My Project

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BusinessHandler Class Reference

`#include <BusinessHandler.h>`

Public Member Functions

- BusinessHandler (const std::string &configfile)
- void run ()
- void heartbeatSending ()
- void recevfromSensor ()

3.1.1 Detailed Description

Definition at line 14 of file BusinessHandler.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BusinessHandler()

```
BusinessHandler::BusinessHandler (
            const std::string & configfile )
```

Definition at line 3 of file BusinessHandler.cpp.

3.1.3 Member Function Documentation

3.1.3.1 heartbeatSending()

```
void BusinessHandler::heartbeatSending ( )
```

Definition at line 8 of file BusinessHandler.cpp.

3.1.3.2 recevfromSensor()

```
void BusinessHandler::recevfromSensor ( )
```

Definition at line 67 of file BusinessHandler.cpp.

3.1.3.3 run()

```
void BusinessHandler::run ( )
```

Definition at line 114 of file BusinessHandler.cpp.

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

- /home/thanh/SensorStation/include/BusinessHandler.h
- /home/thanh/SensorStation/src/BusinessHandler.cpp

3.2 ConfigReader Class Reference

```
#include <ConfigReader.h>
```

Public Member Functions

- ConfigReader (const std::string &filePath)
- std::string getValue (const std::string &section, const std::string &key) const
- int getIntValue (const std::string &section, const std::string &key) const

Public Attributes

- std::string fileName = std::filesystem::path(__FILE__).filename().string()

3.2.1 Detailed Description

Definition at line 14 of file ConfigReader.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ConfigReader()

```
ConfigReader::ConfigReader (
            const std::string & filePath )
```

Definition at line 6 of file ConfigReader.cpp.

3.2.3 Member Function Documentation

3.2.3.1 getIntValue()

```
int ConfigReader::getIntValue (
            const std::string & section,
            const std::string & key ) const
```

Definition at line 75 of file ConfigReader.cpp.

3.2.3.2 getValue()

```
std::string ConfigReader::getValue (
            const std::string & section,
            const std::string & key ) const
```

Definition at line 58 of file ConfigReader.cpp.

3.2.4 Member Data Documentation

3.2.4.1 fileName

```
std::string ConfigReader::fileName = std::filesystem::path(__FILE__).filename().string()
```

Definition at line 22 of file ConfigReader.h.

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

- /home/thanh/SensorStation/include/ConfigReader.h
- /home/thanh/SensorStation/src/ConfigReader.cpp

3.3 Saver Class Reference

```
#include <Saver.h>
```

Public Member Functions

- Saver (const std::string &name, const std::string &nameFile, const std::string &data, std::string name↩ Sensor="")
- void writeError2File ()
- void writeData2File ()

3.3.1 Detailed Description

Definition at line 15 of file Saver.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Saver()

```
Saver::Saver (
            const std::string & name,
            const std::string & nameFile,
            const std::string & data,
            std::string nameSensor = "" )
```

Definition at line 5 of file Saver.cpp.

3.3.3 Member Function Documentation

3.3.3.1 writeData2File()

```
void Saver::writeData2File ( )
```

Definition at line 34 of file Saver.cpp.

3.3.3.2 writeError2File()

```
void Saver::writeError2File ( )
```

Definition at line 24 of file Saver.cpp.

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

- /home/thanh/SensorStation/include/Saver.h
- /home/thanh/SensorStation/src/Saver.cpp

3.4 UDPUnicast Class Reference

```
#include <UDPUnicast.h>
```

Public Member Functions

- UDPUnicast ()
- ~UDPUnicast ()
- void bindToPort (int port)
- void receiveData (std::vector< uint8_t > &data)
- void sendTo (const std::string &ip, int port, const std::vector< uint8_t > &data)
- void stop ()
- void listenfromUnicast ()
- void setOnDataReceived (const std::function< void(const std::vector< uint8_t > &)> &callback)
- void setOnError (const std::function< void(const std::string &)> &callback)

Public Attributes

- std::string fileName = std::filesystem::path(__FILE__).filename().string()

3.4.1 Detailed Description

Definition at line 19 of file UDPUnicast.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 UDPUnicast()

```
UDPUnicast::UDPUnicast ( )
```

Definition at line 5 of file UDPUnicast.cpp.

3.4.2.2 ~UDPUnicast()

```
UDPUnicast::~UDPUnicast ( )
```

Definition at line 14 of file UDPUnicast.cpp.

3.4.3 Member Function Documentation

3.4.3.1 bindToPort()

```
void UDPUnicast::bindToPort (
            int port )
```

Definition at line 90 of file UDPUnicast.cpp.

3.4.3.2 listenfromUnicast()

```
void UDPUnicast::listenfromUnicast ( )
```

Definition at line 122 of file UDPUnicast.cpp.

3.4.3.3 receiveData()

```
void UDPUnicast::receiveData (
            std::vector< uint8_t > & data )
```

Definition at line 60 of file UDPUnicast.cpp.

3.4.3.4 sendTo()

```
void UDPUnicast::sendTo (
            const std::string & ip,
            int port,
            const std::vector< uint8_t > & data )
```

Definition at line 32 of file UDPUnicast.cpp.

3.4.3.5 setOnDataReceived()

```
void UDPUnicast::setOnDataReceived (
            const std::function< void(const std::vector< uint8_t > &)> & callback )
```

Definition at line 118 of file UDPUnicast.cpp.

3.4.3.6 setOnError()

```
void UDPUnicast::setOnError (
            const std::function< void(const std::string &)> & callback )
```

Definition at line 19 of file UDPUnicast.cpp.

3.4.3.7 stop()

```
void UDPUnicast::stop ( )
```

Definition at line 23 of file UDPUnicast.cpp.

3.4.4 Member Data Documentation

3.4.4.1 fileName

```
std::string UDPUnicast::fileName = std::filesystem::path(__FILE__).filename().string()
```

Definition at line 30 of file UDPUnicast.h.

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

- /home/thanh/SensorStation/include/UDPUnicast.h
- /home/thanh/SensorStation/src/UDPUnicast.cpp

Chapter 4

File Documentation

4.1 /home/thanh/SensorStation/CMakeLists.txt File Reference

Functions

- cmake_minimum_required (VERSION 3.10) project(SensorStation) set(CMAKE_CXX_STANDARD 17) set(CMAKE_CXX_STANDARD_REQUIRED True) include_directories(include) set(SOURCES src/main.cpp src/BusinessHandler.cpp src/Saver.cpp src/ConfigReader.cpp src/UDPUnicast.cpp) add_executable(Sensor↩ Station $

4.1.1 Function Documentation

4.1.1.1 cmake_minimum_required()

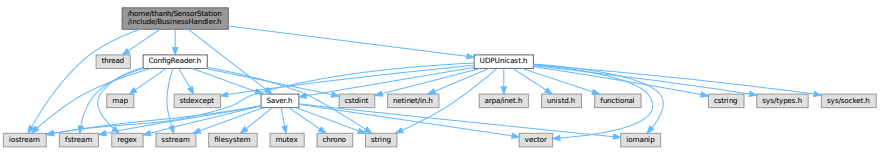
```
cmake_minimum_required (
            VERSION 3.  10 )
```

Definition at line 1 of file CMakeLists.txt.

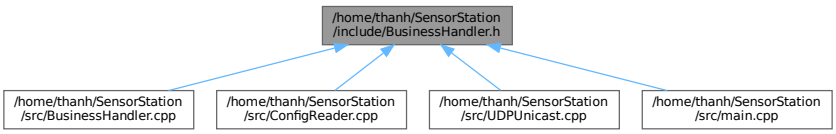
4.2 /home/thanh/SensorStation/include/BusinessHandler.h File Reference

```
#include <iostream>
#include <thread>
#include "ConfigReader.h"
#include "Saver.h"
#include "UDPUnicast.h"
```

Include dependency graph for BusinessHandler.h:



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



Classes

- class BusinessHandler

Functions

- void callError (std::string error)
- void throwError (const std::string &message)
- void cerrError (const std::string &message)

4.2.1 Function Documentation

4.2.1.1 callError()

```
void callError (
            std::string error )
```

Definition at line 133 of file BusinessHandler.cpp.

4.2.1.2 cerrError()

```
void cerrError (
            const std::string & message )
```

Definition at line 145 of file BusinessHandler.cpp.

4.2.1.3 throwError()

```
void throwError (
            const std::string & message )
```

Definition at line 140 of file BusinessHandler.cpp.

4.3 BusinessHandler.h

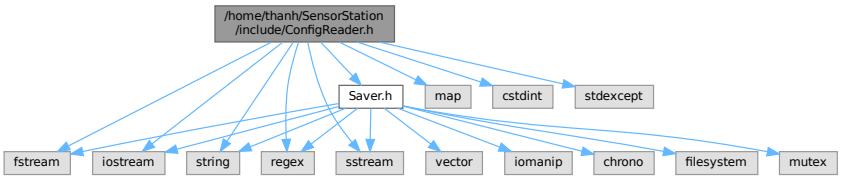
Go to the documentation of this file.

```
00001 #ifndef BUSINESSHANDLER_H
00002 #define BUSINESSHANDLER_H
00003
00004 #include <iostream>
00005 #include <thread>
00006 #include "ConfigReader.h"
00007 #include "Saver.h"
00008 #include "UDPUnicast.h"
00009
00010 void callError(std::string error);
00011 void throwError(const std::string &message);
00012 void cerrError(const std::string &message);
00013
00014 class BusinessHandler{
00015 public:
00016     BusinessHandler(const std::string& configfile);
00017     void run();
00018     void heartbeatSending();
00019     void recevfromSensor();
00020
00021 private:
00022     ConfigReader config;
00023     bool checkConfig();
00024     std::vector<uint8_t> heartbeatData = {0x00, 0x01};
00025     bool running;
00026
00027 };
00028
00029
00030
00031 #endif
```

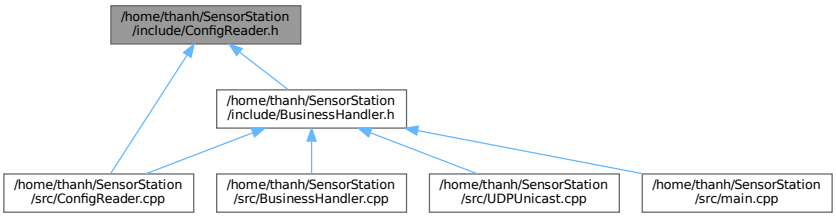
4.4 /home/thanh/SensorStation/include/ConfigReader.h File Reference

```
#include <fstream>
#include <iostream>
#include <string>
#include <map>
#include <cstdint>
#include <stdexcept>
#include <regex>
#include <sstream>
#include "Saver.h"
```

Include dependency graph for ConfigReader.h:



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



Classes

- class ConfigReader

4.5 ConfigReader.h

Go to the documentation of this file.

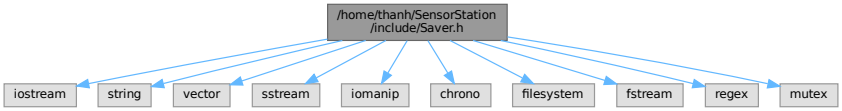
```
00001 #ifndef CONFIGREADER_H
00002 #define CONFIGREADER_H
00003
00004 #include <fstream>
00005 #include <iostream>
00006 #include <string>
00007 #include <map>
00008 #include <cstdint>
00009 #include <stdexcept>
00010 #include <regex>
00011 #include <sstream>
00012 #include "Saver.h"
00013
00014 class ConfigReader{
00015 public:
00016     ConfigReader(const std::string &filePath);
00017
00018     std::string getValue(const std::string &section, const std::string &key) const;
00019
00020     // Check type Integer
00021     int getIntValue(const std::string &section, const std::string &key) const;
00022     std::string fileName = std::filesystem::path(__FILE__).filename().string();
00023
00024 private:
00025     std::map<std::string, std::map<std::string, std::string» configData;
00026     std::string getFilename();
00027     void parseConfig(const std::string &filePath);
00028     std::string trim(const std::string &str);
00029     std::string error;
00030 };
00031
00032 #endif
```

4.6 /home/thanh/SensorStation/include/Saver.h File Reference

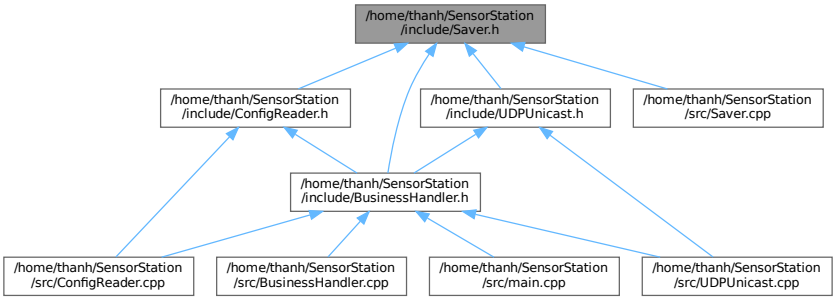
```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <chrono>
#include <filesystem>
```

```
#include <fstream>
#include <regex>
#include <mutex>
```

Include dependency graph for Saver.h:



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



Classes

- class Saver

4.7 Saver.h

Go to the documentation of this file.

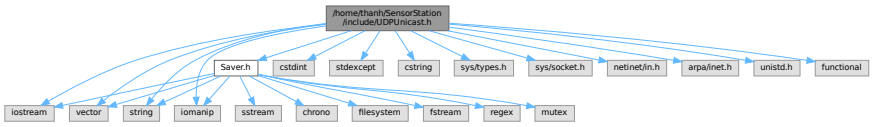
```
00001 #ifndef SAVER_H
00002 #define SAVER_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <vector>
00007 #include <sstream>
00008 #include <iomanip>
00009 #include <chrono>
00010 #include <filesystem>
00011 #include <fstream>
00012 #include <regex>
00013 #include <mutex>
00014
00015 class Saver{
00016 public:
00017     Saver(const std::string& name, const std::string& nameFile, const std::string& data, std::string
      nameSensor = "");
00018     void writeError2File();
00019     void writeData2File();
00020 private:
00021     std::string directoryPath;
00022     std::string namefile;
00023     std::string sensor;
00024     std::string data;
```

```
00025     std::string Pathfile;
00026     std::ofstream file;
00027
00028
00029 };
00030
00031
00032
00033 #endif
```

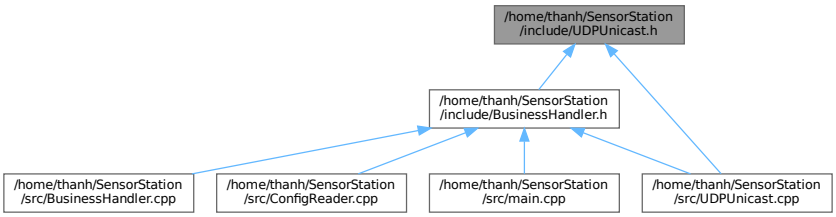
4.8 /home/thanh/SensorStation/include/UDPUnicast.h File Reference

```
#include <iostream>
#include <vector>
#include <cstdint>
#include <stdexcept>
#include <cstring>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <functional>
#include <string>
#include <iomanip>
#include "Saver.h"
```

Include dependency graph for UDPUnicast.h:



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



Classes

- class UDPUnicast

4.9 UDPUnicast.h

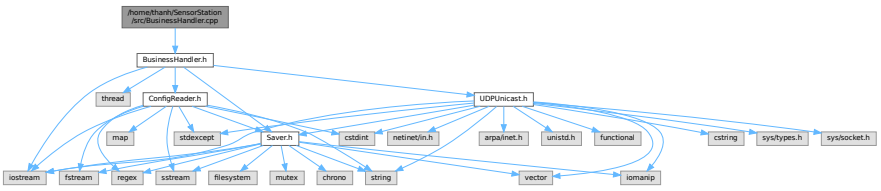
Go to the documentation of this file.

```
00001 #ifndef UDPUNICAST_H
00002 #define UDPUNICAST_H
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <cstdint>
00007 #include <stdexcept>
00008 #include <cstring>        // memset
00009 #include <sys/types.h>    // socket types
00010 #include <sys/socket.h>   // socket(), setsockopt()
00011 #include <netinet/in.h>   // sockaddr_in, INADDR_ANY, htons()
00012 #include <arpa/inet.h>    // inet_addr()
00013 #include <unistd.h>       // close()
00014 #include <functional>
00015 #include <string>
00016 #include <iomanip>
00017 #include "Saver.h"
00018
00019 class UDPUnicast{
00020 public:
00021     UDPUnicast();
00022     ~UDPUnicast();
00023     void bindToPort(int port);
00024     void receiveData(std::vector<uint8_t>& data);
00025     void sendTo(const std::string &ip, int port, const std::vector<uint8_t> &data);
00026     void stop();
00027     void listenfromUnicast();
00028     void setOnDataReceived(const std::function<void(const std::vector<uint8_t> &)> &callback);
00029     void setOnError(const std::function<void(const std::string &)> &callback);
00030     std::string fileName = std::filesystem::path(__FILE__).filename().string();
00031
00032 private:
00033     void processIncomingDatagrams();
00034     int port;
00035     int socketFd;
00036     std::function<void(const std::vector<uint8_t> &)> onDataReceived;
00037     std::function<void(const std::string &)> onError;
00038 };
00039
00040
00041 #endif
```

4.10 /home/thanh/SensorStation/src/BusinessHandler.cpp File Reference

`#include "BusinessHandler.h"`

Include dependency graph for BusinessHandler.cpp:



Functions

- void callError (std::string error)
- void throwError (const std::string &message)
- void cerrError (const std::string &message)

4.10.1 Function Documentation

4.10.1.1 callError()

```
void callError (
            std::string error )
```

Definition at line 133 of file BusinessHandler.cpp.

4.10.1.2 cerrError()

```
void cerrError (
            const std::string & message )
```

Definition at line 145 of file BusinessHandler.cpp.

4.10.1.3 throwError()

```
void throwError (
            const std::string & message )
```

Definition at line 140 of file BusinessHandler.cpp.

4.11 BusinessHandler.cpp

Go to the documentation of this file.

```
00001 #include "BusinessHandler.h"
00002
00003 BusinessHandler::BusinessHandler(const std::string& configfile)
00004     : running(false), config(configfile){}
00005
00006
00007
00008 void BusinessHandler::heartbeatSending() {
00009     try {
00010         // Read the server IP and port from the configuration
00011         std::string serverIp = config.getValue("Server", "ip");
00012         int serverPort = config.getIntValue("Server", "port");
00013
00014         // Create a UDPUnicast instance for sending heartbeats
00015         UDPUnicast udpUnicastHeartbeat;
00016
00017         std::cout << "Starting heartbeat to " << serverIp << ":" << serverPort << std::endl;
00018
00019         // Continuous loop for sending heartbeats periodically
00020         while (true) {
00021             try {
00022                 // Send the heartbeat to the configured server IP and port
00023                 udpUnicastHeartbeat.sendTo(serverIp, serverPort, heartbeatData);
00024
00025                 // Log the successful heartbeat transmission
00026                 std::cout << "Sending heartbeat to " << serverIp << ":" << serverPort << std::endl;
00027
00028                 // Wait for 200 milliseconds before sending the next heartbeat
00029                 std::this_thread::sleep_for(std::chrono::milliseconds(200));
00030             } catch (const std::exception& e) {
00031                 // Catch and log any errors during heartbeat sending
00032                 cerrError(std::string("[Error sending heartbeat: ") + e.what() +"]");
00033
00034             }
00035         }
00036     } catch (const std::exception& e) {
00037         // Catch and log any errors during initialization of heartbeats
00038         // Write Error into log
```

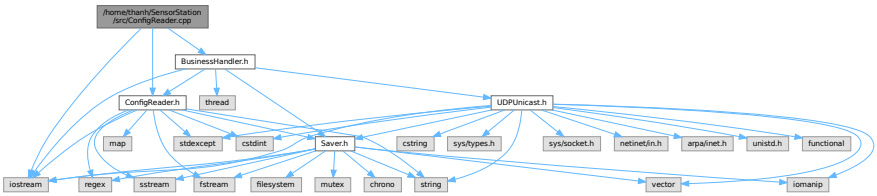
```
00039         cerrError(std::string("[Error in heartbeatSending: ") + e.what() +"]");
00040     }
00041 }
00042
00043 // Function check the configuration
00044 bool BusinessHandler::checkConfig(){
00045     // 1. Validate server configuration
00046     std::string serverIp = config.getValue("Server", "ip");
00047     int serverPort = config.getIntValue("Server", "port");
00048     if (serverIp.empty() || serverPort <= 0 || serverPort > 65535) {
00049         // Write Error into log
00050         throwError("[Invalid server IP or port in configuration.]");
00051     }
00052
00053     // 2. Validate Sensor and Station ports
00054     std::string sensorIp  = config.getValue("Sensor", "ip");
00055     int sensorPort        = config.getIntValue("Sensor", "port");
00056     int stationPort    = config.getIntValue("Station", "port");
00057     if (sensorIp.empty() || sensorPort <= 0 || sensorPort > 65535|| stationPort <= 0 || stationPort >
      65535){
00058         // Write Error into log
00059         throwError( "[Invalid Sensor IP, Sensor port, or Station port in configuration.]");
00060     }
00061
00062     // If execution reaches here, all checks have passed
00063     return true;
00064 }
00065
00066
00067 void BusinessHandler::recevfromSensor(){
00068     std::string fileName = std::filesystem::path(__FILE__).filename().string();
00069
00070     int portStation = config.getIntValue("Station", "port");
00071     std::string nameSensor = config.getValue("Sensor", "name");
00072
00073
00074     std::string ipSever = config.getValue("Server", "ip");
00075     int portServer = config.getIntValue("Server","port");
00076     UDPUnicast udpUnicastreceivedFromSensor;
00077
00078     udpUnicastreceivedFromSensor.bindToPort(portStation);
00079
00080     udpUnicastreceivedFromSensor.setOnDataReceived([fileName,&udpUnicastreceivedFromSensor,
      &nameSensor, ipSever, portServer, this](const std::vector<uint8_t>& data) {
00081         try {
00082             // Log received data
00083             std::cout « "Received data: ";
00084             for (auto byte : data) {
00085                 std::cout « std::hex « static_cast<int>(byte) « "";
00086             }
00087             std::cout « std::dec « std::endl;
00088             udpUnicastreceivedFromSensor.sendTo(ipSever,portServer,data);
00089
00090             // Conver to string
00091             std::string dataStr(data.begin(), data.end());
00092             Saver saver("Data", fileName, dataStr, nameSensor);
00093
00094             saver.writeData2File();
00095             std::cout « "Data sent to " « ipSever « ":" « portServer « std::endl;
00096         } catch (const std::exception& e) {
00097             // Write Error into log
00098             cerrError(std::string("[Error processing data: ") + e.what() + "]");
00099         }
00100
00101     });
00102     udpUnicastreceivedFromSensor.setOnError([this](const std::string &err) {
00103         // Write Error into log
00104         cerrError("[Unicast Error: " + std::string(err) + "]");
00105     });
00106     while (true) {
00107         std::cout«"Station listenning from Sensor......."«std::endl;
00108         udpUnicastreceivedFromSensor.listenfromUnicast();
00109         std::this_thread::sleep_for(std::chrono::milliseconds(200));
00110         udpUnicastreceivedFromSensor.stop();
00111     }
00112 }
00113
00114 void BusinessHandler::run(){
00115     if(!checkConfig()){
00116         std::cerr « "Error in configuration file. Exiting..." « std::endl;
00117     }
00118     running = true;
00119
00120     // Starting Thread
00121     std::thread heartbeatThread(&BusinessHandler::heartbeatSending, this);
00122     std::thread SensorSendThread(&BusinessHandler::recevfromSensor, this);
00123
```

```
00124     if (running == false) {
00125         run();
00126     }
00127     //Stop Thread
00128     if (heartbeatThread.joinable()) heartbeatThread.join();
00129     if(SensorSendThread.joinable()) SensorSendThread.join();
00130 }
00131
00132 // Function log Error
00133 void callError(std::string error){
00134     std::string fileName = std::filesystem::path(__FILE__).filename().string();
00135
00136     Saver saver("Error", fileName, error,"");
00137     saver.writeError2File();
00138 }
00139
00140 void throwError(const std::string &message){
00141     callError(message);
00142     throw std::runtime_error(message);
00143 }
00144
00145 void cerrError(const std::string &message){
00146     callError(message);
00147     std::cerr<<message<<std::endl;
00148 }
00149
00150
```

4.12 /home/thanh/SensorStation/src/ConfigReader.cpp File Reference

```
#include <iostream>
#include "ConfigReader.h"
#include "BusinessHandler.h"
```

Include dependency graph for ConfigReader.cpp:



4.13 ConfigReader.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "ConfigReader.h"
00003 #include "BusinessHandler.h"
00004
00005 // Constructor
00006 ConfigReader::ConfigReader(const std::string &filePath) {
00007     parseConfig(filePath);
00008 }
00009
00010 // Helper function to trim whitespace from a string
00011 std::string ConfigReader::trim(const std::string &str) {
00012     //    (space)
00013     // \t (tab)
00014     // \n (endl)
00015     // \r (return start of line)-
00016     size_t first = str.find_first_not_of(" \t\n\r");
00017     if (first == std::string::npos) return "";
00018     size_t last = str.find_last_not_of(" \t\n\r");
00019     return str.substr(first, last - first + 1);
00020 }
00021
```

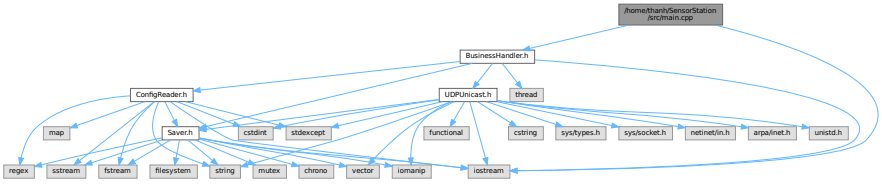
```
00022 void ConfigReader::parseConfig(const std::string &filePath) {
00023     // Open file .ini
00024     std::ifstream file(filePath);
00025     if (!file) {
00026         throwError("[Unable to open file: " +filePath+ "]");
00027     }
00028 
00029     std::string line, currentSection;
00030     while(std::getline(file, line)){
00031         line = trim(line);
00032         // Skip comments and empty lines
00033         if (line.empty() || line[0] == ';' || line[0] == '#') continue;
00034 
00035         if(line[0] == '[' && line.back() == ']'){
00036             currentSection = line.substr(1, line.size() - 2);
00037             currentSection = trim(currentSection);
00038 
00039         }
00040         // Handle key-value pairs
00041         else if (!currentSection.empty()) {
00042             size_t equalPos = line.find('=');
00043             if (equalPos == std::string::npos) continue;
00044             // get Key of Section
00045             std::string key = trim(line.substr(0, equalPos));
00046 
00047             // get Value of Key
00048             std::string value = trim(line.substr(equalPos + 1));
00049 
00050             configData[currentSection][key] = value;
00051         }
00052     }
00053 }
00054 
00055 
00056 
00057 // Function get string value
00058 std::string ConfigReader::getValue(const std::string &section, const std::string &key) const {
00059     auto sectionIt = configData.find(section);
00060     if (sectionIt == configData.end()) {
00061         // Write Error into log
00062         throwError("[Section not found: " + section + "]");
00063     }
00064 
00065     auto keyIt = sectionIt->second.find(key);
00066     if (keyIt == sectionIt->second.end()) {
00067         // Write Error into log
00068         throwError("[Key of "+section+" not found]");
00069     }
00070 
00071     return keyIt->second;
00072 }
00073 
00074 // Function get an integer value
00075 int ConfigReader::getIntValue(const std::string &section, const std::string &key) const {
00076     try {
00077         auto sectionIt = configData.find(section);
00078 
00079         std::string valueStr = getValue(section, key);
00080 
00081         std::regex pattern("^[0-9]+$");
00082         if (!std::regex_match(valueStr, pattern)) {
00083             throwError("[Value contains non-numeric or have special characters: " + valueStr + "]");
00084         }
00085 
00086         int value = std::stoi(valueStr);
00087 
00088         return value;
00089     } catch (const std::invalid_argument &e) {
00090         // Write Error into log
00091         throwError("[Invalid integer value for key: " + section + " - " + key +"]");
00092     } catch (const std::out_of_range &e) {
00093         // Write Error into log
00094         throwError("[Integer value out of range for key: " + section + " - " + key +"]");
00095     }
00096     return 0;
00097 }
```

4.14 /home/thanh/SensorStation/src/main.cpp File Reference

main file of the system.

```
#include "BusinessHandler.h"
#include <iostream>
```

Include dependency graph for main.cpp:



Functions

- int main ()

4.14.1 Detailed Description

main file of the system.

Funcion `main()` initialize `BusinessHandler` with file ini `config.ini`, after that call `BusinessHandler` with handler.run()`.

Returns

return 0;

Definition in file main.cpp.

4.14.2 Function Documentation

4.14.2.1 main()

```
int main ( )
```

Definition at line 13 of file main.cpp.

4.15 main.cpp

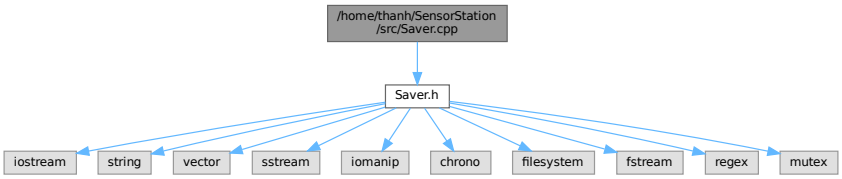
Go to the documentation of this file.

```
00001
00002 #include "BusinessHandler.h"
00003 #include <iostream>
00013 int main() {
00014     // Push file .ini into BusinessHandler
00015     BusinessHandler handler("config.ini");
00016
00017     handler.run();
00018     return 0;
00019 }
```

4.16 /home/thanh/SensorStation/src/Saver.cpp File Reference

```
#include "Saver.h"
```

Include dependency graph for Saver.cpp:



4.17 Saver.cpp

Go to the documentation of this file.

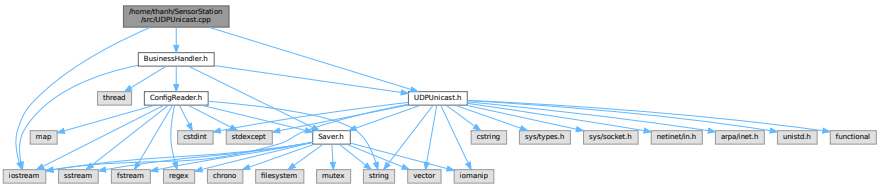
```
00001 #include "Saver.h"
00002
00003
00004
00005 Saver::Saver(const std::string& name, const std::string& nameFile, const std::string& data,
      std::string nameSensor)
00006    : namefile(nameFile),sensor(nameSensor),data(data), directoryPath("logs") {
00007
00008     std::filesystem::create_directories(directoryPath);
00009
00010     if(name == "Error"){
00011         Pathfile = directoryPath + "/[System][SensorStation].csv";
00012     }else if(name == "Data"){
00013         Pathfile = directoryPath + "/[Data]["+sensor+"].csv";
00014     }
00015
00016     file.open(Pathfile, std::ios::out | std::ios::binary | std::ios::app);
00017     if (!file.is_open()) {
00018         std::cerr « "Failed to open file: " « Pathfile « std::endl;
00019     } else {
00020         std::cout « "Saving data to: " « Pathfile « std::endl;
00021     }
00022 }
00023
00024 void Saver::writeError2File() {
00025     auto now = std::chrono::system_clock::now();
00026     auto nowTime = std::chrono::system_clock::to_time_t(now + std::chrono::hours(7));
00027
00028     file « std::put_time(std::localtime(&nowTime), "[%d%m%Y_%H%M%S]")
00029     « "[Error in " « namefile «"]"«data «std::endl;
00030
00031     file.flush();
00032     file.close();
00033 }
00034 void Saver::writeData2File() {
00035     auto now = std::chrono::system_clock::now();
00036     auto nowTime = std::chrono::system_clock::to_time_t(now + std::chrono::hours(7));
00037
00038     file « std::put_time(std::localtime(&nowTime), "[%d%m%Y_%H%M%S]")
00039          «"["«data«"]"«std::endl;
00040     file.flush();
00041     file.close();
00042 }
```

4.18 /home/thanh/SensorStation/src/UDPUnicast.cpp File Reference

```
#include <iostream>
#include "UDPUnicast.h"
```

```
#include "BusinessHandler.h"
```

Include dependency graph for UDPUnicast.cpp:



4.19 UDPUnicast.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "UDPUnicast.h"
00003 #include "BusinessHandler.h"
00004
00005 UDPUnicast::UDPUnicast()
00006     :port(port) {
00007     socketFd = socket(AF_INET, SOCK_DGRAM, 0);
00008     if (socketFd < 0) {
00009         throwError("Socket creation failed: " + std::string(strerror(errno)));
00010     }
00011 }
00012
00013
00014 UDPUnicast::~UDPUnicast() {
00015     stop();
00016 }
00017
00018
00019 void UDPUnicast::setOnError(const std::function<void(const std::string &)> &callback) {
00020     onError = callback;
00021 }
00022
00023 void UDPUnicast::stop() {
00024     // if (!running) return;
00025     // running = false;
00026     if (socketFd >= 0) {
00027         close(socketFd);
00028         socketFd = -1;
00029     }
00030 }
00031
00032 void UDPUnicast::sendTo(const std::string &ip, int port, const std::vector<uint8_t> &data) {
00033     if (socketFd < 0) {
00034         throwError("Socket is not initialized: " + std::string(strerror(errno)));
00035     }
00036
00037     struct sockaddr_in sendAddr{};
00038     memset(&sendAddr, 0, sizeof(sendAddr));
00039     sendAddr.sin_family = AF_INET;
00040     sendAddr.sin_port = htons(port);
00041
00042     if (inet_pton(AF_INET, ip.c_str(), &sendAddr.sin_addr) <= 0) {
00043         throwError("Invalid IP address: " + ip);
00044     }
00045
00046     ssize_t sentLen = sendto(socketFd, data.data(), data.size(), 0, (struct sockaddr *)&sendAddr,
      sizeof(sendAddr));
00047
00048     if (sentLen < 0) {
00049         std::string errorMsg = "[Failed to send data to " + ip + ":" + std::to_string(port) +
00050                                " - " + strerror(errno) + "]";
00051         cerrError(errorMsg);
00052         return;
00053     }
00054
00055     std::cout « "Sent " « sentLen « " bytes to " « ip « ":" « port « std::endl;
00056 }
00057
00058
00059 // Fuction call 1 time to received Data
00060 void UDPUnicast::receiveData(std::vector<uint8_t>& data){
```

```
00061     if (socketFd < 0) {
00062         throw std::runtime_error("Socket is not initialized");
00063     }
00064 
00065     char buffer[1024];
00066     struct  sockaddr_in senderAddr{};
00067     socklen_t senderAddrLen = sizeof(senderAddr);
00068     ssize_t recvData = recvfrom(socketFd, buffer, sizeof(buffer),0,
00069                         (struct sockaddr *)& senderAddr, &senderAddrLen);
00070 
00071     std::cout « "UDPUnicast Received Data: ";
00072     for(int i = 0; i < recvData; i++){
00073         std::cout« std::hex « std::setw(2) « std::setfill('0') « (int)(unsigned char)buffer[i] « " ";
00074     }
00075     std::cout«std::endl;
00076 
00077     if(recvData < 0){
00078         // Write Error into log
00079         cerrError("[Receiving data failed]");
00080     }
00081     data.assign(buffer, buffer + recvData);
00082     // Optional: Log sender information
00083     char senderIP[INET_ADDRSTRLEN];
00084     inet_ntop(AF_INET, &senderAddr.sin_addr, senderIP, sizeof(senderIP));
00085     std::cout « "Received " « std::dec « recvData « " bytes from " « senderIP
00086               « ":" « ntohs(senderAddr.sin_port) « std::endl;
00087 }
00088 
00089 // Fuction bind to Port
00090 void UDPUnicast::bindToPort(int port) {
00091     if (socketFd < 0) {
00092         // Write Error into log
00093         throwError("[Socket is not initialized]");
00094     }
00095 
00096     struct sockaddr_in receivedAddr{};
00097     memset(&receivedAddr, 0, sizeof(receivedAddr));
00098     receivedAddr.sin_family = AF_INET;
00099     receivedAddr.sin_port = htons(port);
00100     receivedAddr.sin_addr.s_addr = INADDR_ANY;
00101 
00102     // Allow address reuse
00103     int optval = 1;
00104     if (setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0) {
00105         // Write Error into log
00106         throwError("[Reuse socket failed]");
00107     }
00108     if (bind(socketFd, (struct sockaddr *)&receivedAddr, sizeof(receivedAddr)) < 0) {
00109         // Write Error into log
00110         throwError("[Socket binding failed]");
00111     } else {
00112         std::cout « "Bind successful on port " « port « std::endl;
00113     }
00114 }
00115 
00116 
00117 
00118 void UDPUnicast::setOnDataReceived(const std::function<void(const std::vector<uint8_t> &)> &callback)
    {
00119     onDataReceived = callback;
00120 }
00121 
00122 void UDPUnicast::listenfromUnicast(){
00123     processIncomingDatagrams();
00124 }
00125 void UDPUnicast::processIncomingDatagrams() {
00126     if (socketFd < 0) {
00127         cerrError("Socket is not initialized, cannot receive data!");
00128         return;
00129     }
00130 
00131     char buffer[4096];
00132     while (true) {
00133         sockaddr_in senderAddr{};
00134         socklen_t addrLen = sizeof(senderAddr);
00135 
00136         ssize_t recvLen = recvfrom(socketFd, buffer, sizeof(buffer), 0,
00137                                    (struct sockaddr *)&senderAddr, &addrLen);
00138 
00139         if (recvLen > 0) {
00140             std::vector<uint8_t> data(buffer, buffer + recvLen);
00141             if (onDataReceived) {
00142                 onDataReceived(data);
00143             }
00144         } else if (recvLen < 0) {
00145             perror("Receive failed");
00146             if (onError) onError("Receive failed");
```

```
00147         }
00148     }
00149 }
```

Index