TemperatureSensor, 90
 readPressure
 PressureSensor, 60
 readSensor
 sensorModule::SensorModuleInternals, 77
 readTemperature
 TemperatureSensor, 90
 receiveEthernetData
 comModule::EthernetCommunication, 32
 receiveSerialData
 comModule::SerialCommunication, 79
 release
 PointerWrapper< T >, 58
 ScopedPointer< T >, 73
 renameFile
 LogManager, 51
 reportError
 reportSystem::ReportSystem, 68
 reportStatus
 reportSystem::ReportSystem, 69
 reportSystem, 12
 reportSystem::ReportSystem, 63
 checkRamLevel, 64
 checkSystemHealth, 65
 checkThresholds, 65
 detectStackOverflow, 66
 getCPULoad, 66
 getCurrentTime, 67
 getLastError, 67
 getLastErrorInfo, 67
 getMemoryStatus, 68
 getStackDump, 68
 reportError, 68
 reportStatus, 69
 saveLastError, 70
 setThreshold, 70
 reset
 PointerWrapper< T >, 58
 ScopedPointer< T >, 73
 run
 vacControlModule::VacControl, 101

 saveLastError
 reportSystem::ReportSystem, 70
 ScopedLock
 locker::ScopedLock, 71, 72
 ScopedPointer< T >, 72
 get, 73
 operator->, 73
 operator*, 73
 release, 73
 reset, 73
 sendCommand
 comModule::EthernetCommunication, 33
 sendEthernetData
 comModule::EthernetCommunication, 33
 sendJsonEthernet
 jsonModule::JsonModuleInternals, 46
 sendJsonResponse
 comModule::EthernetCommunication, 33
 sendSerialData
 comModule::SerialCommunication, 79
 sensorModule, 12
 sensorModule::SensorModuleInternals, 74
 calibrateSensor, 75
 checkSensorStatus, 76
 initialize, 76
 readSensor, 77
 SerialMenu, 79
 getCurrentTime, 80
 load, 82
 printToSerial, 83, 84
 setCompound
 comModule::EthernetCommunication, 34, 35
 setCompoundInternal
 comModule::EthernetCommunication, 35
 setExternDutyCycle
 flybackModule::Flyback, 39
 setExternFrequency
 flybackModule::Flyback, 39
 setExternPressure
 vacControlModule::VacControl, 101
 setExternScenario
 vacControlModule::VacControl, 102
 setLogFileName
 LogManager, 53
 setParameter
 comModule::EthernetCommunication, 36
 setPump
 vacControlModule::VacControl, 102
 setSendDataFlag
 comModule::EthernetCommunication, 37
 setSystemTime
 timeModule::TimeModuleInternals, 96

- setThreshold
 - reportSystem::ReportSystem, 70
- setTimeFromHas
 - timeModule::TimeModuleInternals, 97
- setTimerState
 - flybackModule::Flyback, 40
- setVacuumLed
 - vacControlModule::VacControl, 102
- spiRead
 - comModule::SPICommunication, 87
- spiWrite
 - comModule::SPICommunication, 88
- TemperatureSensor, 88
 - isInitialized, 89
 - readDht11, 89
 - readMLX90614, 90
 - readTemperature, 90
- timeModule, 12
- timeModule::DateTimeStruct, 24
- timeModule::TimeModuleInternals, 91
 - formatTimeString, 92
 - getInstance, 93
 - getSystemTime, 94
 - incrementTime, 95
 - setSystemTime, 96
 - setTimeFromHas, 97
- vacControlModule, 13
- vacControlModule::Pressure, 59
- vacControlModule::VacControl, 97
 - externProcess, 98
 - getExternPressure, 98
 - getExternScenario, 99
 - getScenario, 99
 - getScenarioFromPotValue, 99
 - getSwitchState, 100
 - isInitialized, 100
 - measure, 100
 - run, 101
 - setExternPressure, 101
 - setExternScenario, 102
 - setPump, 102
 - setVacuumLed, 102
- writeToLogFile
 - LogManager, 53