

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	13
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()	17
6.1.1.4 calculatePower()	17
6.1.1.5 calculateResistance()	18
6.1.1.6 calculateStandardDeviation()	18
6.1.1.7 celsiusToFahrenheit()	18
6.1.1.8 celsiusToKelvin()	19
6.1.1.9 extractFloat()	19

6.1.1.10 extractFloatFromResponse()	20
6.1.1.11 fahrenheitToCelsius()	20
6.1.1.12 findMaximum()	20
6.1.1.13 findMedian()	21
6.1.1.14 findMinimum()	21
6.1.1.15 kelvinToCelsius()	21
6.1.1.16 pascalToAtm()	22
6.1.1.17 pascalToPsi()	22
6.1.1.18 psiToPascal()	22
6.2 calcModuleInternals Class Reference	23
6.2.1 Detailed Description	23
6.3 comModule::ComModuleInternals Class Reference	23
6.3.1 Member Function Documentation	24
6.3.1.1 getEthernet()	24
6.3.1.2 getI2C()	24
6.3.1.3 getSerial()	25
6.3.1.4 getSPI()	25
6.4 comModuleInternals Class Reference	26
6.4.1 Detailed Description	26
6.5 timeModule::DateTimeStruct Struct Reference	26
6.5.1 Detailed Description	26
6.6 Error Struct Reference	27
6.7 ErrorSeverity Struct Reference	27
6.7.1 Detailed Description	28
6.8 comModule::EthernetCommunication Class Reference	28
6.8.1 Detailed Description	29
6.8.2 Member Function Documentation	29
6.8.2.1 beginEthernet()	29
6.8.2.2 getClient()	29
6.8.2.3 getCompound() [1/3]	30
6.8.2.4 getCompound() [2/3]	31
6.8.2.5 getCompound() [3/3]	31
6.8.2.6 getCompoundInternal()	32
6.8.2.7 getParameter()	33
6.8.2.8 getParsedCompound() [1/3]	33
6.8.2.9 getParsedCompound() [2/3]	34
6.8.2.10 getParsedCompound() [3/3]	34
6.8.2.11 getRequestedEndpoint()	35
6.8.2.12 getSendDataFlag()	35
6.8.2.13 getSpecificEndpoint()	35
6.8.2.14 isInitialized()	36
6.8.2.15 parseCompoundResponse()	36

6.8.2.16 receiveEthernetData()	37
6.8.2.17 sendCommand()	37
6.8.2.18 sendEthernetData()	38
6.8.2.19 sendJsonResponse()	38
6.8.2.20 setCompound() [1/3]	38
6.8.2.21 setCompound() [2/3]	39
6.8.2.22 setCompound() [3/3]	39
6.8.2.23 setCompoundInternal()	40
6.8.2.24 setParameter()	41
6.8.2.25 setSendDataFlag()	41
6.9 flybackModule::Flyback Class Reference	42
6.9.1 Detailed Description	43
6.9.2 Member Function Documentation	43
6.9.2.1 getExternPSU()	43
6.9.2.2 getHVState()	43
6.9.2.3 getHVSwitchState()	43
6.9.2.4 getHysteresis()	44
6.9.2.5 getMainSwitchState()	44
6.9.2.6 getTargetVoltage()	44
6.9.2.7 getTimerState()	45
6.9.2.8 isInitialized()	45
6.9.2.9 measure()	46
6.9.2.10 regulateVoltage()	46
6.9.2.11 setExternDutyCycle()	47
6.9.2.12 setExternFrequency()	47
6.9.2.13 setExternPSU()	48
6.9.2.14 setHysteresis()	48
6.9.2.15 setTargetVoltage()	48
6.9.2.16 setTimerState()	49
6.10 comModule::I2CCommunication Class Reference	49
6.10.1 Detailed Description	49
6.10.2 Member Function Documentation	50
6.10.2.1 beginI2C()	50
6.10.2.2 i2cRead()	51
6.10.2.3 i2cWrite()	51
6.10.2.4 isInitialized()	51
6.11 jsonModule::JsonModuleInternals Class Reference	52
6.11.1 Detailed Description	53
6.11.2 Member Function Documentation	53
6.11.2.1 createJson()	53
6.11.2.2 getJsonString()	53
6.11.2.3 hasCapacityFor()	54

6.12 LockerBase Class Reference	54
6.12.1 Detailed Description	55
6.12.2 Member Function Documentation	55
6.12.2.1 lockEthernetScoped()	55
6.12.2.2 lockSerialScoped()	55
6.12.2.3 lockTemperatureScoped()	55
6.13 LogManager Class Reference	56
6.13.1 Member Function Documentation	56
6.13.1.1 getCurrentTime()	56
6.13.1.2 getInstance()	57
6.13.1.3 initSDCard()	58
6.13.1.4 isSDCardInitialized()	59
6.13.1.5 renameFile()	60
6.13.1.6 setLogFileName()	61
6.13.1.7 writeToLogFile()	61
6.14 LogMapper Class Reference	63
6.14.1 Detailed Description	63
6.15 flybackModule::Measurement Struct Reference	63
6.15.1 Detailed Description	63
6.16 Measurement Struct Reference	63
6.16.1 Detailed Description	64
6.17 MenuItem Struct Reference	64
6.17.1 Detailed Description	64
6.18 OutputLevel Class Reference	64
6.18.1 Detailed Description	64
6.19 PointerWrapper< T > Class Template Reference	65
6.19.1 Detailed Description	65
6.19.2 Member Function Documentation	65
6.19.2.1 get()	65
6.19.2.2 operator*()	66
6.19.2.3 operator->()	66
6.19.2.4 release()	66
6.19.2.5 reset()	66
6.20 vacControlModule::Pressure Struct Reference	67
6.21 PressureSensor Class Reference	67
6.21.1 Detailed Description	67
6.21.2 Member Function Documentation	68
6.21.2.1 isInitialized()	68
6.21.2.2 readPressure()	68
6.22 PtrUtils Class Reference	69
6.22.1 Detailed Description	69
6.22.2 Member Function Documentation	69

6.22.2.1 IsNullPtr()	69
6.22.2.2 IsValidPtr()	70
6.23 reportSystem::ReportSystem Class Reference	71
6.23.1 Detailed Description	72
6.23.2 Member Function Documentation	72
6.23.2.1 checkRamLevel()	72
6.23.2.2 checkSystemHealth()	73
6.23.2.3 checkThresholds()	74
6.23.2.4 detectStackOverflow()	74
6.23.2.5 getCPULoad()	75
6.23.2.6 getCurrentTime()	75
6.23.2.7 getLastError()	76
6.23.2.8 getLastErrorInfo()	76
6.23.2.9 getMemoryStatus()	77
6.23.2.10 getStackDump()	77
6.23.2.11 hasNoSavedErrors()	77
6.23.2.12 isCommunicationOK()	78
6.23.2.13 isMemoryOK()	78
6.23.2.14 isRamOK()	79
6.23.2.15 isStackSafe()	79
6.23.2.16 isTemperatureSensorOK()	79
6.23.2.17 reportError()	79
6.23.2.18 reportStatus()	80
6.23.2.19 saveLastError()	81
6.23.2.20 setThreshold()	81
6.24 locker::ScopedLock Class Reference	82
6.24.1 Detailed Description	82
6.24.2 Constructor & Destructor Documentation	82
6.24.2.1 ScopedLock() [1/2]	82
6.24.2.2 ScopedLock() [2/2]	83
6.25 ScopedPointer< T > Class Template Reference	83
6.25.1 Detailed Description	83
6.25.2 Member Function Documentation	84
6.25.2.1 get()	84
6.25.2.2 operator*()	84
6.25.2.3 operator->()	84
6.25.2.4 release()	84
6.25.2.5 reset()	84
6.26 sensorModule::SensorModuleInternals Class Reference	85
6.26.1 Detailed Description	86
6.26.2 Member Function Documentation	86
6.26.2.1 calibrateSensor()	86

6.26.2.2 checkSensorStatus()	87
6.26.2.3 readSensor()	88
6.26.2.4 reportUnknownSensorOnce()	89
6.27 comModule::SerialCommunication Class Reference	89
6.27.1 Detailed Description	90
6.27.2 Member Function Documentation	90
6.27.2.1 beginSerial()	90
6.27.2.2 isInitialized()	90
6.27.2.3 receiveSerialData()	91
6.27.2.4 sendSerialData()	91
6.28 SerialMenu Class Reference	91
6.28.1 Detailed Description	92
6.28.2 Member Function Documentation	92
6.28.2.1 getCurrentTime()	92
6.28.2.2 load()	93
6.28.2.3 printToSerial() [1/4]	94
6.28.2.4 printToSerial() [2/4]	94
6.28.2.5 printToSerial() [3/4]	95
6.28.2.6 printToSerial() [4/4]	95
6.29 comModule::SPICommunication Class Reference	98
6.29.1 Detailed Description	98
6.29.2 Member Function Documentation	98
6.29.2.1 isInitialized()	98
6.29.2.2 spiRead()	98
6.29.2.3 spiWrite()	99
6.30 TemperatureSensor Class Reference	99
6.30.1 Detailed Description	100
6.30.2 Member Function Documentation	100
6.30.2.1 calibMCP9601()	100
6.30.2.2 isInitialized()	101
6.30.2.3 readMCP9601()	101
6.30.2.4 readTemperature()	102
6.31 timeModule::TimeModuleInternals Class Reference	103
6.31.1 Detailed Description	103
6.31.2 Member Function Documentation	103
6.31.2.1 formatTimeString()	103
6.31.2.2 getInstance()	104
6.31.2.3 getSystemTime()	105
6.31.2.4 incrementTime()	107
6.31.2.5 setSystemTime()	107
6.31.2.6 setTimeFromHas()	108
6.32 vacControlModule::VacControl Class Reference	108

6.32.1 Detailed Description	109
6.32.2 Member Function Documentation	110
6.32.2.1 externPump()	110
6.32.2.2 getExternPressure()	110
6.32.2.3 getExternPump()	110
6.32.2.4 getExternScenario()	110
6.32.2.5 getMainSwitchState()	111
6.32.2.6 getPumpState()	111
6.32.2.7 getPumpSwitchState()	111
6.32.2.8 getScenario()	112
6.32.2.9 getScenarioFromPotValue()	112
6.32.2.10 isInitialized()	112
6.32.2.11 measure()	113
6.32.2.12 run()	113
6.32.2.13 setExternPressure()	113
6.32.2.14 setExternPump()	114
6.32.2.15 setExternScenario()	114
6.32.2.16 setPump()	114
6.32.2.17 setVacuumLed()	114
7 File Documentation	117
7.1 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp File Reference	117
7.1.1 Detailed Description	117
7.2 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h File Reference	118
7.2.1 Detailed Description	119
7.3 calcModule.h	120
7.4 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp File Reference	121
7.4.1 Detailed Description	121
7.5 comModule.h	121
7.6 ETHH.h	122
7.7 I2CC.h	124
7.8 SER.h	124
7.9 SPII.h	125
7.10 config.h	125
7.11 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h File Reference	126
7.11.1 Detailed Description	127
7.12 flyback.h	127
7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp File Reference	129
7.13.1 Detailed Description	130
7.14 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h File Reference	130
7.14.1 Detailed Description	132
7.15 jsonModule.h	132

7.16 lockerBase.h	133
7.17 scopedLock.h	133
7.18 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.cpp File Reference	134
7.18.1 Detailed Description	135
7.19 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h File Reference	136
7.19.1 Detailed Description	137
7.20 logManager.h	137
7.21 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp File Reference	138
7.21.1 Detailed Description	138
7.22 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h File Reference	139
7.22.1 Detailed Description	140
7.23 pressure.h	140
7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h File Reference	140
7.24.1 Detailed Description	142
7.24.2 Macro Definition Documentation	143
7.24.2.1 RETHROW	143
7.24.2.2 RETURN_IF_ERR	143
7.24.2.3 THROW	143
7.24.2.4 THROW_IF_ERR	143
7.24.2.5 THROW_IF_FALSE	144
7.24.2.6 THROW_IF_NULL	144
7.24.2.7 THROW_SOFT	144
7.24.2.8 TRY	144
7.24.2.9 tryDeletePtr	144
7.25 ptrUtils.h	145
7.26 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp File Reference	149
7.26.1 Detailed Description	149
7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h File Reference	150
7.27.1 Detailed Description	151
7.28 reportSystem.h	151
7.29 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.cpp File Reference	152
7.29.1 Detailed Description	153
7.30 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h File Reference	153
7.30.1 Detailed Description	154
7.31 sensorModule.h	155
7.32 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h File Reference	156
7.32.1 Detailed Description	157
7.33 serialMenu.h	157
7.34 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp File Reference	158
7.34.1 Detailed Description	158
7.35 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.h File Reference	159

7.35.1 Detailed Description	160
7.36 temperature.h	160
7.37 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp File Reference	161
7.37.1 Detailed Description	162
7.38 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h File Reference	162
7.38.1 Detailed Description	163
7.39 timeModule.h	164
7.40 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.cpp File Reference	164
7.40.1 Detailed Description	165
7.41 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h File Reference	165
7.41.1 Detailed Description	167
7.42 vacControl.h	168
Index	171

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	13
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	23
comModule::ComModuleInternals	23
comModuleInternals	26
timeModule::DateTimeStruct	26
Error	27
ErrorSeverity	27
comModule::EthernetCommunication	28
flybackModule::Flyback	42
comModule::I2CCommunication	49
jsonModule::JsonModuleInternals	52
LockerBase	54
LogManager	56
LogMapper	63
flybackModule::Measurement	63
Measurement	63
MenuItem	64
Outputlevel	64
PointerWrapper< T >	65
vacControlModule::Pressure	67
PressureSensor	67
sensorModule::SensorModuleInternals	85
PtrUtils	69
reportSystem::ReportSystem	71
locker::ScopedLock	82
ScopedPointer< T >	83
comModule::SerialCommunication	89
SerialMenu	91
comModule::SPICommunication	98
TemperatureSensor	99
sensorModule::SensorModuleInternals	85
timeModule::TimeModuleInternals	103
vacControlModule::VacControl	108

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	23
comModule::ComModuleInternals	23
comModuleInternals	
Class for the communication module internals	26
timeModule::DateTimeStruct	
Struct to hold the date and time	26
Error	27
ErrorSeverity	
Struct to defined the error settings	27
comModule::EthernetCommunication	
Class to handle Ethernet communication	28
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	42
comModule::I2CCommunication	
Class to handle I2C communication	49
jsonModule::JsonModuleInternals	
Class for the JSON module internals	52
LockerBase	
Base class for the locker system	54
LogManager	56
LogMapper	
Class which handle the printed log messages, maps aka parses them and saves them to the SD card	63
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	63
Measurement	
Structure to store the measured values of the system This structure holds the pressure values measured from the system	63
MenuItem	
Serial menu structure	64

Outputlevel	
Enum Class for the different Outputlevels	64
PointerWrapper< T >	
Template class for wrapping a pointer	65
vacControlModule::Pressure	67
PressureSensor	
Pressure sensor class	67
PtrUtils	
Utility class for pointer operations	69
reportSystem::ReportSystem	
Class for the report system	71
locker::ScopedLock	
Scoped lock class for mutexes	82
ScopedPointer< T >	
Template class for a Scoped Pointer	83
sensorModule::SensorModuleInternals	
Class for the sensor module internals	85
comModule::SerialCommunication	
Class to handle Serial communication	89
SerialMenu	
Class for the serial menu	91
comModule::SPICommunication	
Class to handle SPI communication	98
TemperatureSensor	
Temperature sensor class	99
timeModule::TimeModuleInternals	
Class to handle Systemtime	103
vacControlModule::VacControl	
VacControl class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes	108

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	
Implementation of the calcModule class	117
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h	
Header file for the calculation module handling sensor data	118
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp	
Implementation of the comModule class that utilizes various communication protocols	121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h	121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.h	122
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/l2CC.h	124
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER.h	124
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h	125
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h	125
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h	
Header for the flyback class	126
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp	
Implementation of the jsonModule class	129
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h	
Header for the JsonModuleInternals class	130
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h	133
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h	133
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.cpp	
Implementation of the logManager class	134
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h	
Header file for the logManager	136
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp	
Implementation of the pressure class	138
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h	
Header file for the pressure library	139
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h	
Header only library implementation	140
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp	
Unified system health and error reporting module	149
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h	
Header file for the ReportSystem	150
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.cpp	
Implementation of the sensorModule class	152

C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/ sensorModule.h	
Header file for the sensorModule	153
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/ serialMenu.h	
Header file for the serial menu handling serial menu interaction, logging.. . . .	156
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/ temperature.cpp	
Implementation of the temperature class	158
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/ temperature.h	
Header file for the temperature library	159
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/ timeModule.cpp	
Implementation of the timeModule class	161
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/ timeModule.h	
Header file for the time module handling systemtime for logging, api	162
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/ vacControl.cpp	
Implementation of the vacControl class	164
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/ vacControl.h	
Header for the vacControl class	165

Chapter 5

Namespace Documentation

5.1 calcModule Namespace Reference

Namespace for the calculation module.

Classes

- class [CalcModuleInternals](#)

Enumerations

- enum [Type](#) { **General** , **Pressure** , **Position** }
Enum for the different Types we want to extract from a response.
- enum class [PressureUnit](#) { **Pascal** , **Atmosphere** , **Psi** , **Bar** }
Enum class for different Pressure Units aviable.

5.1.1 Detailed Description

Namespace for the calculation module.

5.2 comModule Namespace Reference

Namespace for the communication module.

Classes

- class [ComModuleInternals](#)
- class [EthernetCommunication](#)
Class to handle Ethernet communication.
- class [I2CCommunication](#)
Class to handle I2C communication.
- class [SerialCommunication](#)
Class to handle Serial communication.
- class [SPICommunication](#)
Class to handle SPI communication.

Enumerations

- enum class [Service](#) : uint8_t {
SET = 0x01 , **GET** = 0x0B , **SET_COMPOUND** = 0x28 , **GET_COMPOUND** = 0x29 ,
SETGET = 0x30 }
Enum class for the service types.
- enum class [Compound1](#) : uint32_t { **CONTROL_MODE** = 0x0F020000 , **TARGET_POSITION** = 0x11020000
, **TARGET_PRESSURE** = 0x07020000 , **NOT_USED** = 0x00000000 }
Enum class for the compound 1 types.
- enum class [Compound2](#) : uint32_t {
ACCESS_MODE = 0x0F0B0000 , **CONTROL_MODE** = 0x0F020000 , **TARGET_POSITION** = 0x11020000 ,
TARGET_PRESSURE = 0x07020000 ,
ACTUAL_POSITION = 0x10010000 , **POSITION_STATE** = 0x00100000 , **ACTUAL_PRESSURE** =
0x07010000 , **TARGET_PRESSURE_USED** = 0x07030000 ,
WARNING_BITMAP = 0x0F300100 , **NOT_USED** = 0x00000000 }
Enum class for the compound 2 types.
- enum class [Compound3](#) : uint32_t {
CONTROL_MODE = 0x0F020000 , **TARGET_POSITION** = 0x11020000 , **TARGET_PRESSURE** =
0x07020000 , **SEPARATION** = 0x00000000 ,
ACCESS_MODE = 0x0F0B0000 , **ACTUAL_POSITION** = 0x10010000 , **POSITION_STATE** = 0x00100000 ,
ACTUAL_PRESSURE = 0x07010000 ,
TARGET_PRESSURE_USED = 0x07030000 , **WARNING_BITMAP** = 0x0F300100 , **NOT_USED** =
0x00000000 }
Enum class for the compound 3 types.
- enum class [Error_Codes](#) : uint8_t {
NO_ERROR = 0x00 , **WRONG_COMMAND_LENGTH** = 0x0C , **VALUE_TOO_LOW** = 0x1C , **VALUE_↵**
TOO_HIGH = 0x1D ,
RESULTING_ZERO_ADJUST_OFFSET = 0x20 , **NO_SENSOR_ENABLED** = 0x21 , **WRONG_ACCESS_↵**
_MODE = 0x50 , **TIMEOUT** = 0x51 ,
NV_MEMORY_NOT_READY = 0x6D , **WRONG_PARAMETER_ID** = 0x6E , **PARAMETER_NOT_↵**
SETTABLE = 0x70 , **PARAMETER_NOT_READABLE** = 0x71 ,
WRONG_PARAMETER_INDEX = 0x73 , **WRONG_VALUE_WITHIN_RANGE** = 0x76 , **NOT_ALLOWED_↵**
IN_THIS_STATE = 0x78 , **SETTING_LOCK** = 0x79 ,
WRONG_SERVICE = 0x7A , **PARAMETER_NOT_ACTIVE** = 0x7B , **PARAMETER_SYSTEM_ERROR** =
0x7C , **COMMUNICATION_ERROR** = 0x7D ,
UNKNOWN_SERVICE = 0x7E , **UNEXPECTED_CHARACTER** = 0x7F , **NO_ACCESS_RIGHTS** = 0x80 ,
NO_ADEQUATE_HARDWARE = 0x81 ,
WRONG_OBJECT_STATE = 0x82 , **NO_SLAVE_COMMAND** = 0x84 , **COMMAND_TO_UNKNOWN_↵**
SLAVE = 0x85 , **COMMAND_TO_MASTER_ONLY** = 0x87 ,
ONLY_G_COMMAND_ALLOWED = 0x88 , **NOT_SUPPORTED** = 0x89 , **FUNCTION_DISABLED** = 0xA0 ,
ALREADY_DONE = 0xA1 }
Enum class for the error codes.

5.2.1 Detailed Description

Namespace for the communication module.

5.3 flybackModule Namespace Reference

Namespace for the [Flyback](#) module.

Classes

- class [Flyback](#)
Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.
- struct [Measurement](#)
Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.

Typedefs

- typedef struct [flybackModule::Measurement](#) **meas**

Enumerations

- enum class [MainSwitchStates](#) : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }
enum for different SwitchStates of HVModule
- enum class [HVSwitchStates](#) : int { **HV_Module_OFF** , **HV_Module_ON** }
enum for SwitchStates of HV_Module on/off
- enum class **HVModule** : int { **powerSupply_OFF** , **powerSupply_ON** }

5.3.1 Detailed Description

Namespace for the [Flyback](#) module.

5.4 jsonModule Namespace Reference

Namespace for the JSON module.

Classes

- class [JsonModuleInternals](#)
Class for the JSON module internals.

5.4.1 Detailed Description

Namespace for the JSON module.

5.5 locker Namespace Reference

Namespace for the locker system.

Classes

- class [ScopedLock](#)
Scoped lock class for mutexes.

5.5.1 Detailed Description

Namespace for the locker system.

5.6 reportSystem Namespace Reference

Namespace for the report system.

Classes

- class [ReportSystem](#)
Class for the report system.

5.6.1 Detailed Description

Namespace for the report system.

5.7 sensorModule Namespace Reference

Namespace for the sensor module.

Classes

- class [SensorModuleInternals](#)
Class for the sensor module internals.

Enumerations

- enum class [SensorType](#) {
 TEMPERATURE , OBJECTTEMPERATURE , AMBIENTTEMPERATURE , PRESSURE ,
 DHT11 , MCP9601_Celsius_Indoor , MCP9601_Fahrenheit_Indoor , MCP9601_Kelvin_Indoor ,
 MCP9601_Celsius_Outdoor , MCP9601_Fahrenheit_Outdoor , MCP9601_Kelvin_Outdoor , UNKNOWN
}
Enum class for the sensor types.

5.7.1 Detailed Description

Namespace for the sensor module.

5.8 timeModule Namespace Reference

namespace for the [timeModule](#)

Classes

- struct [DateTimeStruct](#)
Struct to hold the date and time.
- class [TimeModuleInternals](#)
Class to handle Systemtime.

Typedefs

- typedef struct [timeModule::DateTimeStruct](#) **DateTimeStruct**

5.8.1 Detailed Description

namespace for the [timeModule](#)

5.9 vacControlModule Namespace Reference

Namespace for the [VacControl](#) module.

Classes

- struct [Pressure](#)
- class [VacControl](#)
[VacControl](#) class to manage the vacuum control system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

Typedefs

- typedef struct [vacControlModule::Pressure](#) **meas**

Enumerations

- enum class [MainSwitchStates](#) : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }
Enum to represent the states of the main switch and pump.
- enum class [PumpSwitchStates](#) : int { **Switch_Pump_OFF** , **Switch_Pump_ON** }
Enum for extern Setup for pinMode for the PumpState.
- enum class [PumpState](#) : int { **pump_OFF** , **pump_ON** }
Enum for extern Setup for pinMode for the PumpState.
- enum [Scenarios](#) {
 Scenario_1 = 0 , **Scenario_2** = 1 , **Scenario_3** = 2 , **Scenario_4** = 3 ,
 Scenario_5 = 4 , **Invalid_Scenario** = -1 }
Enum to represent the different operating scenarios of the VacControl system.

5.9.1 Detailed Description

Namespace for the [VacControl](#) module.

Chapter 6

Class Documentation

6.1 calcModule::CalcModuleInternals Class Reference

Static Public Member Functions

- static float [calculateAverage](#) (const float *data, int length)
Function to calculate the average of a data set.
- static float [findMaximum](#) (const float *data, int length)
Function to calculate the maximum value of a data set.
- static float [findMinimum](#) (const float *data, int length)
Function to calculate the minimum value of a data set.
- static float [calculateStandardDeviation](#) (const float *data, int length)
Function to calculate the standard deviation of a data set.
- static float [findMedian](#) (float *data, int length)
Function to calculate the median of a data set.
- static float [celsiusToFahrenheit](#) (float celsius)
Function to convert celsius to fahrenheit.
- static float [fahrenheitToCelsius](#) (float fahrenheit)
Function to convert fahrenheit to celsius.
- static float [celsiusToKelvin](#) (float celsius)
Function to convert celsius to kelvin.
- static float [kelvinToCelsius](#) (float kelvin)
Function to convert kelvin to celsius.
- static float [pascalToAtm](#) (float pascal)
Function to convert pascal to atm.
- static float [atmToPascal](#) (float atm)
Function to convert atm to pascal.
- static float [pascalToPsi](#) (float pascal)
Function to convert pascal to psi.
- static float [psiToPascal](#) (float psi)
Function to convert psi to pascal.
- static float [calculatePower](#) (float voltage, float current)
Function to calculate the power.
- static float [calculateCurrent](#) (float voltage, float resistance)
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)
Extract the float from a VAT uC eth frame, specific for positions and pressures.
- static float **calculatePressureFromSensor** (int sensorValue, [PressureUnit](#) unit=PressureUnit::Pascal)

6.1.1 Member Function Documentation

6.1.1.1 atmToPascal()

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

Function to convert atm to pascal.

Parameters

<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.

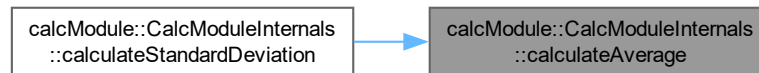
Parameters

<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.

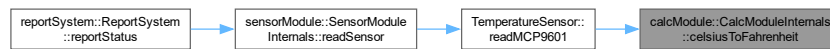
Parameters

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

Returns

float -> The temperature in fahrenheit.

Here is the caller graph for this function:



6.1.1.8 celsiusToKelvin()

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

Function to convert celsius to kelvin.

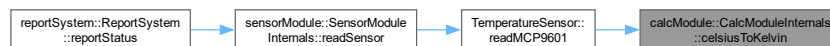
Parameters

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

Returns

float -> The temperature in kelvin.

Here is the caller graph for this function:



6.1.1.9 extractFloat()

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

Extract the float from a VAT uC eth frame.

Parameters

<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 extractFloatFromResponse()

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

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

Parameters

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

Returns

float -> The extracted float.

6.1.1.11 fahrenheitToCelsius()

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

Function to convert fahrenheit to celsius.

Parameters

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

Returns

float -> The temperature in celsius.

6.1.1.12 findMaximum()

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


Function to calculate the maximum value of a data set.

Parameters

<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.

6.1.1.13 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.

6.1.1.14 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.

6.1.1.15 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.

6.1.1.16 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.

6.1.1.17 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.

6.1.1.18 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`&

Here is the caller graph for this function:



6.3.1.2 getI2C()

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

Get the I2C object.

Returns

`I2CCommunication`&

Here is the caller graph for this function:



6.3.1.3 getSerial()

[SerialCommunication](#) & ComModuleInternals::getSerial ()

Get the Serial object.

Returns

[SerialCommunication](#)&

Here is the caller graph for this function:



6.3.1.4 getSPI()

[SPICommunication](#) & ComModuleInternals::getSPI ()

Get the SPI object.

Returns

[SPICommunication](#)&

Here is the caller graph for this function:



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

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

6.4 comModuleInternals Class Reference

Class for the communication module internals.

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

6.4.1 Detailed Description

Class for the communication module internals.

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

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

6.5 timeModule::DateTimeStruct Struct Reference

Struct to hold the date and time.

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

Public Attributes

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

6.5.1 Detailed Description

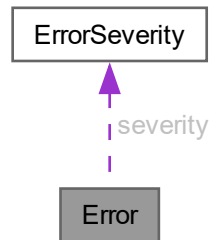
Struct to hold the date and time.

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

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

6.6 Error Struct Reference

Collaboration diagram for Error:



Public Member Functions

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

Public Attributes

- int **code**
- [ErrorSeverity](#) **severity**
- const char * **msg**
- const char * **traceFiles** [TRACE_DEPTH]
- int **traceLines** [TRACE_DEPTH]
- int **traceDepth**

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

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

6.7 ErrorSeverity Struct Reference

Struct to defined the error settings.

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

6.7.1 Detailed Description

Struct to defined the error settings.

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

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

6.8 comModule::EthernetCommunication Class Reference

Class to handle Ethernet communication.

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

Public Member Functions

- void [beginEthernet](#) (uint8_t *macAddress, IPAddress ip)
Function to initialize the Ethernet communication.
- void [sendEthernetData](#) (const char *endpoint, const char *data)
Function to send data over Ethernet.
- void [receiveEthernetData](#) (char *buffer, size_t length)
Function to receive data over Ethernet.
- void [handleEthernetClient](#) ()
Function to handle the Ethernet client.
- String [getRequestedEndpoint](#) ()
Function to get the requested endpoint.
- String [getSpecificEndpoint](#) (const String &jsonBody)
Function to get the specific endpoint.
- void [sendJsonResponse](#) (const String &jsonBody)
Function to send the json response with the measurment data.
- EthernetClient & [getClient](#) ()
Get the currently active Ethernet client.
- bool [isInitialized](#) () const
Function to check if the Ethernet communication is initialized.
- bool [getSendDataFlag](#) () const
Function to get the current status of the flag.
- void [setSendDataFlag](#) (bool flag)
Function to get the current status of the flag.
- void [setCompound](#) ([Compound1](#) id, int index, String value)
Function to set a compound command for the valve uC Slave (Compound1)
- void [setCompound](#) ([Compound2](#) id, int index, String value)
Function to set a compound command for the valve uC Slave (Compound2)
- void [setCompound](#) ([Compound3](#) id, int index, String value)
Function to set a compound command for the valve uC Slave (Compound3)
- void [setCompoundInternal](#) (String compoundType, unsigned long id, int index, String value)
Function to set the 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.

6.8.1 Detailed Description

Class to handle Ethernet communication.

6.8.2 Member Function Documentation

6.8.2.1 beginEthernet()

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

Function to initialize the Ethernet communication.

Parameters

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

6.8.2.2 getClient()

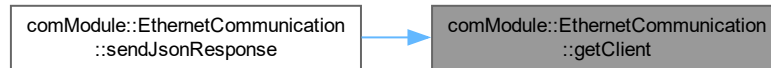
```
EthernetClient & EthernetCommunication::getClient ( )
```

Get the currently active Ethernet client.

Returns

EthernetClient& , Reference to the active Ethernet client

Here is the caller graph for this function:

**6.8.2.3 getCompound() [1/3]**

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

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

Parameters

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

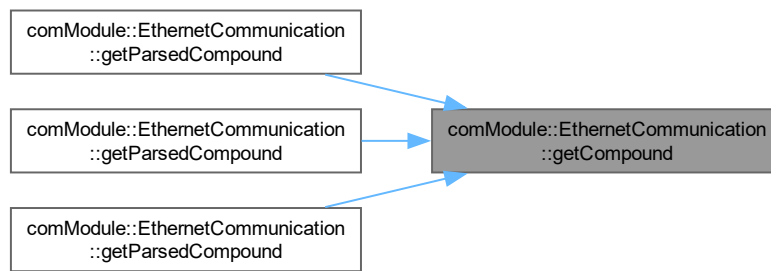
Returns

String -> Response from the valve uC slave

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.2.4 getCompound() [2/3]

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

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

Parameters

<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:



6.8.2.5 getCompound() [3/3]

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

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

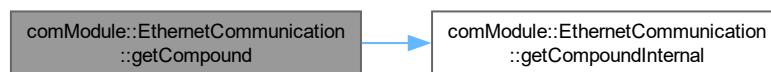
Parameters

<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:

**6.8.2.6 getCompoundInternal()**

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

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

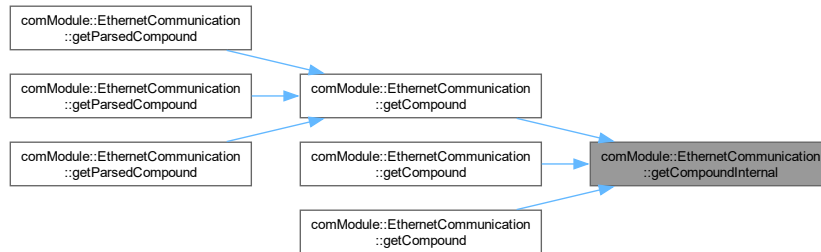
Parameters

<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:

**6.8.2.7 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.

6.8.2.8 getParsedCompound() [1/3]

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

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

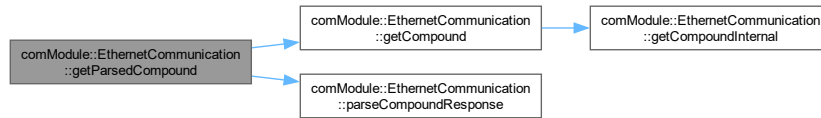
Parameters

<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:

**6.8.2.9 getParsedCompound() [2/3]**

```

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

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

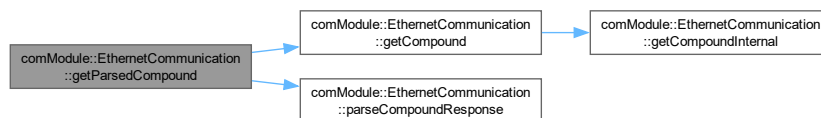
Parameters

<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:

**6.8.2.10 getParsedCompound() [3/3]**

```

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

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

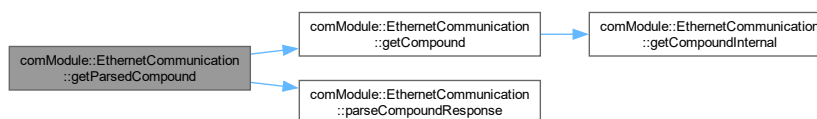
Parameters

<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:



6.8.2.11 getRequestedEndpoint()

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

Function to get the requested endpoint.

Returns

String -> The requested endpoint

6.8.2.12 getSendDataFlag()

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

Function to get the current status of the flag.

Returns

true -> if data should be sent
false -> if data should not be sent

6.8.2.13 getSpecificEndpoint()

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

Function to get the specific endpoint.

Parameters

<i>jsonBody</i>	-> The json body to get the endpoint from
-----------------	---

Returns

String -> The specific endpoint

6.8.2.14 isInitialized()

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

Function to check if the Ethernet communication is initialized.

Returns

true -> if the Ethernet communication is initialized

false -> if the Ethernet communication is not initialized

Here is the caller graph for this function:

**6.8.2.15 parseCompoundResponse()**

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

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

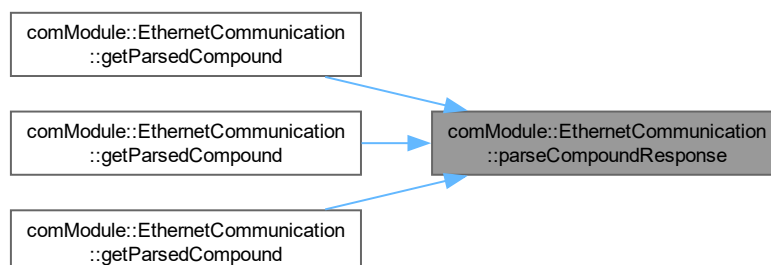
Parameters

<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:

**6.8.2.16 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

6.8.2.17 sendCommand()

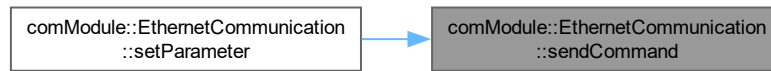
```
void EthernetCommunication::sendCommand (
    String command )
```

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:



6.8.2.18 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

6.8.2.19 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:



6.8.2.20 setCompound() [1/3]

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

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

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

Parameters

<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:



6.8.2.21 setCompound() [2/3]

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

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

Parameters

<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:



6.8.2.22 setCompound() [3/3]

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

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

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

Parameters

<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:



6.8.2.23 setCompoundInternal()

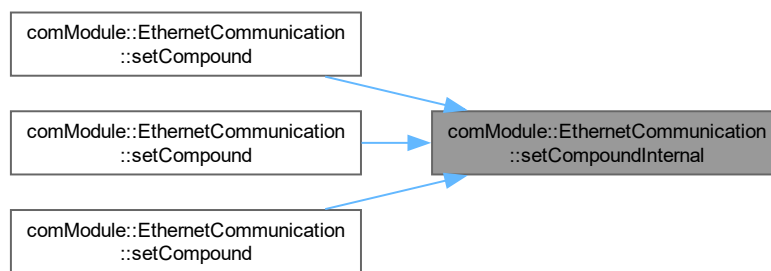
```
void EthernetCommunication::setCompoundInternal (
    String compoundType,
    unsigned long id,
    int index,
    String value )
```

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:



6.8.2.24 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:



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

6.9 flybackModule::Flyback Class Reference

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

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

Public Member Functions

- void **initialize** ()
*Initialize the **Flyback** system This method sets up the pins and prepares the system for operation.*
- void **deinitialize** ()
*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.
- **MainSwitchStates** **getMainSwitchState** ()
Get the state of the Main-Switch.
- **HVSwitchStates** **getHVSwitchState** ()
Get the state of the HV-Switch.
- **HVModule** **getHVState** ()
Get the state of the HV-Output.
- **Measurement** **measure** ()
Measures the voltage, current, power, digitalValue and frequency of the system.
- void **run** ()
Executes logic depending on which Main-Switch state is active.
- void **setExternFrequency** (uint32_t frequency)
Function to get the desired Frequency from HAS.
- uint32_t **getExternFrequency** ()
Getter Function to get the Frequency.
- void **setExternDutyCycle** (int dutyCycle)
Function to get the desired DutyCycle from HAS.
- int **getExternDutyCycle** ()
*Getter Function to get the DutyCycle *.*
- void **setExternPSU** (int state)
Function to get the desired PinMode from HAS.
- int **getExternPSU** ()
Getter Function to get the PinMode.
- void **regulateVoltage** (float targetVoltage, float hysteresis)
Function to regulate voltage to prevent swinging using a hyseresis.
- void **setTargetVoltage** (float voltage)

- Setter Function for the HAS to regulate the target Voltage.*
 - float `getTargetVoltage` () const
 - Getter Function for the HAS to know what the current targetVoltage is.*
 - void `setHysteresis` (float hysteresis)
 - Setter Function for the HAS to regulate the hysteresis.*
 - float `getHysteresis` () const
 - Getter Function for the HAS to know what the current hysteresisVoltage is.*

6.9.1 Detailed Description

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

6.9.2 Member Function Documentation

6.9.2.1 `getExternPSU()`

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

Getter Function to get the PinMode.

Returns

integer with state

6.9.2.2 `getHVState()`

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

Get the state of the HV-Output.

Returns

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

6.9.2.3 `getHVSwitchState()`

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

Get the state of the HV-Switch.

Returns

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

6.9.2.4 getHysteresis()

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

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

Returns

The Voltage the system is using to prevent swinging.

6.9.2.5 getMainSwitchState()

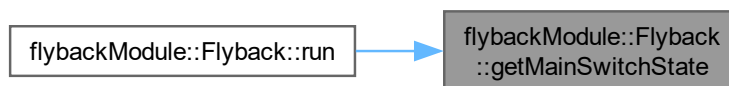
```
MainSwitchStates Flyback::getMainSwitchState ( )
```

Get the state of the Main-Switch.

Returns

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

Here is the caller graph for this function:



6.9.2.6 getTargetVoltage()

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

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

Returns

The Voltage the system is trying to reach.

6.9.2.7 getTimerState()

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

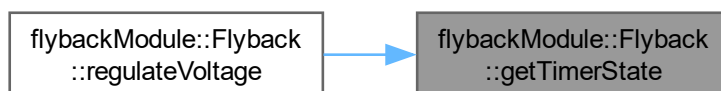
Returns the state of the timer.

Returns

true -> if the timer is initialized

false -> if the timer is not initialized

Here is the caller graph for this function:



6.9.2.8 isInitialized()

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

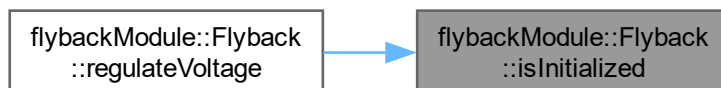
Get the state of the [Flyback](#) system.

Returns

true -> [Flyback](#) is initialized

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

Here is the caller graph for this function:



6.9.2.9 measure()

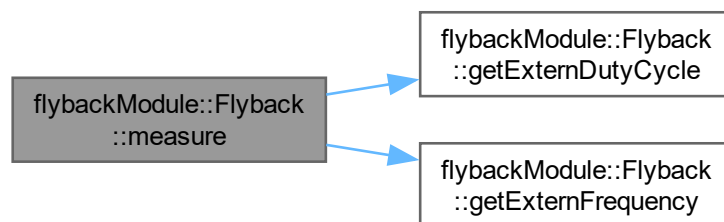
```
Measurement Flyback::measure ( )
```

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

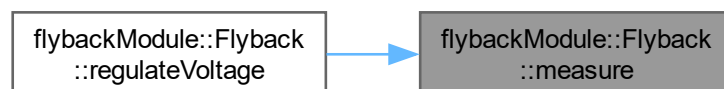
Returns

[Measurement](#) -> A [Measurement](#) object containing voltage, current, and power

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.2.10 regulateVoltage()

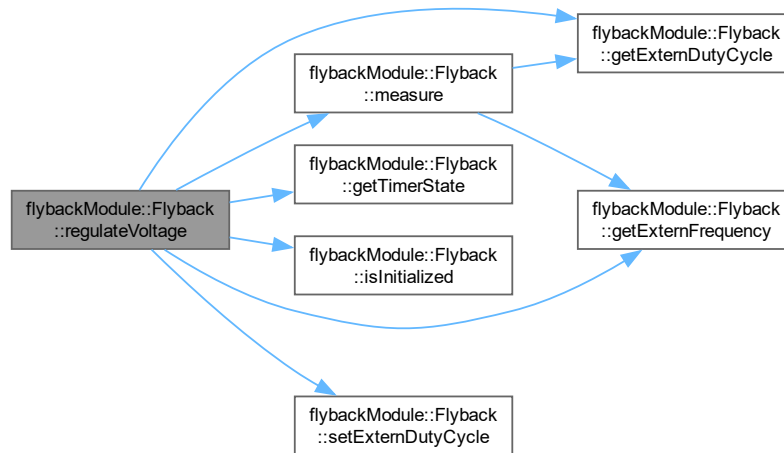
```
void Flyback::regulateVoltage (
    float targetVoltage,
    float hysteresis )
```

Function to regulate voltage to prevent swinging using a hyseresis.

Parameters

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

Here is the call graph for this function:



6.9.2.11 setExternDutyCycle()

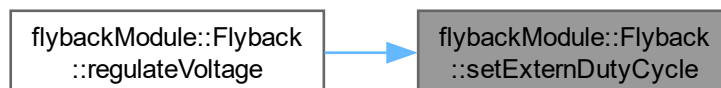
```
void Flyback::setExternDutyCycle (
    int dutyCycle )
```

Function to get the desired DutyCycle from HAS.

Parameters

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

Here is the caller graph for this function:



6.9.2.12 setExternFrequency()

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

Function to get the desired Frequency from HAS.

Parameters

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

6.9.2.13 setExternPSU()

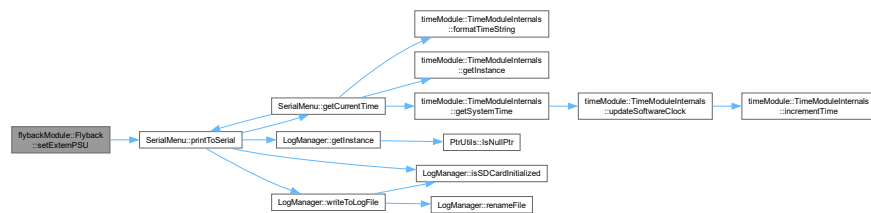
```
void Flyback::setExternPSU (
    int state )
```

Function to get the desired PinMode from HAS.

Parameters

<i>state</i>	-> the state to change the PinMode
--------------	------------------------------------

Here is the call graph for this function:

**6.9.2.14 setHysteresis()**

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

Setter Function for the HAS to regulate the hysteresis.

Parameters

<i>hysteresis</i>	-> The desired hysteresis.
-------------------	----------------------------

6.9.2.15 setTargetVoltage()

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

Setter Function for the HAS to regulate the target Voltage.

Parameters

<i>voltage</i>	-> The desired Voltage to reach.
----------------	----------------------------------

6.9.2.16 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
--------------	---

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

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

6.10 comModule::I2CCommunication Class Reference

Class to handle I2C communication.

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

Public Member Functions

- void [beginI2C](#) (uint8_t address)
Function to initialize the I2C communication.
- void [beginI2CGlobal](#) ()
Function to initialize the I2C communication, without speifying I2C address.
- void [endI2C](#) ()
Function to end the I2C communication.
- void [i2cWrite](#) (uint8_t deviceAddress, uint8_t *data, size_t length)
Function to write data to the I2C device.
- size_t [i2cRead](#) (uint8_t deviceAddress, uint8_t *buffer, size_t length)
Function to read data from the I2C device.
- bool [isInitialized](#) () const
Function to check if the I2C communication is initialized.

6.10.1 Detailed Description

Class to handle I2C communication.

6.10.2 Member Function Documentation

6.10.2.1 beginI2C()

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

Function to initialize the I2C communication.

Parameters

<i>address</i>	-> The address of the I2C device
----------------	----------------------------------

6.10.2.2 i2cRead()

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

Function to read data from the I2C device.

Parameters

<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

6.10.2.3 i2cWrite()

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

Function to write data to the I2C device.

Parameters

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

6.10.2.4 isInitialized()

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

Function to check if the I2C communication is initialized.

Returns

- true -> if the I2C communication is initialized
- false -> if the I2C communication is not initialized

Here is the caller graph for this function:



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

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

6.11 jsonModule::JsonModuleInternals Class Reference

Class for the JSON module internals.

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

Public Member Functions

- template<typename T >
void **createJson** (const char *key, T value)
Add a key-value pair to the Json document.
- void **sendJsonSerial** ()
Function to send the Json object over the Serial connection.
- String **getJSONString** () const
Get the Json String object.
- void **clearJson** ()
Clear the Json object.
- void **printJsonDocMemory** ()
Prints information about the Json object.
- bool **hasCapacityFor** (size_t additionalSize) const
Check if there is enough capacity left in the JsonDocument.

Public Attributes

- size_t **jsonBufferSize**

6.11.1 Detailed Description

Class for the JSON module internals.

6.11.2 Member Function Documentation

6.11.2.1 createJson()

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

Add a key-value pair to the Json document.

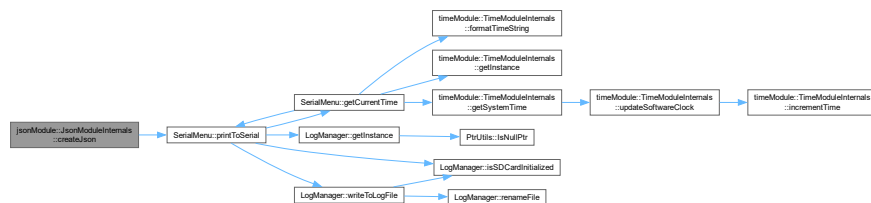
Template Parameters

<i>T</i>	Type of the value.
----------	--------------------

Parameters

<i>key</i>	Key to set.
<i>value</i>	Value to associate with the key.

Here is the call graph for this function:



6.11.2.2 getJsonString()

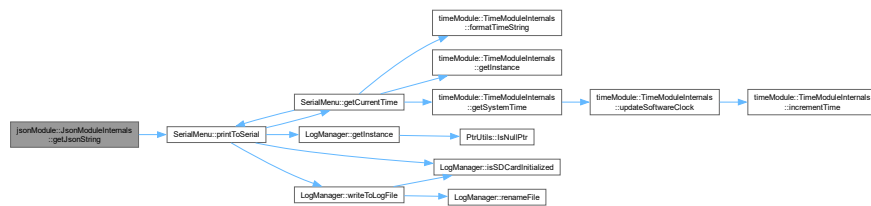
```
String JsonModuleInternals::getJsonString ( ) const
```

Get the Json String object.

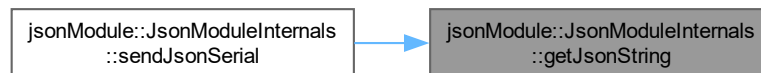
Returns

String -> The Json String object.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.2.3 hasCapacityFor()

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

Check if there is enough capacity left in the JsonDocument.

Parameters

<i>additionalSize</i>	Approximate size of the data to be added.
-----------------------	---

Returns

true If there is enough space.
false If adding may overflow the buffer.

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

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

6.12 LockerBase Class Reference

Base class for the locker system.

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

Public Member Functions

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

6.12.1 Detailed Description

Base class for the locker system.

6.12.2 Member Function Documentation

6.12.2.1 lockEthernetScoped()

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

Returns

[locker::ScopedLock](#)

6.12.2.2 lockSerialScoped()

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

Returns

[locker::ScopedLock](#)

6.12.2.3 lockTemperatureScoped()

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

Returns

[locker::ScopedLock](#)

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

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

6.13 LogManager Class Reference

Public Member Functions

- void `initSDCard` (int cs)
Function to initialize the SD card.
- void `shutdownSDCard` ()
Function to shut down the SD card.
- void `flushLogs` ()
Function to flush the current Logs in special cases.
- bool `isSDCardInitialized` () const
Function to check if the SD card is initialized.
- void `setLogFileName` (const String &fileName)
Set the Log File Name object.
- bool `writeToLogFile` (const String &logMessage)
Function to write a log message to the log file.
- void `renameFile` (const String &oldName, const String &newName)
Function to rename the currently written to file.

Static Public Member Functions

- static `LogManager * getInstance` ()
Get the Instance object.
- static String `getCurrentTime` ()
Getter for the current time.

6.13.1 Member Function Documentation

6.13.1.1 getCurrentTime()

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

Getter for the current time.

Returns

The current time as a String

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.1.2 getInstance()

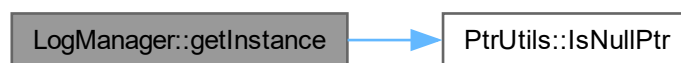
```
LogManager * LogManager::getInstance ( ) [static]
```

Get the Instance object.

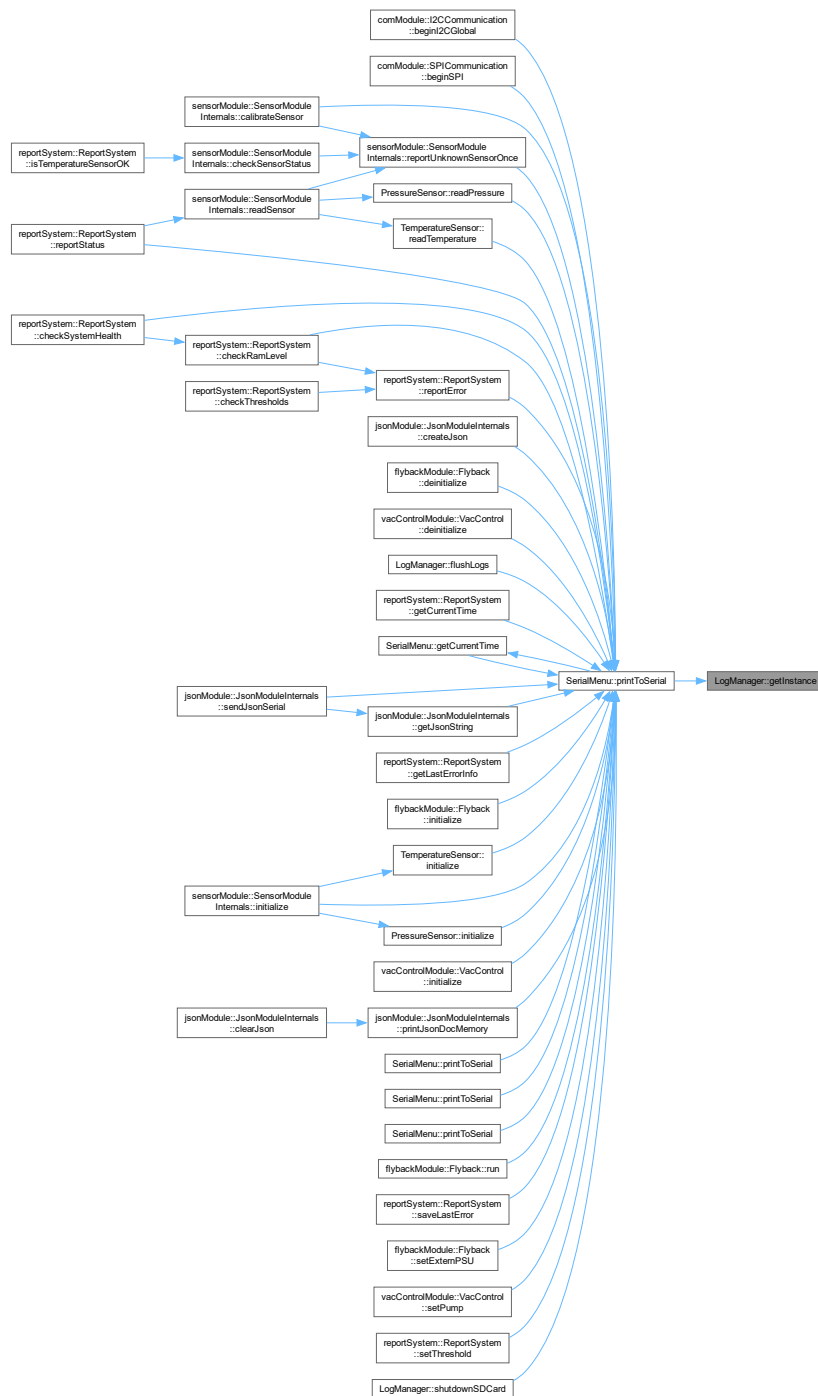
Returns

LogManager*

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.1.3 initSDCard()

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

Function to initialize the SD card.

Parameters

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

6.13.1.4 isSDCardInitialized()

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

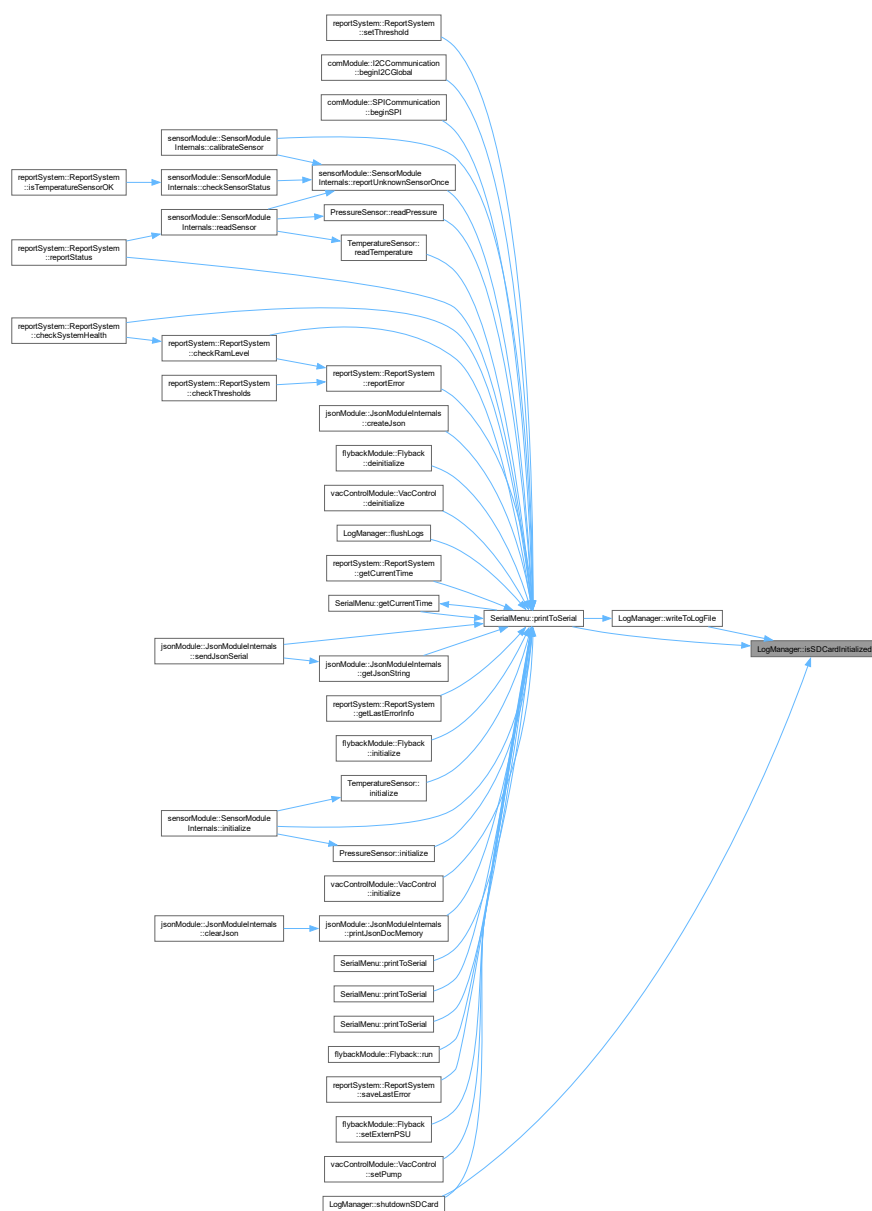
Function to check if the SD card is initialized.

Returns

true -> if the SD card is initialized

false -> if the SD card is not initialized

Here is the caller graph for this function:



6.13.1.5 renameFile()

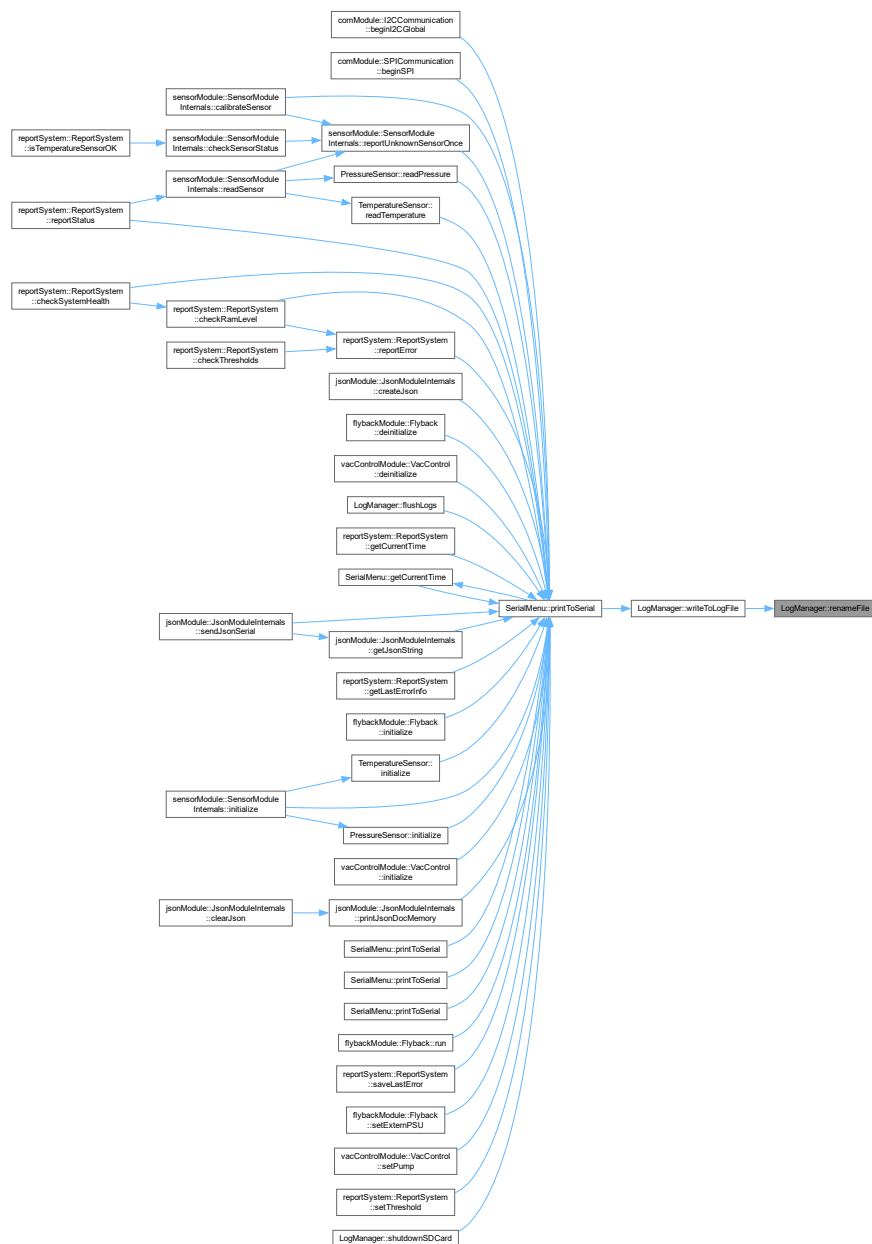
```
void LogManager::renameFile (
    const String & oldName,
    const String & newName )
```

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:



6.13.1.6 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:



6.13.1.7 writeToLogFile()

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

Function to write a log message to the log file.

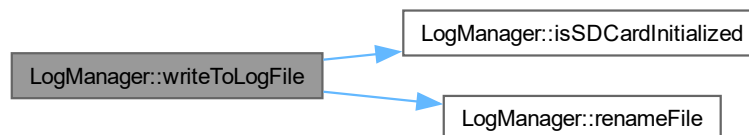
Parameters

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

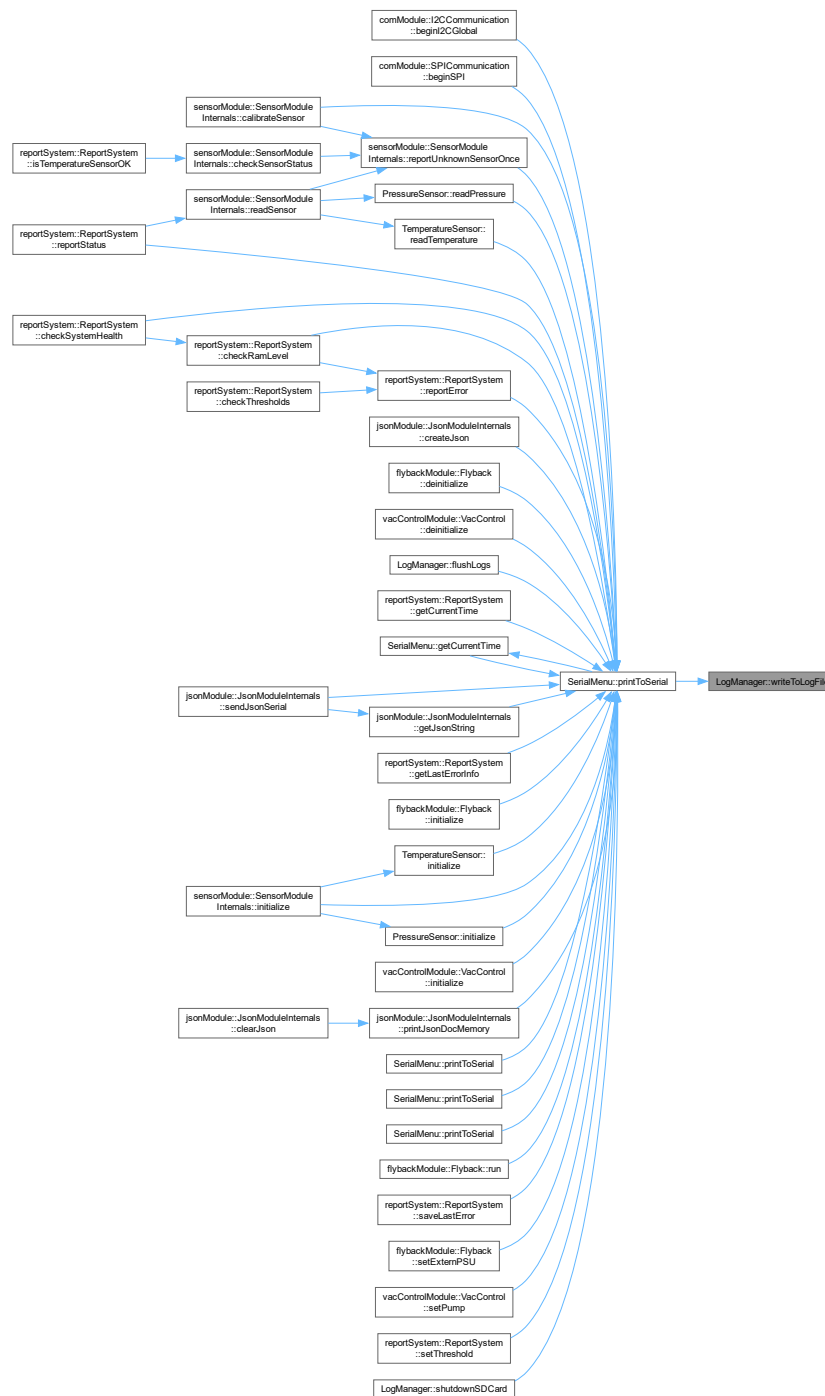
Returns

- 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](#)

6.14 LogMapper Class Reference

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

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

6.14.1 Detailed Description

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

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

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

6.15 flybackModule::Measurement Struct Reference

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

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

Public Attributes

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

6.15.1 Detailed Description

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

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

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

6.16 Measurement Struct Reference

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

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

6.16.1 Detailed Description

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

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

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

6.17 MenuItem Struct Reference

Serial menu structure.

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

Public Attributes

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

6.17.1 Detailed Description

Serial menu structure.

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

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

6.18 Outputlevel Class Reference

Enum Class for the differnet Outputlevels.

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

6.18.1 Detailed Description

Enum Class for the differnet Outputlevels.

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

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

6.19 PointerWrapper< T > Class Template Reference

Tempalte class for wrapping a pointer.

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

Public Member Functions

- **PointerWrapper** (T *p=nullptr)
- T * **get** () const
Function to get the pointer.
- T * **release** ()
Function to release the pointer.
- void **reset** (T *p=nullptr)
Function to reset the pointer.
- T & **operator*** ()
Operator to dereference the pointer.
- T * **operator->** ()
Operator to access the pointer.

6.19.1 Detailed Description

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

Tempalte class for wrapping a pointer.

Template Parameters

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

6.19.2 Member Function Documentation

6.19.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.19.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.19.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.19.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.19.2.5 reset()

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

Function to reset the pointer.

Parameters

<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

6.20 vacControlModule::Pressure Struct Reference

Public Attributes

- float **pressure**

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

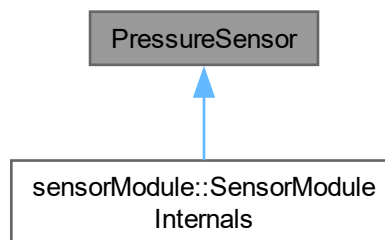
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/[vacControl.h](#)

6.21 PressureSensor Class Reference

Pressure sensor class.

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

Inheritance diagram for PressureSensor:



Public Member Functions

- void **initialize** ()
Function to initialize the pressure sensor.
- float **readPressure** ()
Function to read the pressure from the sensor.
- bool **isInitialized** () const
Function to check if the pressure sensor is initialized.

6.21.1 Detailed Description

Pressure sensor class.

6.21.2 Member Function Documentation

6.21.2.1 isInitialized()

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

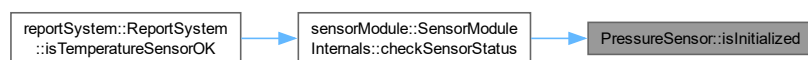
Function to check if the pressure sensor is initialized.

Returns

true -> if the pressure sensor is initialized

false -> if the pressure sensor is not initialized

Here is the caller graph for this function:



6.21.2.2 readPressure()

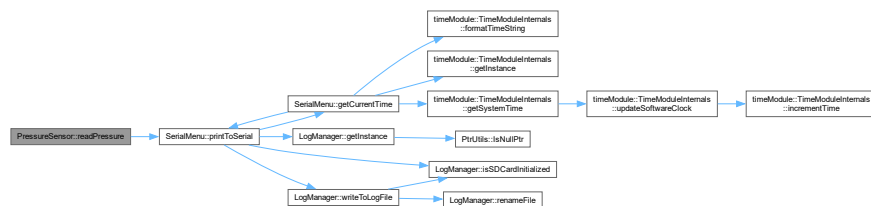
```
float PressureSensor::readPressure ( )
```

Function to read the pressure from the sensor.

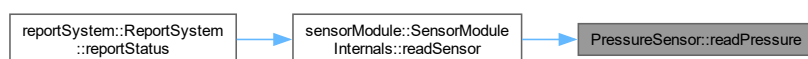
Returns

float -> The pressure value.

Here is the call graph for this function:



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](#)

6.22 PtrUtils Class Reference

Utility class for pointer operations.

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

Static Public Member Functions

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

6.22.1 Detailed Description

Utility class for pointer operations.

6.22.2 Member Function Documentation

6.22.2.1 IsNullPtr()

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

Check if a pointer is nullptr.

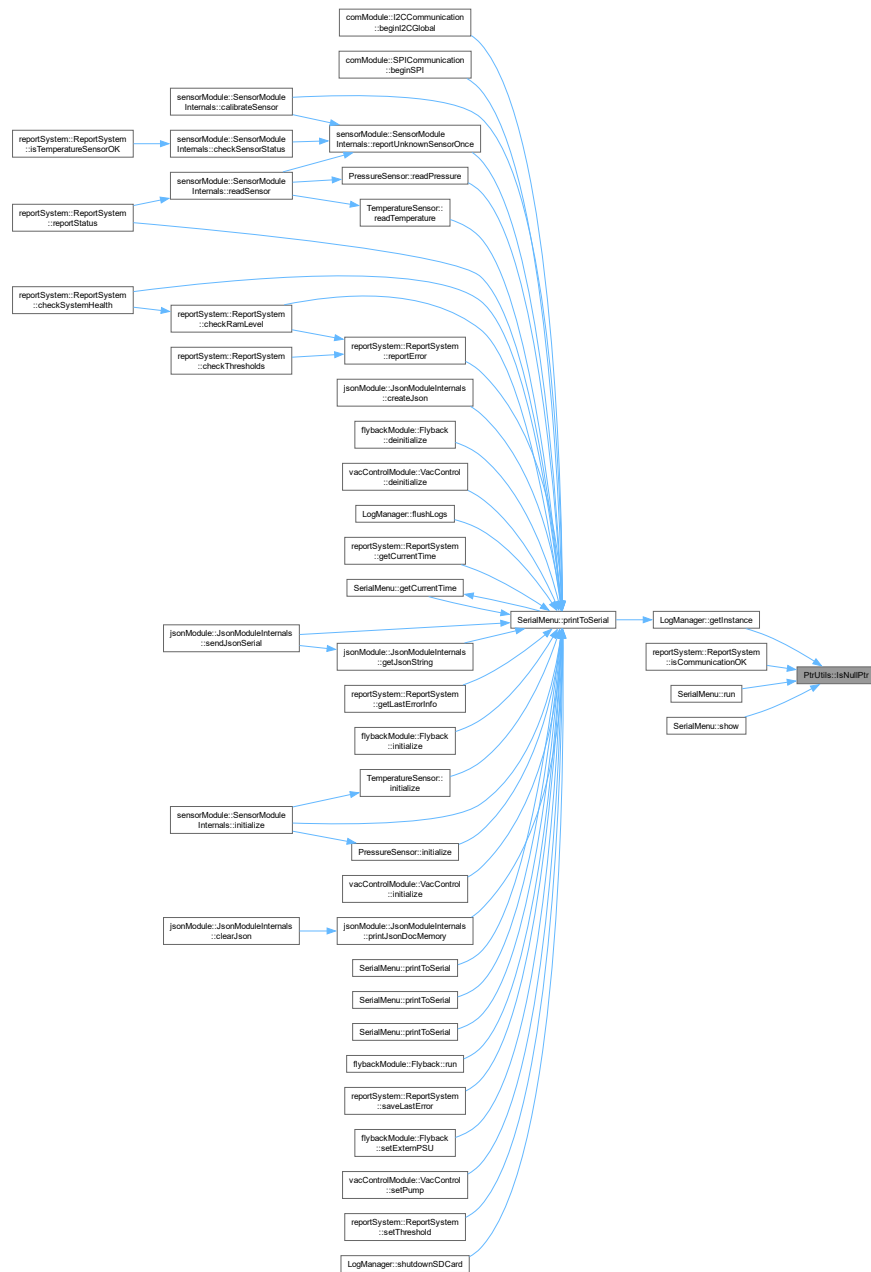
Parameters

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

Returns

true if the pointer is nullptr, false otherwise.

Here is the caller graph for this function:



6.22.2.2 IsValidPtr()

```

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

Check if a pointer is valid (not nullptr).

Parameters

<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](#)

6.23 reportSystem::ReportSystem Class Reference

Class for the report system.

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

Public Member Functions

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

- Get the Last *Error* message from EEPROM.

 - bool `getLastErrorInfo` ()
- Get the Last *Error* message from EEPROM.

 - bool `checkRamLevel` (unsigned int warningThreshold, unsigned int criticalThreshold)

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

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

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

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

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

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

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

Static Public Member Functions

- static void `initStackGuard` ()

Initialize the Stack Guard.
- static bool `detectStackOverflow` ()

Detect Stack Overflow.

6.23.1 Detailed Description

Class for the report system.

6.23.2 Member Function Documentation

6.23.2.1 checkRamLevel()

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

Function to check the SRAM level on the hostsystem.

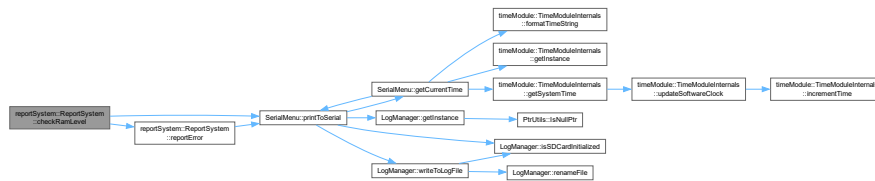
Parameters

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

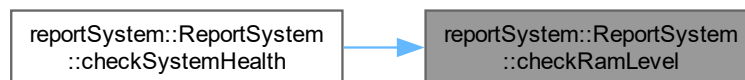
Returns

true -> if the level exceeded
false -> if the levels are withing the thresholds

Here is the call graph for this function:



Here is the caller graph for this function:



6.23.2.2 checkSystemHealth()

```

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

Function to check the system health of the uC.

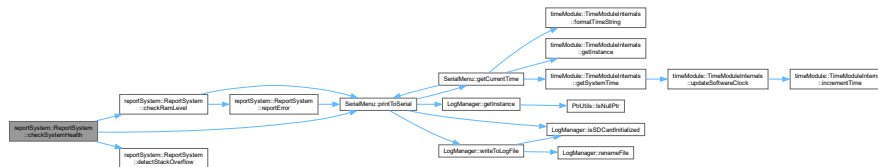
Parameters

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

Returns

true -> if the system is healthy
false -> if the system is not healthy

Here is the call graph for this function:

**6.23.2.3 checkThresholds()**

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

Check the thresholds for the temperature and pressure sensors.

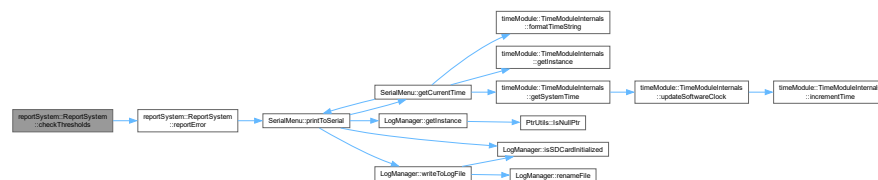
Parameters

<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:

**6.23.2.4 detectStackOverflow()**

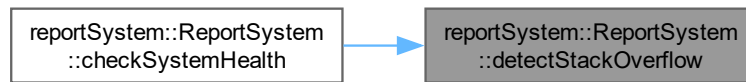
```
bool ReportSystem::detectStackOverflow ( ) [static]
```

Detect Stack Overflow.

Returns

true -> if the stack has overflowed
false -> if the stack has not overflowed

Here is the caller graph for this function:

**6.23.2.5 getCPULoad()**

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

Getter for the CPU Load.

Returns

float -> The CPU Load

6.23.2.6 getCurrentTime()

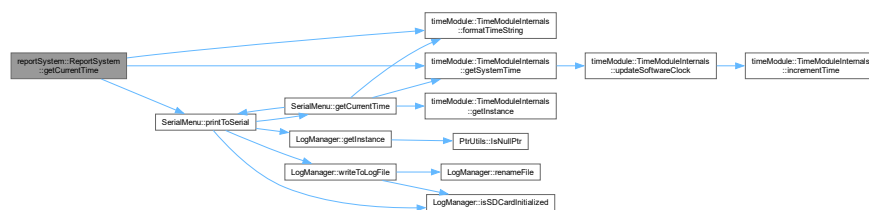
```
String ReportSystem::getCurrentTime ( )
```

Get the Current Time of the system.

Returns

String -> The current time

Here is the call graph for this function:



6.23.2.7 getLastError()

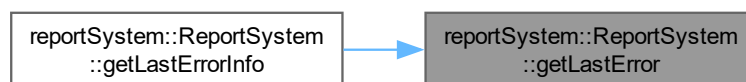
```
String ReportSystem::getLastError ( )
```

Get the Last **Error** message from EEPROM.

Returns

String -> The last error message

Here is the caller graph for this function:



6.23.2.8 getLastErrorInfo()

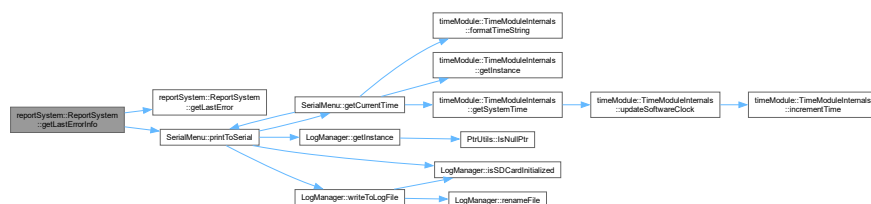
```
bool ReportSystem::getErrorInfo ( )
```

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:



6.23.2.9 getMemoryStatus()

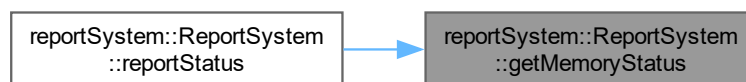
```
String ReportSystem::getMemoryStatus ( )
```

Get the Memory Status of the system.

Returns

String -> The memory status

Here is the caller graph for this function:



6.23.2.10 getStackDump()

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

Get the Stack Dump of the system.

Returns

String -> The stack dump

6.23.2.11 hasNoSavedErrors()

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

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

Returns

True if no errors in EEPROM, false otherwise

6.23.2.12 isCommunicationOK()

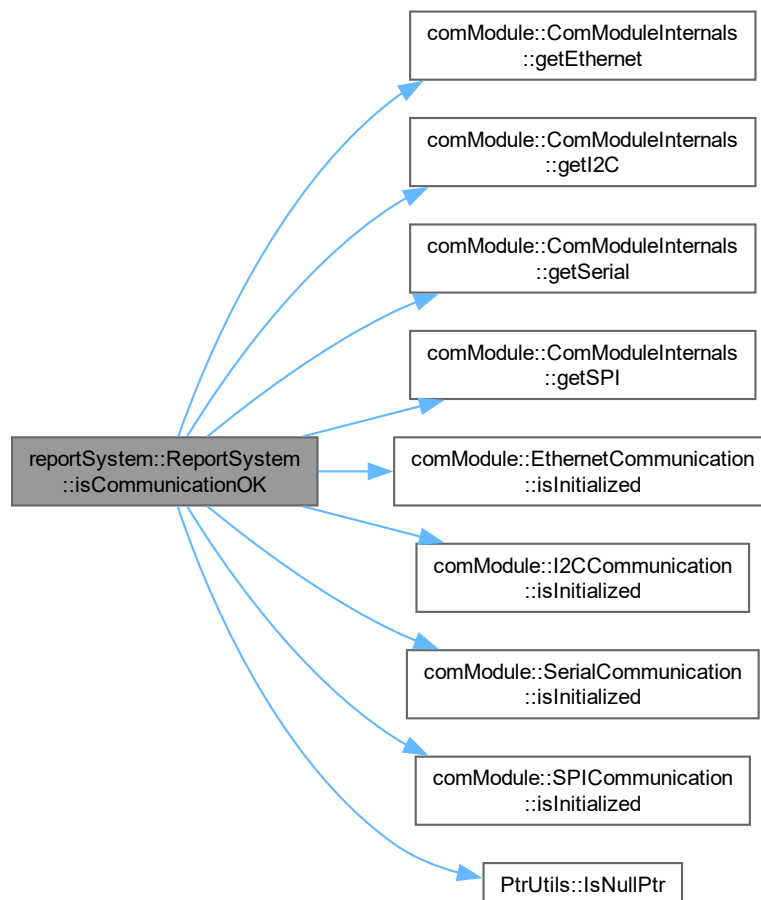
```
bool ReportSystem::isCommunicationOK ( ) const
```

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

Returns

True if communication is up, false otherwise

Here is the call graph for this function:



6.23.2.13 isMemoryOK()

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

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

Returns

True if memory is in between legal bounds, false otherwise

6.23.2.14 isRamOK()

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

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

Returns

True if RAM is in between legal bounds, false otherwise

6.23.2.15 isStackSafe()

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

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

Returns

True if stack is safe, false otherwise

6.23.2.16 isTemperatureSensorOK()

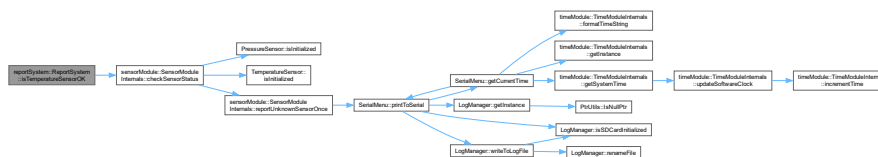
```
bool ReportSystem::isTemperatureSensorOK ( ) const
```

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

Returns

True if sensor is ok, false otherwise

Here is the call graph for this function:



6.23.2.17 reportError()

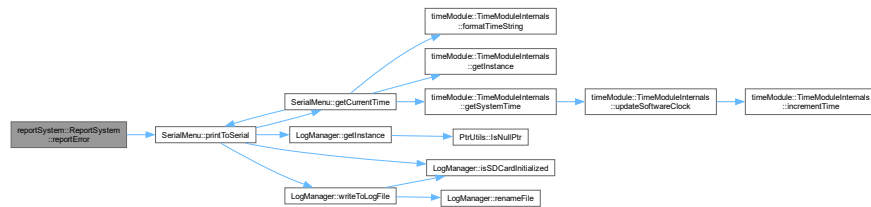
```
void ReportSystem::reportError (
    const char * errorMessage )
```

Function to log an error message.

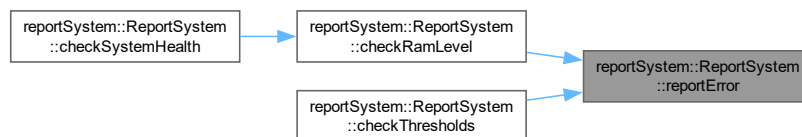
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.23.2.18 reportStatus()

```

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

Function to report the status of the system.

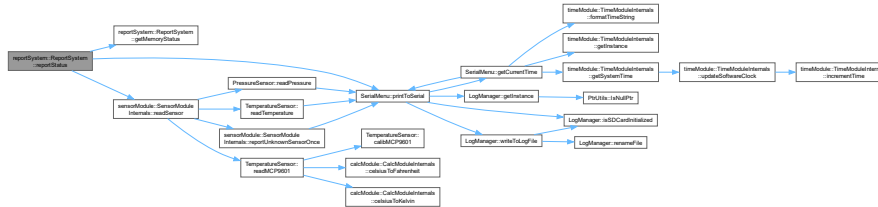
Parameters

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

Returns

bool -> The status of the system

Here is the call graph for this function:



6.23.2.19 saveLastError()

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

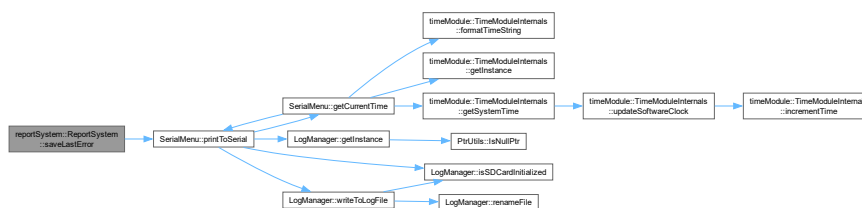
Saves last error message to EEPROM.

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

Parameters

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

Here is the call graph for this function:



6.23.2.20 setThreshold()

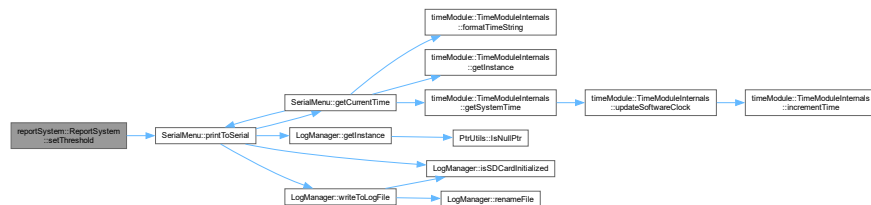
```
void ReportSystem::setThreshold (
    float tempThreshold,
    float pressureThreshold )
```

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](#)

6.24 locker::ScopedLock Class Reference

Scoped lock class for mutexes.

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

Public Member Functions

- [ScopedLock](#) (frt::Mutex &mutex)
Construct a new Scoped Lock object.
- **ScopedLock** (const [ScopedLock](#) &)=delete
- [ScopedLock](#) & **operator=** (const [ScopedLock](#) &)=delete
- [ScopedLock](#) ([ScopedLock](#) &&other) noexcept
Construct a new Scoped Lock object.
- [ScopedLock](#) & **operator=** ([ScopedLock](#) &&)=delete

6.24.1 Detailed Description

Scoped lock class for mutexes.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 ScopedLock() [1/2]

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

Construct a new Scoped Lock object.

Parameters

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

6.24.2.2 ScopedLock() [2/2]

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

Construct a new Scoped Lock object.

Parameters

<i>other</i>	-> The other ScopedLock object to move from
--------------	---

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

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

6.25 ScopedPointer< T > Class Template Reference

Template class for a Scoped Pointer.

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

Public Member Functions

- **ScopedPointer** (T *p=nullptr)
- T * [get](#) () const
Function to get the pointer.
- T * [release](#) ()
Function to release the pointer.
- void [reset](#) (T *p=nullptr)
Function to reset the pointer.
- T & [operator*](#) () const
Operator to dereference the pointer.
- T * [operator->](#) () const
Operator to access the pointer.

6.25.1 Detailed Description

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

Template class for a Scoped Pointer.

Template Parameters

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

6.25.2 Member Function Documentation

6.25.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.25.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.25.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.25.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.25.2.5 reset()

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

Function to reset the pointer.

Parameters

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

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

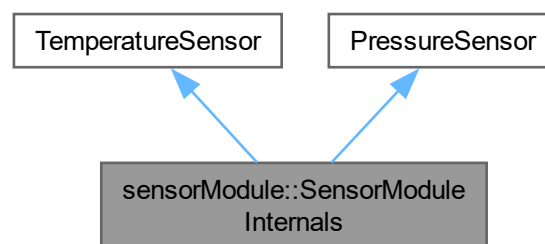
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/[ptrUtils.h](#)

6.26 sensorModule::SensorModuleInternals Class Reference

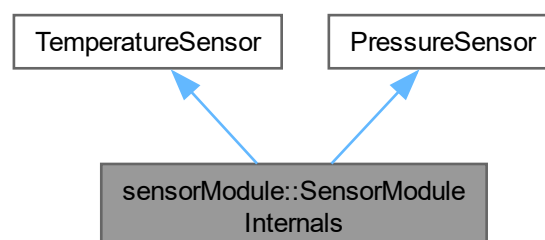
Class for the sensor module internals.

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

Inheritance diagram for sensorModule::SensorModuleInternals:



Collaboration diagram for sensorModule::SensorModuleInternals:



Public Member Functions

- void **initialize** ()
Initialize the sensors.
- float **readSensor** (SensorType type)
Function to read the sensor.
- bool **calibrateSensor** (SensorType type)
Function to calibrate the sensor.
- bool **checkSensorStatus** (SensorType type)
Function to check the status of the sensor.
- void **reportUnknownSensorOnce** (SensorType type, const __FlashStringHelper *context)
Function to Report the faulty sensor only if it changed, saves us some String and doesn't shred the SRAM that much.

Public Member Functions inherited from [TemperatureSensor](#)

- void **initialize** ()
Function to initialize the temperature sensor.
- float **readTemperature** ()
Function to read the temperature from the sensor.
- float **readMCP9601** (Units unit, SensorID sensor)
Function to read form specific sensor MCP9601.
- bool **isInitialized** () const
Check if the temperature sensor is initialized.
- uint8_t **calibMCP9601** (SensorID sensor)
Method to calibrate the MCP9601 sensor, Indoor and Outdoor Env.

Public Member Functions inherited from [PressureSensor](#)

- void **initialize** ()
Function to initialize the pressure sensor.
- float **readPressure** ()
Function to read the pressure from the sensor.
- bool **isInitialized** () const
Function to check if the pressure sensor is initialized.

6.26.1 Detailed Description

Class for the sensor module internals.

6.26.2 Member Function Documentation

6.26.2.1 **calibrateSensor()**

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

Function to calibrate the sensor.

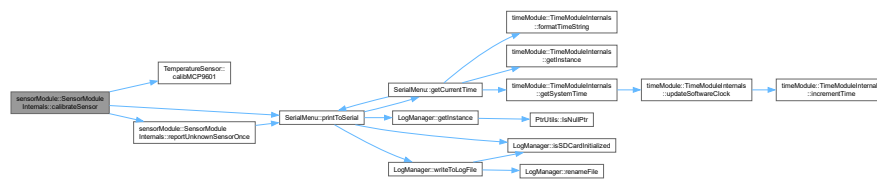
Parameters

<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:



6.26.2.2 checkSensorStatus()

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

Function to check the status of the sensor.

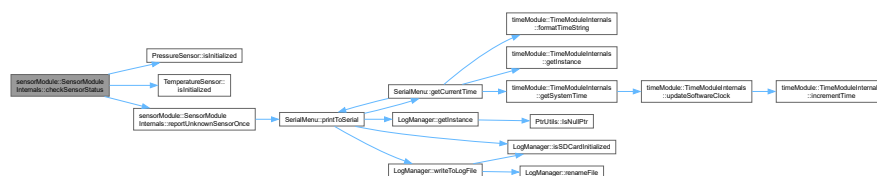
Parameters

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

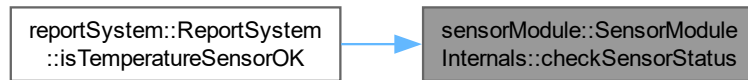
Returns

true -> if the sensor is healthy
false -> if the sensor is not healthy

Here is the call graph for this function:



Here is the caller graph for this function:



6.26.2.3 readSensor()

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

Function to read the sensor.

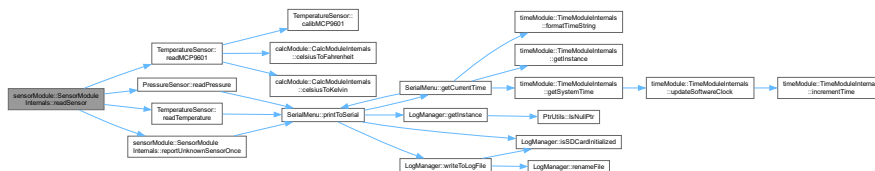
Parameters

<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:



6.26.2.4 reportUnknownSensorOnce()

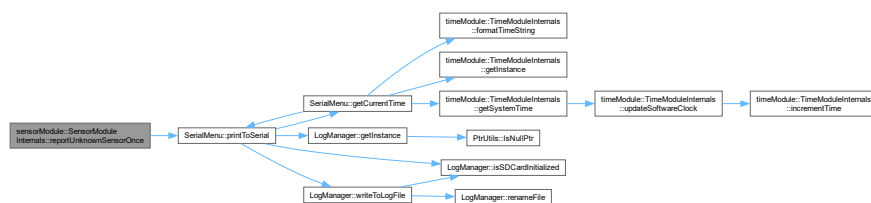
```
void SensorModuleInternals::reportUnknownSensorOnce (
    SensorType type,
    const __FlashStringHelper * context )
```

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

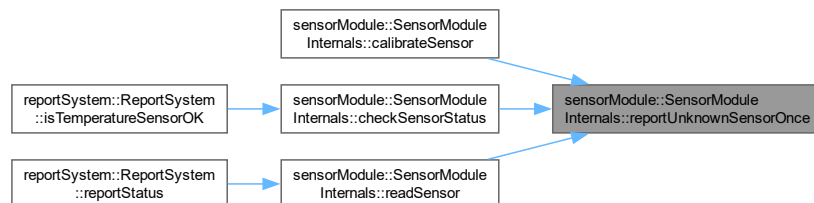
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.27 comModule::SerialCommunication Class Reference

Class to handle Serial communication.

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

Public Member Functions

- void `beginSerial` (long baudRate)
Function to start the serial communication.
- void `endSerial` ()
Function to end the serial communication.
- void `sendSerialData` (const char *data)
Function to end the serial communication.
- void `receiveSerialData` (char *buffer, size_t length)
Function to receive data over serial.
- bool `isInitialized` () const
Function to check if the serial communication is initialized.

6.27.1 Detailed Description

Class to handle Serial communication.

6.27.2 Member Function Documentation

6.27.2.1 `beginSerial()`

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

Function to start the serial communication.

Parameters

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

6.27.2.2 `isInitialized()`

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

Function to check if the serial communication is initialized.

Returns

true -> if the serial communication is initialized
false -> if the serial communication is not initialized

Here is the caller graph for this function:



6.27.2.3 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

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

6.28 SerialMenu Class Reference

Class for the serial menu.

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

Public Types

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

Public Member Functions

- void **load** ([MenuItem](#) *items, size_t size)
 Function to load the menu items.
- void **show** ()
 Function to show the menu.
- void **run** ()
 Function to run the menu.

Static Public Member Functions

- static void [printToSerial](#) (OutputLevel level, const String &message, bool newLine=true, bool 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)
Funtion to print a message to the serial port, using mutexes, output level and new line options.
- static void [printToSerial](#) (const __FlashStringHelper *message, bool newLine=true, bool logMessage=false)
Function to print a message to the serial port, using mutexes, output level and new line options.
- static String [getCurrentTime](#) ()
Getter for the current time.

6.28.1 Detailed Description

Class for the serial menu.

6.28.2 Member Function Documentation

6.28.2.1 getCurrentTime()

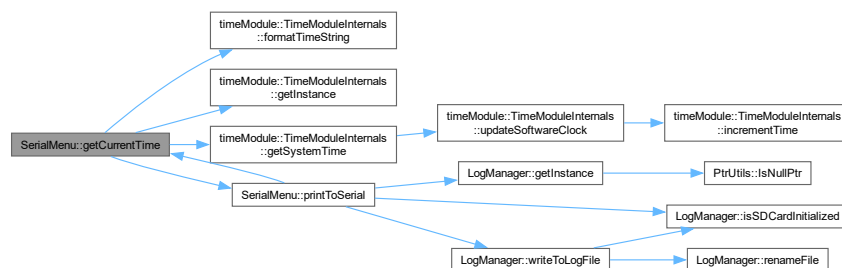
```
String SerialMenu::getCurrentTime ( ) [static]
```

Getter for the current time.

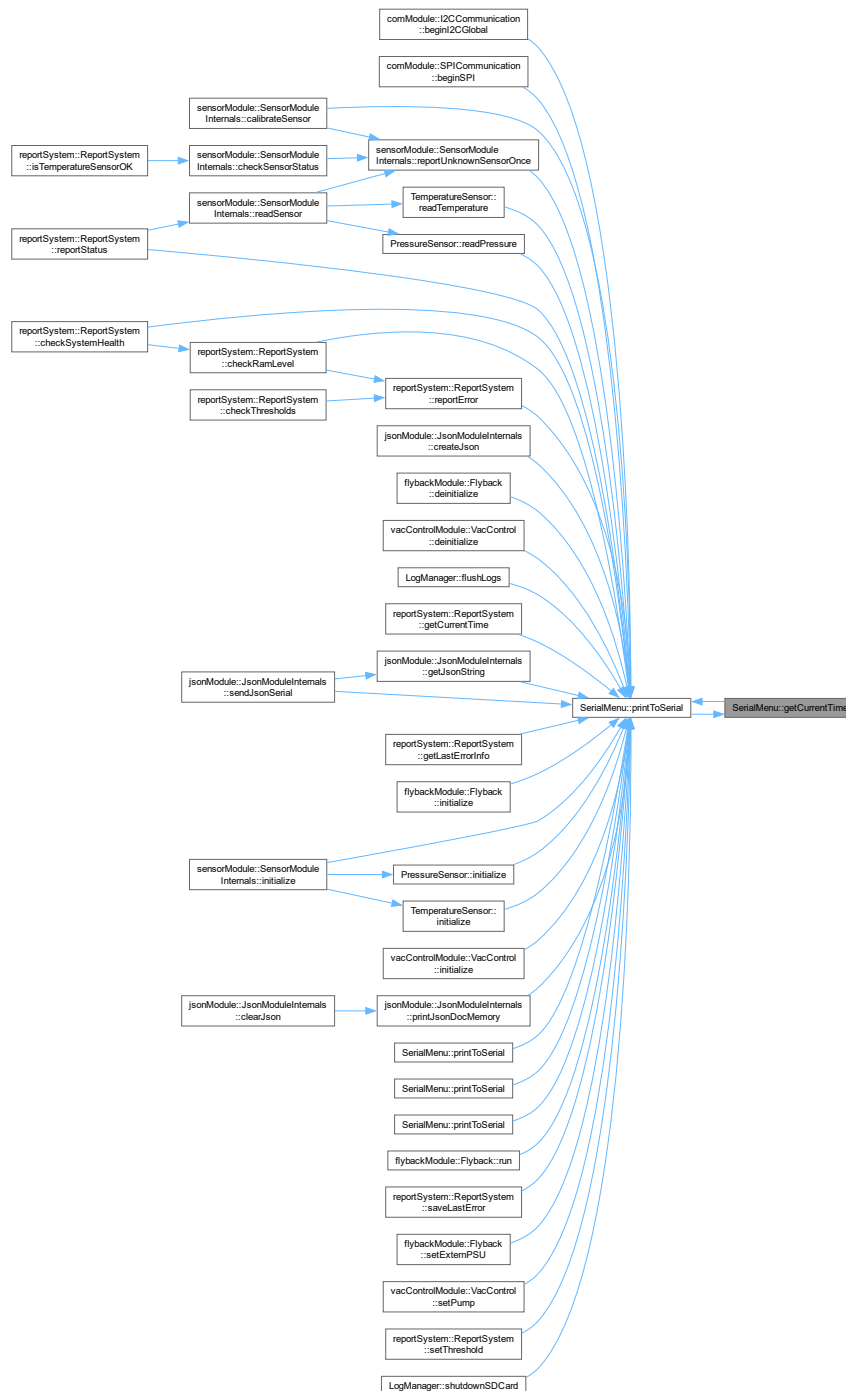
Returns

The current time as a String

Here is the call graph for this function:



Here is the caller graph for this function:



6.28.2.2 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.

6.28.2.3 `printToSerial()` [1/4]

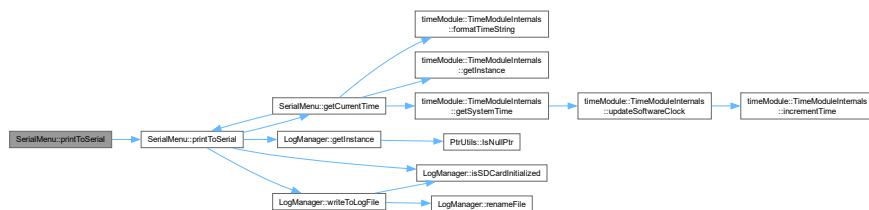
```
void SerialMenu::printToSerial (
    const __FlashStringHelper * message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

Parameters

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

Here is the call graph for this function:

6.28.2.4 `printToSerial()` [2/4]

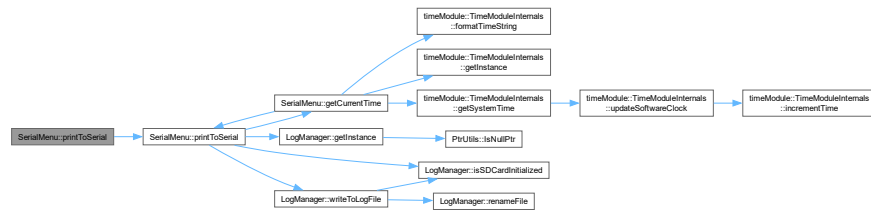
```
void SerialMenu::printToSerial (
    const String & message,
    bool newLine = true,
    bool logMessage = false ) [static]
```

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

Parameters

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

Here is the call graph for this function:



6.28.2.5 printToSerial() [3/4]

```

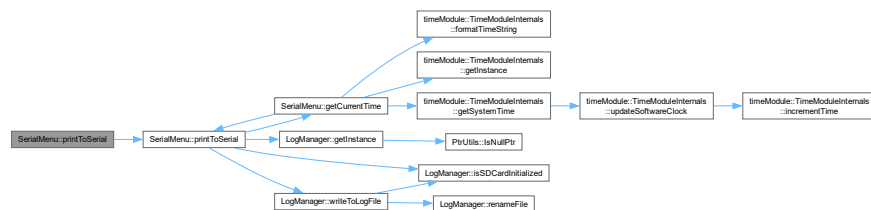
void SerialMenu::printToSerial (
    OutputLevel level,
    const __FlashStringHelper * message,
    bool newLine = true,
    bool logMessage = false ) [static]
  
```

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

Parameters

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

Here is the call graph for this function:



6.28.2.6 printToSerial() [4/4]

```

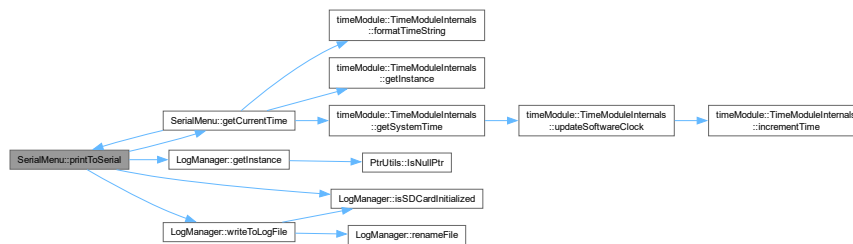
void SerialMenu::printToSerial (
    OutputLevel level,
    const String & message,
    bool newLine = true,
    bool logMessage = false ) [static]
  
```

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

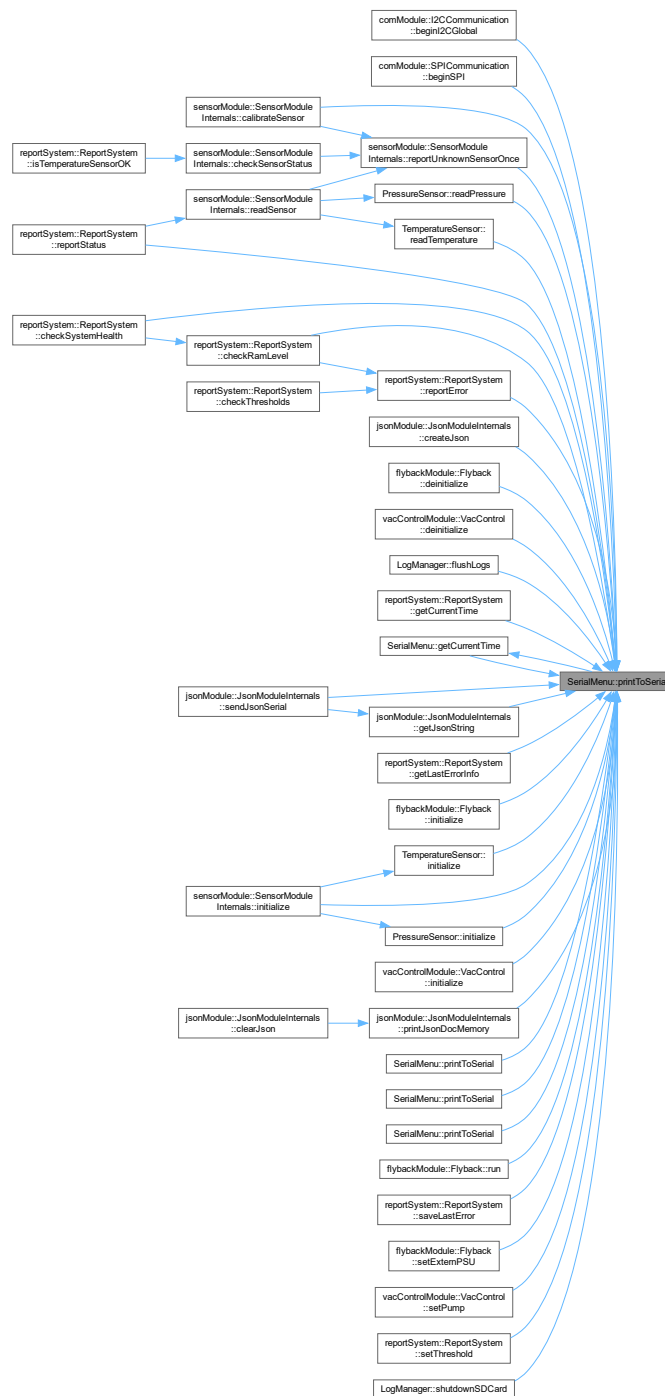
Parameters

<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](#)

6.29 comModule::SPICommunication Class Reference

Class to handle SPI communication.

```
#include <SPII.h>
```

Public Member Functions

- void **beginSPI** ()
Function to initialize the SPI communication.
- void **endSPI** ()
Function to end the SPI communication.
- void **spiWrite** (uint8_t *data, size_t length)
Function to write data over SPI.
- void **spiRead** (uint8_t *buffer, size_t length)
Function to read data over SPI.
- bool **isInitialized** () const
Function to check if the SPI communication is initialized.

6.29.1 Detailed Description

Class to handle SPI communication.

6.29.2 Member Function Documentation

6.29.2.1 isInitialized()

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

Function to check if the SPI communication is initialized.

Returns

true -> if the SPI communication is initialized
false -> if the SPI communication is not initialized

Here is the caller graph for this function:



6.29.2.2 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

6.29.2.3 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:

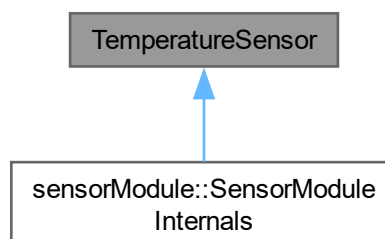
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.cpp

6.30 TemperatureSensor Class Reference

Temperature sensor class.

```
#include <temperature.h>
```

Inheritance diagram for TemperatureSensor:



Public Member Functions

- void **initialize** ()
Function to initialize the temperature sensor.
- float **readTemperature** ()
Function to read the temperature from the sensor.
- float **readMCP9601** (Units unit, SensorID sensor)
Function to read form specific sensor MCP9601.
- bool **isInitialized** () const
Check if the temperature sensor is initialized.
- uint8_t **calibMCP9601** (SensorID sensor)
Method to calibrate the MCP9601 sensor, Indoor and Outdoor Env.

6.30.1 Detailed Description

Temperature sensor class.

6.30.2 Member Function Documentation

6.30.2.1 calibMCP9601()

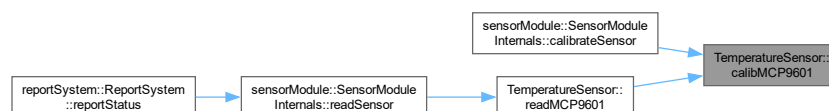
```
uint8_t TemperatureSensor::calibMCP9601 (
    SensorID sensor )
```

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

Returns

uint8_t -> the status of the calibration

Here is the caller graph for this function:



6.30.2.2 isInitialized()

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

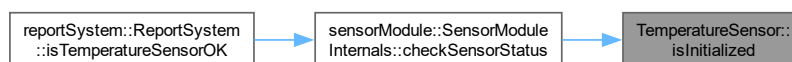
Check if the temperature sensor is initialized.

Returns

true -> if the temperature sensor is initialized

false -> if the temperature sensor is not initialized

Here is the caller graph for this function:



6.30.2.3 readMCP9601()

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

Function to read form specific sensor MCP9601.

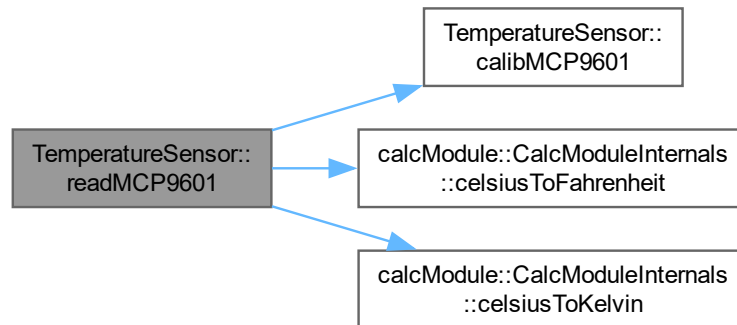
Parameters

<i>unit</i>	-> Choose the unit °C, F, K
-------------	-----------------------------

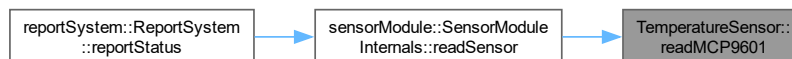
Returns

-> The temperature value

Here is the call graph for this function:



Here is the caller graph for this function:



6.30.2.4 readTemperature()

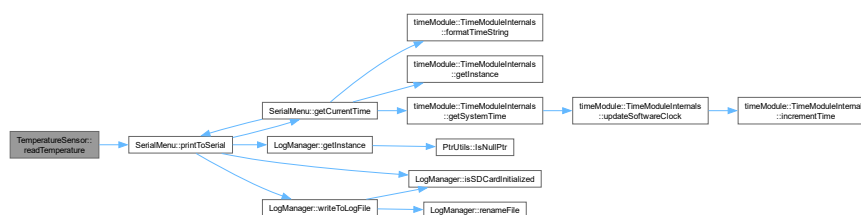
```
float TemperatureSensor::readTemperature ( )
```

Function to read the temperature from the sensor.

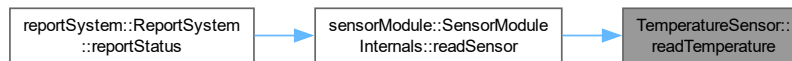
Returns

float -> The temperature value.

Here is the call graph for this function:



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

6.31 timeModule::TimeModuleInternals Class Reference

Class to handle Systemtime.

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

Public Member Functions

- bool [setTimeFromHas](#) (const String &timeString)
Set the Time From Has object to the system time.
- void [setSystemTime](#) (const [DateTimeStruct](#) &dt)
Set the System Time object of the system.
- void **updateSoftwareClock** ()
Updates the software clock.
- [DateTimeStruct](#) [getSystemTime](#) ()
Get the System Time object.

Static Public Member Functions

- static void [incrementTime](#) ([DateTimeStruct](#) *dt)
Function to increment the time of the system.
- static String [formatTimeString](#) (const [DateTimeStruct](#) &dt)
Function to format the time to a string.
- static [TimeModuleInternals](#) * [getInstance](#) ()
Get the Instance object, Singleton pattern.

6.31.1 Detailed Description

Class to handle Systemtime.

6.31.2 Member Function Documentation

6.31.2.1 formatTimeString()

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

Function to format the time to a string.

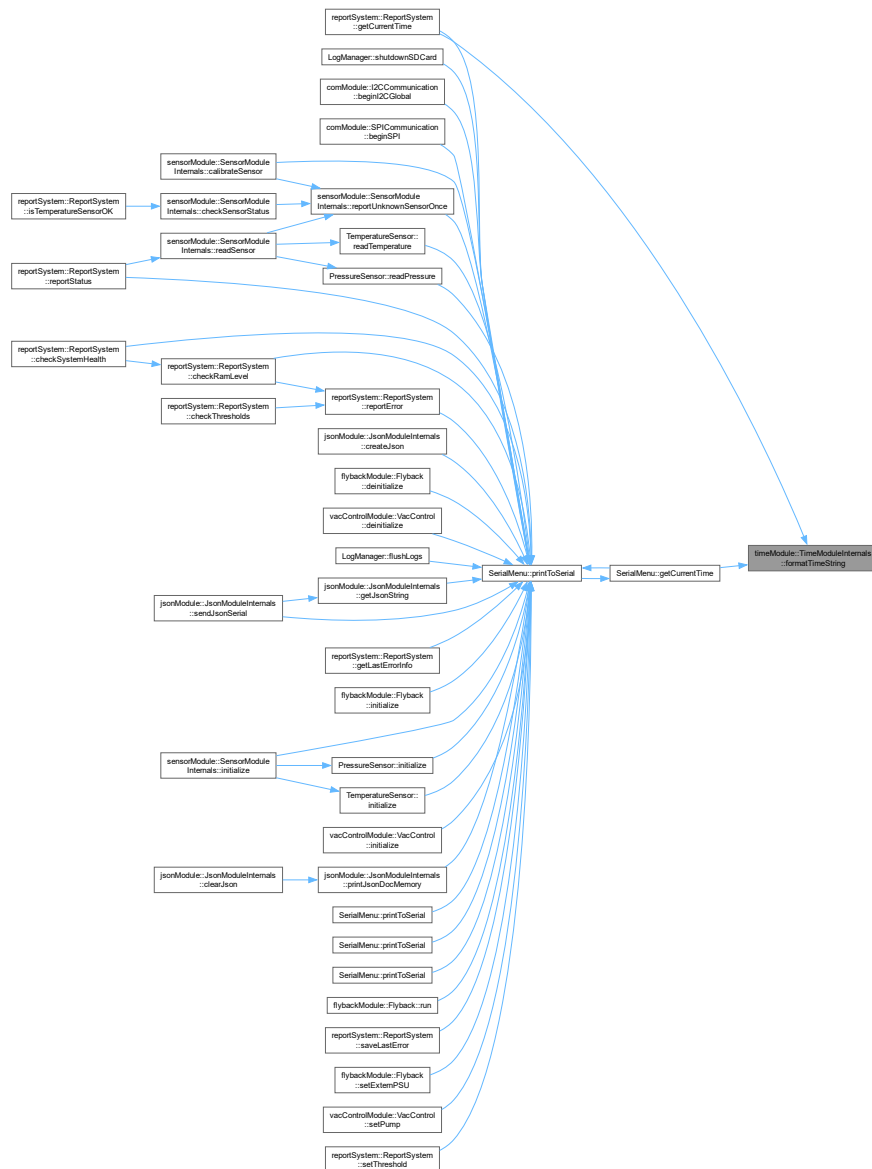
Parameters

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

Returns

String -> The formatted time.

Here is the caller graph for this function:



6.31.2.2 getInstance()

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

Get the Instance object, Singleton pattern.

`TimeModuleInternals*` -> The instance of the `TimeModuleInternals`.

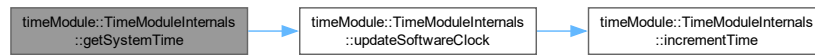
[illegible]

Get the System Time object.

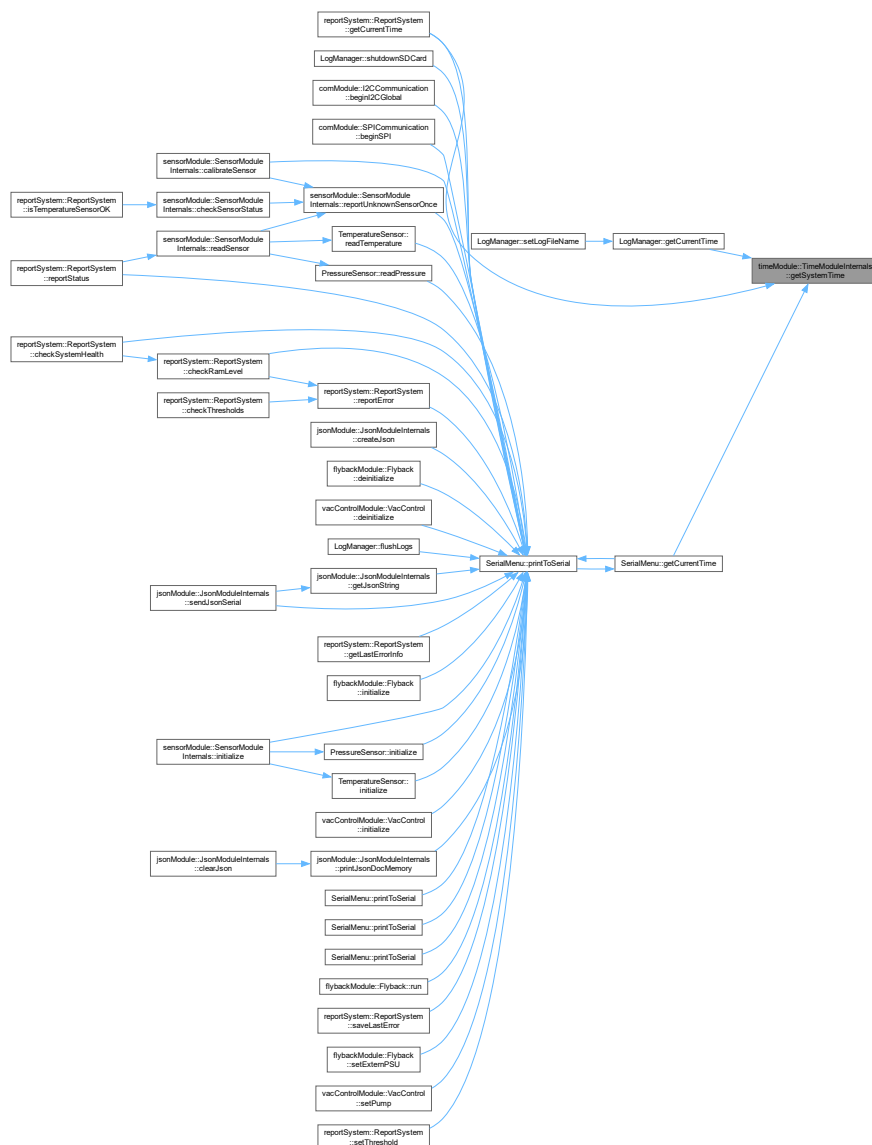
Returns

`DateTimeStruct` -> The system time.

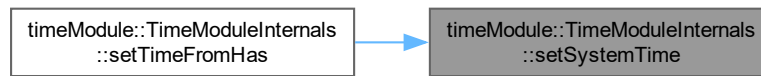
Here is the call graph for this function:



Here is the caller graph for this function:



Here is the caller graph for this function:



6.31.2.6 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](#)

6.32 vacControlModule::VacControl Class Reference

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

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


Public Member Functions

- void **initialize** ()
Initialize the [VacControl](#) System This method sets up the pins and prepares the system for operation.
- void **deinitialize** ()
denitalize the [VacControl](#) System This method shuts down the pins and prepares graceful restart.
- bool **isInitialized** () const
Get the state of the [VacControl](#) system.
- [MainSwitchStates](#) **getMainSwitchState** ()
Returns the state of the main switch.
- [PumpSwitchStates](#) **getPumpSwitchState** ()
Returns the state of the pump switch.
- [PumpState](#) **getPumpState** ()
Returns the state of the pump.
- [Scenarios](#) **getScenario** ()
Executes logic depending on which Main-Switch state is active.
- [Pressure](#) **measure** ()
Measures the actual pressure of the system.
- void **setVacuumLed** (float pressure, float targetPressure)
Controls the vacuum LED based on the current and target pressures.
- int **getScenarioFromPotValue** (int potValue)
Determines the scenario based on the potentiometer value.
- void **setPump** (bool flag)
Set the Pump flag.
- void **run** ()
Runs the main control loop for the [VacControl](#) system.
- void **setExternScenario** (int pressure)
Function to set an external scenario, typically from remote input.
- int **getExternScenario** ()
Getter function to retrieve the current external scenario state.
- void **externPump** (int pumpState)
Process external data for scenarios (currently unused)
- void **setExternPump** (int state)
Function to get the desired PinMode from HAS.
- int **getExternPump** ()
- void **setExternPressure** (float pressure)
Sets the external pressure value.
- float **getExternPressure** ()
Gets the external pressure value.

6.32.1 Detailed Description

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

6.32.2 Member Function Documentation

6.32.2.1 externPump()

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

Process external data for scenarios (currently unused)

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

6.32.2.2 getExternPressure()

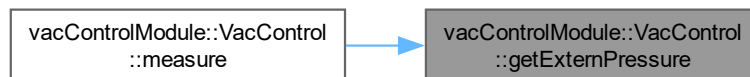
```
float VacControl::getExternPressure ( )
```

Gets the external pressure value.

Returns

The current external pressure value

Here is the caller graph for this function:



6.32.2.3 getExternPump()

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

@ brief Getter Function to get the PinMode

6.32.2.4 getExternScenario()

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

Getter function to retrieve the current external scenario state.

Returns

The current external scenario state (integer)

6.32.2.5 getMainSwitchState()

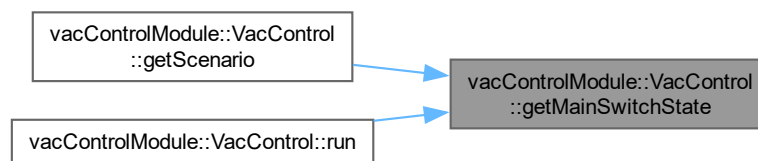
```
MainSwitchStates VacControl::getMainSwitchState ( )
```

Returns the state of the main switch.

Returns

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

Here is the caller graph for this function:



6.32.2.6 getPumpState()

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

Returns the state of the pump.

Returns

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

6.32.2.7 getPumpSwitchState()

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

Returns the state of the pump switch.

Returns

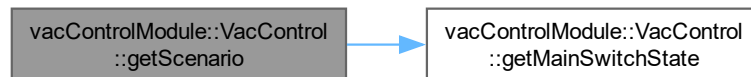
The current state of the switch (Switch_Pump_ON, Switch_PUMP_OFF)

6.32.2.8 getScenario()

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

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

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



6.32.2.9 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

6.32.2.10 isInitialized()

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

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

Returns

true -> [VacControl](#) is initialized and ready

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

6.32.2.11 measure()

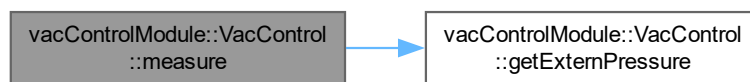
```
Pressure VacControl::measure ( )
```

Measures the actual pressure of the system.

Returns

[Measurement](#) -> A [Measurement](#) object containing the current pressure

Here is the call graph for this function:

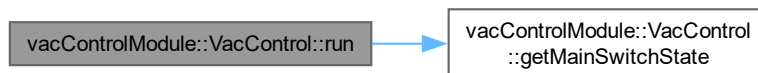


6.32.2.12 run()

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

Runs the main control loop for the [VacControl](#) system.

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



6.32.2.13 setExternPressure()

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

Sets the external pressure value.

Parameters

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

6.32.2.14 setExternPump()

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

Function to get the desired PinMode from HAS.

Parameters

<i>state</i>	-> the state to change the PinMode
--------------	------------------------------------

6.32.2.15 setExternScenario()

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

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

Parameters

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

6.32.2.16 setPump()

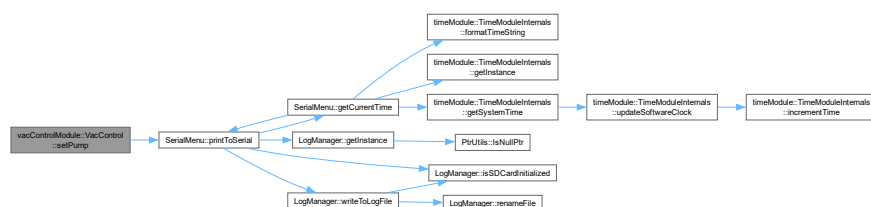
```
void VacControl::setPump (
    bool flag )
```

Set the Pump flag.

Parameters

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

Here is the call graph for this function:



6.32.2.17 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

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

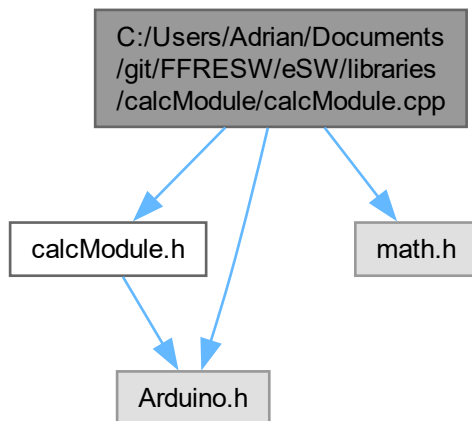
File Documentation

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

Implementation of the `calculatorModule` class.

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

Include dependency graph for calculatorModule.cpp:



7.1.1 Detailed Description

Implementation of the `calculatorModule` class.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

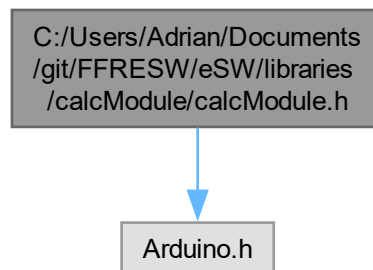
Copyright (c) 2024

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

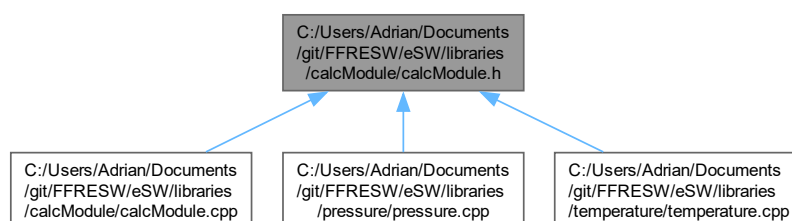
Header file for the calculation module handling sensor data.

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

Include dependency graph for calcModule.h:



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



Classes

- class `calcModule::CalcModuleInternals`

Namespaces

- namespace `calcModule`
Namespace for the calculation module.

Enumerations

- enum `calcModule::Type` { **General** , **Pressure** , **Position** }
Enum for the different Types we want to extract from a response.
- enum class `calcModule::PressureUnit` { **Pascal** , **Atmosphere** , **Psi** , **Bar** }
Enum class for different Pressure Units available.

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

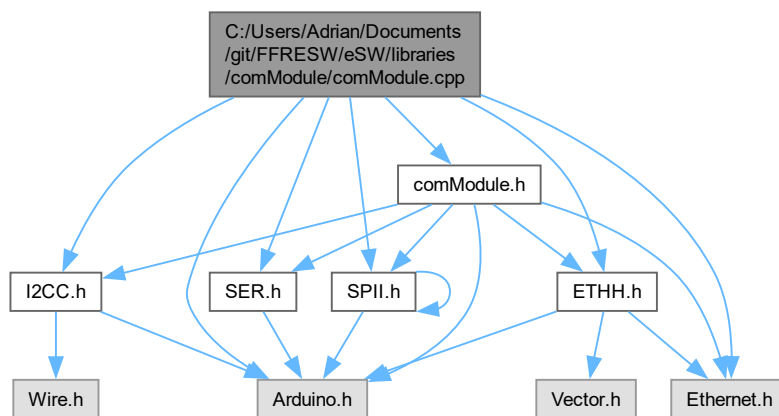
```

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

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

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

Include dependency graph for `comModule.cpp`:



7.4.1 Detailed Description

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

7.5 comModule.h

```
00001 #ifndef COMMODULE_H
00002 #define COMMODULE_H
00003
00004 #include <Arduino.h>
00005 #include <Ethernet.h>
00006 #include "ETHH.h"
00007 #include "I2CC.h"
00008 #include "SER.h"
00009 #include "SPII.h"
00010
00012 namespace comModule
00013 {
00015     class ComModuleInternals
00016     {
00017     public:
00018         ComModuleInternals();
00019         ~ComModuleInternals();
00020
00026         EthernetCommunication& getEthernet();
00027
00033         I2CCCommunication& getI2C();
```

```

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

```

7.6 ETHH.h

```

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

```

```

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

```

```
00329         bool ethernetInitialized = false;
00330         bool sendDataFlag = false;
00331
00332         String floatToIEEE754(float value);
00333
00334         Vector<float> parseResponse(String response);
00335     };
00336 }
00337
00338 #endif // ETHERNET_COMMUNICATION_H
00339
00340
```

7.7 I2CC.h

```
00001
00002 #ifndef I2C_COMMUNICATION_H
00003 #define I2C_COMMUNICATION_H
00004
00005 #include <Arduino.h>
00006 #include <Wire.h>
00007
00008 namespace comModule
00009 {
00010     class I2CCommunication
00011     {
00012     public:
00013         I2CCommunication();
00014         ~I2CCommunication();
00015
00016         void beginI2C(uint8_t address);
00017
00018         void beginI2CGlobal();
00019
00020         void endI2C();
00021
00022         void i2cWrite(uint8_t deviceAddress, uint8_t* data, size_t length);
00023
00024         size_t i2cRead(uint8_t deviceAddress, uint8_t* buffer, size_t length);
00025
00026         bool isInitialized() const;
00027
00028     private:
00029         bool i2cInitialized = false;
00030     };
00031 }
00032
00033 #endif // I2C_COMMUNICATION_H
```

7.8 SER.h

```
00001
00002 #ifndef SERIAL_COMMUNICATION_H
00003 #define SERIAL_COMMUNICATION_H
00004
00005 #include <Arduino.h>
00006
00007 namespace comModule
00008 {
00009     class SerialCommunication
00010     {
00011     public:
00012         SerialCommunication();
00013         ~SerialCommunication();
00014
00015         void beginSerial(long baudRate);
00016
00017         void endSerial();
00018
00019         void sendSerialData(const char* data);
00020
00021         void receiveSerialData(char* buffer, size_t length);
00022
00023         bool isInitialized() const;
00024
00025     private:
00026         bool serInitialized = false;
00027     };
00028 }
```



```

00060     };
00061 }
00062
00063 #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 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h File Reference

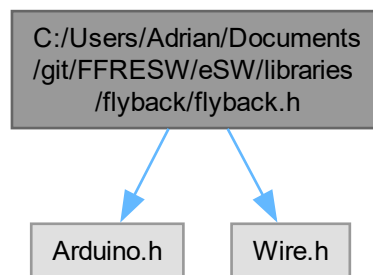
Header for the flyback class.

```

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

```

Include dependency graph for flyback.h:



Classes

- struct [flybackModule::Measurement](#)
Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system.
- class [flybackModule::Flyback](#)
Flyback class to manage the Flyback system This class provides methods for initializing the system, configuring the timer, measuring parameters, and handling different system states such as ON, OFF, HAND, and REMOTE modes.

Namespaces

- namespace [flybackModule](#)
Namespace for the Flyback module.

Typedefs

- typedef struct [flybackModule::Measurement](#) **flybackModule::meas**

Enumerations

- enum class `flybackModule::MainSwitchStates` : int { `Main_Switch_OFF` , `Main_Switch_MANUAL` , `Main_Switch_REMOTE` , `Main_switch_INVALID` }
enum for different SwitchStates of HVModule
- enum class `flybackModule::HVSwitchStates` : int { `HV_Module_OFF` , `HV_Module_ON` }
enum for SwitchStates of HV_Module on/off
- enum class `HVModule` : int { `powerSupply_OFF` , `powerSupply_ON` }

7.11.1 Detailed Description

Header for the flyback class.

Author

Domin

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.12 flyback.h

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

```
00001
00010 #ifndef FLYBACK_H
00011 #define FLYBACK_H
00012
00013 #include <Arduino.h>
00014 #include <Wire.h>
00015
00017 namespace flybackModule
00018 {
00020     enum class MainSwitchStates : int
00021     {
00022         Main_Switch_OFF,
00023         Main_Switch_MANUAL,
00024         Main_Switch_REMOTE,
00025         Main_switch_INVALID
00026     };
00027
00029     enum class HVSwitchStates : int
00030     {
00031         HV_Module_OFF,
00032         HV_Module_ON
00033     };
00034
00036     enum class HVModule : int
00037     {
00038         powerSupply_OFF,
00039         powerSupply_ON
00040     };
00041
```

```

00044     typedef struct Measurement
00045     {
00046         float voltage;
00047         float current;
00048         float power;
00049         int digitalFreqValue;
00050         int digitalDutyValue;
00051         int dutyCycle;
00052         uint32_t frequency;
00053     } meas;
00054
00055     class Flyback
00056     {
00057     public:
00058         Flyback();
00059         ~Flyback();
00060
00061         void initialize();
00062
00063         void deinitialize();
00064
00065         bool isInitialized() const;
00066
00067         bool getTimerState();
00068
00069         void setTimerState(bool state);
00070
00071         MainSwitchStates getMainSwitchState();
00072
00073         HVSwitchStates getHVSwitchState();
00074
00075         HVModule getHVState();
00076
00077         Measurement measure();
00078
00079         void run();
00080
00081         void setExternFrequency(uint32_t frequency);
00082
00083         uint32_t getExternFrequency();
00084
00085         void setExternDutyCycle(int dutyCycle);
00086
00087         int getExternDutyCycle();
00088
00089         void setExternPSU(int state);
00090
00091         int getExternPSU();
00092
00093         void regulateVoltage(float targetVoltage, float hysteresis);
00094
00095         void setTargetVoltage(float voltage);
00096
00097         float getTargetVoltage() const;
00098
00099         void setHysteresis(float hysteresis);
00100
00101         float getHysteresis() const;
00102
00103     private:
00104         Measurement meas;
00105
00106         //Define Pins -->Inputs
00107         static const int Main_Switch_OFF = 27;
00108         static const int Main_Switch_MANUAL = 28;
00109         static const int Main_Switch_REMOTE = 29;
00110         static const int HV_Module_ON = 37;
00111         static const int Measure_ADC = A0;
00112         static const int PWM_Frequency = A1;
00113         static const int PWM_DutyCycle = A2;
00114
00115         //Define Pins -> Outputs
00116         static const int PWM_OUT = 11;
00117         static const int PWM_INV = 12;
00118         static const int PSU = 36; //Power Supply ON OFF
00119         static const int HV_Module_Working = 35; //Signal LED
00120
00121         //Variables for calculating HV
00122         const float R1 = 100000000;
00123         const float R2 = 10000;
00124         const float ADC_Max_Value = 1023.0;
00125         const float Vcc = 5.0;
00126
00127         bool _flybackInitialized;
00128         bool _timerInitialized;
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
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
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234

```

```

00235         // States
00236         static MainSwitchStates lastState;
00237         static bool lastTimerState;
00238         static int lastPWMPFrequency;
00239         static int lastPWMDutyCycle;
00240         int currentPsuState = -1;
00241
00242         // Default targetVoltage and hysteresisVoltage for to prevent swinging
00243         float _targetVoltage = 0.0f;
00244         float _hysteresis = 0.0f;
00245
00246         // Rate limiting
00247         unsigned long _lastRegulationTime = 0;
00248         const unsigned long _regulationInterval = 100;
00249
00250         // PID vars
00251         float _integral = 0.0f;
00252         float _lastError = 0.0f;
00253         const float _Kp = 1.5f;
00254         const float _Ki = 0.05f;
00255         const float _Kd = 0.1f;
00256
00257         float _integralMin = -10.0f;
00258         float _integralMax = 10.0f;
00259
00260         // Soft start
00261         bool _softStartActive = true;
00262         int _softStartDuty = 1;
00263
00264         void timerConfig();
00265
00266         void setPWMPFrequency(uint32_t frequency, int dutyCycle);
00267
00268         void setHVandPSU(int hvLedState, int powerSupplyState);
00269
00270         void handleState(MainSwitchStates state);
00271
00272         void handleOffState();
00273
00274         void handleManualState();
00275
00276         void handleRemoteState();
00277
00278         void handleInvalidState();
00279     };
00280 }
00281 #endif //FLYBACK_H

```

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

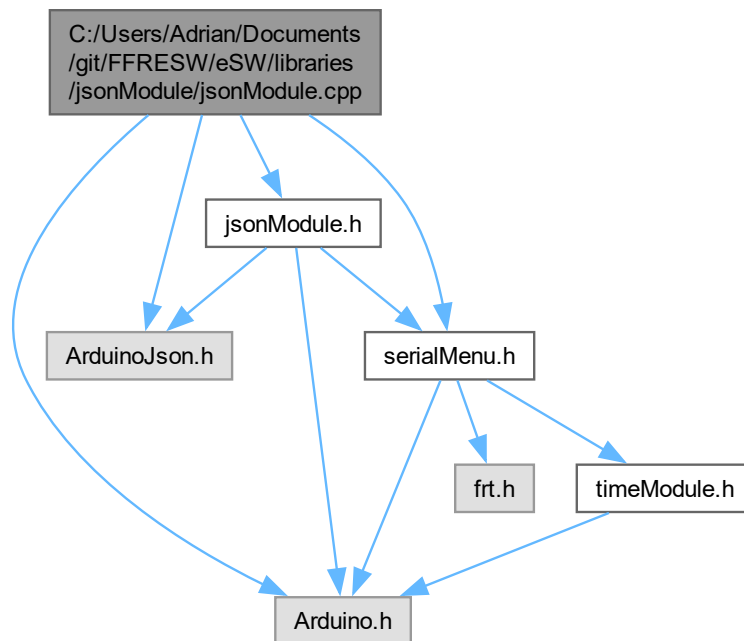
Implementation of the `jsonModule` class.

```

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

```

Include dependency graph for jsonModule.cpp:



7.13.1 Detailed Description

Implementation of the `jsonModule` class.

Version

0.2

Date

2025-05-18

Copyright

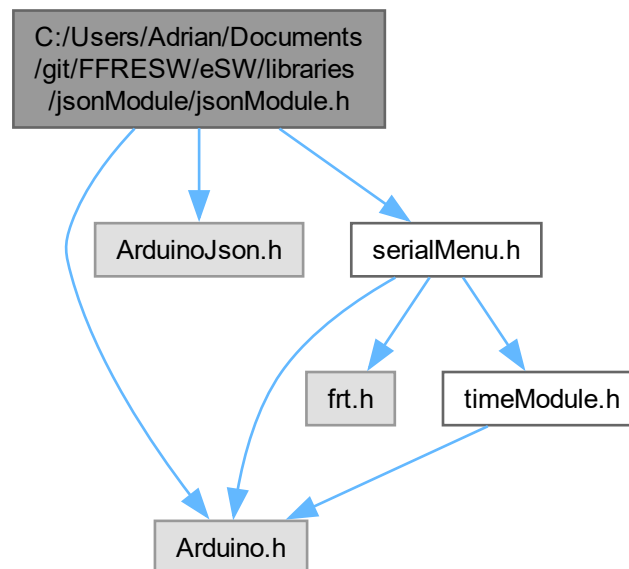
Copyright (c) 2025

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

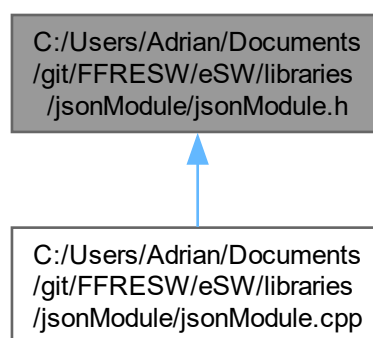
Header for the `JsonModuleInternals` class.

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

Include dependency graph for jsonModule.h:



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



Classes

- class `jsonModule::JsonModuleInternals`
Class for the JSON module internals.

Namespaces

- namespace [jsonModule](#)
Namespace for the JSON module.

7.14.1 Detailed Description

Header for the JsonModuleInternals class.

Author

Adrian Goessl

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.15 jsonModule.h

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

```

00001
00010 #ifndef JSONMODULE_H
00011 #define JSONMODULE_H
00012
00013 #include <Arduino.h>
00014 #include <ArduinoJson.h>
00015 #include <serialMenu.h>
00016
00017 namespace jsonModule
00018 {
00019     class JsonModuleInternals
00020     {
00021     public:
00022         JsonModuleInternals();
00023         ~JsonModuleInternals();
00024
00025         template<typename T>
00026         void createJson(const char* key, T value)
00027         {
00028             if (!jsonDoc[key].set(value))
00029             {
00030                 SerialMenu::printToSerial(SerialMenu::OutputLevel::ERROR, F("Failed to set JSON
00031 key."));
00032             }
00033         }
00034
00035         void sendJsonSerial();
00036
00037         String getJsonString() const;
00038
00039         void clearJson();
00040
00041         void printJsonDocMemory();
00042
00043         bool hasCapacityFor(size_t additionalSize) const;
00044
00045         size_t jsonBufferSize;
00046
00047     private:
00048         StaticJsonDocument<512> jsonDoc;
00049     };
00050 }
00051
00052 #endif // JSONMODULE_H

```


7.16 lockerBase.h

```

00001 #ifndef LOCKER_BASE_H
00002 #define LOCKER_BASE_H
00003
00004 #include <frt.h>
00005 #include <Arduino.h>
00006 #include <scopedLock.h>
00007 #include <logManager.h>
00008
00009 class LockerBase
00010 {
00011 public:
00012     LockerBase()
00013     {
00014         if (_logger->isSDCardInitialized())
00015         {
00016             _logger->setLogFileName("log_LockerBase.txt");
00017         }
00018     }
00019
00020     ~LockerBase() {}
00021
00022     locker::ScopedLock lockEthernetScoped()
00023     {
00024         ethernetConnected = true;
00025         logState("Ethernet connected", ethernetConnected);
00026         return locker::ScopedLock(ethernetMutex);
00027     }
00028
00029     locker::ScopedLock lockTemperatureScoped()
00030     {
00031         temperatureReading = true;
00032         logState("Temperature reading", temperatureReading);
00033         return locker::ScopedLock(temperaturQueueMutex);
00034     }
00035
00036     locker::ScopedLock lockSerialScoped()
00037     {
00038         serialReading = true;
00039         logState("Serial reading", serialReading);
00040         return locker::ScopedLock(serialMutex);
00041     }
00042
00043 private:
00044     // Mutexes for the different resources
00045     frt::Mutex temperaturQueueMutex;
00046     frt::Mutex ethernetMutex;
00047     frt::Mutex serialMutex;
00048
00049     // Semaphores for the different resources
00050     frt::Semaphore ethernetSemaphore;
00051     frt::Semaphore temperatureSemaphore;
00052     frt::Semaphore serialSemaphore;
00053
00054     // Flags for the different resources
00055     bool ethernetConnected = false;
00056     bool temperatureReading = false;
00057     bool serialReading = false;
00058
00059     // Logger instance
00060     LogManager* _logger = LogManager::getInstance();
00061
00062     void logState(const char* label, bool state)
00063     {
00064         char buffer[64];
00065         snprintf(buffer, sizeof(buffer), "[LockerBase] %s: %s", label, state ? "true" : "false");
00066         _logger->writeToLogFile(buffer);
00067     }
00068 };
00069
00070 #endif // LOCKER_BASE_H

```

7.17 scopedLock.h

```

00001 #ifndef LOCKER_BASE_SCOPEDLOCK_H
00002 #define LOCKER_BASE_SCOPEDLOCK_H
00003
00004 #include <frt.h>
00005
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

```

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

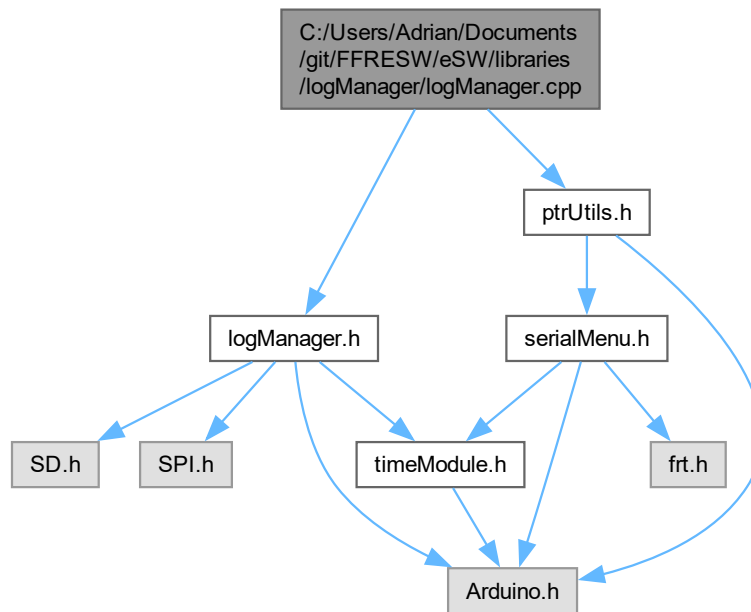
Implementation of the logManager class.

```

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

```

Include dependency graph for logManager.cpp:



Variables

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

7.18.1 Detailed Description

Implementation of the logManager class.

Version

0.1

Date

2024-01-26

Copyright

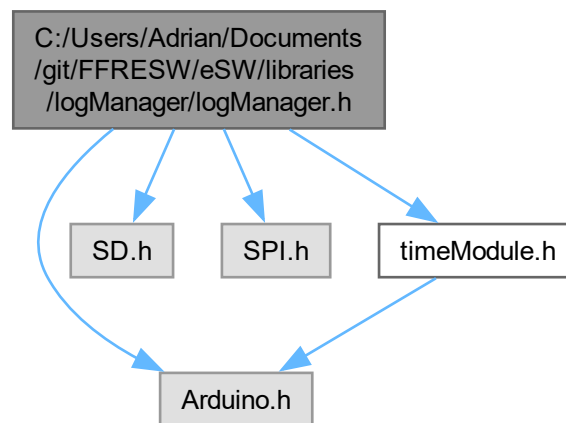
Copyright (c) 2024

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

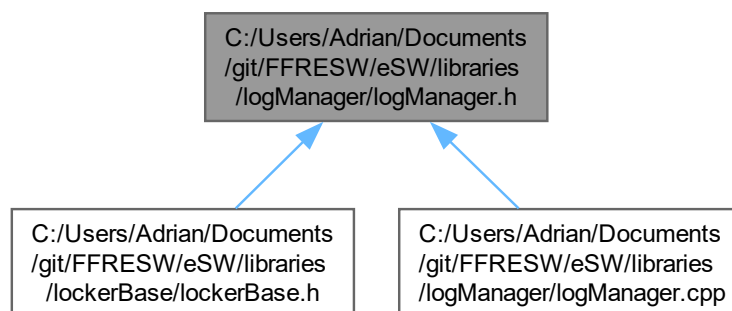
header file for the logManager.

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

Include dependency graph for logManager.h:



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



Classes

- class [LogManager](#)

7.19.1 Detailed Description

header file for the logManager.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.20 logManager.h

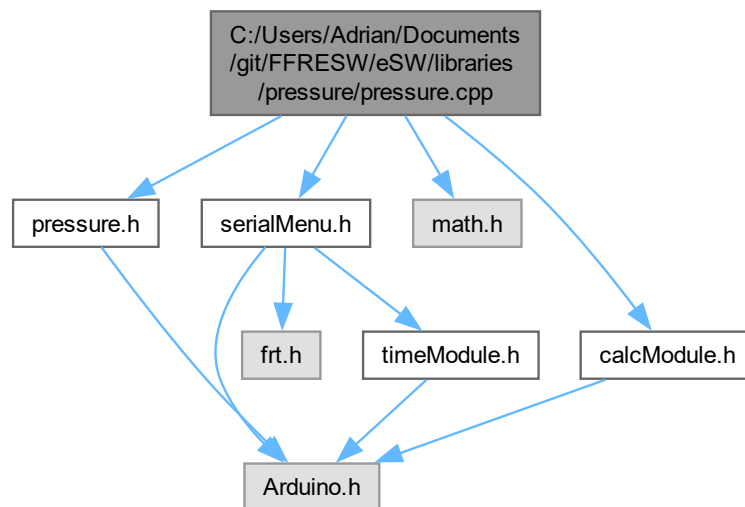
[Go to the documentation of this file.](#)

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

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

Implementation of the pressure class.

```
#include "pressure.h"  
#include <serialMenu.h>  
#include <math.h>  
#include <calcModule.h>  
Include dependency graph for pressure.cpp:
```



7.21.1 Detailed Description

Implementation of the pressure class.

Version

0.1

Date

2024-01-26

Copyright

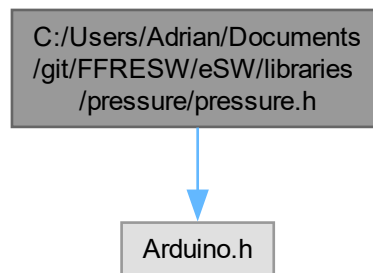
Copyright (c) 2024

7.22 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h File Reference

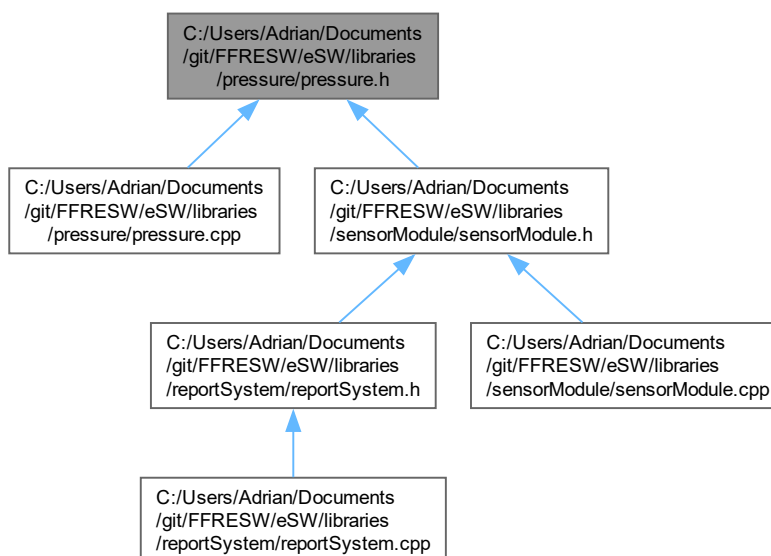
Header file for the pressure library.

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

Include dependency graph for pressure.h:



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



Classes

- class [PressureSensor](#)
Pressure sensor class.

7.22.1 Detailed Description

Header file for the pressure library.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.23 pressure.h

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

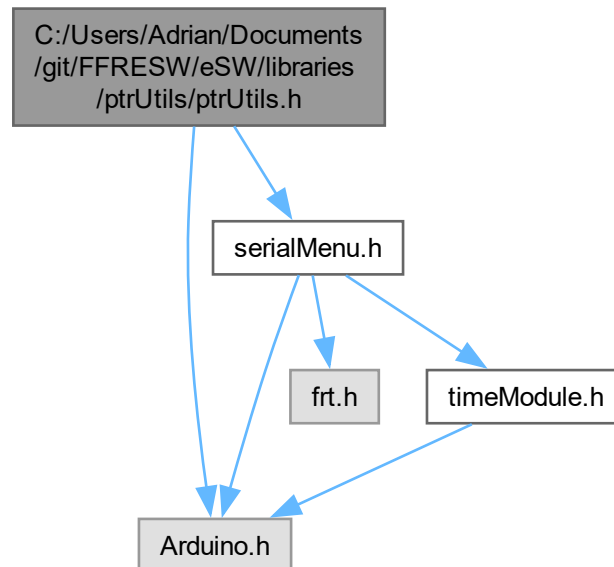
```
00001
00011 #ifndef PRESSURESENSOR_H
00012 #define PRESSURESENSOR_H
00013
00014 #include <Arduino.h>
00015
00017 class PressureSensor
00018 {
00019 public:
00020     PressureSensor();
00021     ~PressureSensor();
00022
00027     void initialize();
00028
00034     float readPressure();
00035
00042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

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

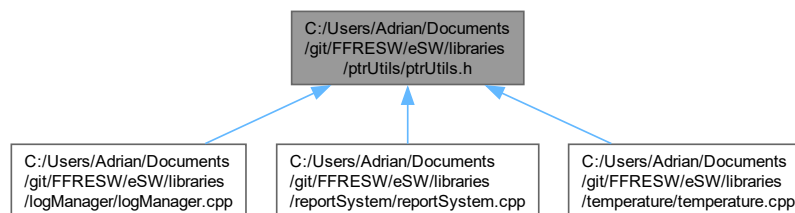
Header only library implementation.


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

Include dependency graph for ptrUtils.h:



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



Classes

- class `PtrUtils`
Utility class for pointer operations.
- class `ScopedPointer< T >`
Template class for a Scoped Pointer.
- class `PointerWrapper< T >`
Tempalte class for wrapping a pointer.
- struct `Error`

Macros

- #define `tryDeletePtr(ptr)`
Macro to safely delete a pointer and verify it is nullptr.
- #define `TRACE_DEPTH` 4
- #define `UNIQUE_NAME(base)` `__UNIQUE_NAME(base, __LINE__)`
Various macros for errorhandling.
- #define `__UNIQUE_NAME(base, line)` `base##line`
- #define `TRY`
- #define `CATCH` else if (`__SAFE_ERR_PTR__` && `__SAFE_ERR_PTR__->hasError()`)
- #define `FINALLY` else
- #define `THROW(code, msg)`
- #define `THROW_SOFT(code, msg)`
- #define `RETHROW(code, msg)`
- #define `THROW_IF_ERR(err)`
- #define `THROW_IF_NULL(ptr, code, msg)`
- #define `THROW_IF_FALSE(cond, code, msg)`
- #define `RETURN_IF_ERR(expr)`

Enumerations

- enum `ErrorSeverity` { `ERROR_NONE` = 0 , `ERROR_SOFT` = 1 , `ERROR_FATAL` = 2 }
- Enum to declare the severity of the error.*

7.24.1 Detailed Description

Header only library implementation.

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.24.2 Macro Definition Documentation

7.24.2.1 RETHROW

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

Value:

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

7.24.2.2 RETURN_IF_ERR

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

Value:

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

7.24.2.3 THROW

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

Value:

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

7.24.2.4 THROW_IF_ERR

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

Value:

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

7.24.2.5 THROW_IF_FALSE

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

Value:

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

7.24.2.6 THROW_IF_NULL

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

Value:

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

7.24.2.7 THROW_SOFT

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

Value:

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

7.24.2.8 TRY

```
#define TRY
```

Value:

```
for (Error UNIQUE_NAME(_safetry_err_) = {}, *prev = __SAFE_ERR_PTR__, *err =
    &UNIQUE_NAME(_safetry_err_); \
    __SAFE_ERR_PTR__ = err, err != nullptr; \
    __SAFE_ERR_PTR__ = prev, err = nullptr)
```

7.24.2.9 tryDeletePtr

```
#define tryDeletePtr(
    ptr )
```

Value:

```
if (PtrUtils::IsValidPtr(ptr))
    SafeDelete(ptr);
```

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

7.25 ptrUtils.h

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

```

00001
00011 #ifndef PTRUTILS_H
00012 #define PTRUTILS_H
00013
00014 #include <Arduino.h>
00015 #include <serialMenu.h>
00016
00023 template <typename T>
00024 static inline void SafeDelete(T*& ptr)
00025 {
00026     if (ptr != nullptr)
00027     {
00028         delete ptr;
00029         ptr = nullptr;
00030     }
00031 }
00032
00039 template <typename T>
00040 static inline void SafeDeleteArray(T*& ptr)
00041 {
00042     if (ptr != nullptr)
00043     {
00044         delete[] ptr;
00045         ptr = nullptr;
00046     }
00047 }
00048
00057 template <typename T>
00058 static inline void Verify(const T& value, const T& expected, const char* errorMsg = nullptr)
00059 {
00060     if (value != expected)
00061     {
00062         if (errorMsg)
00063         {
00064             SerialMenu::printToSerial(errorMsg);
00065         }
00066         else
00067         {
00068             String errStr;
00069             errStr += "[ERROR] Verification failed: Value ";
00070             errStr += value;
00071             errStr += ") does not match expected ";
00072             errStr += expected;
00073             errStr += ").";
00074             SerialMenu::printToSerial(errStr);
00075         }
00076         while (true); // Halt execution
00077     }
00078 }
00079
00088 template <typename T>
00089 static inline void Verify(T* value, T* expected, const char* errorMsg = nullptr)
00090 {
00091     if (value != expected)
00092     {
00093         if (errorMsg)
00094         {
00095             SerialMenu::printToSerial(errorMsg);
00096         }
00097         else
00098         {
00099             String errStr;
00100             errStr += "[ERROR] Verification failed: Pointer ";
00101             errStr += (unsigned long)value, HEX;
00102             errStr += ") does not match expected pointer ";
00103             errStr += (unsigned long)expected, HEX;
00104             errStr += ").";
00105             SerialMenu::printToSerial(errStr);
00106         }
00107         while (true); // Halt execution
00108     }
00109 }
00110
00118 template <typename T>
00119 static inline void Verify(T* value, const char* errorMsg = nullptr)
00120 {
00121     if (value != nullptr) // Directly compare with nullptr (no std::nullptr_t)
00122     {
00123         if (errorMsg)
00124         {
00125             SerialMenu::printToSerial(errorMsg);
00126         }
00127     }

```

```

00127         else
00128         {
00129             String errStr;
00130             errStr += "[ERROR] Verification failed: Pointer (";
00131             errStr += (unsigned long)value, HEX;
00132             errStr += ") is not null.";
00133             SerialMenu::printToSerial(errStr);
00134         }
00135         while (true); // Halt execution
00136     }
00137 }
00138
00139
00140 #define tryDeletePtr(ptr) \
00141     if (PtrUtils::IsValidPtr(ptr)) \
00142         SafeDelete(ptr); \
00143
00144
00145 class PtrUtils
00146 {
00147 public:
00148     template <typename T>
00149     static inline bool IsNullPtr(T* ptr)
00150     {
00151         return ptr == nullptr;
00152     }
00153
00154     template <typename T>
00155     static inline bool IsValidPtr(T* ptr)
00156     {
00157         return ptr != nullptr;
00158     }
00159 };
00160
00161 template <typename T>
00162 static inline void ClearArray(T* array, size_t size)
00163 {
00164     for (size_t i = 0; i < size; ++i)
00165     {
00166         array[i] = T();
00167     }
00168 }
00169
00170 template <typename T>
00171 static inline void PrintPtrInfo(T* ptr, const char* ptrName = "Pointer")
00172 {
00173     if (ptr == nullptr)
00174     {
00175         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String("is nullptr"));
00176     }
00177     else
00178     {
00179         SerialMenu::printToSerial("[INFO] " + String(ptrName) + String(" points to address: 0x") +
00180             (uintptr_t)ptr, HEX);
00181     }
00182 }
00183
00184 template <typename T>
00185 class ScopedPointer
00186 {
00187 private:
00188     T* ptr;
00189 public:
00190     explicit ScopedPointer(T* p = nullptr) : ptr(p) {}
00191     ~ScopedPointer() { SafeDelete(ptr); }
00192
00193     T* get() const { return ptr; }
00194
00195     T* release()
00196     {
00197         T* temp = ptr;
00198         ptr = nullptr;
00199         return temp;
00200     }
00201
00202     void reset(T* p = nullptr)
00203     {
00204         SafeDelete(ptr);
00205         ptr = p;
00206     }
00207
00208     T& operator*() const { return *ptr; }
00209
00210     T* operator->() const { return ptr; }
00211 };

```

```

00274
00280 template <typename T>
00281 class PointerWrapper
00282 {
00283 private:
00284     T* ptr;
00285
00286 public:
00287     explicit PointerWrapper(T* p = nullptr) : ptr(p) {}
00288     ~PointerWrapper() { SafeDelete(ptr); }
00289
00295     T* get() const { return ptr; }
00296
00302     T* release()
00303     {
00304         T* temp = ptr;
00305         ptr = nullptr;
00306         return temp;
00307     }
00308
00314     void reset(T* p = nullptr)
00315     {
00316         SafeDelete(ptr);
00317         ptr = p;
00318     }
00319
00325     T& operator*() { return *ptr; }
00326
00332     T* operator->() { return ptr; }
00333 };
00334
00335 #define TRACE_DEPTH 4
00336
00338 enum ErrorSeverity
00339 {
00340     ERROR_NONE = 0,
00341     ERROR_SOFT = 1,
00342     ERROR_FATAL = 2
00343 };
00344
00346 struct Error
00347 {
00348     int code;
00349     ErrorSeverity severity;
00350     const char* msg;
00351     const char* traceFiles[TRACE_DEPTH];
00352     int traceLines[TRACE_DEPTH];
00353     int traceDepth;
00354
00355     Error()
00356         : code(0)
00357         , severity(ERROR_NONE)
00358         , msg("")
00359         , traceDepth(0)
00360     {
00361     }
00362
00363     Error(int c, const char* m, ErrorSeverity s, const char* file, int line)
00364         : code(c)
00365         , severity(s)
00366         , msg(m)
00367         , traceDepth(0)
00368     {
00369         addTrace(file, line);
00370     }
00371
00372     void addTrace(const char* file, int line)
00373     {
00374         if (traceDepth < TRACE_DEPTH)
00375         {
00376             traceFiles[traceDepth] = file;
00377             traceLines[traceDepth] = line;
00378             traceDepth++;
00379         }
00380     }
00381
00382     bool hasError() const
00383     {
00384         return code != 0;
00385     }
00386
00387     bool isFatal() const
00388     {
00389         return severity == ERROR_FATAL;
00390     }
00391
00392

```

```

00393     void log() const
00394     {
00395         Serial.print(F("[ERROR] "));
00396         Serial.print(code);
00397         Serial.print(F(": "));
00398         Serial.print(msg);
00399         Serial.print(F(" | Severity: "));
00400         Serial.println(severity == ERROR_FATAL ? F("FATAL") : F("SOFT"));
00401
00402         for (int i = 0; i < traceDepth; ++i)
00403         {
00404             Serial.print(F(" -> at "));
00405             Serial.print(traceFiles[i]);
00406             Serial.print(F(":"));
00407             Serial.println(traceLines[i]);
00408         }
00409     }
00410 };
00411
00415 #define UNIQUE_NAME(base) __UNIQUE_NAME(base, __LINE__)
00416 #define __UNIQUE_NAME(base, line) base##line
00417
00418 static Error* __SAFE_ERR_PTR__ = nullptr;
00419
00420 #define TRY
00421 \
00422     for (Error UNIQUE_NAME(_safetry_err_) = {}, *prev = __SAFE_ERR_PTR__, *err =
00423     &UNIQUE_NAME(_safetry_err_); \
00424     __SAFE_ERR_PTR__ = err, err != nullptr;
00425 \
00426     __SAFE_ERR_PTR__ = prev, err = nullptr)
00427
00428 #define CATCH \
00429     else if (__SAFE_ERR_PTR__ && __SAFE_ERR_PTR__->hasError())
00430
00431 #define FINALLY \
00432     else
00433
00434 #define THROW(code, msg)
00435     do {
00436         if (__SAFE_ERR_PTR__)
00437         {
00438             *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_FATAL, __FILE__, __LINE__);
00439             break;
00440         }
00441     } while (0)
00442
00443 #define THROW_SOFT(code, msg)
00444     do {
00445         if (__SAFE_ERR_PTR__)
00446         {
00447             *__SAFE_ERR_PTR__ = Error(code, msg, ERROR_SOFT, __FILE__, __LINE__);
00448             break;
00449         }
00450     } while (0)
00451
00452 #define RETHROW(code, msg)
00453     do {
00454         if (__SAFE_ERR_PTR__)
00455         {
00456             __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__);
00457             __SAFE_ERR_PTR__->code = code;
00458             __SAFE_ERR_PTR__->msg = msg;
00459             break;
00460         }
00461     } while (0)
00462
00463 #define THROW_IF_ERR(err)
00464     do {
00465         if ((err).hasError() && __SAFE_ERR_PTR__)
00466         {
00467             *__SAFE_ERR_PTR__ = err;
00468             __SAFE_ERR_PTR__->addTrace(__FILE__, __LINE__);
00469             break;
00470         }
00471     } while (0)
00472
00473 #define THROW_IF_NULL(ptr, code, msg)
00474     do {
00475         if ((ptr) == nullptr) { THROW(code, msg); }
00476     } while (0)
00477
00478 #define THROW_IF_FALSE(cond, code, msg)
00479     do {
00480         if (!(cond)) { THROW(code, msg); }
00481     } while (0)

```

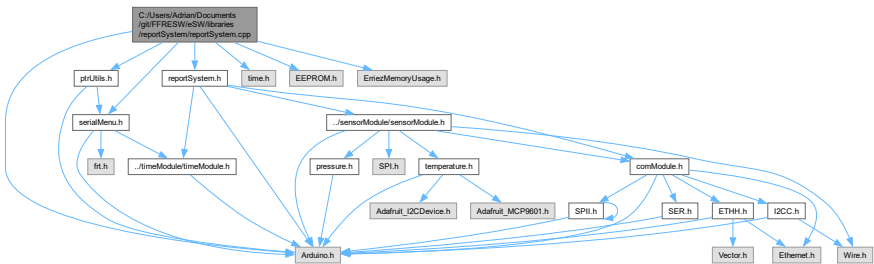


```
00480 #define RETURN_IF_ERR(expr)           \
00481     do {                               \
00482         Error _e = (expr);             \
00483         if (_e.hasError()) return _e;   \
00484     } while (0)                         \
00485                                         \
00486                                         \
00487 #endif // PTRUTILS_H
```

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

Unified system health and error reporting module.

```
#include "reportSystem.h"
#include "ptrUtils.h"
#include <Arduino.h>
#include <time.h>
#include <EEPROM.h>
#include <ErriezMemoryUsage.h>
#include <serialMenu.h>
Include dependency graph for reportSystem.cpp:
```



Functions

- volatile uint16_t stackCheck **__attribute__**__((section(".noinit")))

7.26.1 Detailed Description

Unified system health and error reporting module.

Author

Adrian Goessl

Version

0.3

Date

2024-09-28

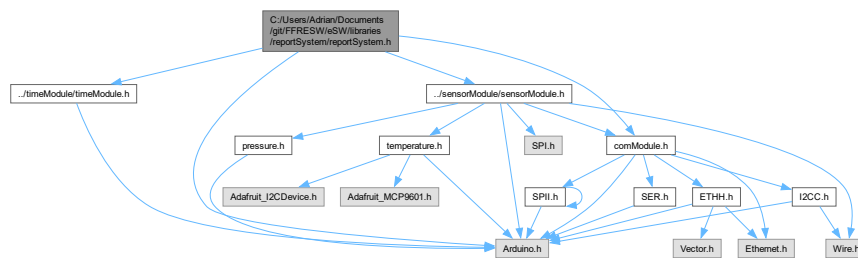
Copyright

Copyright (c) 2024

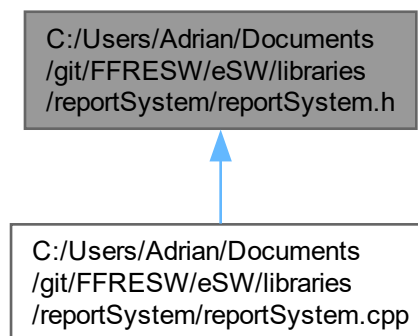
7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h File Reference

Header file for the ReportSystem.

```
#include <Arduino.h>
#include "../sensorModule/sensorModule.h"
#include "../comModule/comModule.h"
#include "../timeModule/timeModule.h"
Include dependency graph for reportSystem.h:
```



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



Classes

- class [reportSystem::ReportSystem](#)
Class for the report system.

Namespaces

- namespace [reportSystem](#)
Namespace for the report system.

Macros

- `#define STACK_GUARD 0xDEAD`
- `#define EEPROM_ERROR_ADDR 0`

Variables

- `volatile uint16_t stackCheck`

7.27.1 Detailed Description

Header file for the ReportSystem.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.28 reportSystem.h

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

```
00001
00011 #ifndef REPORTSYSTEM_H
00012 #define REPORTSYSTEM_H
00013
00014 #include <Arduino.h>
00015 #include "../sensorModule/sensorModule.h"
00016 #include "../comModule/comModule.h"
00017 #include "../timeModule/timeModule.h"
00018
00019 #define STACK_GUARD 0xDEAD // Stack guard value
00020 extern volatile uint16_t stackCheck; // Stack check variable
00021
00022 #define EEPROM_ERROR_ADDR 0
00023
00025 namespace reportSystem
00026 {
00028     class ReportSystem
00029     {
00030     public:
00031         ReportSystem();
00032         ~ReportSystem();
00033
00039         void reportError(const char* errorMessage);
00040
00053         bool checkSystemHealth(size_t memoryThreshold, bool checkEth,
00054                               bool checkSpi, bool checkI2c,
00055                               bool checkTemp, bool checkPress);
00056
```

```

00063     bool reportStatus(bool active);
00064
00071     void setThreshold(float tempThreshold, float pressureThreshold);
00072
00081     bool checkThresholds(float currentTemp, float currentPressure);
00082
00088     String getCurrentTime();
00089
00095     String getMemoryStatus();
00096
00102     String getStackDump();
00103
00108     void startBusyTime();
00109
00114     void startIdleTime();
00115
00121     float getCPULoad();
00122
00127     void resetUsage();
00128
00133     static void initStackGuard();
00134
00141     static bool detectStackOverflow();
00142
00149     void saveLastError(const char* error);
00150
00156     String getLastError();
00157
00163     bool getLastErrorInfo();
00164
00174     bool checkRamLevel(unsigned int warningThreshold, unsigned int criticalThreshold);
00175
00181     bool isTemperatureSensorOK() const;
00182
00188     bool isCommunicationOK() const;
00189
00195     bool isMemoryOK() const;
00196
00202     bool isRamOK() const;
00203
00209     bool isStackSafe() const;
00210
00216     bool hasNoSavedErrors() const;
00217
00218 private:
00219     float tempThreshold;
00220     float pressureThreshold;
00221     unsigned long lastHealthCheck;
00222     const unsigned long healthCheckInterval = 10000; // 10 seconds
00223     unsigned long busyTime = 0;
00224     unsigned long idleTime = 0;
00225     unsigned long lastTimestamp = 0;
00226     bool lastHealthCheckPassed = true;
00227     bool firstHealthCheckDone = false;
00228
00237     bool checkSensors(bool checkTemp, bool checkPress);
00238
00248     bool checkCommunication(bool checkEth, bool checkSpi, bool checkI2c);
00249
00257     bool checkMemory(unsigned int threshold);
00258
00259     sensorModule::SensorModuleInternals* _sens;
00260     comModule::ComModuleInternals* _com;
00261     timeModule::TimeModuleInternals* _time;
00262 };
00263 }
00264
00265 #endif // REPORTSYSTEM_H

```

7.29 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.cpp File Reference

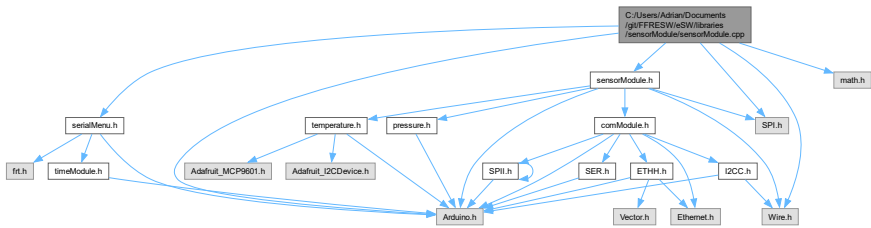
Implementation of the [sensorModule](#) class.

```

#include "sensorModule.h"
#include <math.h>
#include <Wire.h>
#include <SPI.h>

```

```
#include <Arduino.h>
#include <serialMenu.h>
Include dependency graph for sensorModule.cpp:
```



7.29.1 Detailed Description

Implementation of the [sensorModule](#) class.

Version

0.1

Date

2024-01-26

Copyright

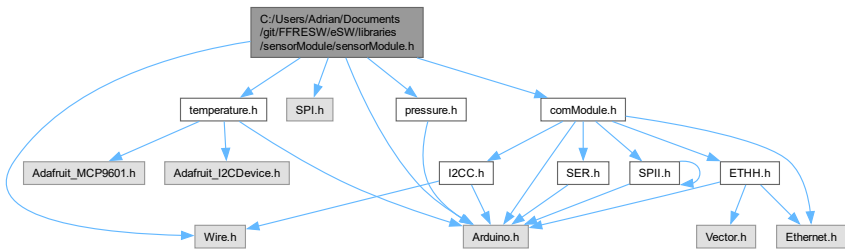
Copyright (c) 2024

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

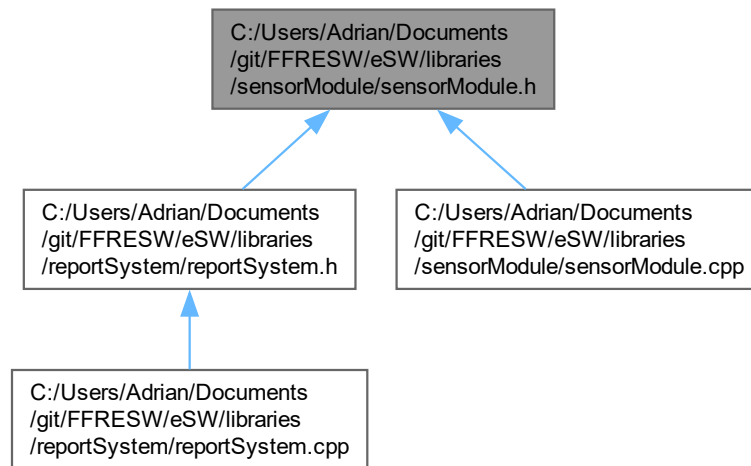
header file for the [sensorModule](#).

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

Include dependency graph for sensorModule.h:



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



Classes

- class [sensorModule::SensorModuleInternals](#)

Class for the sensor module internals.

Namespaces

- namespace [sensorModule](#)

Namespace for the sensor module.

Enumerations

- enum class [sensorModule::SensorType](#) {
TEMPERATURE , **OBJECTTEMPERATURE** , **AMBIENTTEMPERATURE** , **PRESSURE** ,
DHT11 , **MCP9601_Celsius_Indoor** , **MCP9601_Fahrenheit_Indoor** , **MCP9601_Kelvin_Indoor** ,
MCP9601_Celsius_Outdoor , **MCP9601_Fahrenheit_Outdoor** , **MCP9601_Kelvin_Outdoor** , **UNKNOWN**
}

Enum class for the sensor types.

7.30.1 Detailed Description

header file for the [sensorModule](#).

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.31 sensorModule.h

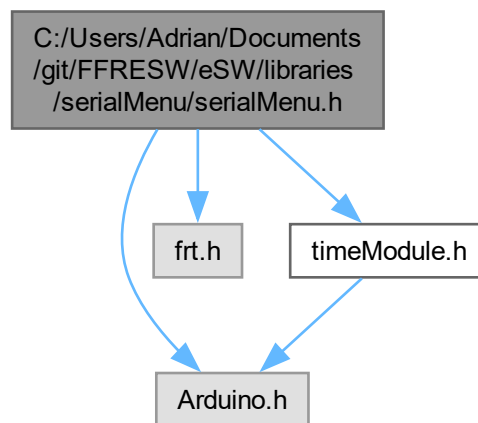
[Go to the documentation of this file.](#)

```
00001
00011 #ifndef SENSORMODULE_H
00012 #define SENSORMODULE_H
00013
00014 #include <Arduino.h>
00015 #include <Wire.h>
00016 #include <SPI.h>
00017 #include <pressure.h>
00018 #include <temperature.h>
00019 #include <comModule.h>
00020
00021
00023 namespace sensorModule
00024 {
00026     enum class SensorType
00027     {
00028         TEMPERATURE,
00029         OBJECTTEMPERATURE,
00030         AMBIENTTEMPERATURE,
00031         PRESSURE,
00032         DHT11,
00033         MCP9601_Celsius_Indoor,
00034         MCP9601_Fahrenheit_Indoor,
00035         MCP9601_Kelvin_Indoor,
00036         MCP9601_Celsius_Outdoor,
00037         MCP9601_Fahrenheit_Outdoor,
00038         MCP9601_Kelvin_Outdoor,
00039         UNKNOWN
00040     };
00041
00043     class SensorModuleInternals : public TemperatureSensor, public PressureSensor
00044     {
00045     public:
00046         SensorModuleInternals();
00047         ~SensorModuleInternals();
00048
00053         void initialize();
00054
00061         float readSensor(SensorType type);
00062
00070         bool calibrateSensor(SensorType type);
00071
00079         bool checkSensorStatus(SensorType type);
00080
00087         void reportUnknownSensorOnce(SensorType type, const __FlashStringHelper* context);
00088
00089     private:
00090         TemperatureSensor _temperatureSensor;
00091         PressureSensor _pressureSensor;
00092
00093         SensorType _lastUnknownSensorType;
00094         bool _unknownSensorReported;
00095     };
00096 }
00097
00098 #endif // SENSORMODULE_H
```

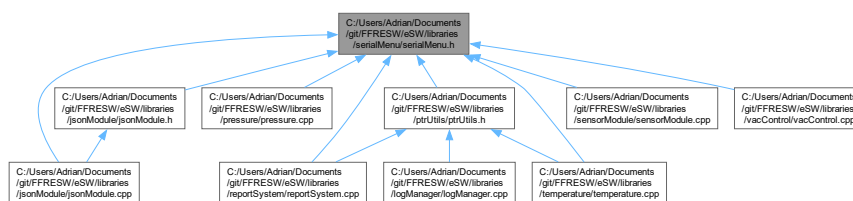
7.32 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h File Reference

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

```
#include <Arduino.h>
#include <frt.h>
#include <timeModule.h>
Include dependency graph for serialMenu.h:
```



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



Classes

- struct `MenuItem`
Serial menu structure.
- class `SerialMenu`
Class for the serial menu.

7.32.1 Detailed Description

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

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.33 serialMenu.h

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

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

```

00084
00091     static void printToSerial(const String& message, bool newLine = true, bool logMessage = false);
00092
00099     static void printToSerial(const __FlashStringHelper* message, bool newLine = true, bool logMessage
= false);
00100
00105     static String getCurrentTime();
00106
00107 private:
00108     MenuItem* currentMenu;
00109     size_t menuSize;
00110 };
00111
00112 #endif // SERIAL_MENU_H

```

7.34 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp File Reference

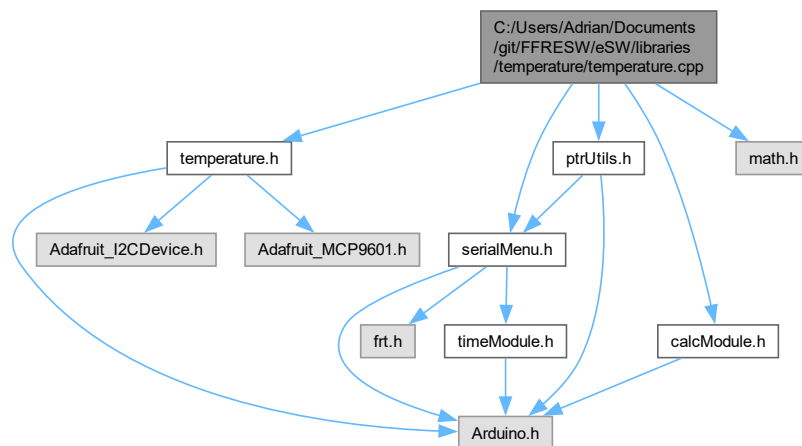
Implementation of the temperature class.

```

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

```

Include dependency graph for temperature.cpp:



7.34.1 Detailed Description

Implementation of the temperature class.

Version

0.1

Date

2024-01-26

Copyright

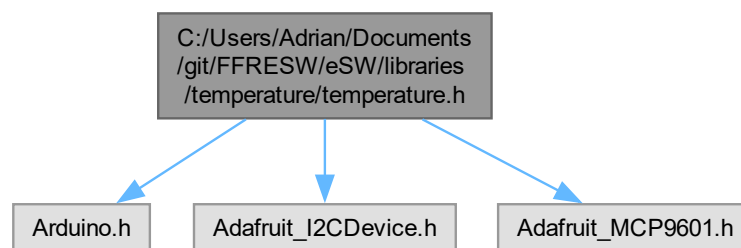
Copyright (c) 2024

7.35 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.h File Reference

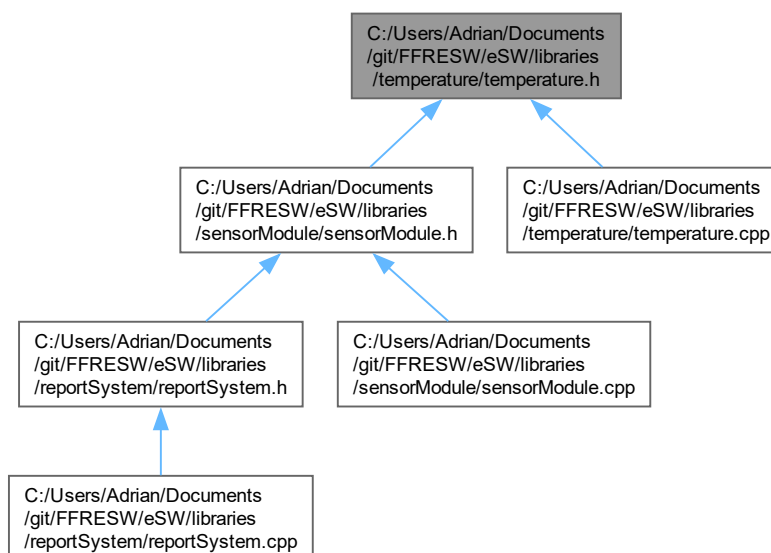
Header file for the temperature library.

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

Include dependency graph for temperature.h:



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



Classes

- class [TemperatureSensor](#)
Temperature sensor class.

Enumerations

- enum `Units` { `Celsius` , `Kelvin` , `Fahrenheit` }
Enum for different units used by methods as parameters.
- enum `MCP9601_Status` : `uint8_t` { `MCP9601_OPENCIRCUIT` = 0x10 , `MCP9601_SHORTCIRCUIT` = 0x20 }
Enum for the different status codes of the MCP9601 sensor.
- enum `SensorID` { `INDOOR` , `OUTDOOR` }

7.35.1 Detailed Description

Header file for the temperature library.

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.36 temperature.h

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

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

```

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

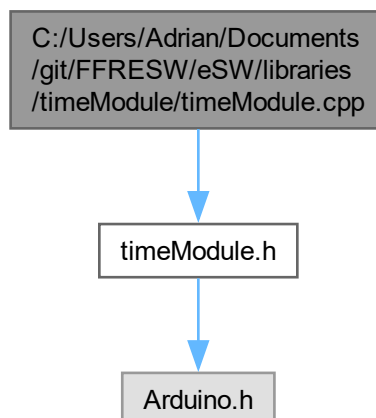
```

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

Implementation of the `timeModule` class.

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

Include dependency graph for `timeModule.cpp`:



7.37.1 Detailed Description

Implementation of the `timeModule` class.

Version

0.1

Date

2024-01-26

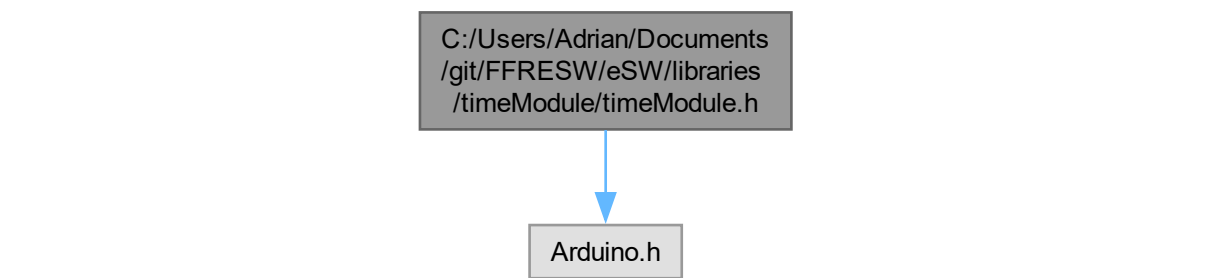
Copyright

Copyright (c) 2024

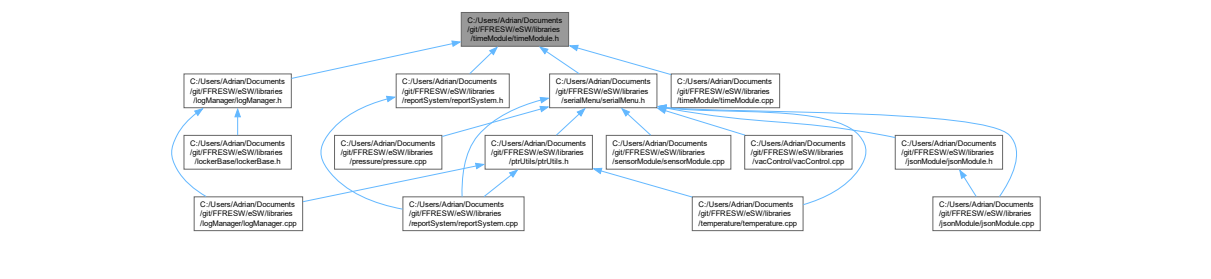
7.38 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h File Reference

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

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



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



Classes

- struct [timeModule::DateTimeStruct](#)
Struct to hold the date and time.
- class [timeModule::TimeModuleInternals](#)
Class to handle Systemtime.

Namespaces

- namespace [timeModule](#)
namespace for the [timeModule](#)

Typedefs

- typedef struct [timeModule::DateTimeStruct](#) **timeModule::DateTimeStruct**

7.38.1 Detailed Description

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

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.39 timeModule.h

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

```

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

```

7.40 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.cpp File Reference

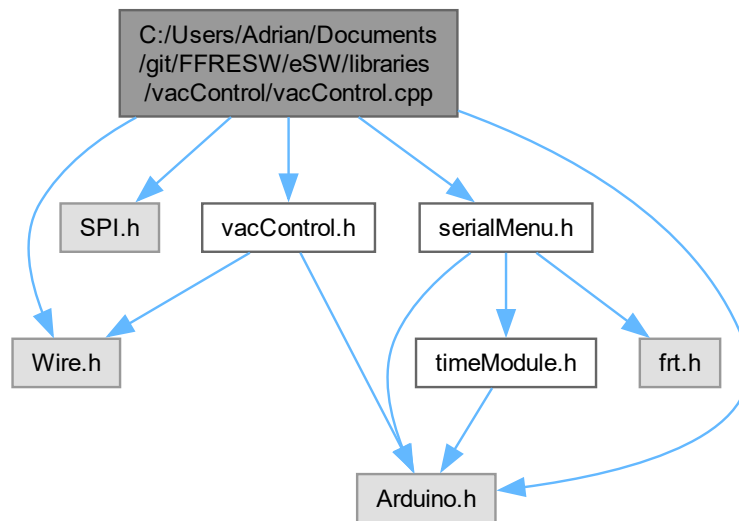
Implementation of the vacControl class.

```

#include <Wire.h>
#include <SPI.h>
#include <Arduino.h>
#include <vacControl.h>
#include <serialMenu.h>

```


Include dependency graph for vacControl.cpp:



7.40.1 Detailed Description

Implementation of the `vacControl` class.

Version

0.1

Date

2024-01-26

Copyright

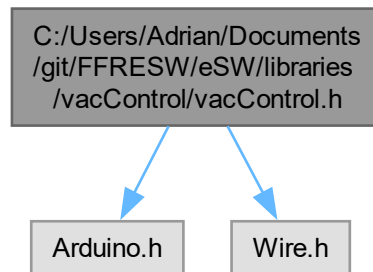
Copyright (c) 2024

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

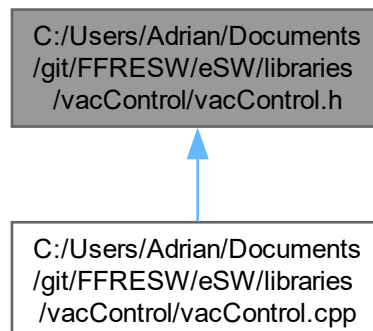
Header for the `vacControl` class.

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

Include dependency graph for vacControl.h:



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



Classes

- struct [vacControlModule::Pressure](#)
- class [vacControlModule::VacControl](#)

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

Namespaces

- namespace [vacControlModule](#)
Namespace for the [VacControl](#) module.

Typedefs

- typedef struct [vacControlModule::Pressure](#) [vacControlModule::meas](#)

Enumerations

- enum class [vacControlModule::MainSwitchStates](#) : int { **Main_Switch_OFF** , **Main_Switch_MANUAL** , **Main_Switch_REMOTE** , **Main_switch_INVALID** }
Enum to represent the states of the main switch and pump.
- enum class [vacControlModule::PumpSwitchStates](#) : int { **Switch_Pump_OFF** , **Switch_Pump_ON** }
Enum for extern Setup for pinMode for the PumpState.
- enum class [vacControlModule::PumpState](#) : int { **pump_OFF** , **pump_ON** }
Enum for extern Setup for pinMode for the PumpState.
- enum [vacControlModule::Scenarios](#) {
Scenario_1 = 0 , **Scenario_2** = 1 , **Scenario_3** = 2 , **Scenario_4** = 3 ,
Scenario_5 = 4 , **Invalid_Scenario** = -1 }
Enum to represent the different operating scenarios of the VacControl system.

7.41.1 Detailed Description

Header for the vacControl class.

Author

Domin

Version

0.2

Date

2025-05-18

Copyright

Copyright (c) 2025

7.42 vacControl.h

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

```

00001
00010 #ifndef VACCONTROL_H
00011 #define VACCONTROL_H
00012
00013 #include <Arduino.h>
00014 #include <Wire.h>
00015
00016
00018 namespace vacControlModule
00019 {
00021     enum class MainSwitchStates : int
00022     {
00023         Main_Switch_OFF,
00024         Main_Switch_MANUAL,
00025         Main_Switch_REMOTE,
00026         Main_switch_INVALID
00027     };
00028
00030     enum class PumpSwitchStates : int
00031     {
00032         Switch_Pump_OFF,
00033         Switch_Pump_ON
00034     };
00035
00037     enum class PumpState : int
00038     {
00039         pump_OFF,
00040         pump_ON
00041     };
00042
00045     typedef struct Pressure
00046     {
00047         float pressure;
00048     } meas;
00049
00050
00052     enum Scenarios
00053     {
00054         Scenario_1 = 0,
00055         Scenario_2 = 1,
00056         Scenario_3 = 2,
00057         Scenario_4 = 3,
00058         Scenario_5 = 4,
00059         Invalid_Scenario = -1
00060     };
00061
00065     class VacControl
00066     {
00067     public:
00068
00069         VacControl();
00070         ~VacControl();
00071
00076         void initialize();
00077
00082         void deinitialize();
00083
00090         bool isInitialized() const;
00091
00097         MainSwitchStates getMainSwitchState();
00098
00104         PumpSwitchStates getPumpSwitchState();
00105
00111         PumpState getPumpState();
00112
00118         Scenarios getScenario();
00119
00125         Pressure measure();
00126
00133         void setVacuumLed(float pressure, float targetPressure);
00134
00141         int getScenarioFromPotValue(int potValue);
00142
00148         void setPump(bool flag);
00149
00155         void run();
00156
00162         void setExternScenario(int pressure);
00163
00169         int getExternScenario();
00170
00176         void externPump(int pumpState);

```

```

00177
00183     void setExternPump(int state);
00184
00189     int getExternPump();
00190
00196     void setExternPressure(float pressure);
00197
00203     float getExternPressure();
00204
00205
00206 private:
00207     Pressure meas;
00208
00209     //Define Pins --> Input
00210     static const int Main_Switch_OFF = 27;           //Main_Switch OFF Mode 27
00211     static const int Main_Switch_MANUAL = 28;        //Main_Switch Manual Mode 28
00212     static const int Main_Switch_REMOTE = 29;        //Main_Switch Remote Mode 29
00213     static const int Switch_Pump_ON = 23;           //Button to turn Pump ON 23
00214     static const int targetPressure = A3;           //Potentiometer for Regulation
00215
00216     //Define Pins --> Output
00217     static const int Pump_Status_LED = 24;           //OUTPUT to see State off Pump
00218     static const int Pump_Relay = 25;               //OUTPUT to turn on/off Relais
00219     static const int targetVacuumLED = 26;           //OUTPUT to see Vacuum reached
00220
00221     //Variables to save Values
00222     int currentScenario = -1;
00223     int currentPumpState = -1;
00224
00225     //Variables for TargetPressure
00226     static const float TARGET_PRESSURE_1 = 1;
00227     static const float TARGET_PRESSURE_2 = 0.8f;
00228     static const float TARGET_PRESSURE_3 = 0.5;
00229     static const float TARGET_PRESSURE_4 = 0.01f;
00230
00231     static int lastState;
00232     static int lastPumpState;
00233
00234     bool _vacControlInitialized;
00235
00239     void handleOffMode();
00240
00244     void handleManualMode();
00245
00249     void handleRemoteMode();
00250
00254     void handleInvalidMode();
00255 };
00256 }
00257
00258 #endif //VACCONTROL_H

```


Index

atmToPascal
 calcModule::CalcModuleInternals, 16

beginEthernet
 comModule::EthernetCommunication, 29

beginI2C
 comModule::I2CCommunication, 50

beginSerial
 comModule::SerialCommunication, 90

C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp
 117
 159, 160
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h
 118, 120
 161
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp
 121
 162, 164
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h
 121
 164
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.h
 122
 165, 168
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC.h,
 calcModule, 9
 calcModule::CalcModuleInternals, 15
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SEB.h
 124
 atmToPascal, 16
 calculateAverage, 16
 calculateCurrent, 17
 calculatePower, 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h
 125
 calculateResistance, 17
 calculateStandardDeviation, 18
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h
 126, 127
 celsiusToFahrenheit, 18
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp,
 extractFloat, 19
 extractFloatFromResponse, 20
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h
 130, 132
 fahrenheitToCelsius, 20
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h,
 findMaximum, 20
 findMedian, 21
 findMinimum, 21
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h
 133
 kelvinToCelsius, 21
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.cpp,
 pascalToPsi, 22
 psiToPascal, 22
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h,
 calcModuleInternals, 23
 calculateAverage
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp
 138
 calcModule::CalcModuleInternals, 16
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h
 139, 140
 calculateCurrent
 calcModule::CalcModuleInternals, 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h
 140, 145
 calculatePower
 calcModule::CalcModuleInternals, 17
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp,
 calcModule::CalcModuleInternals, 17
 149
 calculateResistance
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem/rep
 150, 151
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sen
 152
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sen
 153, 155
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialM
 156, 157
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temp
 158
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temp
 159, 160
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeM
 161
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeM
 162, 164
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacCo
 164
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacCo
 165, 168

- calculateStandardDeviation
 - calcModule::CalcModuleInternals, 18
- calibMCP9601
 - TemperatureSensor, 100
- calibrateSensor
 - sensorModule::SensorModuleInternals, 86
- celsiusToFahrenheit
 - calcModule::CalcModuleInternals, 18
- celsiusToKelvin
 - calcModule::CalcModuleInternals, 19
- checkRamLevel
 - reportSystem::ReportSystem, 72
- checkSensorStatus
 - sensorModule::SensorModuleInternals, 87
- checkSystemHealth
 - reportSystem::ReportSystem, 73
- checkThresholds
 - reportSystem::ReportSystem, 74
- comModule, 9
- comModule::ComModuleInternals, 23
 - getEthernet, 24
 - getI2C, 24
 - getSerial, 24
 - getSPI, 25
- comModule::EthernetCommunication, 28
 - beginEthernet, 29
 - getClient, 29
 - getCompound, 30, 31
 - getCompoundInternal, 32
 - getParameter, 33
 - getParsedCompound, 33, 34
 - getRequestedEndpoint, 35
 - getSendDataFlag, 35
 - getSpecificEndpoint, 35
 - isInitialized, 36
 - parseCompoundResponse, 36
 - receiveEthernetData, 37
 - sendCommand, 37
 - sendEthernetData, 38
 - sendJsonResponse, 38
 - setCompound, 38, 39
 - setCompoundInternal, 40
 - setParameter, 41
 - setSendDataFlag, 41
- comModule::I2CCommunication, 49
 - beginI2C, 50
 - i2cRead, 51
 - i2cWrite, 51
 - isInitialized, 51
- comModule::SerialCommunication, 89
 - beginSerial, 90
 - isInitialized, 90
 - receiveSerialData, 91
 - sendSerialData, 91
- comModule::SPICommunication, 98
 - isInitialized, 98
 - spiRead, 98
 - spiWrite, 99
- comModuleInternals, 26
- createJson
 - jsonModule::JsonModuleInternals, 53
- detectStackOverflow
 - reportSystem::ReportSystem, 74
- Error, 27
- ErrorSeverity, 27
- externPump
 - vacControlModule::VacControl, 110
- extractFloat
 - calcModule::CalcModuleInternals, 19
- extractFloatFromResponse
 - calcModule::CalcModuleInternals, 20
- fahrenheitToCelsius
 - calcModule::CalcModuleInternals, 20
- findMaximum
 - calcModule::CalcModuleInternals, 20
- findMedian
 - calcModule::CalcModuleInternals, 21
- findMinimum
 - calcModule::CalcModuleInternals, 21
- flybackModule, 10
- flybackModule::Flyback, 42
 - getExternPSU, 43
 - getHVState, 43
 - getHVSwitchState, 43
 - getHysteresis, 43
 - getMainSwitchState, 44
 - getTargetVoltage, 44
 - getTimerState, 44
 - isInitialized, 45
 - measure, 45
 - regulateVoltage, 46
 - setExternDutyCycle, 47
 - setExternFrequency, 47
 - setExternPSU, 48
 - setHysteresis, 48
 - setTargetVoltage, 48
 - setTimerState, 49
- flybackModule::Measurement, 63
- formatTimeString
 - timeModule::TimeModuleInternals, 103
- get
 - PointerWrapper< T >, 65
 - ScopedPointer< T >, 84
- getClient
 - comModule::EthernetCommunication, 29
- getCompound
 - comModule::EthernetCommunication, 30, 31
- getCompoundInternal
 - comModule::EthernetCommunication, 32
- getCPULoad
 - reportSystem::ReportSystem, 75
- getCurrentTime
 - LogManager, 56

- reportSystem::ReportSystem, 75
- SerialMenu, 92
- getEthernet
 - comModule::ComModuleInternals, 24
- getExternPressure
 - vacControlModule::VacControl, 110
- getExternPSU
 - flybackModule::Flyback, 43
- getExternPump
 - vacControlModule::VacControl, 110
- getExternScenario
 - vacControlModule::VacControl, 110
- getHVState
 - flybackModule::Flyback, 43
- getHVSwitchState
 - flybackModule::Flyback, 43
- getHysteresis
 - flybackModule::Flyback, 43
- getI2C
 - comModule::ComModuleInternals, 24
- getInstance
 - LogManager, 56
 - timeModule::TimeModuleInternals, 104
- getJsonString
 - jsonModule::JsonModuleInternals, 53
- getLastError
 - reportSystem::ReportSystem, 75
- getLastErrorInfo
 - reportSystem::ReportSystem, 76
- getMainSwitchState
 - flybackModule::Flyback, 44
 - vacControlModule::VacControl, 110
- getMemoryStatus
 - reportSystem::ReportSystem, 76
- getParameter
 - comModule::EthernetCommunication, 33
- getParsedCompound
 - comModule::EthernetCommunication, 33, 34
- getPumpState
 - vacControlModule::VacControl, 111
- getPumpSwitchState
 - vacControlModule::VacControl, 111
- getRequestedEndpoint
 - comModule::EthernetCommunication, 35
- getScenario
 - vacControlModule::VacControl, 111
- getScenarioFromPotValue
 - vacControlModule::VacControl, 112
- getSendDataFlag
 - comModule::EthernetCommunication, 35
- getSerial
 - comModule::ComModuleInternals, 24
- getSpecificEndpoint
 - comModule::EthernetCommunication, 35
- getSPI
 - comModule::ComModuleInternals, 25
- getStackDump
 - reportSystem::ReportSystem, 77
- getSystemTime
 - timeModule::TimeModuleInternals, 105
- getTargetVoltage
 - flybackModule::Flyback, 44
- getTimerState
 - flybackModule::Flyback, 44
- hasCapacityFor
 - jsonModule::JsonModuleInternals, 54
- hasNoSavedErrors
 - reportSystem::ReportSystem, 77
- i2cRead
 - comModule::I2CCommunication, 51
- i2cWrite
 - comModule::I2CCommunication, 51
- incrementTime
 - timeModule::TimeModuleInternals, 106
- initSDCard
 - LogManager, 58
- isCommunicationOK
 - reportSystem::ReportSystem, 77
- isInitialized
 - comModule::EthernetCommunication, 36
 - comModule::I2CCommunication, 51
 - comModule::SerialCommunication, 90
 - comModule::SPICommunication, 98
 - flybackModule::Flyback, 45
 - PressureSensor, 68
 - TemperatureSensor, 100
 - vacControlModule::VacControl, 112
- isMemoryOK
 - reportSystem::ReportSystem, 78
- IsNullPtr
 - PtrUtils, 69
- isRamOK
 - reportSystem::ReportSystem, 78
- isSDCardInitialized
 - LogManager, 59
- isStackSafe
 - reportSystem::ReportSystem, 79
- isTemperatureSensorOK
 - reportSystem::ReportSystem, 79
- IsValidPtr
 - PtrUtils, 70
- jsonModule, 11
- jsonModule::JsonModuleInternals, 52
 - createJson, 53
 - getJsonString, 53
 - hasCapacityFor, 54
- kelvinToCelsius
 - calcModule::CalcModuleInternals, 21
- load
 - SerialMenu, 93
- locker, 11
- locker::ScopedLock, 82

- ScopedLock, [82](#), [83](#)
- LockerBase, [54](#)
 - lockEthernetScoped, [55](#)
 - lockSerialScoped, [55](#)
 - lockTemperatureScoped, [55](#)
- lockEthernetScoped
 - LockerBase, [55](#)
- lockSerialScoped
 - LockerBase, [55](#)
- lockTemperatureScoped
 - LockerBase, [55](#)
- LogManager, [56](#)
 - getCurrentTime, [56](#)
 - getInstance, [56](#)
 - initSDCard, [58](#)
 - isSDCardInitialized, [59](#)
 - renameFile, [59](#)
 - setLogFileName, [60](#)
 - writeToLogFile, [61](#)
- LogMapper, [63](#)
- measure
 - flybackModule::Flyback, [45](#)
 - vacControlModule::VacControl, [112](#)
- Measurement, [63](#)
- MenuItem, [64](#)
- operator->
 - PointerWrapper< T >, [66](#)
 - ScopedPointer< T >, [84](#)
- operator*
 - PointerWrapper< T >, [65](#)
 - ScopedPointer< T >, [84](#)
- Outputlevel, [64](#)
- parseCompoundResponse
 - comModule::EthernetCommunication, [36](#)
- pascalToAtm
 - calcModule::CalcModuleInternals, [22](#)
- pascalToPsi
 - calcModule::CalcModuleInternals, [22](#)
- PointerWrapper< T >, [65](#)
 - get, [65](#)
 - operator->, [66](#)
 - operator*, [65](#)
 - release, [66](#)
 - reset, [66](#)
- PressureSensor, [67](#)
 - isInitialized, [68](#)
 - readPressure, [68](#)
- printToSerial
 - SerialMenu, [94](#), [95](#)
- psiToPascal
 - calcModule::CalcModuleInternals, [22](#)
- PtrUtils, [69](#)
 - IsNullPtr, [69](#)
 - IsValidPtr, [70](#)
- ptrUtils.h
 - RETHROW, [143](#)
- RETURN_IF_ERR, [143](#)
- THROW, [143](#)
- THROW_IF_ERR, [143](#)
- THROW_IF_FALSE, [143](#)
- THROW_IF_NULL, [144](#)
- THROW_SOFT, [144](#)
- TRY, [144](#)
- tryDeletePtr, [144](#)
- readMCP9601
 - TemperatureSensor, [101](#)
- readPressure
 - PressureSensor, [68](#)
- readSensor
 - sensorModule::SensorModuleInternals, [88](#)
- readTemperature
 - TemperatureSensor, [102](#)
- receiveEthernetData
 - comModule::EthernetCommunication, [37](#)
- receiveSerialData
 - comModule::SerialCommunication, [91](#)
- regulateVoltage
 - flybackModule::Flyback, [46](#)
- release
 - PointerWrapper< T >, [66](#)
 - ScopedPointer< T >, [84](#)
- renameFile
 - LogManager, [59](#)
- reportError
 - reportSystem::ReportSystem, [79](#)
- reportStatus
 - reportSystem::ReportSystem, [80](#)
- reportSystem, [12](#)
- reportSystem::ReportSystem, [71](#)
 - checkRamLevel, [72](#)
 - checkSystemHealth, [73](#)
 - checkThresholds, [74](#)
 - detectStackOverflow, [74](#)
 - getCPULoad, [75](#)
 - getCurrentTime, [75](#)
 - getLastError, [75](#)
 - getLastErrorInfo, [76](#)
 - getMemoryStatus, [76](#)
 - getStackDump, [77](#)
 - hasNoSavedErrors, [77](#)
 - isCommunicationOK, [77](#)
 - isMemoryOK, [78](#)
 - isRamOK, [78](#)
 - isStackSafe, [79](#)
 - isTemperatureSensorOK, [79](#)
 - reportError, [79](#)
 - reportStatus, [80](#)
 - saveLastError, [81](#)
 - setThreshold, [81](#)
- reportUnknownSensorOnce
 - sensorModule::SensorModuleInternals, [88](#)
- reset
 - PointerWrapper< T >, [66](#)
 - ScopedPointer< T >, [84](#)

- RETHROW
 - ptrUtils.h, [143](#)
- RETURN_IF_ERR
 - ptrUtils.h, [143](#)
- run
 - vacControlModule::VacControl, [113](#)
- saveLastError
 - reportSystem::ReportSystem, [81](#)
- ScopedLock
 - locker::ScopedLock, [82](#), [83](#)
- ScopedPointer< T >, [83](#)
 - get, [84](#)
 - operator->, [84](#)
 - operator*, [84](#)
 - release, [84](#)
 - reset, [84](#)
- sendCommand
 - comModule::EthernetCommunication, [37](#)
- sendEthernetData
 - comModule::EthernetCommunication, [38](#)
- sendJsonResponse
 - comModule::EthernetCommunication, [38](#)
- sendSerialData
 - comModule::SerialCommunication, [91](#)
- sensorModule, [12](#)
 - sensorModule::SensorModuleInternals, [85](#)
 - calibrateSensor, [86](#)
 - checkSensorStatus, [87](#)
 - readSensor, [88](#)
 - reportUnknownSensorOnce, [88](#)
- SerialMenu, [91](#)
 - getCurrentTime, [92](#)
 - load, [93](#)
 - printToSerial, [94](#), [95](#)
- setCompound
 - comModule::EthernetCommunication, [38](#), [39](#)
- setCompoundInternal
 - comModule::EthernetCommunication, [40](#)
- setExternDutyCycle
 - flybackModule::Flyback, [47](#)
- setExternFrequency
 - flybackModule::Flyback, [47](#)
- setExternPressure
 - vacControlModule::VacControl, [113](#)
- setExternPSU
 - flybackModule::Flyback, [48](#)
- setExternPump
 - vacControlModule::VacControl, [114](#)
- setExternScenario
 - vacControlModule::VacControl, [114](#)
- setHysteresis
 - flybackModule::Flyback, [48](#)
- setLogFileName
 - LogManager, [60](#)
- setParameter
 - comModule::EthernetCommunication, [41](#)
- setPump
 - vacControlModule::VacControl, [114](#)
- setSendDataFlag
 - comModule::EthernetCommunication, [41](#)
- setSystemTime
 - timeModule::TimeModuleInternals, [107](#)
- setTargetVoltage
 - flybackModule::Flyback, [48](#)
- setThreshold
 - reportSystem::ReportSystem, [81](#)
- setTimeFromHas
 - timeModule::TimeModuleInternals, [108](#)
- setTimerState
 - flybackModule::Flyback, [49](#)
- setVacuumLed
 - vacControlModule::VacControl, [114](#)
- spiRead
 - comModule::SPICommunication, [98](#)
- spiWrite
 - comModule::SPICommunication, [99](#)
- TemperatureSensor, [99](#)
 - calibMCP9601, [100](#)
 - isInitialized, [100](#)
 - readMCP9601, [101](#)
 - readTemperature, [102](#)
- THROW
 - ptrUtils.h, [143](#)
- THROW_IF_ERR
 - ptrUtils.h, [143](#)
- THROW_IF_FALSE
 - ptrUtils.h, [143](#)
- THROW_IF_NULL
 - ptrUtils.h, [144](#)
- THROW_SOFT
 - ptrUtils.h, [144](#)
- timeModule, [13](#)
 - timeModule::DateTimeStruct, [26](#)
 - timeModule::TimeModuleInternals, [103](#)
 - formatTimeString, [103](#)
 - getInstance, [104](#)
 - getSystemTime, [105](#)
 - incrementTime, [106](#)
 - setSystemTime, [107](#)
 - setTimeFromHas, [108](#)
- TRY
 - ptrUtils.h, [144](#)
- tryDeletePtr
 - ptrUtils.h, [144](#)
- vacControlModule, [13](#)
 - vacControlModule::Pressure, [67](#)
 - vacControlModule::VacControl, [108](#)
 - externPump, [110](#)
 - getExternPressure, [110](#)
 - getExternPump, [110](#)
 - getExternScenario, [110](#)
 - getMainSwitchState, [110](#)
 - getPumpState, [111](#)
 - getPumpSwitchState, [111](#)
 - getScenario, [111](#)

- getScenarioFromPotValue, [112](#)
- isInitialized, [112](#)
- measure, [112](#)
- run, [113](#)
- setExternPressure, [113](#)
- setExternPump, [114](#)
- setExternScenario, [114](#)
- setPump, [114](#)
- setVacuumLed, [114](#)

writeToLogFile

- LogManager, [61](#)