FFRESW

Generated by Doxygen 1.9.8

1 Namespace Index 1

1.1 Namespace List . . . 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Namespace Documentation 9

5.1 calcModule Namespace Reference . . . 9

5.1.1 Detailed Description . . . 9

5.2 comModule Namespace Reference . . . 9

5.2.1 Detailed Description . . . 10

5.3 flybackModule Namespace Reference . . . 10

5.3.1 Detailed Description . . . 11

5.4 jsonModule Namespace Reference . . . 11

5.4.1 Detailed Description . . . 11

5.5 locker Namespace Reference . . . 11

5.5.1 Detailed Description . . . 12

5.6 reportSystem Namespace Reference . . . 12

5.6.1 Detailed Description . . . 12

5.7 sensorModule Namespace Reference . . . 12

5.7.1 Detailed Description . . . 12

5.8 timeModule Namespace Reference . . . 12

5.8.1 Detailed Description . . . 13

5.9 vacControlModule Namespace Reference . . . 13

5.9.1 Detailed Description . . . 13

6 Class Documentation 15

6.1 calcModule::CalcModuleInternals Class Reference . . . 15

6.1.1 Member Function Documentation . . . 16

6.1.1.1 atmToPascal() . . . 16

6.1.1.2 calculateAverage() . . . 16

6.1.1.3 calculateCurrent() . . . 17

6.1.1.4 calculatePower() . . . 17

6.1.1.5 calculateResistance() . . . 17

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() . . . 24
6.3.1.4 getSPI() . . . 24
6.4 comModuleInternals Class Reference . . . 25
6.4.1 Detailed Description . . . 25
6.5 timeModule::DateTimeStruct Struct Reference . . . 25
6.5.1 Detailed Description . . . 25
6.6 comModule::EthernetCommunication Class Reference . . . 25
6.6.1 Detailed Description . . . 27
6.6.2 Member Function Documentation . . . 27
6.6.2.1 beginEthernet() . . . 27
6.6.2.2 getClient() . . . 27
6.6.2.3 getCompound() [1/3] . . . 27
6.6.2.4 getCompound() [2/3] . . . 28
6.6.2.5 getCompound() [3/3] . . . 29
6.6.2.6 getCompoundInternal() . . . 29
6.6.2.7 getParameter() . . . 30
6.6.2.8 getParsedCompound() [1/3] . . . 30
6.6.2.9 getParsedCompound() [2/3] . . . 31
6.6.2.10 getParsedCompound() [3/3] . . . 32
6.6.2.11 getRequestedEndpoint() . . . 32
6.6.2.12 getSendDataFlag() . . . 32
6.6.2.13 getSpecificEndpoint() . . . 32
6.6.2.14 isInitialized() . . . 33
6.6.2.15 parseCompoundResponse() . . . 33
6.6.2.16 receiveEthernetData() . . . 34
6.6.2.17 sendCommand() . . . 34
6.6.2.18 sendEthernetData() . . . 34

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

6.12.1 Member Function Documentation . . . 48
6.12.1.1 getCurrentTime() . . . 48
6.12.1.2 getInstance() . . . 49
6.12.1.3 initSDCard() . . . 50
6.12.1.4 isSDCardInitialized() . . . 51
6.12.1.5 renameFile() . . . 51
6.12.1.6 setLogFileName() . . . 52
6.12.1.7 writeToLogFile() . . . 53
6.13 LogMapper Class Reference . . . 54
6.13.1 Detailed Description . . . 55
6.14 flybackModule::Measurement Struct Reference . . . 55
6.14.1 Detailed Description . . . 55
6.15 Measurement Struct Reference . . . 55
6.15.1 Detailed Description . . . 55
6.16 MenuItem Struct Reference . . . 56
6.16.1 Detailed Description . . . 56
6.17 Outputlevel Class Reference . . . 56
6.17.1 Detailed Description . . . 56
6.18 PointerWrapper< T > Class Template Reference . . . 56
6.18.1 Detailed Description . . . 57
6.18.2 Member Function Documentation . . . 57
6.18.2.1 get() . . . 57
6.18.2.2 operator*() . . . 57
6.18.2.3 operator->() . . . 58
6.18.2.4 release() . . . 58
6.18.2.5 reset() . . . 58
6.19 vacControlModule::Pressure Struct Reference . . . 58
6.20 PressureSensor Class Reference . . . 59
6.20.1 Detailed Description . . . 59
6.20.2 Member Function Documentation . . . 59
6.20.2.1 isInitialized() . . . 59
6.20.2.2 readPressure() . . . 60
6.21 PtrUtils Class Reference . . . 61
6.21.1 Detailed Description . . . 61
6.21.2 Member Function Documentation . . . 61
6.21.2.1 IsNullPtr() . . . 61
6.21.2.2 IsValidPtr() . . . 62
6.22 reportSystem::ReportSystem Class Reference . . . 63
6.22.1 Detailed Description . . . 64
6.22.2 Member Function Documentation . . . 64
6.22.2.1 checkRamLevel() . . . 64
6.22.2.2 checkSystemHealth() . . . 65

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

6.27.2.2 load() . . . 81
6.27.2.3 printToSerial() [1/4] . . . 82
6.27.2.4 printToSerial() [2/4] . . . 82
6.27.2.5 printToSerial() [3/4] . . . 83
6.27.2.6 printToSerial() [4/4] . . . 83
6.28 comModule::SPICommunication Class Reference . . . 86
6.28.1 Detailed Description . . . 86
6.28.2 Member Function Documentation . . . 86
6.28.2.1 isInitialized() . . . 86
6.28.2.2 spiRead() . . . 86
6.28.2.3 spiWrite() . . . 87
6.29 TemperatureSensor Class Reference . . . 87
6.29.1 Detailed Description . . . 88
6.29.2 Member Function Documentation . . . 88
6.29.2.1 calibMCP9601() . . . 88
6.29.2.2 isInitialized() . . . 88
6.29.2.3 readMCP9601() . . . 88
6.29.2.4 readTemperature() . . . 89
6.30 timeModule::TimeModuleInternals Class Reference . . . 90
6.30.1 Detailed Description . . . 91
6.30.2 Member Function Documentation . . . 91
6.30.2.1 formatTimeString() . . . 91
6.30.2.2 getInstance() . . . 92
6.30.2.3 getSystemTime() . . . 93
6.30.2.4 incrementTime() . . . 94
6.30.2.5 setSystemTime() . . . 95
6.30.2.6 setTimeFromHas() . . . 96
6.31 vacControlModule::VacControl Class Reference . . . 96
6.31.1 Detailed Description . . . 97
6.31.2 Member Function Documentation . . . 97
6.31.2.1 externProcess() . . . 97
6.31.2.2 getExternPressure() . . . 98
6.31.2.3 getExternScenario() . . . 98
6.31.2.4 getScenario() . . . 98
6.31.2.5 getScenarioFromPotValue() . . . 98
6.31.2.6 getSwitchState() . . . 99
6.31.2.7 isInitialized() . . . 99
6.31.2.8 measure() . . . 100
6.31.2.9 run() . . . 100
6.31.2.10 setExternPressure() . . . 100
6.31.2.11 setExternScenario() . . . 101
6.31.2.12 setPump() . . . 101

6.31.2.13 setVacuumLed() . . . 102

7 File Documentation **103**

7.1 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp File Reference . 103
7.1.1 Detailed Description . . . 103
7.2 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h File Reference . . 104
7.2.1 Detailed Description . . . 105
7.3 calcModule.h . . . 105
7.4 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp File Reference . 106
7.4.1 Detailed Description . . . 107
7.5 comModule.h . . . 107
7.6 ETHH.h . . . 108
7.7 I2CC.h . . . 110
7.8 SER.h . . . 110
7.9 SPII.h . . . 110
7.10 config.h . . . 111
7.11 flyback.h . . . 111
7.12 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp File Reference 113
7.12.1 Detailed Description . . . 113
7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h File Reference . 114
7.13.1 Detailed Description . . . 115
7.14 jsonModule.h . . . 115
7.15 lockerBase.h . . . 116
7.16 scopedLock.h . . . 116
7.17 logManager.h . . . 117
7.18 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp File Reference . . . 118
7.18.1 Detailed Description . . . 118
7.19 pressure.h . . . 119
7.20 ptrUtils.h . . . 119
7.21 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp File Reference . . . 121
7.21.1 Detailed Description . . . 122
7.22 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h File Reference 122
7.22.1 Detailed Description . . . 124
7.23 reportSystem.h . . . 124
7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h File Reference125
7.24.1 Detailed Description . . . 126
7.25 sensorModule.h . . . 127
7.26 serialMenu.h . . . 128
7.27 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp File Reference 128
7.27.1 Detailed Description . . . 129
7.28 temperature.h . . . 129
7.29 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp File Reference 130

7.29.1 Detailed Description 131
7.30 timeModule.h 131
7.31 vacControl.h 132

Index **135**

Chapter 1

Namespace Index

1.1 Namespace List

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

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

calcModule::CalcModuleInternals . . . 15
calcModuleInternals . . . 23
comModule::ComModuleInternals . . . 23
comModuleInternals . . . 25
timeModule::DateTimeStruct . . . 25
comModule::EthernetCommunication . . . 25
flybackModule::Flyback . . . 38
comModule::I2CCommunication . . . 41
jsonModule::JsonModuleInternals . . . 43
jsonModuleInternals . . . 46
LockerBase . . . 47
LogManager . . . 48
LogMapper . . . 54
flybackModule::Measurement . . . 55
Measurement . . . 55
MenuItem . . . 56
Outputlevel . . . 56
PointerWrapper< T > . . . 56
vacControlModule::Pressure . . . 58
PressureSensor . . . 59
    sensorModule::SensorModuleInternals . . . 74
PtrUtils . . . 61
reportSystem::ReportSystem . . . 63
locker::ScopedLock . . . 71
ScopedPointer< T > . . . 72
comModule::SerialCommunication . . . 77
SerialMenu . . . 79
comModule::SPICommunication . . . 86
TemperatureSensor . . . 87
    sensorModule::SensorModuleInternals . . . 74
timeModule::TimeModuleInternals . . . 90
vacControlModule::VacControl . . . 96

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 25
timeModule::DateTimeStruct
Struct to hold the date and time 25
comModule::EthernetCommunication
Class to handle Ethernet communication 25
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 38
comModule::I2CCommunication
Class to handle I2C communication 41
jsonModule::JsonModuleInternals 43
jsonModuleInternals
Class for the JSON module internals 46
LockerBase
Base class for the locker system 47
LogManager 48
LogMapper
Class which handle the printed log messages, maps aka parses them and saves them to the SD card 54
flybackModule::Measurement
Structure to store the measured values of the system This structure holds the voltage, current, power, frequency and dutycycle values measured from the system 55
Measurement
Structure to store the measured values of the system This structure holds the pressure values measured from the system 55
MenuItem
Serial menu structure 56
Outputlevel
Enum Class for the differnet Outputlevels 56

PointerWrapper< T >
Tempalte class for wrapping a pointer . . . 56
vacControlModule::Pressure . . . 58
PressureSensor
Pressure sensor class . . . 59
PtrUtils
Utility class for pointer operations . . . 61
reportSystem::ReportSystem
Class for the report system . . . 63
locker::ScopedLock
Scoped lock class for mutexes . . . 71
ScopedPointer< T >
Template class for a Scoped Pointer . . . 72
sensorModule::SensorModuleInternals
Class for the sensor module internals . . . 74
comModule::SerialCommunication
Class to handle Serial communication . . . 77
SerialMenu
Class for the serial menu . . . 79
comModule::SPICommunication
Class to handle SPI communication . . . 86
TemperatureSensor
Temperature sensor class . . . 87
timeModule::TimeModuleInternals
Class to handle Systemtime . . . 90
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 . . . 96

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 . . . 103
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h
Header file for the calculation module handling sensor data . . . 104
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp
Implementation of the comModule class that utilizes various communication protocols . . . 106
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h . . . 107
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.h . . . 108
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC.h . . . 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SER.h . . . 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h . . . 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h . . . 111
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h . . . 111
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp
Implementation of the jsonModule class . . . 113
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h . . . 114
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h . . . 116
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h . . . 116
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h . . . 117
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp
Implementation of the pressure class . . . 118
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h . . . 119
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h . . . 119
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp
Unified system health and error reporting module . . . 121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h . . . 122
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h . . . 125
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialMenu.h . . . 128
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.cpp
Implementation of the temperature class . . . 128
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temperature.h . . . 129
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.cpp
Implementation of the timeModule class . . . 130
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeModule.h . . . 131
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h . . . 132

Chapter 5

Namespace Documentation

5.1 calcModule Namespace Reference

Namespace for the calculation module.

Classes

- class CalcModuleInternals

Enumerations

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

  Enum for the different Types we want to extract from a response.

5.1.1 Detailed Description

Namespace for the calculation module.

5.2 comModule Namespace Reference

Namespace for the communication module.

Classes

- class ComModuleInternals
- class EthernetCommunication

  Class to handle Ethernet communication.
- class I2CCommunication

  Class to handle I2C communication.
- class SerialCommunication

  Class to handle Serial communication.
- class SPICommunication

  Class to handle SPI communication.

Enumerations

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

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

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

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

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

  Enum class for the error codes.

5.2.1 Detailed Description

Namespace for the communication module.

5.3 flybackModule Namespace Reference

Namespace for the Flyback module.

Classes

- class Flyback

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

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

Typedefs

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

Enumerations

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

  enum for different SwitchStates of HVModule

5.3.1 Detailed Description

Namespace for the Flyback module.

5.4 jsonModule Namespace Reference

Namespace for the JSON module.

Classes

- class JsonModuleInternals

5.4.1 Detailed Description

Namespace for the JSON module.

5.5 locker Namespace Reference

Namespace for the locker system.

Classes

- class ScopedLock

  Scoped lock class for mutexes.

5.5.1 Detailed Description

Namespace for the locker system.

5.6 reportSystem Namespace Reference

Namespace for the report system.

Classes

- class ReportSystem

  Class for the report system.

5.6.1 Detailed Description

Namespace for the report system.

5.7 sensorModule Namespace Reference

Namespace for the sensor module.

Classes

- class SensorModuleInternals

  Class for the sensor module internals.

Enumerations

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

  Enum class for the sensor types.

5.7.1 Detailed Description

Namespace for the sensor module.

5.8 timeModule Namespace Reference

namespace for the timeModule

Classes

- struct DateTimeStruct

  Struct to hold the date and time.
- class TimeModuleInternals

  Class to handle Systemtime.

Typedefs

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

5.8.1 Detailed Description

namespace for the timeModule

5.9 vacControlModule Namespace Reference

Namespace for the VacControl module.

Classes

- struct Pressure
- class VacControl

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

Typedefs

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

Enumerations

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

  Enum to represent the states of the main switch and pump.
- enum Scenarios {
  Scenario_1 = 0 , **Scenario_2** = 1 , **Scenario_3** = 2 , **Scenario_4** = 3 ,
  Scenario_5 = 4 , **Invalid_Scenario** = -1 }

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

5.9.1 Detailed Description

Namespace for the VacControl module.

Chapter 6

Class Documentation

6.1 calcModule::CalcModuleInternals Class Reference

Static Public Member Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Function to caculate the Resistance.

- static float extractFloat (String response, int id)

  Extract the float from a VAT uC eth frame.

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

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

6.1.1 Member Function Documentation

6.1.1.1 atmToPascal()

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

Function to convert atm to pascal.

Parameters

atm	-> The pressure in atm.

Returns

float -> The pressure in pascal.

6.1.1.2 calculateAverage()

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

Function to calculate the average of a data set.

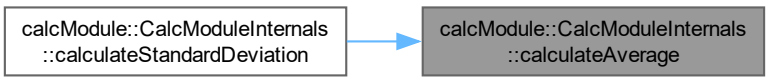
Parameters

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

Returns

float -> The average of the data set.

Here is the caller graph for this function:



6.1.1.3 calculateCurrent()

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

Funcion to calculate the current.

Parameters

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

Returns

float -> The calculated current.

6.1.1.4 calculatePower()

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

Function to calculate the power.

Parameters

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

Returns

float -> The calculated power.

6.1.1.5 calculateResistance()

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

Function to caculate the Resistance.

Parameters

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

Returns

float -> The calculated resistance.

6.1.1.6 calculateStandardDeviation()

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

Function to calculate the standard deviation of a data set.

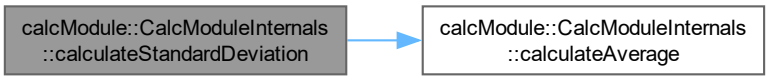
Parameters

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

Returns

float -> The standard deviation of the data set.

Here is the call graph for this function:



6.1.1.7 celsiusToFahrenheit()

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

Function to convert celsius to fahrenheit.

Parameters

celsius	-> The temperature in celsius.

Returns

float -> The temperature in fahrenheit.

Here is the caller graph for this function:

6.1.1.8 celsiusToKelvin()

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

Function to convert celsius to kelvin.

Parameters

celsius	-> The temperature in celsius.

Returns

float -> The temperature in kelvin.

Here is the caller graph for this function:

6.1.1.9 extractFloat()

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

Extract the float from a VAT uC eth frame.

Parameters

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

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

Returns

float -> The extracted float.

6.1.1.10 extractFloatFromResponse()

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

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

Parameters

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

Returns

float -> The extracted float.

6.1.1.11 fahrenheitToCelsius()

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

Function to convert fahrenheit to celsius.

Parameters

fahrenheit	-> The temperature in fahrenheit.

Returns

float -> The temperature in celsius.

6.1.1.12 findMaximum()

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

Function to calculate the maximum value of a data set.

Parameters

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

Returns

float -> The maximum value of the data set.

6.1.1.13 findMedian()

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

Function to calculate the median of a data set.

Parameters

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

Returns

float -> The median of the data set.

6.1.1.14 findMinimum()

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

Function to calculate the minimum value of a data set.

Parameters

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

Returns

float -> The minimum value of the data set.

6.1.1.15 kelvinToCelsius()

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

Function to convert kelvin to celsius.

Parameters

kelvin	-> The temperature in kelvin.

Returns

float -> The temperature in celsius.

6.1.1.16 pascalToAtm()

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

Function to convert pascal to atm.

Parameters

pascal	-> The pressure in pascal.

Returns

float -> The pressure in atm.

6.1.1.17 pascalToPsi()

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

Function to convert pascal to psi.

Parameters

pascal	-> The pressure in pascal.

Returns

float -> The pressure in psi.

6.1.1.18 psiToPascal()

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

Function to convert psi to pascal.

Parameters

psi	-> The pressure in psi.

Returns

float -> The pressure in pascal.

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

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

6.2 calcModuleInternals Class Reference

Class for the calculation module internals.

`#include <calcModule.h>`

6.2.1 Detailed Description

Class for the calculation module internals.

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

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

6.3 comModule::ComModuleInternals Class Reference

Public Member Functions

- EthernetCommunication & getEthernet ()
  Get the Ethernet object.
- I2CCommunication & getI2C ()
  Get the I2C object.
- SPICommunication & getSPI ()
  Get the SPI object.
- SerialCommunication & getSerial ()
  Get the Serial object.

6.3.1 Member Function Documentation

6.3.1.1 getEthernet()

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

Get the Ethernet object.

Returns

EthernetCommunication&

6.3.1.2 getI2C()

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

Get the I2C object.

Returns

I2CCommunication&

6.3.1.3 getSerial()

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

Get the Serial object.

Returns

SerialCommunication&

6.3.1.4 getSPI()

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

Get the SPI object.

Returns

SPICommunication&

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

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

6.4 comModuleInternals Class Reference

Class for the communication module internals.

`#include <comModule.h>`

6.4.1 Detailed Description

Class for the communication module internals.

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

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

6.5 timeModule::DateTimeStruct Struct Reference

Struct to hold the date and time.

`#include <timeModule.h>`

Public Attributes

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

6.5.1 Detailed Description

Struct to hold the date and time.

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

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

6.6 comModule::EthernetCommunication Class Reference

Class to handle Ethernet communication.

`#include <ETHH.h>`

Public Member Functions

- void beginEthernet (uint8_t *macAddress, IPAddress ip)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Getter for a compound command response from the valve uC Slave (Compound2)
- String getCompound (Compound3 id, int index)

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

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

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

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

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

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

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

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

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

6.6.1 Detailed Description

Class to handle Ethernet communication.

6.6.2 Member Function Documentation

6.6.2.1 beginEthernet()

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

Function to initialize the Ethernet communication.

Parameters

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

6.6.2.2 getClient()

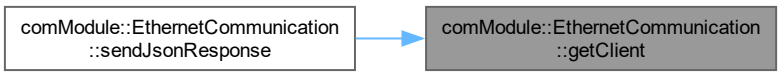
```
EthernetClient & EthernetCommunication::getClient ( )
```

Get the currently active Ethernet client.

Returns

EthernetClient& , Reference to the active Ethernet client

Here is the caller graph for this function:



6.6.2.3 getCompound() [1/3]

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

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

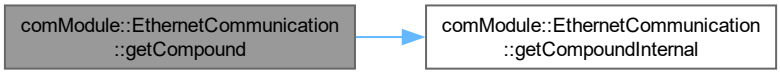
Parameters

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

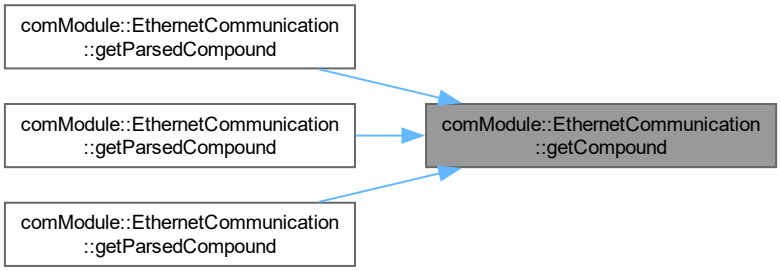
Returns

String -> Response from the valve uC slave

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.4 getCompound() [2/3]

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

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

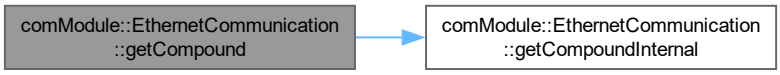
Parameters

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

Returns

String -> Response from the valve uC slave

Here is the call graph for this function:



6.6.2.5 getCompound() [3/3]

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

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

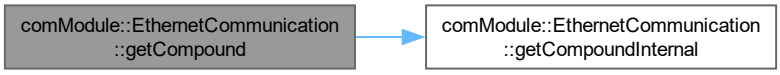
Parameters

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

Returns

String -> Response from the valve uC slave

Here is the call graph for this function:



6.6.2.6 getCompoundInternal()

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

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

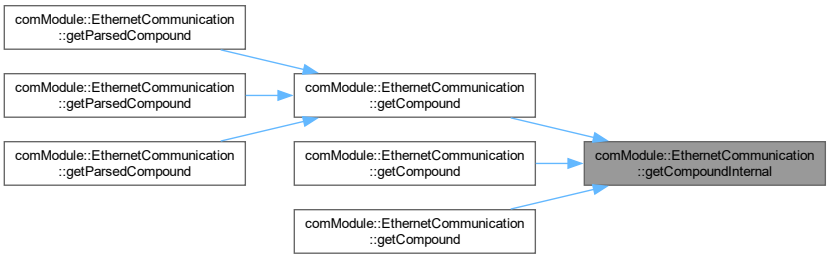
Parameters

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

Returns

String -> Response from the valve uC slave

Here is the caller graph for this function:



6.6.2.7 getParameter()

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

Getter for a parameter from the VAT slave.

Parameters

id	-> The ID of the parameter to get

Returns

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

6.6.2.8 getParsedCompound() [1/3]

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

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

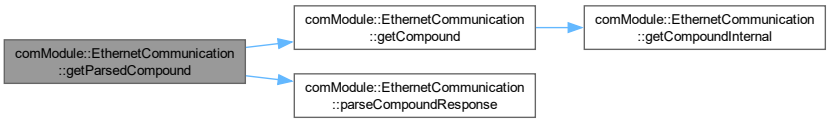
Parameters

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

Returns

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

Here is the call graph for this function:



6.6.2.9 getParsedCompound() [2/3]

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

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

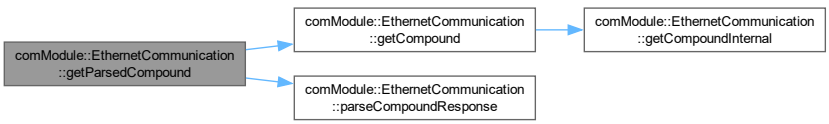
Parameters

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

Returns

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

Here is the call graph for this function:



6.6.2.10 getParsedCompound() [3/3]

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

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

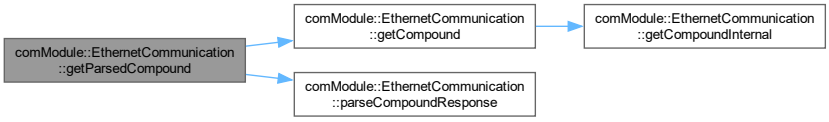
Parameters

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

Returns

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

Here is the call graph for this function:



6.6.2.11 getRequestedEndpoint()

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

Function to get the requested endpoint.

Returns

String -> The requested endpoint

6.6.2.12 getSendDataFlag()

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

Function to get the current status of the flag.

Returns

true -> if data should be sent

false -> if data should not be sent

6.6.2.13 getSpecificEndpoint()

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

Function to get the specific endpoint.

Parameters

jsonBody	-> The json body to get the endpoint from

Returns

String -> The specific endpoint

6.6.2.14 isInitialized()

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

Function to check if the Ethernet communication is initialized.

Returns

true -> if the Ethernet communication is initialized

false -> if the Ethernet communication is not initialized

6.6.2.15 parseCompoundResponse()

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

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

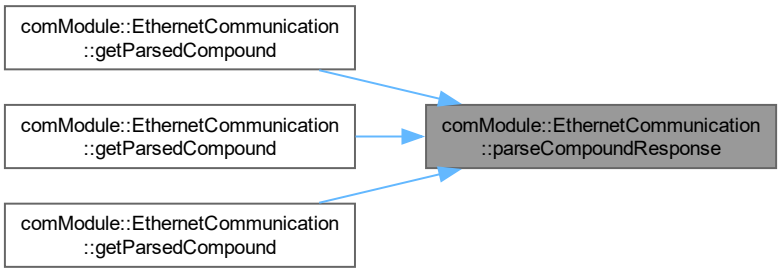
Parameters

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

Returns

Vector<float> -> containing parsed float values.

Here is the caller graph for this function:



6.6.2.16 receiveEthernetData()

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

Function to receive data over Ethernet.

Parameters

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

6.6.2.17 sendCommand()

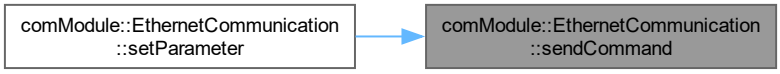
```
void EthernetCommunication::sendCommand (
            String command )
```

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

Parameters

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

Here is the caller graph for this function:



6.6.2.18 sendEthernetData()

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

Function to send data over Ethernet.

Parameters

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

6.6.2.19 sendJsonResponse()

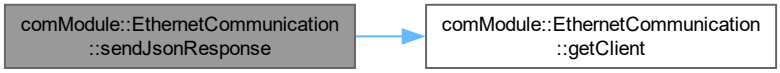
```
void EthernetCommunication::sendJsonResponse (
            const String & jsonBody )
```

Function to send the json response with the measurment data.

Parameters

jsonBody	-> jsonstring with the content needed

Here is the call graph for this function:



6.6.2.20 setCompound() [1/3]

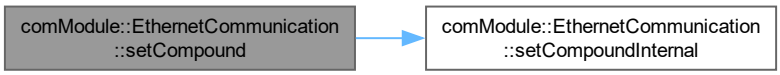
```
void EthernetCommunication::setCompound (
            Compound1 id,
            int index,
            String value )
```

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

Parameters

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

Here is the call graph for this function:



6.6.2.21 setCompound() [2/3]

```
void EthernetCommunication::setCompound (
```

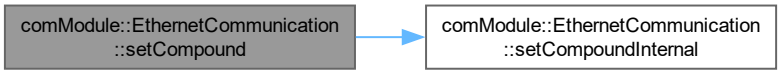
```
            Compound2 id,
            int index,
            String value )
```

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

Parameters

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

Here is the call graph for this function:



6.6.2.22 setCompound() [3/3]

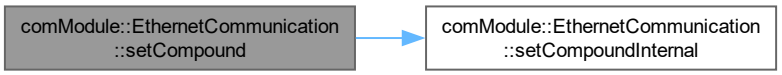
```
void EthernetCommunication::setCompound (
            Compound3 id,
            int index,
            String value )
```

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

Parameters

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

Here is the call graph for this function:



6.6.2.23 setCompoundInternal()

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

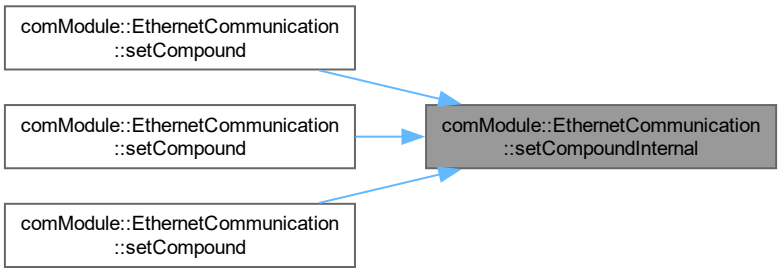
```
String compoundType,
unsigned long id,
int index,
String value )
```

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

Parameters

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

Here is the caller graph for this function:



6.6.2.24 setParameter()

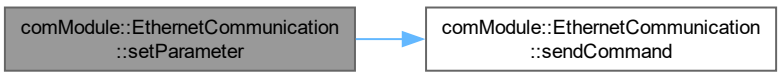
```
void EthernetCommunication::setParameter (
            Compound2 id,
            String value )
```

Setter for a parameter from the VAT slave.

Parameters

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

Here is the call graph for this function:



6.6.2.25 setSendDataFlag()

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

Function to get the current status of the flag.

Parameters

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

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

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

6.7 flybackModule::Flyback Class Reference

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

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

Public Member Functions

- void **initialize** ()

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

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

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

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

  Sets the state of the timer.
- SwitchStates getSwitchState ()

  Get the state of the Main-Switch.

- Measurement measure ()

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

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

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

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

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

  Getter Function to get the DutyCycle ∗.

6.7.1 Detailed Description

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

6.7.2 Member Function Documentation

6.7.2.1 getSwitchState()

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

Get the state of the Main-Switch.

Returns

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

6.7.2.2 getTimerState()

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

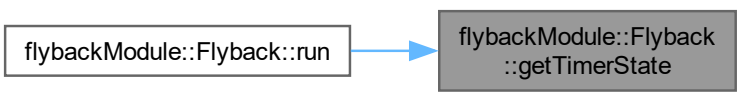
Returns the state of the timer.

Returns

true -> if the timer is initialized

false -> if the timer is not initialized

Here is the caller graph for this function:



6.7.2.3 isInitialized()

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

Get the state of the Flyback system.

Returns

true -> Flyback is initialized

false -> Flyback is not initialized

6.7.2.4 measure()

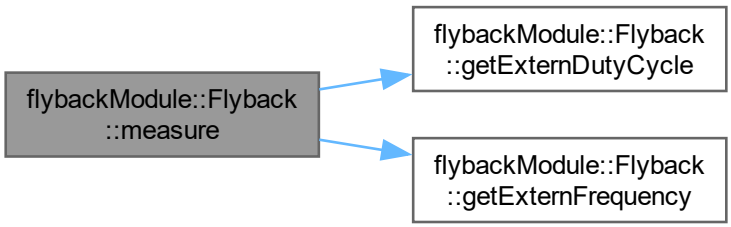
```
Measurement Flyback::measure ( )
```

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

Returns

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

Here is the call graph for this function:



6.7.2.5 setExternDutyCycle()

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

Function to get the desired DutyCycle from HAS.

Parameters

dutyCycle	-> the dutyCycle to change

6.7.2.6 setExternFrequency()

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

Function to get the desired Frequency from HAS.

Parameters

frequency	-> the frequency to change

6.7.2.7 setTimerState()

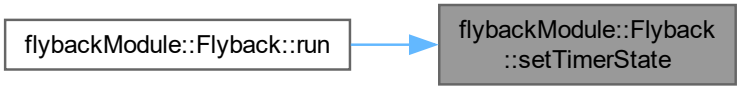
```
void Flyback::setTimerState (
            bool state )
```

Sets the state of the timer.

Parameters

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

Here is the caller graph for this function:



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

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

6.8 comModule::I2CCommunication Class Reference

Class to handle I2C communication.

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

Public Member Functions

- void beginI2C (uint8_t address)

  Function to initialize the I2C communication.
- void **endI2C** ()

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

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

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

  Function to check if the I2C communication is initialized.

6.8.1 Detailed Description

Class to handle I2C communication.

6.8.2 Member Function Documentation

6.8.2.1 beginI2C()

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

Function to initialize the I2C communication.

Parameters

address	-> The address of the I2C device

6.8.2.2 i2cRead()

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

Function to read data from the I2C device.

Parameters

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

Returns

size_t -> The number of bytes read

6.8.2.3 i2cWrite()

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

Function to write data to the I2C device.

Parameters

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

6.8.2.4 isInitialized()

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

Function to check if the I2C communication is initialized.

Returns

true -> if the I2C communication is initialized

false -> if the I2C communication is not initialized

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

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

6.9 jsonModule::JsonModuleInternals Class Reference

Public Member Functions

- void createJson (const char *key, const char *value)

  Create a Json object with a key and a value.
- void createJsonFloat (const char *key, float value)

  Create a Json Float object.
- void createJsonInt (const char *key, int value)

  Create a Json Int object.
- void createJsonString (const char *key, String &value)

  Create a Json String object.
- void createJsonStringConst (const char *key, const String &value)

Create a Json String Const object.

- void **sendJsonSerial** ()

  Function to send the Json object over the Serial connection.

- void sendJsonEthernet (const char *endpoint)

  Function to send the Json object over the Ethernet connection.

- String getJsonString () const

  Get the Json String object.

- void **clearJson** ()

  Clear the Json object.

- void **printJsonDocMemory** ()

  Prints information about the Json object.

Public Attributes

- size_t **jsonBuffer**

6.9.1 Member Function Documentation

6.9.1.1 createJson()

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

Create a Json object with a key and a value.

Parameters

key	-> The key of the Json object
value	-> The value of the Json object

6.9.1.2 createJsonFloat()

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

Create a Json Float object.

Parameters

key	-> The key of the Json object
value	-> The value of the Json object

6.9.1.3 createJsonInt()

```
void JsonModuleInternals::createJsonInt (
```

```
            const char * key,
            int value )
```

Create a Json Int object.

Parameters

key	-> The key of the Json object
value	-> The value of the Json object

6.9.1.4 createJsonString()

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

Create a Json String object.

Parameters

key	-> The key of the Json object
value	-> The value of the Json object

6.9.1.5 createJsonStringConst()

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

Create a Json String Const object.

Parameters

key	-> The key of the Json object
value	-> The value of the Json object

6.9.1.6 getJsonString()

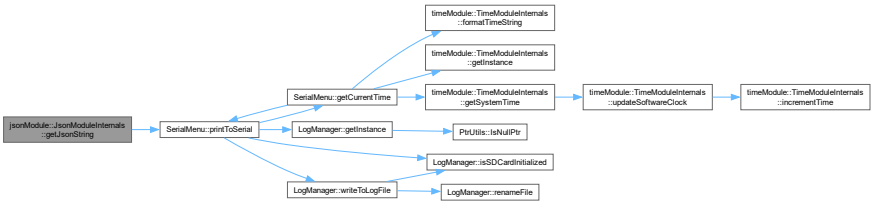
```
String JsonModuleInternals::getJsonString ( ) const
```

Get the Json String object.

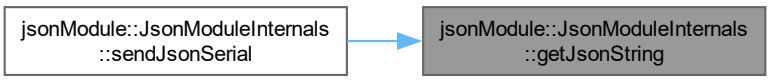
Returns

String -> The Json String object

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.1.7 sendJsonEthernet()

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

Function to send the Json object over the Ethernet connection.

Parameters

endpoint	-> The endpoint to send the Json object to

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

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

6.10 jsonModuleInternals Class Reference

Class for the JSON module internals.

`#include <jsonModule.h>`

6.10.1 Detailed Description

Class for the JSON module internals.

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

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

6.11 LockerBase Class Reference

Base class for the locker system.

`#include <lockerBase.h>`

Public Member Functions

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

6.11.1 Detailed Description

Base class for the locker system.

6.11.2 Member Function Documentation

6.11.2.1 lockEthernetScoped()

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

Returns

locker::ScopedLock

6.11.2.2 lockSerialScoped()

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

Returns

locker::ScopedLock

6.11.2.3 lockTemperatureScoped()

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

Returns

locker::ScopedLock

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

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

6.12 LogManager Class Reference

Public Member Functions

- void initSDCard (int cs)

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

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

  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.

6.12.1 Member Function Documentation

6.12.1.1 getCurrentTime()

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

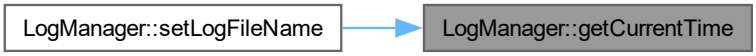
Getter for the current time.

Returns

The current time as a String

Here is the call graph for this function:

Here is the caller graph for this function:



6.12.1.2 getInstance()

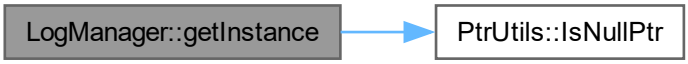
```
LogManager * LogManager::getInstance ( )  [static]
```

Get the Instance object.

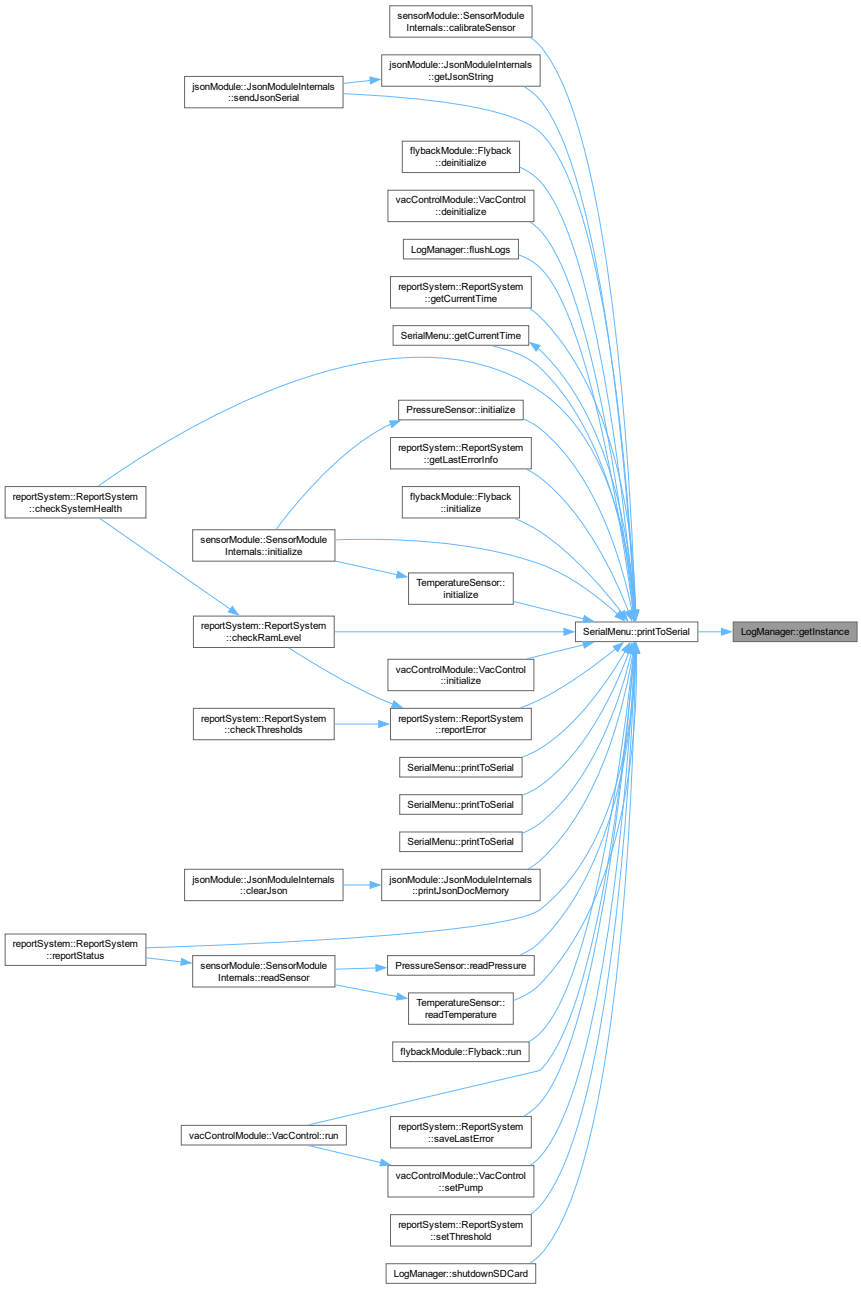
Returns

LogManager*

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.1.3 initSDCard()

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

Function to initialize the SD card.

Parameters

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

6.12.1.4 isSDCardInitialized()

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

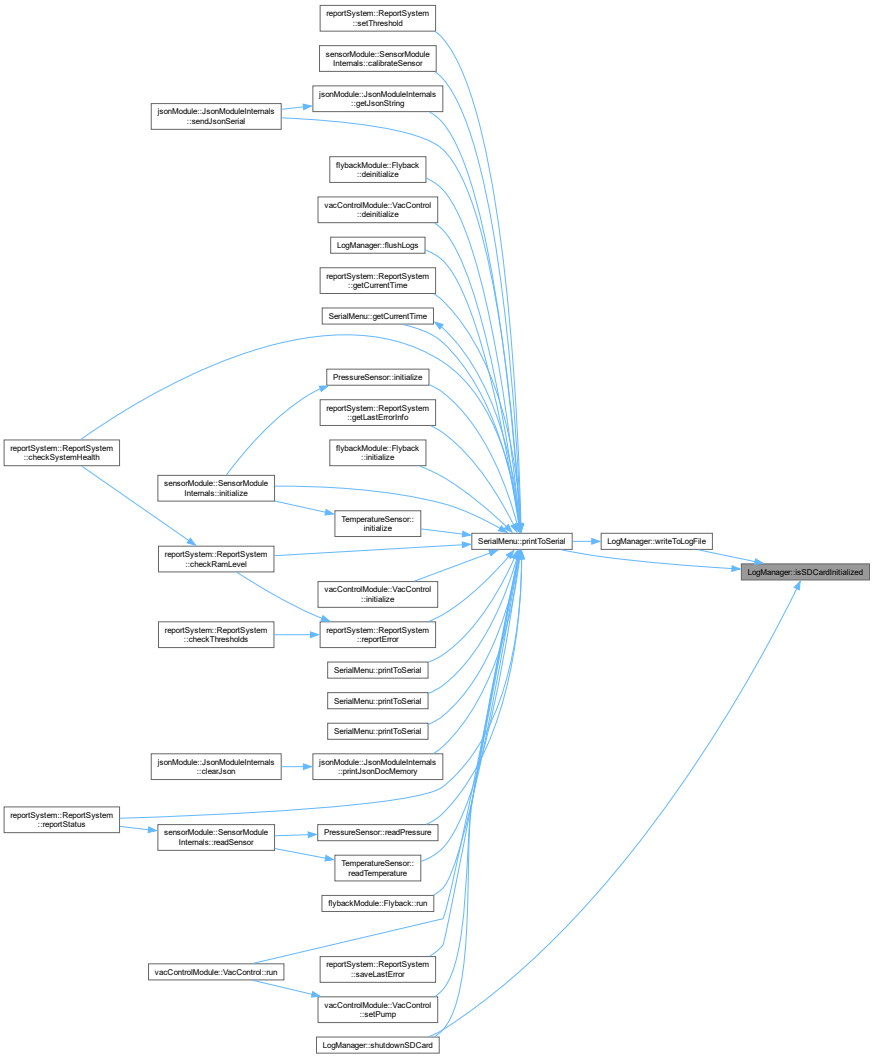
Function to check if the SD card is initialized.

Returns

true -> if the SD card is initialized

false -> if the SD card is not initialized

Here is the caller graph for this function:



6.12.1.5 renameFile()

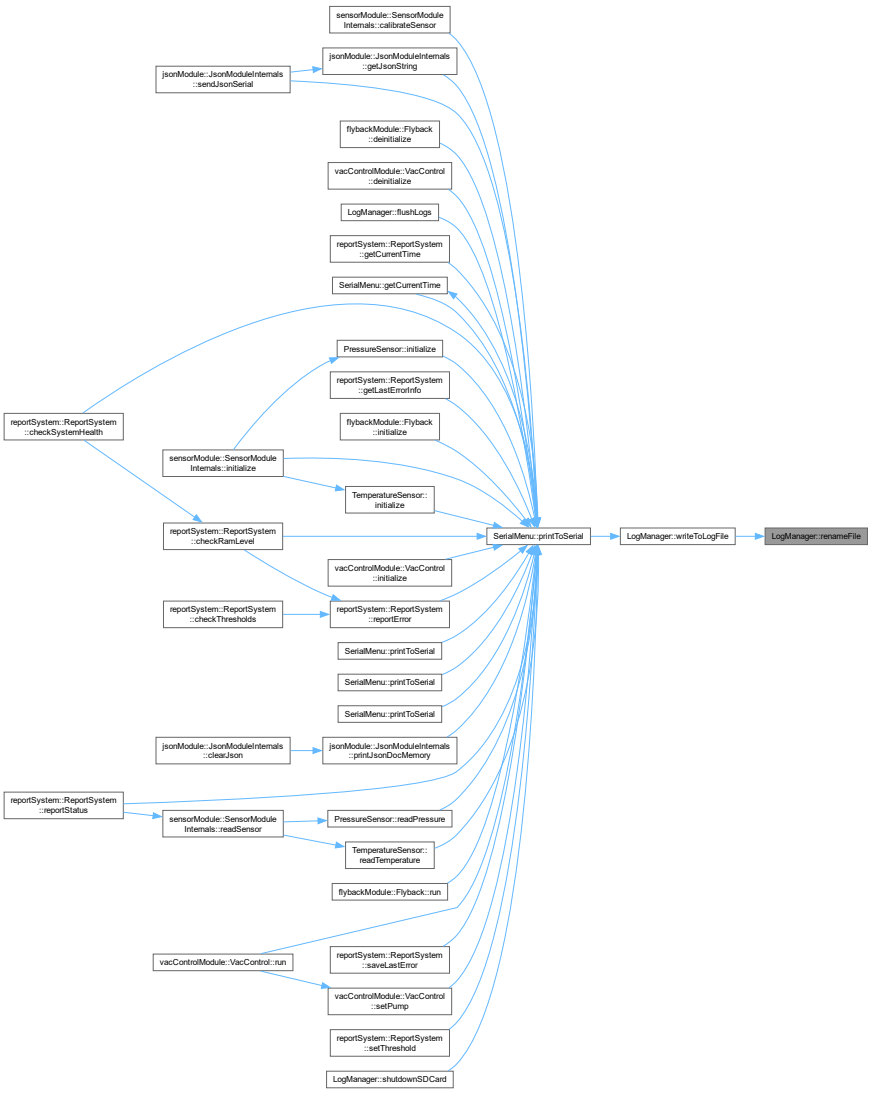
```
void LogManager::renameFile (
            const String & oldName,
            const String & newName )
```

Function to rename the currently written to file.

Parameters

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

Here is the caller graph for this function:



6.12.1.6 setLogFileName()

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

Set the Log File Name object.

Parameters

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

Here is the call graph for this function:

6.12.1.7 writeToLogFile()

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

Function to write a log message to the log file.

Parameters

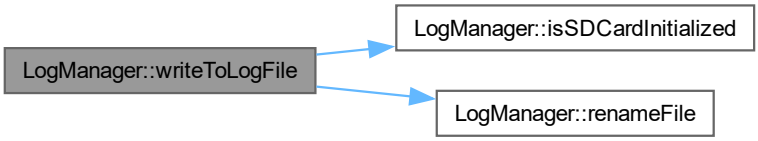
logMessage	-> The log message to write to the log file.

Returns

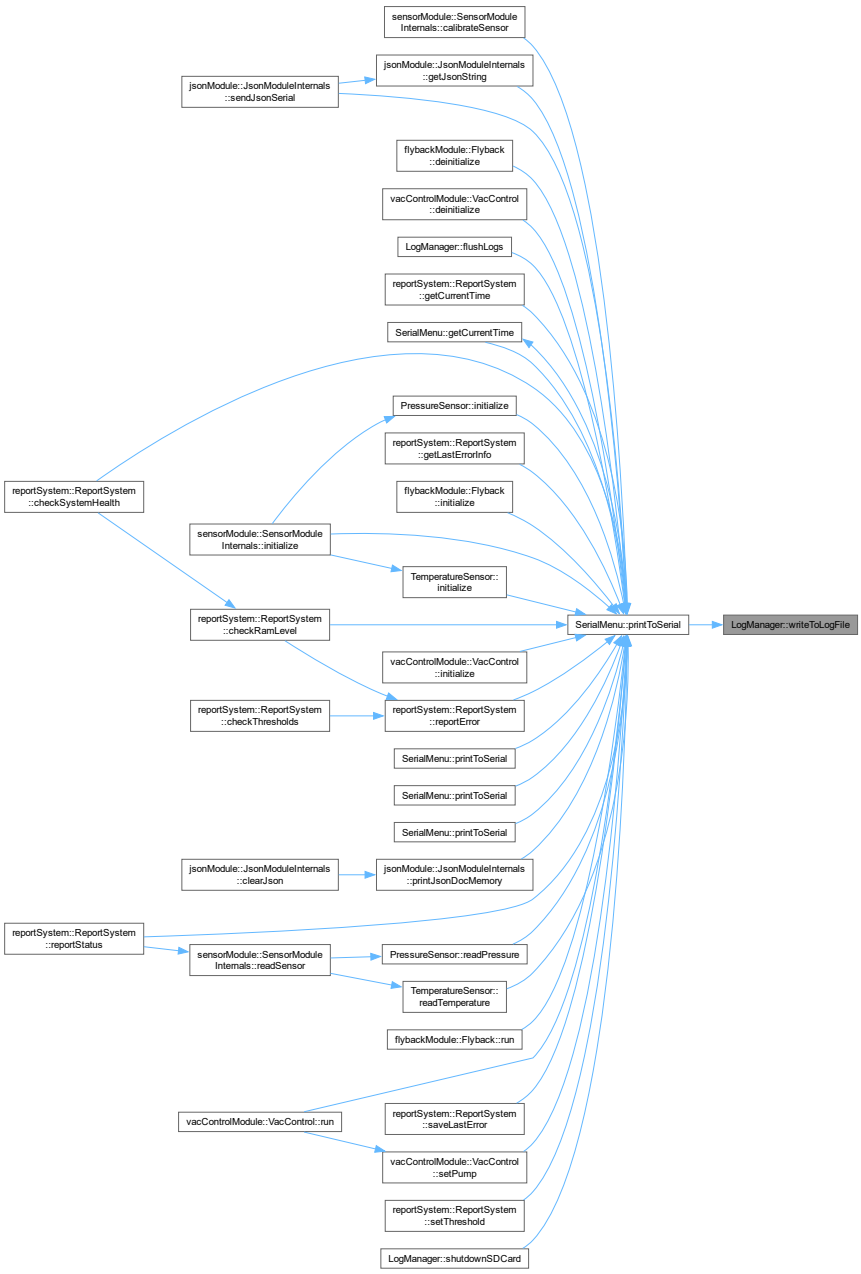
true -> if the log message was written successfully

false -> if the log message was not written successfully

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.13 LogMapper Class Reference

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

`#include <logManager.h>`

6.13.1 Detailed Description

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

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

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

6.14 flybackModule::Measurement Struct Reference

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

`#include <flyback.h>`

Public Attributes

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

6.14.1 Detailed Description

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

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

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

6.15 Measurement Struct Reference

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

`#include <vacControl.h>`

6.15.1 Detailed Description

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

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

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

6.16 MenuItem Struct Reference

Serial menu structure.

`#include <serialMenu.h>`

Public Attributes

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

6.16.1 Detailed Description

Serial menu structure.

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

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

6.17 Outputlevel Class Reference

Enum Class for the differnet Outputlevels.

`#include <serialMenu.h>`

6.17.1 Detailed Description

Enum Class for the differnet Outputlevels.

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

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

6.18 PointerWrapper< T > Class Template Reference

Tempalte class for wrapping a pointer.

`#include <ptrUtils.h>`

Public Member Functions

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

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

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

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

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

  Operator to access the pointer.

6.18.1 Detailed Description

template<typename T>
class PointerWrapper< T >

Tempalte class for wrapping a pointer.

Template Parameters

T	

6.18.2 Member Function Documentation

6.18.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.18.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.18.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.18.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.18.2.5 reset()

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

Function to reset the pointer.

Parameters

p	-> The pointer to reset to.

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

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

6.19 vacControlModule::Pressure Struct Reference

Public Attributes

- float **pressure**

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

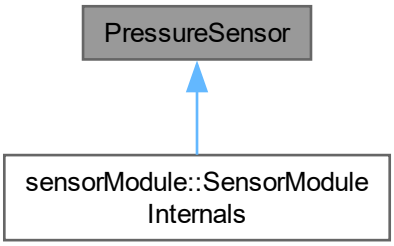
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacControl.h

6.20 PressureSensor Class Reference

Pressure sensor class.

`#include <pressure.h>`

Inheritance diagram for PressureSensor:



Public Member Functions

- void **initialize** ()

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

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

  Function to check if the pressure sensor is initialized.

6.20.1 Detailed Description

Pressure sensor class.

6.20.2 Member Function Documentation

6.20.2.1 isInitialized()

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

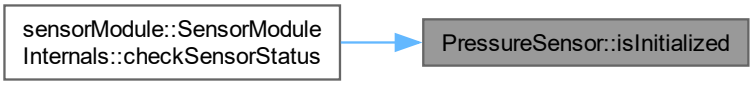
Function to check if the pressure sensor is initialized.

Returns

true -> if the pressure sensor is initialized

false -> if the pressure sensor is not initialized

Here is the caller graph for this function:



6.20.2.2 readPressure()

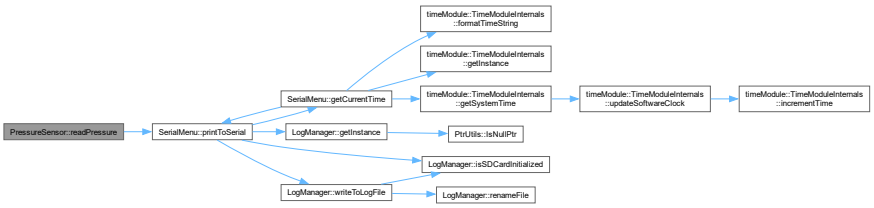
```
float PressureSensor::readPressure ( )
```

Function to read the pressure from the sensor.

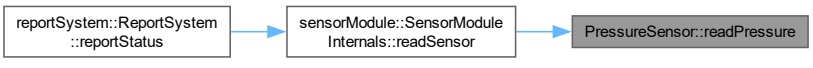
Returns

float -> The pressure value.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.21 PtrUtils Class Reference

Utility class for pointer operations.

`#include <ptrUtils.h>`

Static Public Member Functions

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

6.21.1 Detailed Description

Utility class for pointer operations.

6.21.2 Member Function Documentation

6.21.2.1 IsNullPtr()

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

Check if a pointer is nullptr.

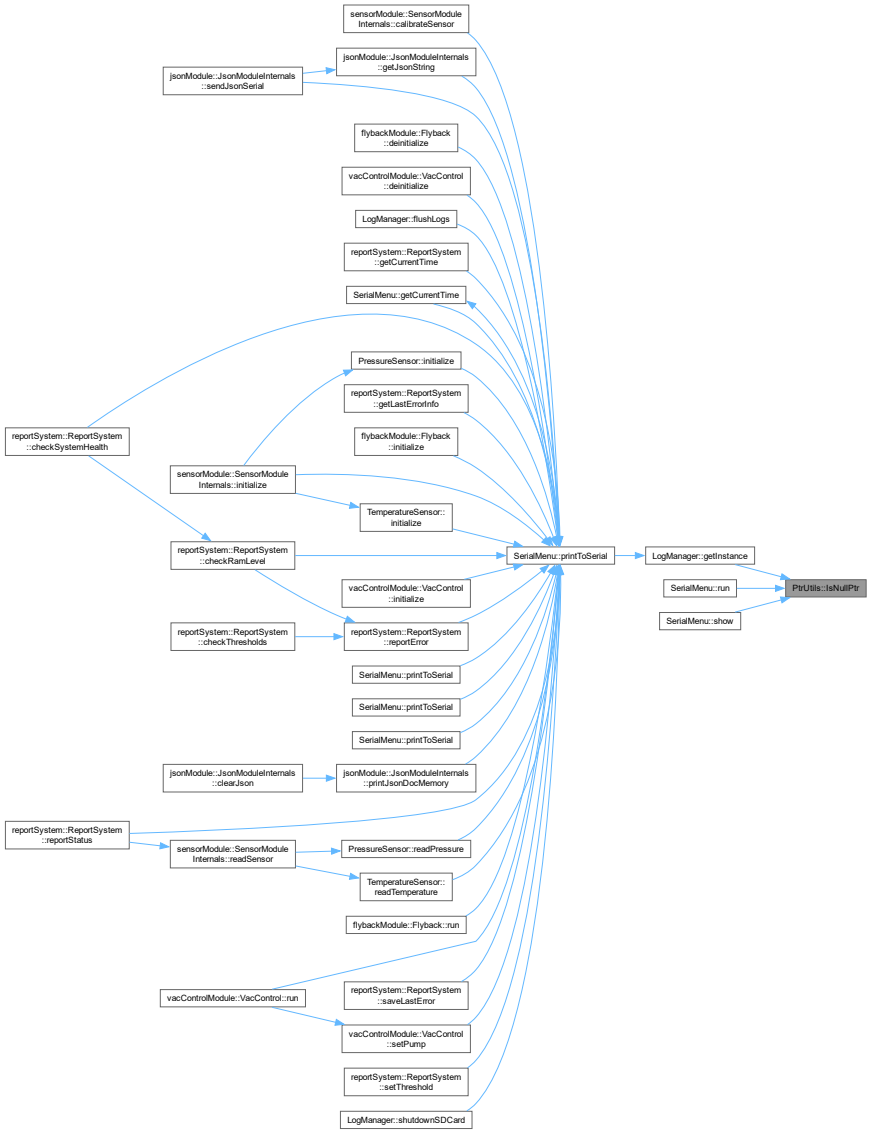
Parameters

ptr	Pointer to check.

Returns

true if the pointer is nullptr, false otherwise.

Here is the caller graph for this function:



6.21.2.2 IsValidPtr()

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

Check if a pointer is valid (not nullptr).

Parameters

ptr	Pointer to check.

Returns

true if the pointer is not nullptr, false otherwise.

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

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

6.22 reportSystem::ReportSystem Class Reference

Class for the report system.

`#include <reportSystem.h>`

Public Member Functions

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

Static Public Member Functions

- static void **initStackGuard** ()

  Initialize the Stack Guard.
- static bool detectStackOverflow ()

  Detect Stack Overflow.

6.22.1 Detailed Description

Class for the report system.

6.22.2 Member Function Documentation

6.22.2.1 checkRamLevel()

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

Function to check the SRAM level on the hostsystem.

Parameters

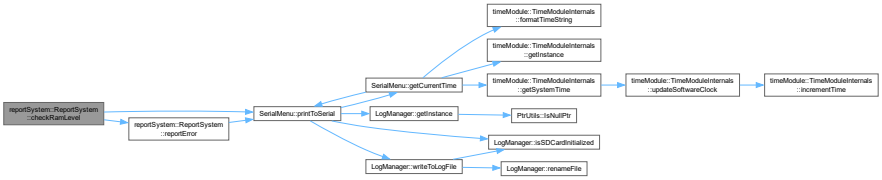
warningThreshold	-> first warning to get
criticalThreshold	-> last warning to get

Returns

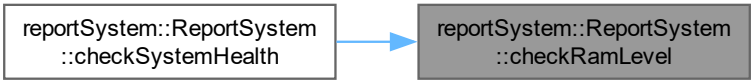
true -> if the level exceeded

false -> if the levels are withing the thresholds

Here is the call graph for this function:



Here is the caller graph for this function:



6.22.2.2 checkSystemHealth()

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

Function to check the system health of the uC.

Parameters

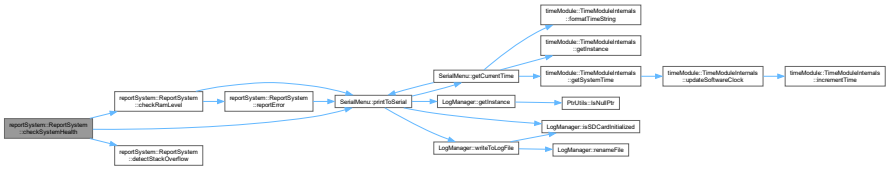
memoryThreshold	-> The memory threshold to check
checkEth	-> Check the Ethernet connection
checkSpi	-> Check the SPI connection
checkI2c	-> Check the I2C connection
checkTemp	-> Check the temperature sensor
checkPress	-> Check the pressure sensor

Returns

true -> if the system is healthy

false -> if the system is not healthy

Here is the call graph for this function:



6.22.2.3 checkThresholds()

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

Check the thresholds for the temperature and pressure sensors.

Parameters

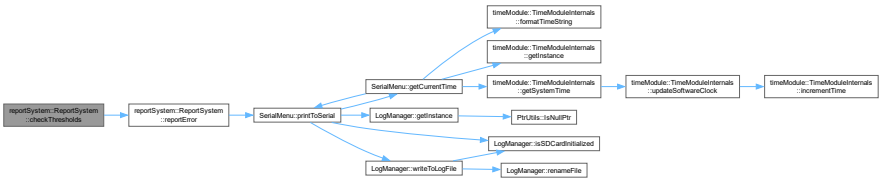
currentTemp	-> The current temperature
currentPressure	-> The current pressure

Returns

true -> if the thresholds are met

false -> if the thresholds are not met

Here is the call graph for this function:



6.22.2.4 detectStackOverflow()

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

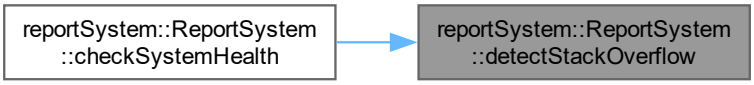
Detect Stack Overflow.

Returns

true -> if the stack has overflowed

false -> if the stack has not overflowed

Here is the caller graph for this function:



6.22.2.5 getCPULoad()

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

Getter for the CPU Load.

Returns

float -> The CPU Load

6.22.2.6 getCurrentTime()

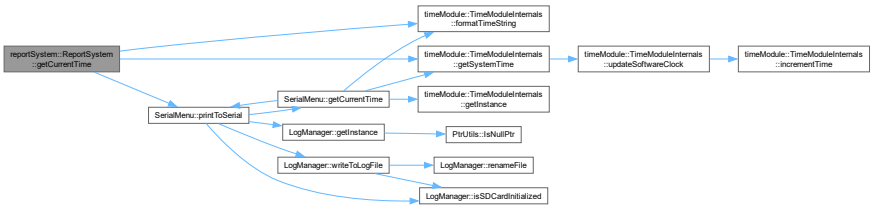
```
String ReportSystem::getCurrentTime ( )
```

Get the Current Time of the system.

Returns

String -> The current time

Here is the call graph for this function:



6.22.2.7 getLastError()

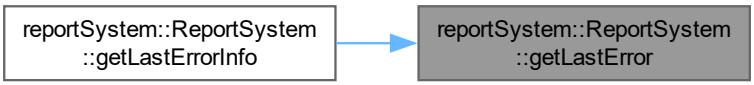
```
String ReportSystem::getLastError ( )
```

Get the Last Error message from EEPROM.

Returns

String -> The last error message

Here is the caller graph for this function:



6.22.2.8 getLastErrorInfo()

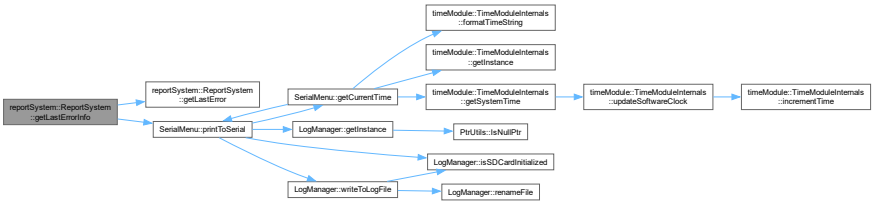
```
bool ReportSystem::getLastErrorInfo ( )
```

Get the Last Error message from EEPROM.

Returns

bool -> used by the Endpoint to report to HAS

Here is the call graph for this function:



6.22.2.9 getMemoryStatus()

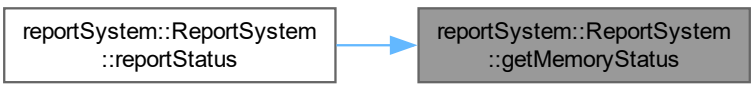
```
String ReportSystem::getMemoryStatus ( )
```

Get the Memory Status of the system.

Returns

String -> The memory status

Here is the caller graph for this function:



6.22.2.10 getStackDump()

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

Get the Stack Dump of the system.

Returns

String -> The stack dump

6.22.2.11 reportError()

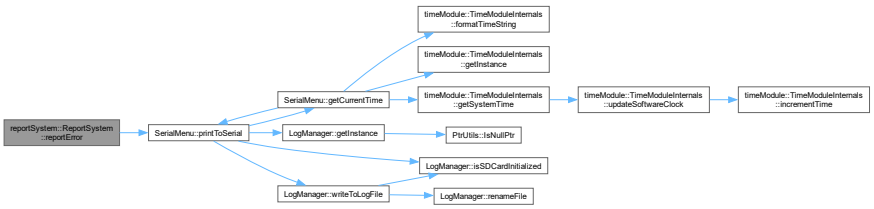
```
void ReportSystem::reportError (
            const char * errorMessage )
```

Function to log an error message.

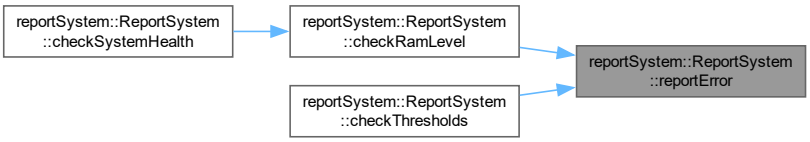
Parameters

errorMessage	-> The error message to log

Here is the call graph for this function:



Here is the caller graph for this function:



6.22.2.12 reportStatus()

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

Function to report the status of the system.

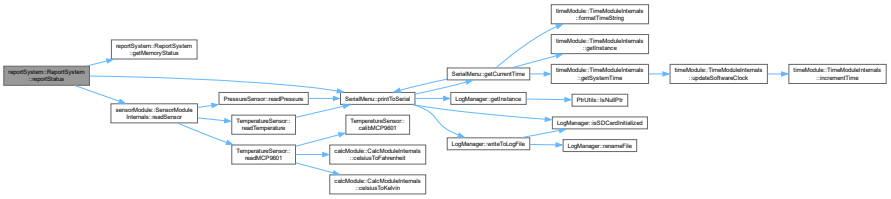
Parameters

active	-> The status of the system

Returns

String -> The status of the system

Here is the call graph for this function:



6.22.2.13 saveLastError()

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

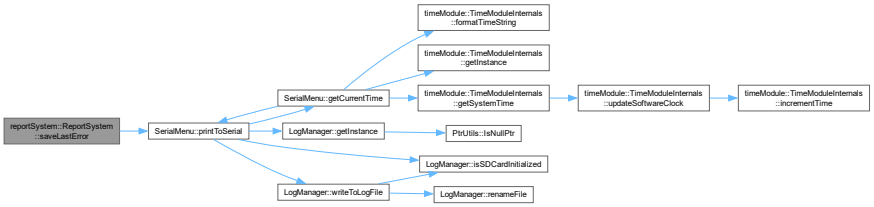
Saves last error message to EEPROM.

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

Parameters

error	-> The error message to save

Here is the call graph for this function:



6.22.2.14 setThreshold()

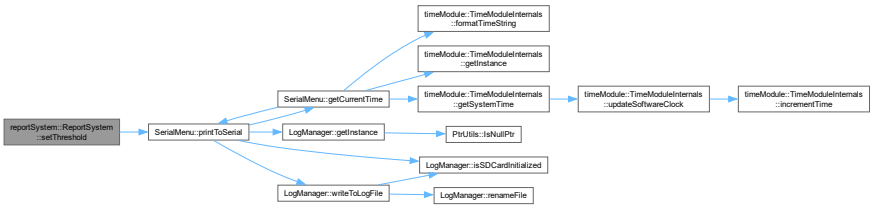
```
void ReportSystem::setThreshold (
            float tempThreshold,
            float pressureThreshold )
```

Set Thresholds for the pressure and temperature sensors.

Parameters

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

Here is the call graph for this function:



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

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

6.23 locker::ScopedLock Class Reference

Scoped lock class for mutexes.

`#include <scopedLock.h>`

Public Member Functions

- ScopedLock (frt::Mutex &mutex)

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

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

6.23.1 Detailed Description

Scoped lock class for mutexes.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 ScopedLock() [1/2]

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

Construct a new Scoped Lock object.

Parameters

mutex	-> The mutex to lock

6.23.2.2 ScopedLock() [2/2]

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

Construct a new Scoped Lock object.

Parameters

other	-> The other ScopedLock object to move from

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

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

6.24 ScopedPointer< T > Class Template Reference

Template class for a Scoped Pointer.

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

Public Member Functions

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

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

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

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

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

  Operator to access the pointer.

6.24.1 Detailed Description

template<typename T>
class ScopedPointer< T >

Template class for a Scoped Pointer.

Template Parameters

T	-> The type of the pointer.

6.24.2 Member Function Documentation

6.24.2.1 get()

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

Function to get the pointer.

Returns

T* -> The pointer.

6.24.2.2 operator*()

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

Operator to dereference the pointer.

Returns

T& -> The dereferenced pointer.

6.24.2.3 operator->()

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

Operator to access the pointer.

Returns

T* -> The pointer.

6.24.2.4 release()

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

Function to release the pointer.

Returns

T* -> The released pointer.

6.24.2.5 reset()

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

Function to reset the pointer.

Parameters

p	-> The pointer to reset to.

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

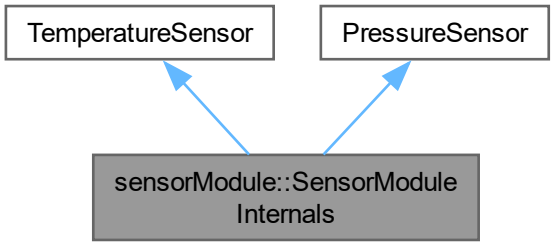
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h

6.25 sensorModule::SensorModuleInternals Class Reference

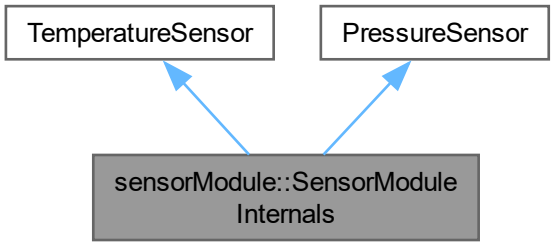
Class for the sensor module internals.

`#include <sensorModule.h>`

Inheritance diagram for sensorModule::SensorModuleInternals:



Collaboration diagram for sensorModule::SensorModuleInternals:



Public Member Functions

- void **initialize** ()

  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, String context)

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.

6.25.1 Detailed Description

Class for the sensor module internals.

6.25.2 Member Function Documentation

6.25.2.1 calibrateSensor()

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

Function to calibrate the sensor.

Parameters

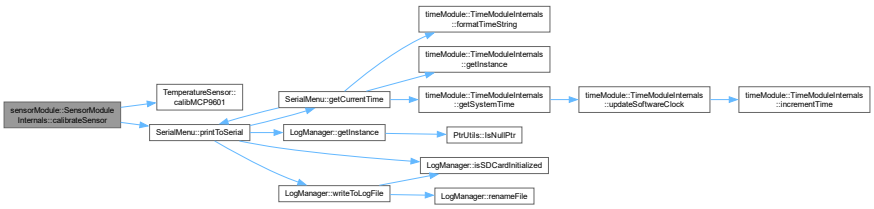
type	-> The type of the sensor to calibrate.

Returns

true -> if the sensor was calibrated successfully

false -> if the sensor was not calibrated successfully

Here is the call graph for this function:



6.25.2.2 checkSensorStatus()

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

Function to check the status of the sensor.

Parameters

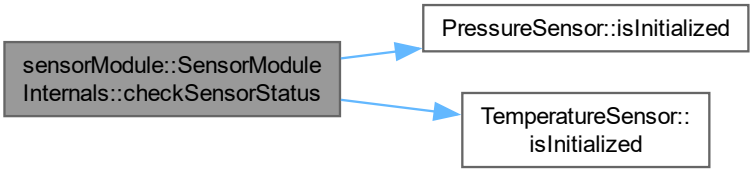
type	-> The type of the sensor to check.

Returns

true -> if the sensor is healthy

false -> if the sensor is not healthy

Here is the call graph for this function:



6.25.2.3 readSensor()

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

Function to read the sensor.

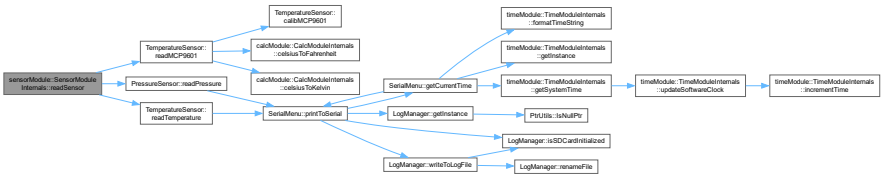
Parameters

type	-> The type of the sensor to read.

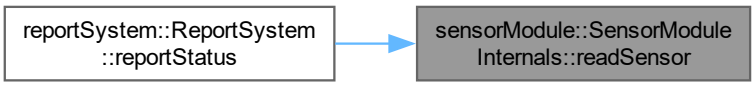
Returns

float -> The value of the sensor.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.26 comModule::SerialCommunication Class Reference

Class to handle Serial communication.

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

Public Member Functions

- void beginSerial (long baudRate)

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

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

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

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

  Function to check if the serial communication is initialized.

6.26.1 Detailed Description

Class to handle Serial communication.

6.26.2 Member Function Documentation

6.26.2.1 beginSerial()

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

Function to start the serial communication.

Parameters

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

6.26.2.2 isInitialized()

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

Function to check if the serial communication is initialized.

Returns

true -> if the serial communication is initialized

false -> if the serial communication is not initialized

6.26.2.3 receiveSerialData()

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

Function to receive data over serial.

Parameters

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

6.26.2.4 sendSerialData()

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

Function to end the serial communication.

Parameters

data	-> The data to send

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

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

6.27 SerialMenu Class Reference

Class for the serial menu.

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

Public Types

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

Public Member Functions

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

Static Public Member Functions

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

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

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

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

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

  Getter for the current time.

6.27.1 Detailed Description

Class for the serial menu.

6.27.2 Member Function Documentation

6.27.2.1 getCurrentTime()

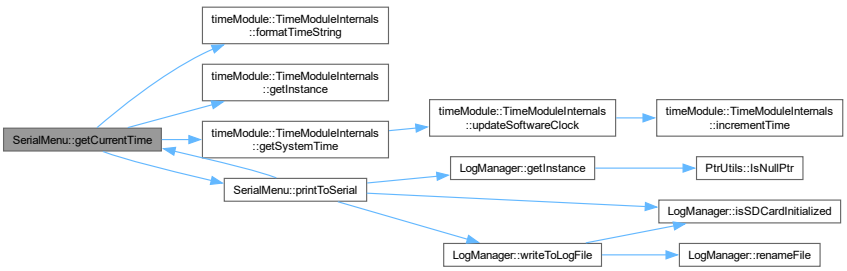
```
String SerialMenu::getCurrentTime ( )  [static]
```

Getter for the current time.

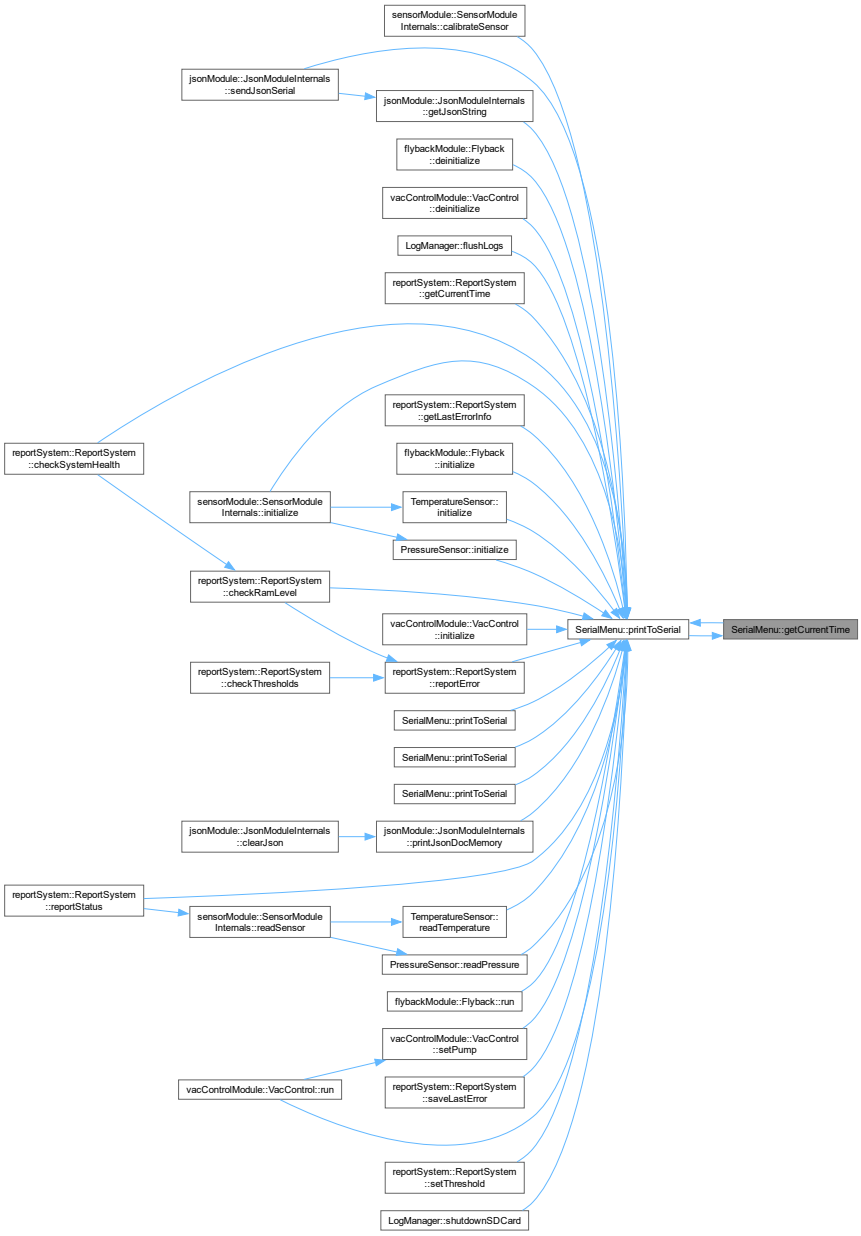
Returns

The current time as a String

Here is the call graph for this function:



Here is the caller graph for this function:



6.27.2.2 load()

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

Function to load the menu items.

Parameters

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

6.27.2.3 printToSerial() [1/4]

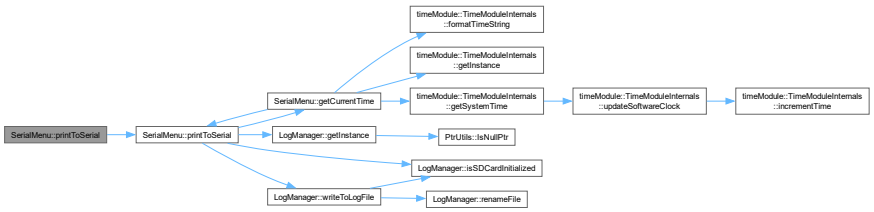
```
void SerialMenu::printToSerial (
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.4 printToSerial() [2/4]

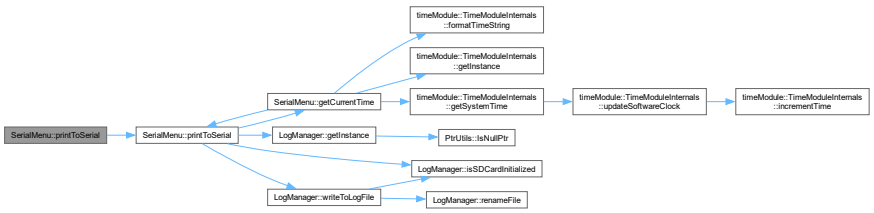
```
void SerialMenu::printToSerial (
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.5 printToSerial() [3/4]

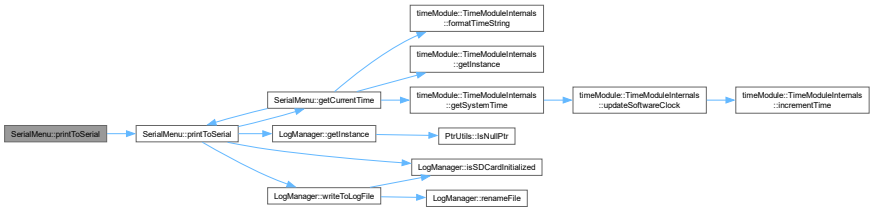
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const __FlashStringHelper * message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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

Parameters

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

Here is the call graph for this function:



6.27.2.6 printToSerial() [4/4]

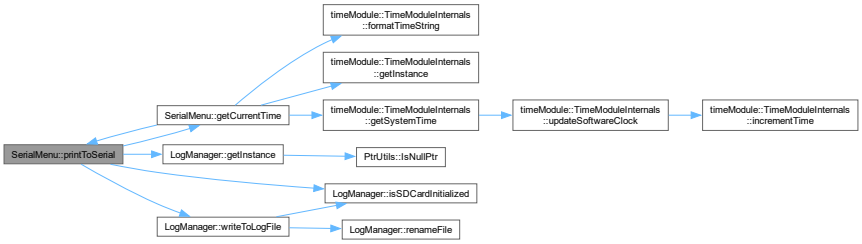
```
void SerialMenu::printToSerial (
            OutputLevel level,
            const String & message,
            bool newLine = true,
            bool logMessage = false )  [static]
```

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

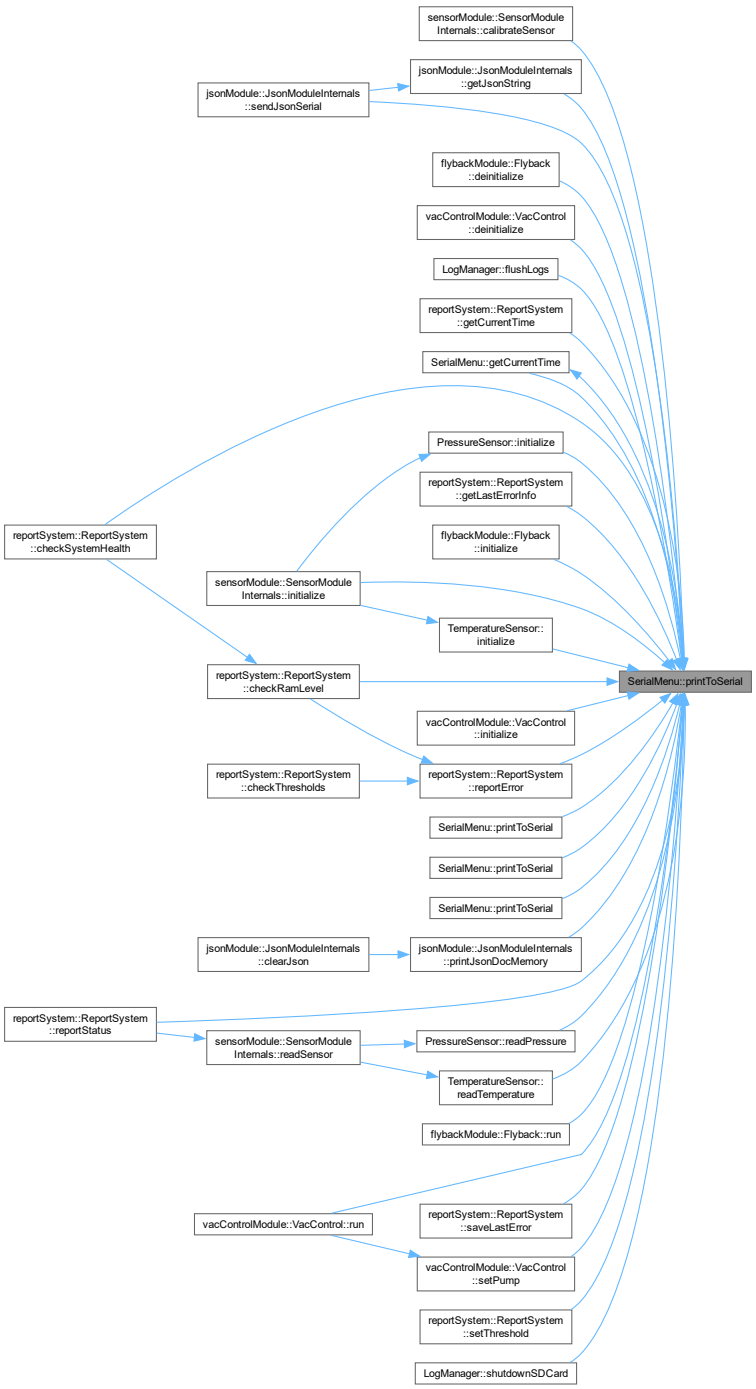
Parameters

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.28 comModule::SPICommunication Class Reference

Class to handle SPI communication.

`#include <SPII.h>`

Public Member Functions

- void **beginSPI** ()

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

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

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

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

  Function to check if the SPI communication is initialized.

6.28.1 Detailed Description

Class to handle SPI communication.

6.28.2 Member Function Documentation

6.28.2.1 isInitialized()

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

Function to check if the SPI communication is initialized.

Returns

true -> if the SPI communication is initialized

false -> if the SPI communication is not initialized

6.28.2.2 spiRead()

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

Function to read data over SPI.

Parameters

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

6.28.2.3 spiWrite()

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

Function to write data over SPI.

Parameters

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

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

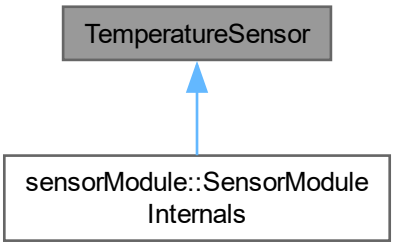
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h
- C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.cpp

6.29 TemperatureSensor Class Reference

Temperature sensor class.

`#include <temperature.h>`

Inheritance diagram for TemperatureSensor:



Public Member Functions

- void **initialize** ()

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

  Function to read the temperature from the sensor.
- float 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.

6.29.1 Detailed Description

Temperature sensor class.

6.29.2 Member Function Documentation

6.29.2.1 calibMCP9601()

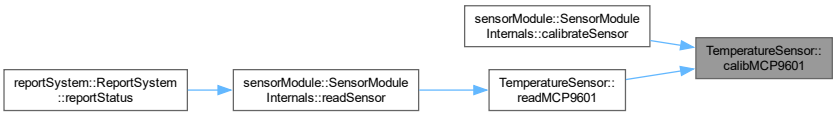
```
uint8_t TemperatureSensor::calibMCP9601 (
            SensorID sensor )
```

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:



6.29.2.2 isInitialized()

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

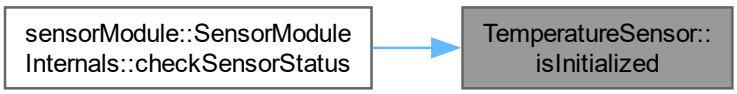
Check if the temperature sensor is initialized.

Returns

true -> if the temperature sensor is initialized

false -> if the temperature sensor is not initialized

Here is the caller graph for this function:



6.29.2.3 readMCP9601()

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

Function to read form specific sensor MCP9601.

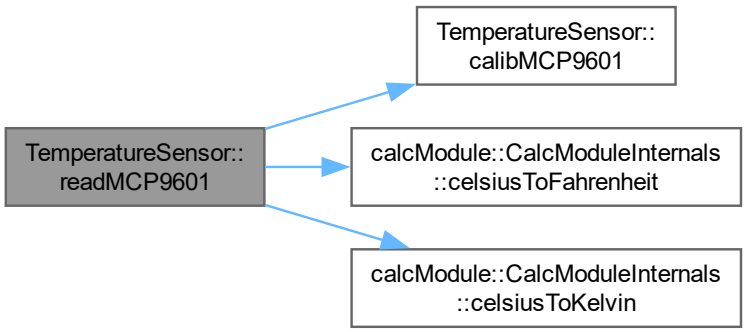
Parameters

unit	-> Choose the unit ℃, F, K

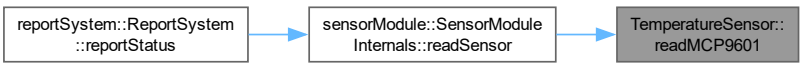
Returns

-> The temperature value

Here is the call graph for this function:



Here is the caller graph for this function:



6.29.2.4 readTemperature()

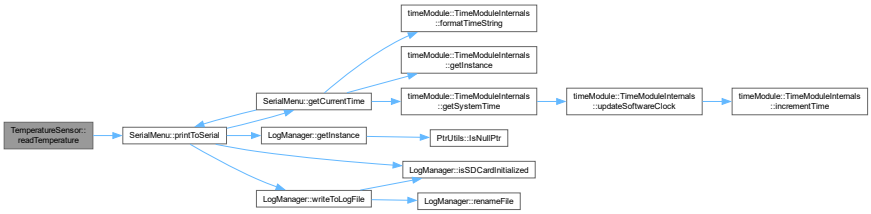
```
float TemperatureSensor::readTemperature ( )
```

Function to read the temperature from the sensor.

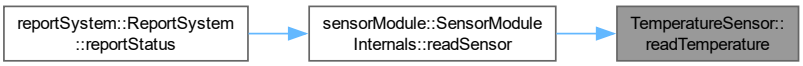
Returns

float -> The temperature value.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

6.30 timeModule::TimeModuleInternals Class Reference

Class to handle Systemtime.

`#include <timeModule.h>`

Public Member Functions

- bool setTimeFromHas (const String &timeString)

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

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

  Updates the software clock.
- DateTimeStruct getSystemTime ()

  Get the System Time object.

Static Public Member Functions

- static void incrementTime (DateTimeStruct *dt)

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

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

  Get the Instance object, Singleton pattern.

6.30.1 Detailed Description

Class to handle Systemtime.

6.30.2 Member Function Documentation

6.30.2.1 formatTimeString()

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

Function to format the time to a string.

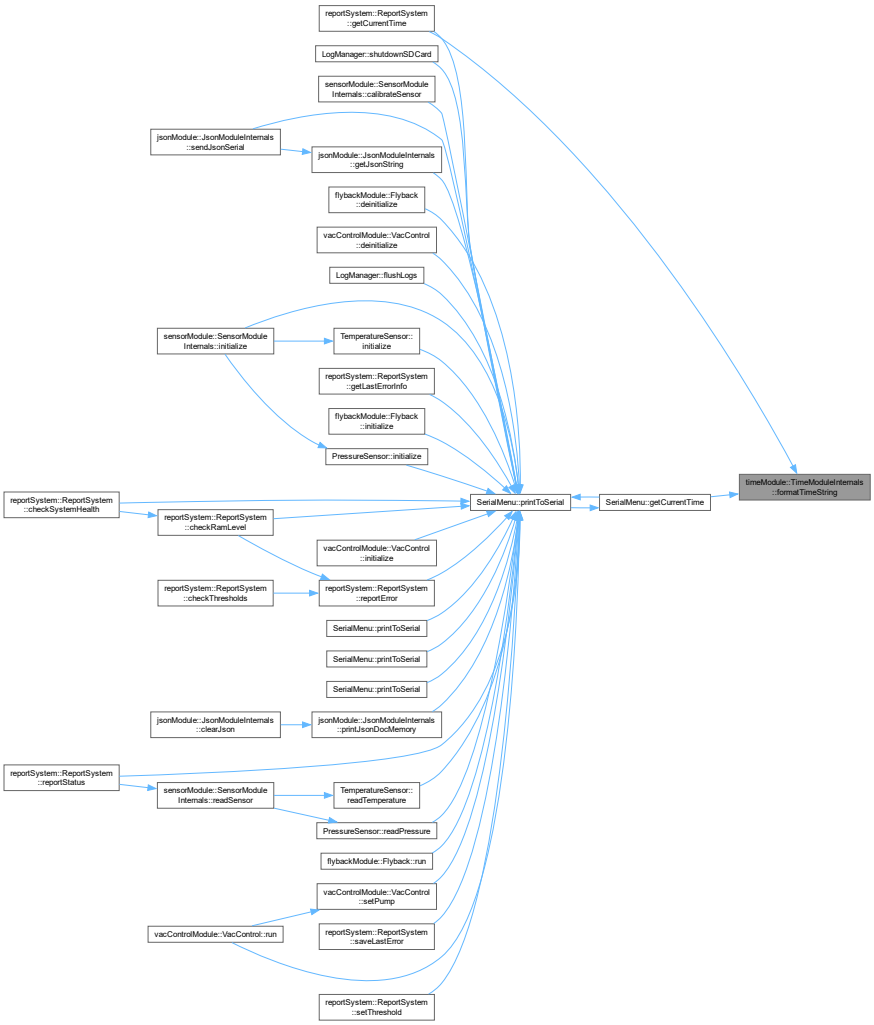
Parameters

dt	-> DateTimeStruct to format

Returns

String -> The formatted time.

Here is the caller graph for this function:



6.30.2.2 getInstance()

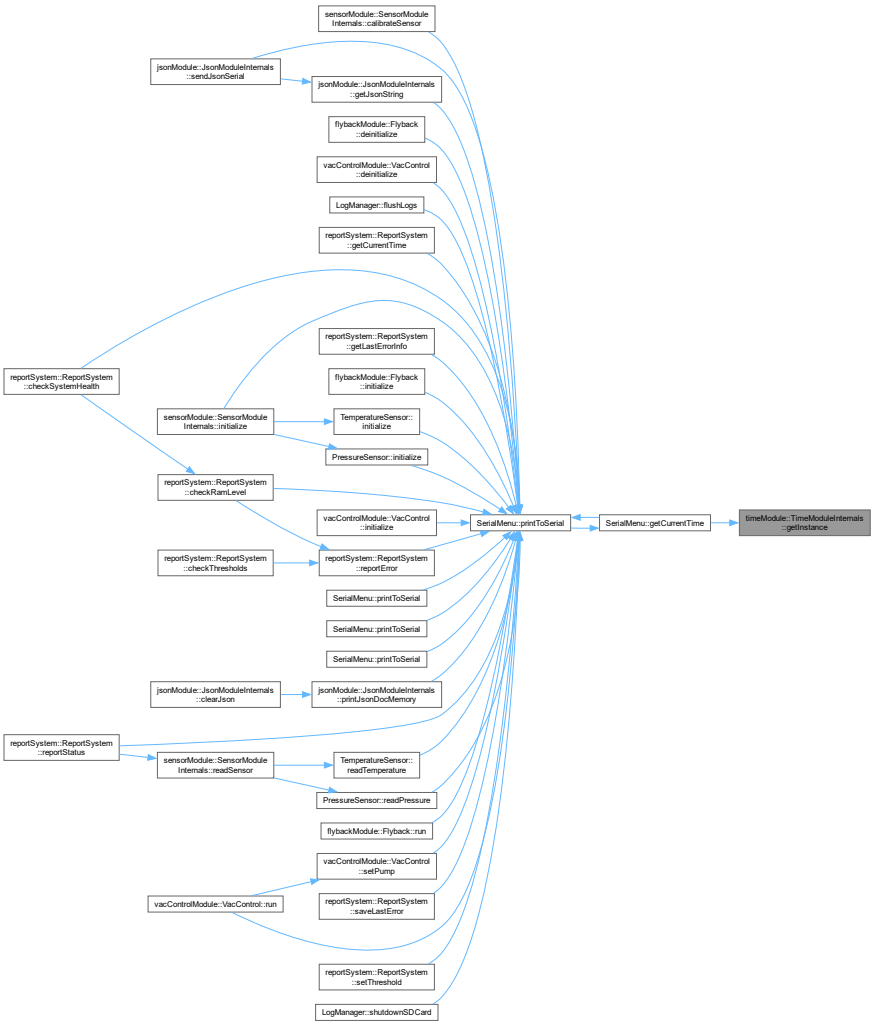
```
TimeModuleInternals * TimeModuleInternals::getInstance ( )  [static]
```

Get the Instance object, Singleton pattern.

Returns

TimeModuleInternals* -> The instance of the TimeModuleInternals.

Here is the caller graph for this function:



6.30.2.3 getSystemTime()

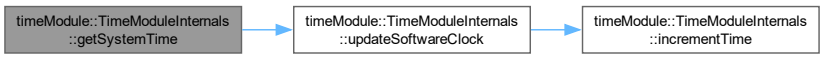
```
DateTimeStruct TimeModuleInternals::getSystemTime ( )
```

Get the System Time object.

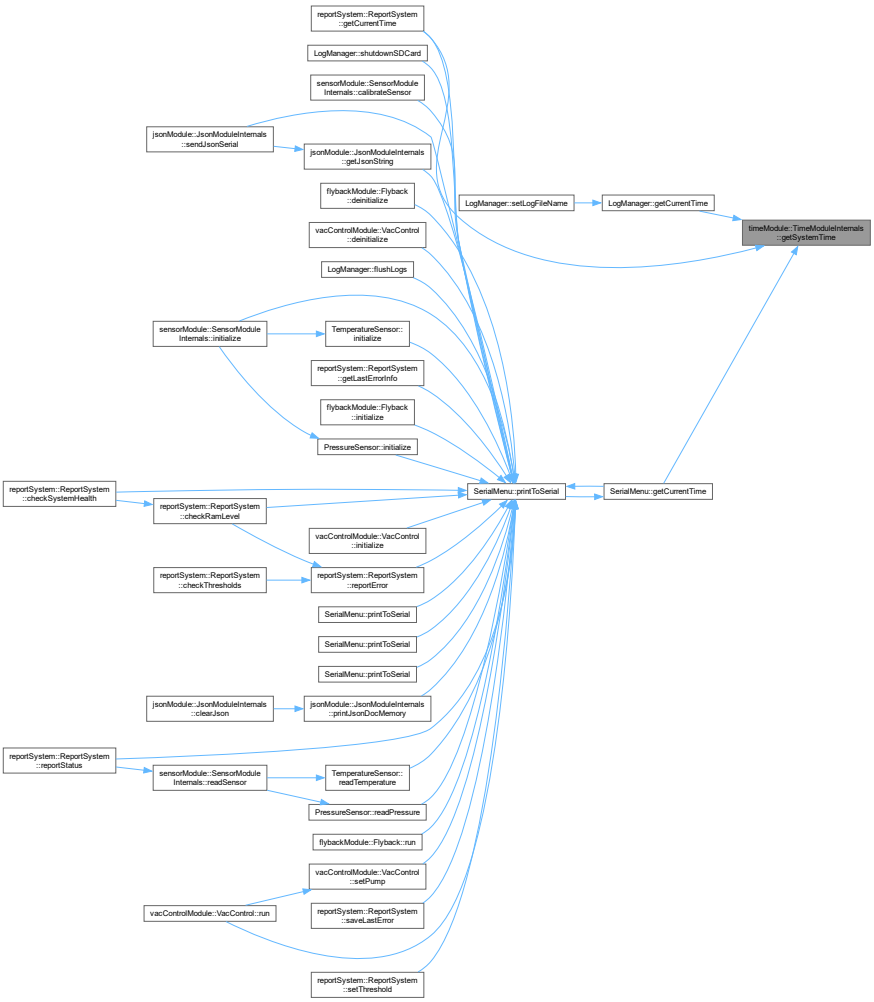
Returns

DateTimeStruct -> The system time.

Here is the call graph for this function:



Here is the caller graph for this function:



6.30.2.4 incrementTime()

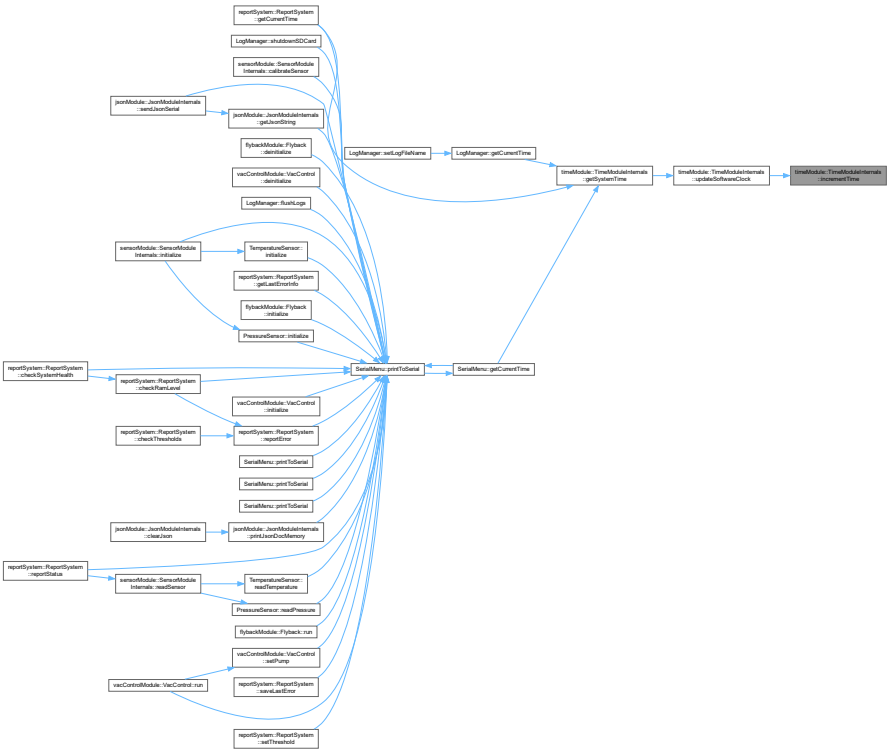
```
void TimeModuleInternals::incrementTime (
            DateTimeStruct * dt )  [static]
```

Function to increment the time of the system.

Parameters

dt	-> DateTimeStruct to increment time

Here is the caller graph for this function:



6.30.2.5 setSystemTime()

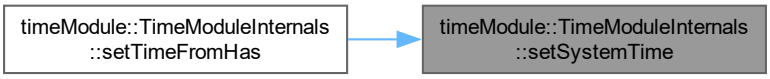
```
void TimeModuleInternals::setSystemTime (
            const DateTimeStruct & dt )
```

Set the System Time object of the system.

Parameters

dt	-> DateTimeStruct to set the system time to.

Here is the caller graph for this function:



6.30.2.6 setTimeFromHas()

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

Set the Time From Has object to the system time.

Parameters

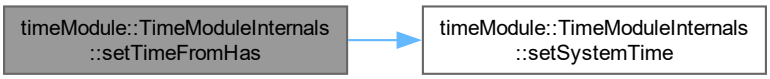
timeString	-> The time string to set the system time to.

Returns

true -> if the time was set successfully

false -> if the time was not set successfully

Here is the call graph for this function:



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

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

6.31 vacControlModule::VacControl Class Reference

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

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

Public Member Functions

- void **initialize** ()

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

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

  Get the state of the VacControl system.
- SwitchStates getSwitchState ()

  Returns the state of the main switch.
- Scenarios getScenario ()

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

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

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

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

  Set the Pump flag.
- void run ()

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

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

  Getter function to retrieve the current external scenario state.
- void externProcess ()

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

  Sets the external pressure value.
- float getExternPressure ()

  Gets the external pressure value.

6.31.1 Detailed Description

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

6.31.2 Member Function Documentation

6.31.2.1 externProcess()

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

Process external data for scenarios (currently unused)

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

6.31.2.2 getExternPressure()

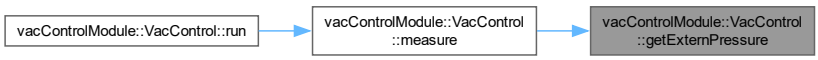
```
float VacControl::getExternPressure ( )
```

Gets the external pressure value.

Returns

The current external pressure value

Here is the caller graph for this function:



6.31.2.3 getExternScenario()

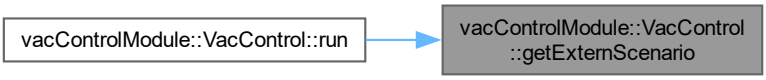
```
int VacControl::getExternScenario ( )
```

Getter function to retrieve the current external scenario state.

Returns

The current external scenario state (integer)

Here is the caller graph for this function:



6.31.2.4 getScenario()

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

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

This function decides which scenario to run based on the current state of the system.

6.31.2.5 getScenarioFromPotValue()

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

Determines the scenario based on the potentiometer value.

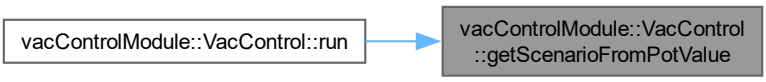
Parameters

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

Returns

The corresponding scenario based on the potentiometer value

Here is the caller graph for this function:



6.31.2.6 getSwitchState()

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

Returns the state of the main switch.

Returns

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

6.31.2.7 isInitialized()

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

Get the state of the VacControl system.

Returns

true -> VacControl is initialized and ready

false -> VacControl is not initialized

6.31.2.8 measure()

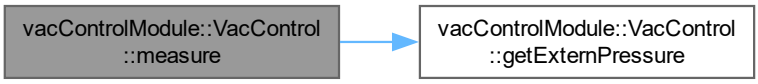
```
Pressure VacControl::measure ( )
```

Measures the actual pressure of the system.

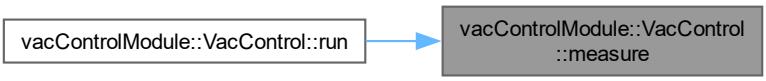
Returns

Measurement -> A Measurement object containing the current pressure

Here is the call graph for this function:



Here is the caller graph for this function:

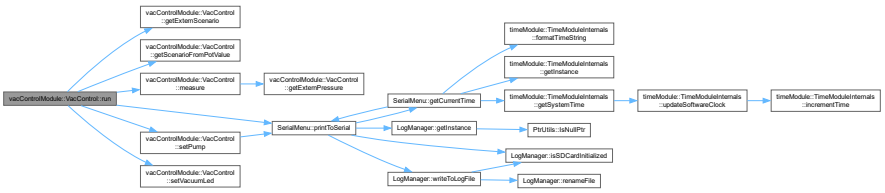


6.31.2.9 run()

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

Runs the main control loop for the VacControl system.

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



6.31.2.10 setExternPressure()

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

Sets the external pressure value.

Parameters

pressure	The external pressure value to set

6.31.2.11 setExternScenario()

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

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

Parameters

pressure	The external scenario pressure value

6.31.2.12 setPump()

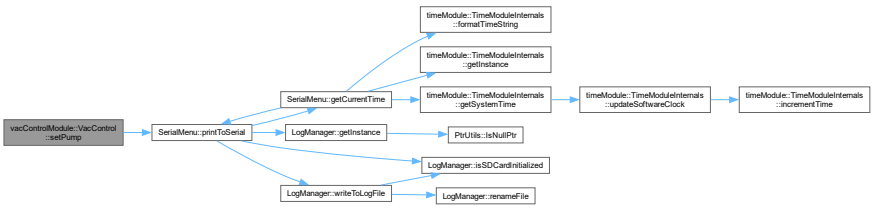
```
void VacControl::setPump (
            bool flag )
```

Set the Pump flag.

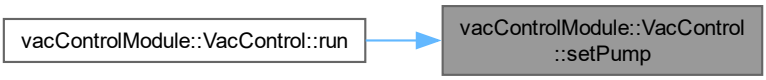
Parameters

flag	This is the boolean flag to set

Here is the call graph for this function:



Here is the caller graph for this function:



6.31.2.13 setVacuumLed()

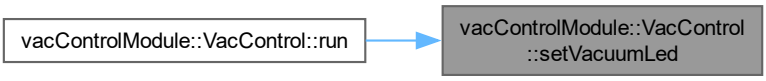
```
void VacControl::setVacuumLed (
            float pressure,
            float targetPressure )
```

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

Parameters

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

Here is the caller graph for this function:



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

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

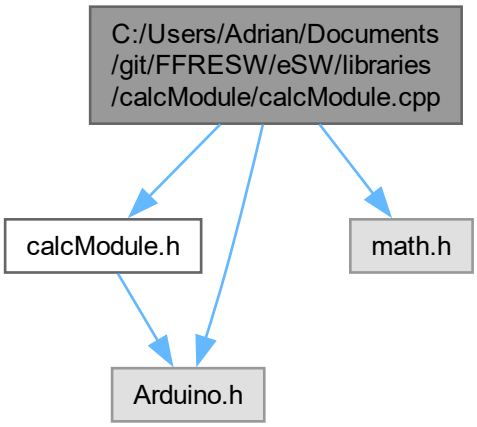
Chapter 7

File Documentation

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

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

Include dependency graph for calcModule.cpp:



7.1.1 Detailed Description

Author

your name (you@domain.com)

Version

0.1

Date

2024-09-28

Copyright

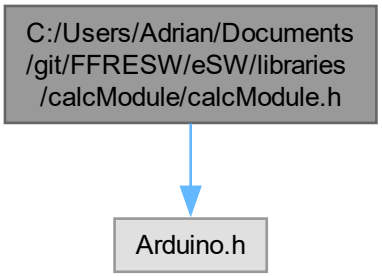
Copyright (c) 2024

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

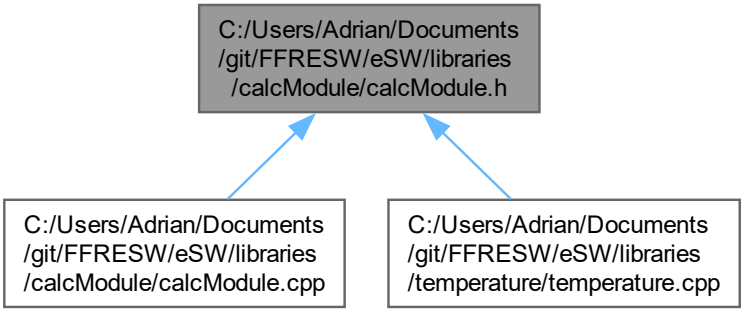
Header file for the calculation module handling sensor data.

```
#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.

7.2.1 Detailed Description

Header file for the calculation module handling sensor data.

Author

Adrian Goessl

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.3 calcModule.h

Go to the documentation of this file.

```
00001
00012 #ifndef CALCMODULE_H
00013 #define CALCMODULE_H
00014
00015 #include <Arduino.h>
00016
00018 namespace calcModule
00019 {
00021     enum Type
00022     {
00023         General,
00024         Pressure,
00025         Position
00026     };
00027
00029     class CalcModuleInternals
00030     {
00031     public:
```

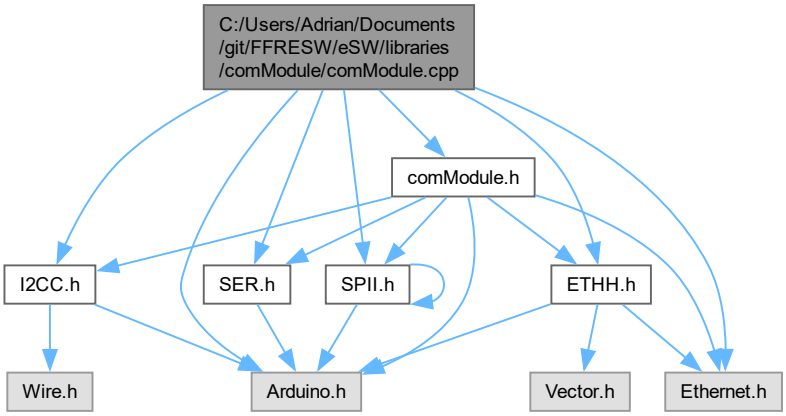
```
00032         CalcModuleInternals();
00033         ~CalcModuleInternals();
00034
00042         static float calculateAverage(const float* data, int length);
00043
00051         static float findMaximum(const float* data, int length);
00052
00060         static float findMinimum(const float* data, int length);
00061
00069         static float calculateStandardDeviation(const float* data, int length);
00070
00078         static float findMedian(float* data, int length);
00079
00086         static float celsiusToFahrenheit(float celsius);
00087
00094         static float fahrenheitToCelsius(float fahrenheit);
00095
00102         static float celsiusToKelvin(float celsius);
00103
00110         static float kelvinToCelsius(float kelvin);
00111
00118         static float pascalToAtm(float pascal);
00119
00126         static float atmToPascal(float atm);
00127
00134         static float pascalToPsi(float pascal);
00135
00142         static float psiToPascal(float psi);
00143
00151         static float calculatePower(float voltage, float current);
00152
00160         static float calculateCurrent(float voltage, float resistance);
00161
00169         static float calculateResistance(float voltage, float current);
00170
00184         static float extractFloat(String response, int id);
00185
00194         static float extractFloatFromResponse(const String& response, Type type);
00195
00196     private:
00197
00204         static void sortArray(float* data, int length);
00205
00212         static float roundToPrecision(float value, int precision);
00213     };
00214 }
00215
00216 #endif // CALCMODULE_H
```

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

Implementation of the comModule class that utilizes various communication protocols.

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

Include dependency graph for comModule.cpp:



7.4.1 Detailed Description

Implementation of the comModule class that utilizes various communication protocols.

7.5 comModule.h

```
00001 #ifndef COMMODULE_H
00002 #define COMMODULE_H
00003
00004 #include <Arduino.h>
00005 #include <Ethernet.h>
00006 #include "ETHH.h"
00007 #include "I2CC.h"
00008 #include "SER.h"
00009 #include "SPII.h"
00010
00012 namespace comModule
00013 {
00015     class ComModuleInternals
00016     {
00017     public:
00018         ComModuleInternals();
00019         ~ComModuleInternals();
00020
00026         EthernetCommunication& getEthernet();
00027
00033         I2CCommunication& getI2C();
00034
00040         SPICommunication& getSPI();
00041
00047         SerialCommunication& getSerial();
00048
00049     private:
00050         EthernetCommunication eth;
00051         I2CCommunication i2c;
00052         SPICommunication spi;
00053         SerialCommunication ser;
00054     };
00055 }
00056
00057 #endif // COMMODULE_H
```

7.6 ETHH.h

```
00001
00008 #ifndef ETHERNET_COMMUNICATION_H
00009 #define ETHERNET_COMMUNICATION_H
00010
00011 #include <Arduino.h>
00012 #include <Ethernet.h>
00013 #include <Vector.h>
00014
00016 namespace comModule
00017 {
00019     enum class Service : uint8_t
00020     {
00021         SET = 0x01,
00022         GET = 0x0B,
00023         SET_COMPOUND = 0x28,
00024         GET_COMPOUND = 0x29,
00025         SETGET = 0x30
00026     };
00027
00029     enum class Compound1 : uint32_t
00030     {
00031         CONTROL_MODE = 0x0F020000,
00032         TARGET_POSITION = 0x11020000,
00033         TARGET_PRESSURE = 0x07020000,
00034         NOT_USED = 0x00000000
00035     };
00036
00038     enum class Compound2 : uint32_t
00039     {
00040         ACCESS_MODE = 0x0F0B0000,
00041         CONTROL_MODE = 0x0F020000,
00042         TARGET_POSITION = 0x11020000,
00043         TARGET_PRESSURE = 0x07020000,
00044         ACTUAL_POSITION = 0x10010000,
00045         POSITION_STATE = 0x00100000,
00046         ACTUAL_PRESSURE = 0x07010000,
00047         TARGET_PRESSURE_USED = 0x07030000,
00048         WARNING_BITMAP = 0x0F300100,
00049         NOT_USED = 0x00000000
00050     };
00051
00053     enum class Compound3 : uint32_t
00054     {
00055         CONTROL_MODE = 0x0F020000,
00056         TARGET_POSITION = 0x11020000,
00057         TARGET_PRESSURE = 0x07020000,
00058         SEPARATION = 0x00000000,
00059         ACCESS_MODE = 0x0F0B0000,
00060         ACTUAL_POSITION = 0x10010000,
00061         POSITION_STATE = 0x00100000,
00062         ACTUAL_PRESSURE = 0x07010000,
00063         TARGET_PRESSURE_USED = 0x07030000,
00064         WARNING_BITMAP = 0x0F300100,
00065         NOT_USED = 0x00000000
00066     };
00067
00069     enum class Error_Codes : uint8_t
00070     {
00071         NO_ERROR = 0x00,
00072         WRONG_COMMAND_LENGTH = 0x0C,
00073         VALUE_TOO_LOW = 0x1C,
00074         VALUE_TOO_HIGH = 0x1D,
00075         RESULTING_ZERO_ADJUST_OFFSET = 0x20,
00076         NO_SENSOR_ENABLED = 0x21,
00077         WRONG_ACCESS_MODE = 0x50,
00078         TIMEOUT = 0x51,
00079         NV_MEMORY_NOT_READY = 0x6D,
00080         WRONG_PARAMETER_ID = 0x6E,
00081         PARAMETER_NOT_SETTABLE = 0x70,
00082         PARAMETER_NOT_READABLE = 0x71,
00083         WRONG_PARAMETER_INDEX = 0x73,
00084         WRONG_VALUE_WITHIN_RANGE = 0x76,
00085         NOT_ALLOWED_IN_THIS_STATE = 0x78,
00086         SETTING_LOCK = 0x79,
00087         WRONG_SERVICE = 0x7A,
00088         PARAMETER_NOT_ACTIVE = 0x7B,
00089         PARAMETER_SYSTEM_ERROR = 0x7C,
00090         COMMUNICATION_ERROR = 0x7D,
00091         UNKNOWN_SERVICE = 0x7E,
00092         UNEXPECTED_CHARACTER = 0x7F,
00093         NO_ACCESS_RIGHTS = 0x80,
00094         NO_ADEQUATE_HARDWARE = 0x81,
00095         WRONG_OBJECT_STATE = 0x82,
00096         NO_SLAVE_COMMAND = 0x84,
00097         COMMAND_TO_UNKNOWN_SLAVE = 0x85,
```

```
00098         COMMAND_TO_MASTER_ONLY = 0x87,
00099         ONLY_G_COMMAND_ALLOWED = 0x88,
00100         NOT_SUPPORTED = 0x89,
00101         FUNCTION_DISABLED = 0xA0,
00102         ALREADY_DONE = 0xA1
00103     };
00104
00106     class EthernetCommunication
00107     {
00108     public:
00109         EthernetCommunication();
00110         ~EthernetCommunication();
00111
00118         void beginEthernet(uint8_t* macAddress, IPAddress ip);
00119
00126         void sendEthernetData(const char* endpoint, const char* data);
00127
00134         void receiveEthernetData(char* buffer, size_t length);
00135
00140         void handleEthernetClient();
00141
00147         String getRequestedEndpoint();
00148
00155         String getSpecificEndpoint(const String& jsonBody);
00156
00162         void sendJsonResponse(const String& jsonBody);
00163
00169         EthernetClient& getClient();
00170
00177         bool isInitialized() const;
00178
00185         bool getSendDataFlag() const;
00186
00192         void setSendDataFlag(bool flag);
00193
00201         void setCompound(Compound1 id, int index, String value);
00202
00210         void setCompound(Compound2 id, int index, String value);
00211
00219         void setCompound(Compound3 id, int index, String value);
00220
00229         void setCompoundInternal(String compoundType, unsigned long id, int index, String value);
00230
00238         String getCompound(Compound1 id, int index);
00239
00247         String getCompound(Compound2 id, int index);
00248
00256         String getCompound(Compound3 id, int index);
00257
00266         String getCompoundInternal(String compoundType, unsigned long id, int index);
00267
00275         Vector<float> getParsedCompound(Compound1 id, int index);
00276
00284         Vector<float> getParsedCompound(Compound2 id, int index);
00285
00293         Vector<float> getParsedCompound(Compound3 id, int index);
00294
00301         Vector<float> parseCompoundResponse(String response);
00302
00309         void setParameter(Compound2 id, String value);
00310
00317         String getParameter(Compound2 id);
00318
00324         void sendCommand(String command);
00325
00326     private:
00327         EthernetServer server;
00328         EthernetClient client;
00329         bool ethernetInitialized = false;
00330         bool sendDataFlag = false;
00331
00338         String floatToIEEE754(float value);
00339
00346         Vector<float> parseResponse(String response);
00347
00348     };
00349 }
00350
00351 #endif // ETHERNET_COMMUNICATION_H
00352
00353
```

7.7 I2CC.h

```
00001
00008 #ifndef I2C_COMMUNICATION_H
00009 #define I2C_COMMUNICATION_H
00010
00011 #include <Arduino.h>
00012 #include <Wire.h>
00013
00014 namespace comModule
00015 {
00017     class I2CCommunication
00018     {
00019     public:
00020         I2CCommunication();
00021         ~I2CCommunication();
00022
00028         void beginI2C(uint8_t address);
00029
00034         void endI2C();
00035
00043         void i2cWrite(uint8_t deviceAddress, uint8_t* data, size_t length);
00044
00053         size_t i2cRead(uint8_t deviceAddress, uint8_t* buffer, size_t length);
00054
00061         bool isInitialized() const;
00062
00063     private:
00064         bool i2cInitialized = false;
00065     };
00066 }
00067
00068 #endif // I2C_COMMUNICATION_H
```

7.8 SER.h

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

7.9 SPII.h

```
00001
00008 #ifndef SPI_COMMUNICATION_H
00009 #define SPI_COMMUNICATION_H
00010
00011 #include "SPII.h"
00012
00013 #include <Arduino.h>
00014
00016 namespace comModule
00017 {
00019     class SPICommunication
00020     {
```

```
00021     public:
00022         SPICommunication();
00023         ~SPICommunication();
00024 
00029         void beginSPI();
00030 
00035         void endSPI();
00036 
00043         void spiWrite(uint8_t* data, size_t length);
00044 
00051         void spiRead(uint8_t* buffer, size_t length);
00052 
00059         bool isInitialized() const;
00060 
00061     private:
00062         bool spiInitialized = false;
00063     };
00064 }
00065 
00066 #endif // SPI_COMMUNICATION_H
```

7.10 config.h

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

7.11 flyback.h

```
00001 /*
00002  * flyback.h
00003  *
00004  *  Created on: 07.12.2024
00005  *      Author: domin
00006  */
00007 #ifndef FLYBACK_H
00008 #define FLYBACK_H
00009 
00010 #include <Arduino.h>
00011 #include <Wire.h>
00012 
00014 namespace flybackModule
```

```
00015 {
00017     enum class SwitchStates : int
00018     {
00019         HV_Module_OFF,
00020         HV_Module_MANUAL,
00021         HV_Module_REMOTE,
00022         HV_Module_INVALID
00023     };
00024
00027     typedef struct Measurement
00028     {
00029         float voltage;
00030         float current;
00031         float power;
00032         int digitalFreqValue;
00033         int digitalDutyValue;
00034         int dutyCycle;
00035         uint32_t frequency;
00036     } meas;
00037
00041     class Flyback
00042     {
00043     public:
00044         Flyback();
00045         ~Flyback();
00046
00051         void initialize();
00052
00057         void deinitialize();
00058
00065         bool isInitialized() const;
00066
00073         bool getTimerState();
00074
00080         void setTimerState(bool state);
00081
00088         SwitchStates getSwitchState();
00089
00095         Measurement measure();
00096
00101         void run();
00102
00108         void setExternFrequency(uint32_t frequency);
00109
00114         uint32_t getExternFrequency();
00115
00121         void setExternDutyCycle(int dutyCycle);
00122
00127         int getExternDutyCycle();
00128
00129
00130     private:
00131         Measurement meas;
00132
00133         //Define Pins -->Signaltable
00134         static const int Main_Switch_OFF = 27;
00135         static const int Main_Switch_MANUAL = 28;
00136         static const int Main_Switch_REMOTE = 29;
00137         static const int Measure_ADC = A0;          //ADC PIN for Voltage Measurement
00138         static const int PWM_OUT = 11;
00139         static const int PWM_INV = 12;
00140
00141         //Variables for calculating HV
00142         const float R1 = 100000000;
00143         const float R2 = 10000;
00144         const float ADC_Max_Value = 1023.0;
00145         const float Vcc = 5.0;
00146
00147         //Define Pins --> Signaltable
00148         static const int HV_Module_ON = 37;
00149         static const int HV_Module_OFF = 36;
00150         static const int HV_Module_Working = 35;
00151         static const int PWM_Frequency = A1;
00152         static const int PWM_DutyCycle = A2;
00153
00154         bool _flybackInitialized;
00155         bool _timerInitialized;
00156
00157         // States
00158         static SwitchStates lastState;
00159         static bool lastTimerState;
00160         static int lastPWMFrequency;
00161         static int lastPWMDutyCycle;
00162
00167         void timerConfig();
00168
00175         void setPWMFrequency(uint32_t frequency, int dutyCycle);
```

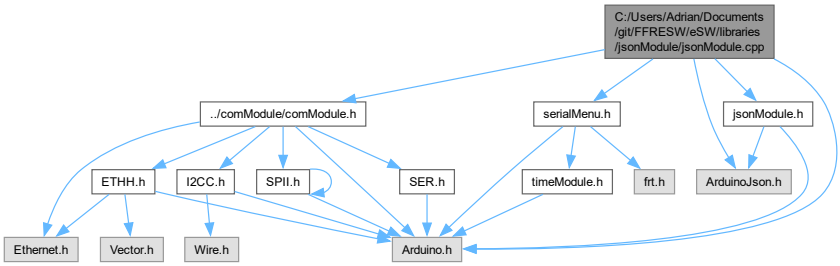
```
00176     };
00177 }
00178
00179 #endif //FLYBACK_H
```

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

Implementation of the jsonModule class.

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include "../comModule/comModule.h"
#include <jsonModule.h>
#include <serialMenu.h>
```

Include dependency graph for jsonModule.cpp:



7.12.1 Detailed Description

Implementation of the jsonModule class.

Version

0.1

Date

2024-01-26

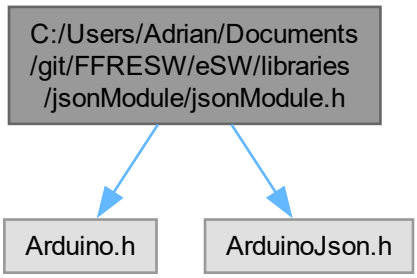
Copyright

Copyright (c) 2024

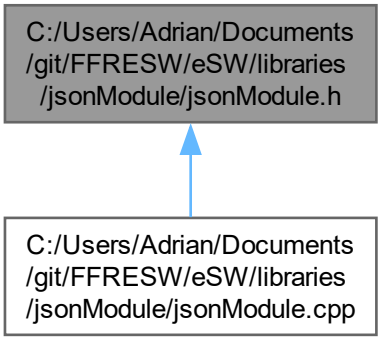
7.13 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h File Reference

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

Include dependency graph for jsonModule.h:



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



Classes

- class jsonModule::JsonModuleInternals

Namespaces

- namespace jsonModule

  Namespace for the JSON module.

7.13.1 Detailed Description

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.14 jsonModule.h

Go to the documentation of this file.

```
00001
00012 #ifndef JSONMODULE_H
00013 #define JSONMODULE_H
00014
00015 #include <Arduino.h>
00016 #include <ArduinoJson.h>
00017
00019 namespace jsonModule
00020 {
00022     class JsonModuleInternals
00023     {
00024     public:
00025         JsonModuleInternals();
00026         ~JsonModuleInternals();
00027
00034         void createJson(const char* key, const char* value);
00035
00042         void createJsonFloat(const char* key, float value);
00043
00050         void createJsonInt(const char* key, int value);
00051
00058         void createJsonString(const char* key, String& value);
00059
00066         void createJsonStringConst(const char* key, const String& value);
00067
00072         void sendJsonSerial();
00073
00079         void sendJsonEthernet(const char* endpoint);
00080
00086         String getJsonString() const;
00087
00092         void clearJson();
00093
00098         void printJsonDocMemory();
00099
00100         size_t jsonBuffer;
00101
00102     private:
00103         StaticJsonDocument<512> jsonDoc;
00104
00105     };
00106 }
00107
00108 #endif // JSONMODULE_H
```

7.15 lockerBase.h

```
00001 #ifndef LOCKER_BASE_H
00002 #define LOCKER_BASE_H
00003
00004 #include <frt.h>
00005 #include <Arduino.h>
00006 #include <scopedLock.h>
00007 #include <logManager.h>
00008
00010 class LockerBase
00011 {
00012 public:
00013     LockerBase()
00014     {
00015         if (_logger->isSDCardInitialized())
00016         {
00017             _logger->setLogFileName("log_LockerBase.txt");
00018         }
00019     }
00020
00021     ~LockerBase() {}
00022
00028     locker::ScopedLock lockEthernetScoped()
00029     {
00030         ethernetConnected = true;
00031         logState("Ethernet connected", ethernetConnected);
00032         return locker::ScopedLock(ethernetMutex);
00033     }
00034
00040     locker::ScopedLock lockTemperatureScoped()
00041     {
00042         temperatureReading = true;
00043         logState("Temperature reading", temperatureReading);
00044         return locker::ScopedLock(temperaturQueueMutex);
00045     }
00046
00052     locker::ScopedLock lockSerialScoped()
00053     {
00054         serialReading = true;
00055         logState("Serial reading", serialReading);
00056         return locker::ScopedLock(serialMutex);
00057     }
00058
00059 private:
00060
00061     // Mutexes for the different resources
00062     frt::Mutex temperaturQueueMutex;
00063     frt::Mutex ethernetMutex;
00064     frt::Mutex serialMutex;
00065
00066     // Semaphores for the different resources
00067     frt::Semaphore ethernetSemaphore;
00068     frt::Semaphore temperatureSemaphore;
00069     frt::Semaphore serialSemaphore;
00070
00071     // Flags for the different resources
00072     bool ethernetConnected = false;
00073     bool temperatureReading = false;
00074     bool serialReading = false;
00075
00076     // Logger instance
00077     LogManager* _logger = LogManager::getInstance();
00078
00085     void logState(const char* label, bool state)
00086     {
00087         char buffer[64];
00088         snprintf(buffer, sizeof(buffer), "[LockerBase] %s: %s", label, state ? "true" : "false");
00089         _logger->writeToLogFile(buffer);
00090     }
00091 };
00092
00093 #endif // LOCKER_BASE_H
```

7.16 scopedLock.h

```
00001 #ifndef LOCKER_BASE_SCOPEDLOCK_H
00002 #define LOCKER_BASE_SCOPEDLOCK_H
00003
00004 #include <frt.h>
00005
00007 namespace locker
00008 {
```

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

7.17 logManager.h

```
00001 #ifndef LOGMANAGER_H
00002 #define LOGMANAGER_H
00003
00004 #include <Arduino.h>
00005 #include <SD.h>
00006 #include <SPI.h>
00007 #include <timeModule.h>
00008
00010 class LogManager
00011 {
00012 public:
00013
00019     static LogManager* getInstance();
00020
00026     void initSDCard(int cs);
00027
00031     void shutdownSDCard();
00032
00036     void flushLogs();
00037
00044     bool isSDCardInitialized() const;
00045
00050     static String getCurrentTime();
00051
00057     void setLogFileName(const String& fileName);
00058
00066     bool writeToLogFile(const String& logMessage);
00067
00074     void renameFile(const String& oldName, const String& newName);
00075
00076 private:
00077     LogManager();
00078     ~LogManager();
00079
00080     static LogManager* _instance;
00081     File logFile;
00082     bool sdCardInitialized = false;
00083     String logFileName;
00084     String baseLogFileName;
00085
00086     static const int chipSelectPinEth = 10; // Default CS pin for SD card
00087     static const long maxLogFileSize = 1024 * 1024 * 100; // 100 MB
00088
00089     LogManager(const LogManager&) = delete;
00090     LogManager& operator=(const LogManager&) = delete;
00091 };
00092
```

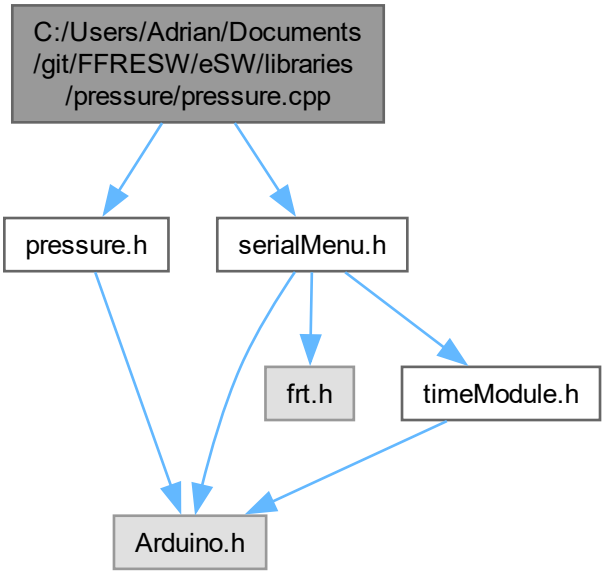
```
00093
00094 #endif // LOGMANAGER_H
```

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

Implementation of the pressure class.

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

Include dependency graph for pressure.cpp:



7.18.1 Detailed Description

Implementation of the pressure class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.19 pressure.h

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

7.20 ptrUtils.h

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

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

```
00210 template <typename T>
00211 class ScopedPointer
00212 {
00213 private:
00214     T* ptr;
00215
00216 public:
00217     explicit ScopedPointer(T* p = nullptr) : ptr(p) {}
00218     ~ScopedPointer() { SafeDelete(ptr); }
00219
00225     T* get() const { return ptr; }
00226
00232     T* release()
00233     {
00234         T* temp = ptr;
00235         ptr = nullptr;
00236         return temp;
00237     }
00238
00244     void reset(T* p = nullptr)
00245     {
00246         SafeDelete(ptr);
00247         ptr = p;
00248     }
00249
00255     T& operator*() const { return *ptr; }
00256
00262     T* operator->() const { return ptr; }
00263 };
00264
00270 template <typename T>
00271 class PointerWrapper
00272 {
00273 private:
00274     T* ptr;
00275
00276 public:
00277     explicit PointerWrapper(T* p = nullptr) : ptr(p) {}
00278     ~PointerWrapper() { SafeDelete(ptr); }
00279
00285     T* get() const { return ptr; }
00286
00292     T* release()
00293     {
00294         T* temp = ptr;
00295         ptr = nullptr;
00296         return temp;
00297     }
00298
00304     void reset(T* p = nullptr)
00305     {
00306         SafeDelete(ptr);
00307         ptr = p;
00308     }
00309
00315     T& operator*() { return *ptr; }
00316
00322     T* operator->() { return ptr; }
00323 };
00324
00325 #endif // PTRUTILS_H
```

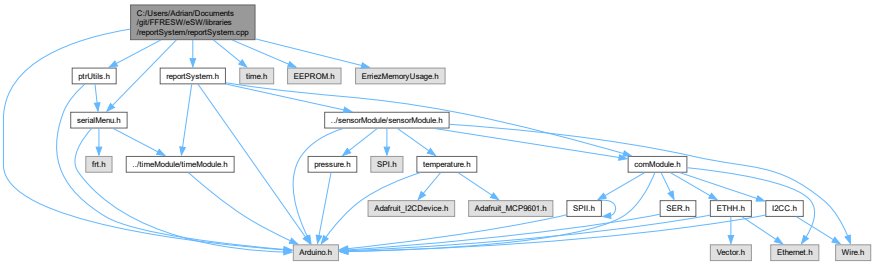
7.21 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp File Reference

Unified system health and error reporting module.

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

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

Include dependency graph for reportSystem.cpp:



Functions

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

7.21.1 Detailed Description

Unified system health and error reporting module.

Author

Adrian Goessl

Version

0.3

Date

2024-09-28

Copyright

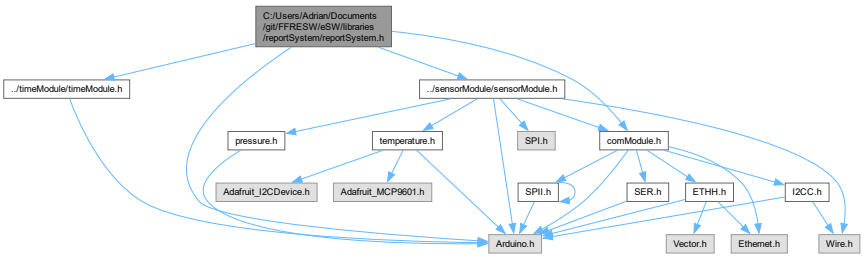
Copyright (c) 2024

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

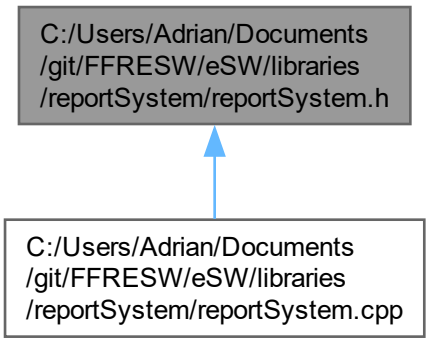
```
#include <Arduino.h>
#include "../sensorModule/sensorModule.h"
#include "../comModule/comModule.h"
```

```
#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**

7.22.1 Detailed Description

Author

your name (you@domain.com)

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.23 reportSystem.h

Go to the documentation of this file.

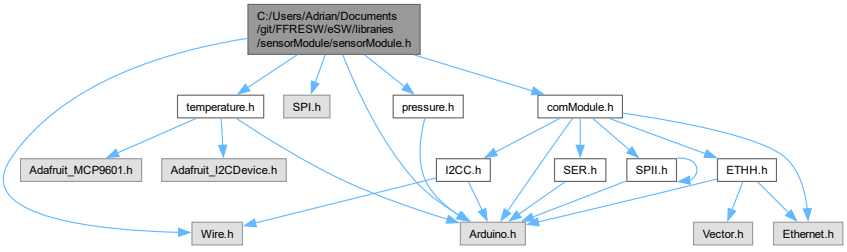
```
00001
00011 #ifndef REPORTSYSTEM_H
00012 #define REPORTSYSTEM_H
00013
00014 #include <Arduino.h>
00015 #include "../sensorModule/sensorModule.h"
00016 #include "../comModule/comModule.h"
00017 #include "../timeModule/timeModule.h"
00018
00019 #define STACK_GUARD 0xDEAD // Stack guard value
00020 extern volatile uint16_t stackCheck; // Stack check variable
00021
00022 #define EEPROM_ERROR_ADDR 0
00023
00025 namespace reportSystem
00026 {
00028     class ReportSystem
00029     {
00030     public:
00031         ReportSystem();
00032         ~ReportSystem();
00033
00039         void reportError(const char* errorMessage);
00040
00053         bool checkSystemHealth(size_t memoryThreshold, bool checkEth,
00054                                bool checkSpi, bool checkI2c,
00055                                bool checkTemp, bool checkPress);
00056
00063         String reportStatus(bool active);
00064
00071         void setThreshold(float tempThreshold, float pressureThreshold);
00072
00081         bool checkThresholds(float currentTemp, float currentPressure);
00082
00088         String getCurrentTime();
00089
00095         String getMemoryStatus();
00096
00102         String getStackDump();
00103
00108         void startBusyTime();
00109
00114         void startIdleTime();
00115
00121         float getCPULoad();
00122
00127         void resetUsage();
00128
00133         static void initStackGuard();
```

```
00134
00141         static bool detectStackOverflow();
00142
00149         void saveLastError(const char* error);
00150
00156         String getLastError();
00157
00163         bool getLastErrorInfo();
00164
00174         bool checkRamLevel(unsigned int warningThreshold, unsigned int criticalThreshold);
00175
00176     private:
00177         float tempThreshold;
00178         float pressureThreshold;
00179         unsigned long lastHealthCheck;
00180         const unsigned long healthCheckInterval = 10000; // 10 seconds
00181         unsigned long busyTime = 0;
00182         unsigned long idleTime = 0;
00183         unsigned long lastTimestamp = 0;
00184
00193         bool checkSensors(bool checkTemp, bool checkPress);
00194
00204         bool checkCommunication(bool checkEth, bool checkSpi, bool checkI2c);
00205
00213         bool checkMemory(unsigned int threshold);
00214
00215         sensorModule::SensorModuleInternals* _sens;
00216         comModule::ComModuleInternals* _com;
00217         timeModule::TimeModuleInternals* _time;
00218     };
00219 }
00220
00221 #endif // REPORTSYSTEM_H
```

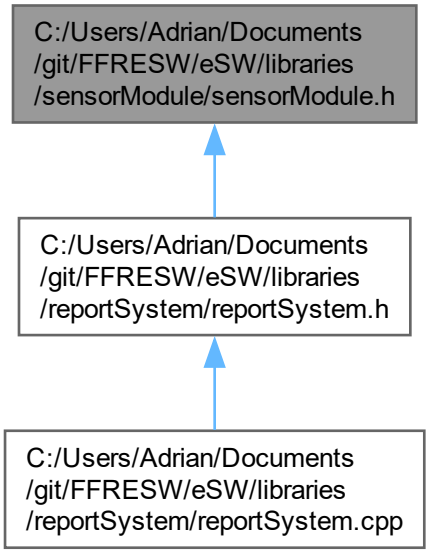
7.24 C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/sensorModule.h File Reference

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

Include dependency graph for sensorModule.h:



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



Classes

- class sensorModule::SensorModuleInternals

  Class for the sensor module internals.

Namespaces

- namespace sensorModule

  Namespace for the sensor module.

Enumerations

- enum class sensorModule::SensorType {
  TEMPERATURE , **OBJECTTEMPERATURE** , **AMBIENTTEMPERATURE** , **PRESSURE** ,
  DHT11 , **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.

7.24.1 Detailed Description

Author

Adrian Goessl

Version

0.1

Date

2024-09-28

Copyright

Copyright (c) 2024

7.25 sensorModule.h

Go to the documentation of this file.

```
00001
00012 #ifndef SENSORMODULE_H
00013 #define SENSORMODULE_H
00014
00015 #include <Arduino.h>
00016 #include <Wire.h>
00017 #include <SPI.h>
00018 #include <pressure.h>
00019 #include <temperature.h>
00020 #include <comModule.h>
00021
00022
00024 namespace sensorModule
00025 {
00027     enum class SensorType
00028     {
00029         TEMPERATURE,
00030         OBJECTTEMPERATURE,
00031         AMBIENTTEMPERATURE,
00032         PRESSURE,
00033         DHT11,
00034         MCP9601_Celsius_Indoor,
00035         MCP9601_Fahrenheit_Indoor,
00036         MCP9601_Kelvin_Indoor,
00037         MCP9601_Celsius_Outdoor,
00038         MCP9601_Fahrenheit_Outdoor,
00039         MCP9601_Kelvin_Outdoor,
00040         UNKNOWN
00041     };
00042
00044     class SensorModuleInternals : public TemperatureSensor, public PressureSensor
00045     {
00046     public:
00047         SensorModuleInternals();
00048         ~SensorModuleInternals();
00049
00054         void initialize();
00055
00062         float readSensor(SensorType type);
00063
00071         bool calibrateSensor(SensorType type);
00072
00080         bool checkSensorStatus(SensorType type);
00081
00082         void reportUnknownSensorOnce(SensorType type, String context);
00083
00084     private:
00085         TemperatureSensor _temperatureSensor;
00086         PressureSensor _pressureSensor;
00087
00088         bool _i2cSensorInitialized;
00089         bool _spiSensorInitialized;
00090
00091         SensorType _lastUnknownSensorType;
00092         bool _unknownSensorReported;
00093     };
00094 }
00095
00096 #endif // SENSORMODULE_H
```

7.26 serialMenu.h

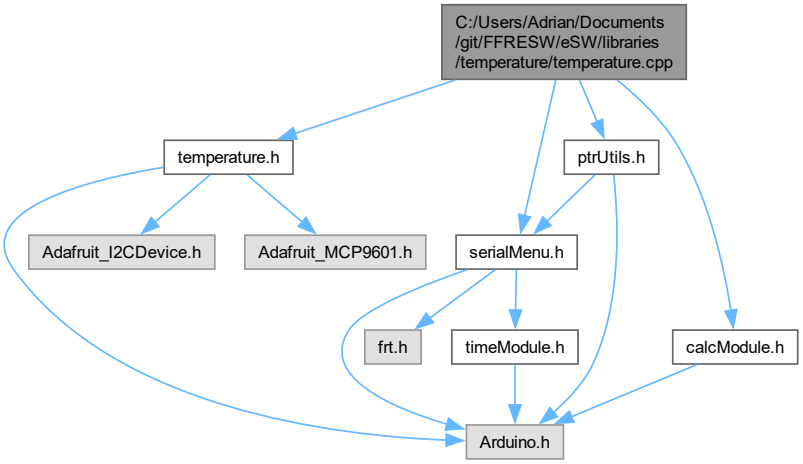
```
00001 #ifndef SERIAL_MENU_H
00002 #define SERIAL_MENU_H
00003
00004 #include <Arduino.h>
00005 #include <frt.h>
00006 #include <timeModule.h>
00007
00008
00010 struct MenuItem
00011 {
00012     const char* label;
00013     char key;
00014     void (*callback)();
00015 };
00016
00018 class SerialMenu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

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

Implementation of the temperature class.

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

Include dependency graph for temperature.cpp:



7.27.1 Detailed Description

Implementation of the temperature class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.28 temperature.h

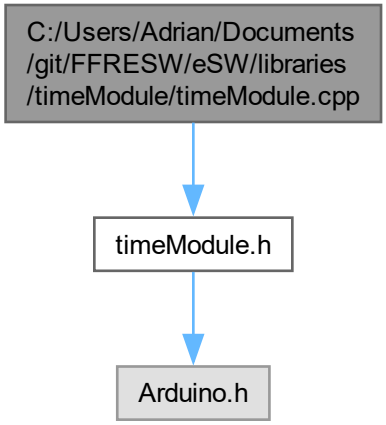
```
00001 #ifndef TEMPERATURESENSOR_H
00002 #define TEMPERATURESENSOR_H
00003
00004 #include <Arduino.h>
00005 #include <Adafruit_I2CDevice.h>
00006 #include "Adafruit_MCP9601.h"
00007
00009 enum Units
00010 {
00011     Celsius,
00012     Kelvin,
00013     Fahrenheit
00014 };
00015
00017 enum MCP9601_Status : uint8_t
00018 {
00019     MCP9601_OPENCIRCUIT = 0x10,
00020     MCP9601_SHORTCIRCUIT = 0x20
00021 };
```

```
00022
00023 // @brief Enum for the Different Sensors in different Environments \enum SensorID
00024 enum SensorID
00025 {
00026     INDOOR, // Black-Color Patch on Cable
00027     OUTDOOR // Yellow-Color Patch on Cable
00028 };
00029
00031 class TemperatureSensor
00032 {
00033 public:
00034     TemperatureSensor();
00035     ~TemperatureSensor();
00036
00041     void initialize();
00042
00048     float readTemperature();
00049
00056     float readMCP9601(Units unit, SensorID sensor);
00057
00064     bool isInitialized() const;
00065
00066
00072     uint8_t calibMCP9601(SensorID sensor);
00073
00074 private:
00075     bool _temperatureSensorInitialized;
00076     static const int TEMP_SENSOR_PIN = A0;
00077     static const int TEMP_SENSOR_PIN_DIG = 4;
00078     static const int DHT11_PIN = 7;
00079
00080     static const uint8_t MLX90614 = 0x5A;
00081     static const uint8_t AMBIENT_TEMP = 0x06;
00082     static const uint8_t OBJECT_TEMP = 0x07;
00083
00084     // Settings for the MCP9601 Sensor board
00085     Adafruit_MCP9601 _mcp1;
00086     Adafruit_MCP9601 _mcp2;
00087     Ambient_Resolution _ambientREs = RES_ZERO_POINT_0625;
00088     static const uint8_t MCP9601_I2C = 0x67;
00089
00090
00097     float readAnalogSensor(int pin);
00098
00105     float readDigitalSensor(int pin);
00106 };
00107
00108 #endif // TEMPERATURESENSOR_H
```

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

Implementation of the timeModule class.

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



7.29.1 Detailed Description

Implementation of the timeModule class.

Version

0.1

Date

2024-01-26

Copyright

Copyright (c) 2024

7.30 timeModule.h

```
00001 #ifndef TIMEMODULE_H
00002 #define TIMEMODULE_H
00003
00004 #include <Arduino.h>
00005
00007 namespace timeModule
00008 {
00010     typedef struct DateTimeStruct
00011     {
00012         int year;
00013         int month;
00014         int day;
00015         int hour;
00016         int minute;
00017         int second;
00018     } DateTimeStruct;
```

```
00019
00021     class TimeModuleInternals
00022     {
00023     public:
00024         TimeModuleInternals();
00025         ~TimeModuleInternals();
00026
00032         static void incrementTime(DateTimeStruct *dt);
00033
00040         static String formatTimeString(const DateTimeStruct &dt);
00041
00049         bool setTimeFromHas(const String& timeString);
00050
00056         void setSystemTime(const DateTimeStruct& dt);
00057
00062         void updateSoftwareClock();
00063
00069         DateTimeStruct getSystemTime();
00070
00076         static TimeModuleInternals* getInstance();
00077
00078     private:
00079         DateTimeStruct dt;
00080         unsigned long startMillis = 0;
00081     };
00082 }
00083
00084 #endif // TIMEMODULE
```

7.31 vacControl.h

```
00001 /*
00002 * flyback.h
00003 *
00004 *  Created on: 28.03.2025
00005 *      Author: domin
00006 */
00007 #ifndef VACCONTROL_H
00008 #define VACCONTROL_H
00009
00010 #include <Arduino.h>
00011 #include <Wire.h>
00012
00013
00015 namespace vacControlModule
00016 {
00018     enum class SwitchStates : int
00019     {
00020         Main_Switch_OFF,
00021         Main_Switch_MANUAL,
00022         Main_Switch_REMOTE,
00023         Main_switch_INVALID,
00024         PUMP_ON,
00025         PUMP_OFF
00026     };
00029     typedef struct Pressure
00030     {
00031         float pressure;
00032
00033     } meas;
00034
00036     enum Scenarios
00037     {
00038         Scenario_1 = 0,
00039         Scenario_2 = 1,
00040         Scenario_3 = 2,
00041         Scenario_4 = 3,
00042         Scenario_5 = 4,
00043         Invalid_Scenario = -1
00044     };
00045
00049     class VacControl
00050     {
00051     public:
00052
00053         VacControl();
00054         ~VacControl();
00055
00060         void initialize();
00061
00066         void deinitialize();
00067
00074         bool isInitialized() const;
```

```
00075
00081         SwitchStates getSwitchState();
00082
00088         Scenarios getScenario();
00089
00095         Pressure measure();
00096
00103         void setVacuumLed(float pressure, float targetPressure);
00104
00111         int getScenarioFromPotValue(int potValue);
00112
00118         void setPump(bool flag);
00119
00125         void run();
00126
00127
00133         void setExternScenario(int pressure);
00134
00140         int getExternScenario();
00141
00147         void externProcess();
00148
00154         void setExternPressure(float pressure);
00155
00161         float getExternPressure();
00162
00163
00164     private:
00165         Pressure meas;
00166
00167         //Define Pins --> Main_Switch
00168         static const int Main_Switch_OFF = 27;       //Main_Switch OFF Mode 27
00169         static const int Main_Switch_MANUAL = 28;    //Main_Switch Manual Mode 28
00170         static const int Main_Switch_REMOTE = 29;    //Main_Switch Remote Mode 29
00171
00172         //Define Pins --> Vacuum Logic
00173         static const int Switch_Pump_ON = 23;        //Button to turn Pump ON 37
00174         static const int Pump_Status_LED = 24;       //OUTPUT to see State off Pump
00175         static const int Pump_Relay = 25;            //OUTPUT to turn on/off Relais
00176         static const int targetVacuumLED = 26;       //OUTPUT to see Vacuum reached
00177         static const int targetPressure = A4;    //Potentiometer for Regulation
00178
00179         //Variables to save Values
00180         int currentScenario = -1;
00181
00182         //Variables for TargetPressure
00183         static const float TARGET_PRESSURE_1 = 1;
00184         static const float TARGET_PRESSURE_2 = 0.8f;
00185         static const float TARGET_PRESSURE_3 = 0.5;
00186         static const float TARGET_PRESSURE_4 = 0.01f;
00187
00188         // TODO NEW ADDED-> CHECK FUNCTIONALTY WITH FRANIC
00189         static int lastState;
00190         static int lastPumpState;
00191
00192
00193         bool _vacControlInitialized;
00194     };
00195 }
00196
00197 #endif //VACCONTROL_H
```

Index

atmToPascal
calcModule::CalcModuleInternals, 16

beginEthernet
comModule::EthernetCommunication, 27
beginI2C
comModule::I2CCommunication, 42
beginSerial
comModule::SerialCommunication, 78

C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.cpp, 103
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/calcModule/calcModule.h, 104, 105
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.cpp, 106
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/comModule.h, 107
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/ETHH.h, 108
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/I2CC.h, 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SERH.h, 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/comModule/SPII.h, 110
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/config/config.h, 111
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/flyback/flyback.h, 111
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.cpp, 113
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/jsonModule/jsonModule.h, 114, 115
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/lockerBase.h, 116
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/lockerBase/scopedLock.h, 116
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/logManager/logManager.h, 117
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.cpp, 118
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/pressure/pressure.h, 119
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/ptrUtils/ptrUtils.h, 119
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.cpp, 121
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/reportSystem/reportSystem.h, 122, 124
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/sensorModule/ser
125, 127
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/serialMenu/serialM
128
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temp
128
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/temperature/temp
129
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeM
130
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/timeModule/timeM
131
C:/Users/Adrian/Documents/git/FFRESW/eSW/libraries/vacControl/vacCo
132
calcModule, 9
calcModule::CalcModuleInternals, 15
atmToPascal, 16
calculateAverage, 16
calculateCurrent, 17
calculatePower, 17
calculateResistance, 17
calculateStandardDeviation, 18
celsiusToFahrenheit, 18
celsiusToKelvin, 19
extractFloat, 19
extractFloatFromResponse, 20
fahrenheitToCelsius, 20
findMaximum, 20
findMedian, 21
findMinimum, 21
kelvinToCelsius, 21
pascalToAtm, 22
pascalToPsi, 22
psiToPascal, 22
calcModuleInternals, 23
calculateAverage
calcModule::CalcModuleInternals, 16
calculateCurrent
calcModule::CalcModuleInternals, 17
calculatePower
calcModule::CalcModuleInternals, 17
calculateResistance
calcModule::CalcModuleInternals, 17
calculateStandardDeviation
calcModule::CalcModuleInternals, 18
calibMCP9601
temperatureSensor, 88
calibrateSensor
sensorModule::SensorModuleInternals, 75

celsiusToFahrenheit
calcModule::CalcModuleInternals, 18
celsiusToKelvin
calcModule::CalcModuleInternals, 19
checkRamLevel
reportSystem::ReportSystem, 64
checkSensorStatus
sensorModule::SensorModuleInternals, 76
checkSystemHealth
reportSystem::ReportSystem, 65
checkThresholds
reportSystem::ReportSystem, 65
comModule, 9
comModule::ComModuleInternals, 23
getEthernet, 24
getI2C, 24
getSerial, 24
getSPI, 24
comModule::EthernetCommunication, 25
beginEthernet, 27
getClient, 27
getCompound, 27–29
getCompoundInternal, 29
getParameter, 30
getParsedCompound, 30, 31
getRequestedEndpoint, 32
getSendDataFlag, 32
getSpecificEndpoint, 32
isInitialized, 33
parseCompoundResponse, 33
receiveEthernetData, 33
sendCommand, 34
sendEthernetData, 34
sendJsonResponse, 34
setCompound, 35, 36
setCompoundInternal, 36
setParameter, 37
setSendDataFlag, 38
comModule::I2CCommunication, 41
beginI2C, 42
i2cRead, 42
i2cWrite, 43
isInitialized, 43
comModule::SerialCommunication, 77
beginSerial, 78
isInitialized, 78
receiveSerialData, 78
sendSerialData, 79
comModule::SPICommunication, 86
isInitialized, 86
spiRead, 86
spiWrite, 87
comModuleInternals, 25
createJson
jsonModule::JsonModuleInternals, 44
createJsonFloat
jsonModule::JsonModuleInternals, 44
createJsonInt
jsonModule::JsonModuleInternals, 44
createJsonString
jsonModule::JsonModuleInternals, 45
createJsonStringConst
jsonModule::JsonModuleInternals, 45

detectStackOverflow
reportSystem::ReportSystem, 66

externProcess
vacControlModule::VacControl, 97
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, 38
getSwitchState, 39
getTimerState, 39
isInitialized, 39
measure, 40
setExternDutyCycle, 40
setExternFrequency, 40
setTimerState, 41
flybackModule::Measurement, 55
formatTimeString
timeModule::TimeModuleInternals, 91

get
PointerWrapper< T >, 57
ScopedPointer< T >, 73
getClient
comModule::EthernetCommunication, 27
getCompound
comModule::EthernetCommunication, 27–29
getCompoundInternal
comModule::EthernetCommunication, 29
getCPULoad
reportSystem::ReportSystem, 66
getCurrentTime
LogManager, 48
reportSystem::ReportSystem, 67
SerialMenu, 80
getEthernet
comModule::ComModuleInternals, 24
getExternPressure
vacControlModule::VacControl, 97
getExternScenario
vacControlModule::VacControl, 98
getI2C

comModule::ComModuleInternals, 24
getInstance
LogManager, 48
timeModule::TimeModuleInternals, 92
getJsonString
jsonModule::JsonModuleInternals, 45
getLastError
reportSystem::ReportSystem, 67
getLastErrorInfo
reportSystem::ReportSystem, 67
getMemoryStatus
reportSystem::ReportSystem, 68
getParameter
comModule::EthernetCommunication, 30
getParsedCompound
comModule::EthernetCommunication, 30, 31
getRequestedEndpoint
comModule::EthernetCommunication, 32
getScenario
vacControlModule::VacControl, 98
getScenarioFromPotValue
vacControlModule::VacControl, 98
getSendDataFlag
comModule::EthernetCommunication, 32
getSerial
comModule::ComModuleInternals, 24
getSpecificEndpoint
comModule::EthernetCommunication, 32
getSPI
comModule::ComModuleInternals, 24
getStackDump
reportSystem::ReportSystem, 68
getSwitchState
flybackModule::Flyback, 39
vacControlModule::VacControl, 99
getSystemTime
timeModule::TimeModuleInternals, 93
getTimerState
flybackModule::Flyback, 39

i2cRead
comModule::I2CCommunication, 42
i2cWrite
comModule::I2CCommunication, 43
incrementTime
timeModule::TimeModuleInternals, 94
initSDCard
LogManager, 50
isInitialized
comModule::EthernetCommunication, 33
comModule::I2CCommunication, 43
comModule::SerialCommunication, 78
comModule::SPICommunication, 86
flybackModule::Flyback, 39
PressureSensor, 59
TemperatureSensor, 88
vacControlModule::VacControl, 99
IsNullPtr
PtrUtils, 61

isSDCardInitialized
LogManager, 51
IsValidPtr
PtrUtils, 62

jsonModule, 11
jsonModule::JsonModuleInternals, 43
createJson, 44
createJsonFloat, 44
createJsonInt, 44
createJsonString, 45
createJsonStringConst, 45
getJsonString, 45
sendJsonEthernet, 46
jsonModuleInternals, 46

kelvinToCelsius
calcModule::CalcModuleInternals, 21

load
SerialMenu, 81
locker, 11
locker::ScopedLock, 71
ScopedLock, 71, 72
LockerBase, 47
lockEthernetScoped, 47
lockSerialScoped, 47
lockTemperatureScoped, 47
lockEthernetScoped
LockerBase, 47
lockSerialScoped
LockerBase, 47
lockTemperatureScoped
LockerBase, 47
LogManager, 48
getCurrentTime, 48
getInstance, 48
initSDCard, 50
isSDCardInitialized, 51
renameFile, 51
setLogFileName, 52
writeToLogFile, 53
LogMapper, 54

measure
flybackModule::Flyback, 40
vacControlModule::VacControl, 99
Measurement, 55
MenuItem, 56

operator->
PointerWrapper< T >, 57
ScopedPointer< T >, 73
operator*
PointerWrapper< T >, 57
ScopedPointer< T >, 73
Outputlevel, 56

parseCompoundResponse
comModule::EthernetCommunication, 33

pascalToAtm
calcModule::CalcModuleInternals, 22
pascalToPsi
calcModule::CalcModuleInternals, 22
PointerWrapper< T >, 56
get, 57
operator->, 57
operator*, 57
release, 58
reset, 58
PressureSensor, 59
isInitialized, 59
readPressure, 60
printToSerial
SerialMenu, 82, 83
psiToPascal
calcModule::CalcModuleInternals, 22
PtrUtils, 61
IsNullPtr, 61
IsValidPtr, 62

readMCP9601
TemperatureSensor, 88
readPressure
PressureSensor, 60
readSensor
sensorModule::SensorModuleInternals, 76
readTemperature
TemperatureSensor, 89
receiveEthernetData
comModule::EthernetCommunication, 33
receiveSerialData
comModule::SerialCommunication, 78
release
PointerWrapper< T >, 58
ScopedPointer< T >, 73
renameFile
LogManager, 51
reportError
reportSystem::ReportSystem, 68
reportStatus
reportSystem::ReportSystem, 69
reportSystem, 12
reportSystem::ReportSystem, 63
checkRamLevel, 64
checkSystemHealth, 65
checkThresholds, 65
detectStackOverflow, 66
getCPULoad, 66
getCurrentTime, 67
getLastError, 67
getLastErrorInfo, 67
getMemoryStatus, 68
getStackDump, 68
reportError, 68
reportStatus, 69
saveLastError, 70
setThreshold, 70
reset
PointerWrapper< T >, 58
ScopedPointer< T >, 73
run
vacControlModule::VacControl, 100

saveLastError
reportSystem::ReportSystem, 70
ScopedLock
locker::ScopedLock, 71, 72
ScopedPointer< T >, 72
get, 73
operator->, 73
operator*, 73
release, 73
reset, 73
sendCommand
comModule::EthernetCommunication, 34
sendEthernetData
comModule::EthernetCommunication, 34
sendJsonEthernet
jsonModule::JsonModuleInternals, 46
sendJsonResponse
comModule::EthernetCommunication, 34
sendSerialData
comModule::SerialCommunication, 79
sensorModule, 12
sensorModule::SensorModuleInternals, 74
calibrateSensor, 75
checkSensorStatus, 76
readSensor, 76
SerialMenu, 79
getCurrentTime, 80
load, 81
printToSerial, 82, 83
setCompound
comModule::EthernetCommunication, 35, 36
setCompoundInternal
comModule::EthernetCommunication, 36
setExternDutyCycle
flybackModule::Flyback, 40
setExternFrequency
flybackModule::Flyback, 40
setExternPressure
vacControlModule::VacControl, 100
setExternScenario
vacControlModule::VacControl, 101
setLogFileName
LogManager, 52
setParameter
comModule::EthernetCommunication, 37
setPump
vacControlModule::VacControl, 101
setSendDataFlag
comModule::EthernetCommunication, 38
setSystemTime
timeModule::TimeModuleInternals, 95
setThreshold
reportSystem::ReportSystem, 70
setTimeFromHas

timeModule::TimeModuleInternals, 96
setTimerState
flybackModule::Flyback, 41
setVacuumLed
vacControlModule::VacControl, 101
spiRead
comModule::SPICommunication, 86
spiWrite
comModule::SPICommunication, 87

TemperatureSensor, 87
calibMCP9601, 88
isInitialized, 88
readMCP9601, 88
readTemperature, 89
timeModule, 12
timeModule::DateTimeStruct, 25
timeModule::TimeModuleInternals, 90
formatTimeString, 91
getInstance, 92
getSystemTime, 93
incrementTime, 94
setSystemTime, 95
setTimeFromHas, 96

vacControlModule, 13
vacControlModule::Pressure, 58
vacControlModule::VacControl, 96
externProcess, 97
getExternPressure, 97
getExternScenario, 98
getScenario, 98
getScenarioFromPotValue, 98
getSwitchState, 99
isInitialized, 99
measure, 99
run, 100
setExternPressure, 100
setExternScenario, 101
setPump, 101
setVacuumLed, 101

writeToLogFile
LogManager, 53