

Сервер Стригин 22ПТ1
1.0

Создано системой Doxygen 1.9.1

1 Иерархический список классов	1
1.1 Иерархия классов	1
2 Алфавитный указатель классов	1
2.1 Классы	1
3 Список файлов	2
3.1 Файлы	2
4 Классы	2
4.1 Класс base	2
4.1.1 Подробное описание	4
4.1.2 Конструктор(ы)	4
4.1.3 Методы	4
4.2 Класс communicator	5
4.2.1 Подробное описание	7
4.2.2 Конструктор(ы)	7
4.2.3 Методы	8
4.3 Класс critical_error	10
4.3.1 Подробное описание	11
4.3.2 Конструктор(ы)	11
4.4 Класс data_handler	12
4.4.1 Подробное описание	13
4.4.2 Конструктор(ы)	13
4.4.3 Методы	13
4.4.4 Данные класса	14
4.5 Класс logger	14
4.5.1 Подробное описание	15
4.5.2 Методы	15
4.6 Класс UI	15
4.6.1 Подробное описание	16
4.6.2 Конструктор(ы)	17
4.6.3 Методы	17
5 Файлы	19
5.1 Файл base.cpp	19
5.1.1 Подробное описание	19
5.2 Файл base.h	20
5.2.1 Подробное описание	21
5.3 Файл communicator.cpp	21
5.3.1 Подробное описание	21
5.4 Файл communicator.h	22
5.4.1 Подробное описание	23
5.5 Файл data_handler.cpp	23
5.5.1 Подробное описание	24

5.6 Файл <code>error.h</code>	24
5.6.1 Подробное описание	25
5.7 Файл <code>logger.cpp</code>	26
5.7.1 Подробное описание	26
5.8 Файл <code>logger.h</code>	26
5.8.1 Подробное описание	27
5.9 Файл <code>main.cpp</code>	28
5.9.1 Подробное описание	28
5.10 Файл <code>ui.cpp</code>	29
5.10.1 Подробное описание	29
5.11 Файл <code>ui.h</code>	29
5.11.1 Подробное описание	30
Предметный указатель	31

1 Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

<code>base</code>	2
<code>communicator</code>	5
<code>data_handler</code>	12
<code>logger</code>	14
<code>std::runtime_error</code>	
<code>critical_error</code>	10
<code>UI</code>	15

2 Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

<code>base</code>	
Класс базы данных	2
<code>communicator</code>	
Класс коммуникатор	5
<code>critical_error</code>	
Класс ошибок	10

data_handler	Класс обработчик данных	12
logger	Класс логгера	14
UI	Класс пользовательского интерфейса	15

3 Список файлов

3.1 Файлы

Полный список документированных файлов.

base.cpp	Исполняемый файл для модуля base	19
base.h	Заголовочный файл для модуля base	20
communicator.cpp	Исполняемый файл для модуля communicator	21
communicator.h	Заголовочный файл для модулей communicator и data_handler	22
data_handler.cpp	Исполняемый файл для модуля data_handler	23
error.h	Заголовочный файл для модуля error	24
logger.cpp	Исполняемый файл для модуля logger	26
logger.h	Заголовочный файл для модуля logger	26
main.cpp	Главный файл проекта	28
ui.cpp	Исполняемый файл для модуля ui	29
ui.h	Заголовочный файл для модуля UI	29

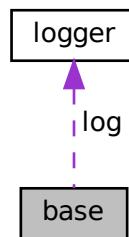
4 Классы

4.1 Класс base

Класс базы данных

```
#include <base.h>
```

Граф связей класса base:



Открытые члены

- `std::vector< std::string > get_logins ()`
Метод для получения вектора с логинами
- `std::vector< std::string > get_passwords ()`
Метод для получения вектора с паролями
- `void read_base (std::ifstream &b)`
Метод чтения базы данных
- `base (std::string base_loc, std::string log_loc)`
Конструктор инициализации
- `~base ()`
Деструктор класса

Открытые атрибуты

- `logger log`
Объекта класса logger для ведения журнала
- `std::string log_location`
Расположение лог файла
- `std::ifstream cl_base`
Объект потока ifstream для чтения базы данных

Закрытые члены

- `std::vector< std::string > id_s (std::vector< std::string > logins)`
Метод разбора логинов
- `std::vector< std::string > password_s (std::vector< std::string > pswds)`
Метод разбора паролей

Закрытые данные

- `std::vector< std::string > logins`
Вектор для хранения логинов
- `std::vector< std::string > passwords`
Вектор для хранения паролей
- `std::vector< std::string > clients`
Вектор для хранения логинов/паролей

4.1.1 Подробное описание

Класс базы данных

Осуществляется работа с базой данных. Чтение осуществляется в методе `read_base` Получение данных о пользователе в методах `id_s` и `password_s`

4.1.2 Конструктор(ы)

4.1.2.1 `base()` `base::base (`
`std::string base_loc,`
`std::string log_loc)`

Конструктор инициализации

Открывается файл базы данных

Аргументы

in	base_loc	Расположения файла базы данных
in	log_loc	Расположения лог файла

Исключения

<code>critical_error</code> ,если	не удалось открыть файл с базой данных
-----------------------------------	--

4.1.2.2 `~base()` `base::~base ()`

Деструктор класса

Закрывается файл с базой данных

4.1.3 Методы

```
4.1.3.1 id_s() std::vector< std::string > base::id_s (
        std::vector< std::string > logins ) [private]
```

Метод разбора логинов

Из строки логин/пароль получается логин

Аргументы

in	logins	Вектор для хранения логинов
----	--------	-----------------------------

```
4.1.3.2 password_s() std::vector< std::string > base::password_s (
        std::vector< std::string > pswds ) [private]
```

Метод разбора паролей

Из строки логин/пароль получается пароль

Аргументы

in	pswds	Вектор для хранения паролей
----	-------	-----------------------------

```
4.1.3.3 read_base() void base::read_base (
        std::ifstream & b )
```

Метод чтения базы данных

Построчно считывается содержимое базы данных. Содержимое в формате логин/пароль

Аргументы

in	base	Объект потока ifstream с открытой базой
----	------	---

Исключения

critical_error , если	в строке встретиться разделитель ":" больше 1 раза или не встретиться
---------------------------------------	---

Объявления и описания членов классов находятся в файлах:

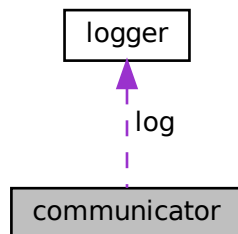
- [base.h](#)
- [base.cpp](#)

4.2 Класс communicator

Класс коммуникатор

```
#include <communicator.h>
```

Граф связей класса communicator:



Открытые члены

- void `connect_to_cl` ()
Подсоединение к клиенту
- int `client_auth` ()
Аутентификация клиента
- std::string `SALT_generate` ()
Генерация соли
- std::string `convert_to_hex` (uint64_t)
Конвертирование в 16-ричный формат
- void `send_data` (std::string data, std::string msg)
Отправка данных клиенту
- std::string `recv_data` (std::string messg)
Прием данных от клиента
- std::string `hash_gen` (std::string &salt, std::string &password)
Генерация хеша
- void `close_sock` ()
Закрытие сокета
- void `start` ()
Запуск сервера
- `communicator` (uint port, std::string base_loc, std::string log_loc)
Конструктор инициализации

Открытые атрибуты

- int `serverSocket`
Сокеты сервера и клиента
- int `clientSocket`
- `logger log`
Объекта класса `loger` для ведения журнала
- std::string `cl_id`
Айди клиента, расположение лог файла
- std::string `log_location`

Закрытые данные

- struct sockaddr_in serverAddr clientAddr
Структуры адреса сервера и клиента
- socklen_t addr_size
Размер адреса
- std::string base_location
Расположение базы данных
- std::vector< std::string > cl_ids
Вектора с логинами и паролями
- std::vector< std::string > cl_passes
- size_t buflen =1024
Размер буффера
- std::unique_ptr< char[]> buffer {new char[buflen]}
- Unique_ptr для приема и отправки сообщений
- uint p
Порт
- std::string digits [16] = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"}
Набор 16-ричных символов для генерации соли

4.2.1 Подробное описание

Класс коммуникатор

Порт, расположение логфайла и базы данных устанавливаются в конструкторе

Для начала работы сервера используется метод start Сетевое взаимодействие осуществляется в методах connect_to_cl и client_auth

4.2.2 Конструктор(ы)

4.2.2.1 communicator() communicator::communicator (
uint port,
std::string base_loc,
std::string log_loc)

Конструктор инициализации

Аргументы

in	port	Номер порта для привязки сокета
in	base_loc	Расположение базы данных
in	log_loc	Расположение лог файла

Инициализируется значение порта, расположения базы данных и лог файла

4.2.3 Методы

4.2.3.1 `client_auth()` `int communicator::client_auth ()`

Аутентификация клиента

Осуществляется аутентификация клиента посредством генерации хеша из соль+пароль

Возвращает

Функция возвращает 1, если не удалось аутентифицировать клиента

4.2.3.2 `close_sock()` `void communicator::close_sock ()`

Закрытие сокета

Метод для закрытия сокета клиента, используемый вне модуля `communicator`

4.2.3.3 `connect_to_cl()` `void communicator::connect_to_cl ()`

Подсоединение к клиенту

Осуществляется установление соединения с клиентом. Сервер работает в режиме `listen`

Возвращает

Функция ничего не возвращает

Исключения

<code>critical_error</code> ,если	не удалось встать на прослушку порта
-----------------------------------	--------------------------------------

4.2.3.4 `convert_to_hex()` `std::string communicator::convert_to_hex (` `uint64_t x)`

Конвертирование в 16-ричный формат

Осуществляется конвертирование числа в 16-ричную строку. Используются в методе [SALT_generate\(\)](#)

Аргументы

<code>in</code>	<code>d_salt</code>	Случайно сгенерированное число для конвертации
-----------------	---------------------	--

Возвращает

Функция возвращает сгенерированную соль

```
4.2.3.5 hash_gen() std::string communicator::hash_gen (
    std::string & salt,
    std::string & password )
```

Генерация хеша

Хеш генерируется на основе соли и пароля

Аргументы

in	salt	Соль, используемая для генерации
in	pswd	Пароль клиента, используемый для генерации

Возвращает

Функция возвращает сгенерированный хеш

```
4.2.3.6 recv_data() std::string communicator::recv_data (
    std::string messg )
```

Прием данных от клиента

Осуществляется прием данных от клиента с помощью `unique_ptr`. Во внутреннем цикле обрабатывается ситуация, когда размер сообщения, больше размера буфера

Аргументы

in	msg	Строка для записи принятого сообщения
----	-----	---------------------------------------

Исключения

В	случае возникновения ошибки обрывается соединение с клиентом и записывается соответствующее сообщение в лог файл
---	--

```
4.2.3.7 SALT_generate() std::string communicator::SALT_generate ( )
```

Генерация соли

Осуществляется генерация 64-разрядного числа, затем число конвертируется в 16-ричную строку, которая дополняется 0 до длины 16 при необходимости

Возвращает

Функция возвращает сгенерированную соль

```
4.2.3.8  send_data()  void communicator::send_data (
                        std::string data,
                        std::string msg )
```

Отправка данных клиенту

Осуществляется отправка данных клиенту с помощью `unique_ptr`

Аргументы

in	data	Строка, которую необходимо отправить
in	log_msg	Строка, которая запишется в лог файл при возникновении ошибки во время отправки

Исключения

В	случае возникновения ошибки обрывается соединение с клиентом и записывается соответствующее сообщение в лог файл
---	--

```
4.2.3.9  start()  void communicator::start ( )
```

Запуск сервера

Создание сокета сервера и его привязка к локальному адресу

Исключения

critical_error ,если	не удалось создать или привязать сокет
--------------------------------------	--

Объявления и описания членов классов находятся в файлах:

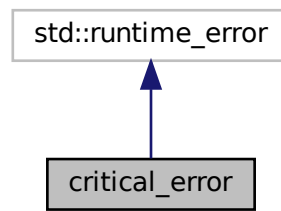
- [communicator.h](#)
- [communicator.cpp](#)

4.3 Класс `critical_error`

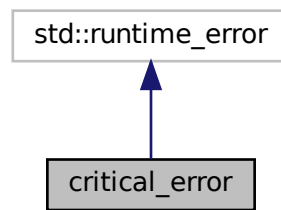
Класс ошибок

```
#include <error.h>
```

Граф наследования: `critical_error`:



Граф связей класса `critical_error`:



Открытые члены

- `critical_error` (`const std::string &s`)
Конструктор ошибки

4.3.1 Подробное описание

Класс ошибок

Используется для отлова специфических ошибок, возникающих в ходе работы модулей В конструкторе указывается строка с сообщением ошибки

4.3.2 Конструктор(ы)

4.3.2.1 `critical_error()` `critical_error::critical_error (`
`const std::string & s) [inline]`

Конструктор ошибки

Аргументы

in	s	Сообщение об ошибке
----	---	---------------------

Объявления и описания членов класса находятся в файле:

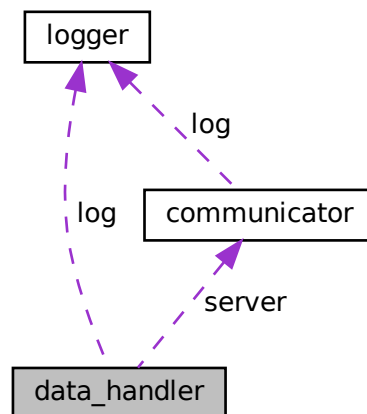
- [error.h](#)

4.4 Класс data_handler

Класс обработчик данных

```
#include <communicator.h>
```

Граф связей класса data_handler:



Открытые члены

- double [calculation](#) (double number)
Калькулятор
- int [handle_data](#) ()
Обработка данных векторов
- [data_handler](#) ([communicator](#) &[server](#), std::string [log](#))
Конструктор инициализации

Открытые атрибуты

- [logger log](#)
Объекта класса logger для ведения журнала
- std::string [log_location](#)
Расположение лог файла
- double [result](#)
Переменная для хранения результата вычислений

Закрытые данные

- `communicator` & `server`
Объект модуля `communicator`.
- `uint32_t` `nums`
Количество векторов, которые необходимо обработать

4.4.1 Подробное описание

Класс обработчик данных

Обработка данных, необходимых для вычислений: прием, отправка, вычисления Прием и отправка осуществляются в методе `handle_data`. Вычисления в методе `calculation`

4.4.2 Конструктор(ы)

4.4.2.1 `data_handler()` `data_handler::data_handler (`
`communicator & server,`
`std::string log)`

Конструктор инициализации

Инициализируется объект модуля `communicator`, расположение лог файла Принимается от клиента значение числа векторов

Аргументы

in	server	Объект модуля <code>communicator</code>
in	log	Расположение лог файла

Предупреждения

В случае приема неверного типа данных соединение с клиентом обрывается

4.4.3 Методы

4.4.3.1 `calculation()` `double data_handler::calculation (`
`double number)`

Калькулятор

Производит возведение элемента вектора в квадрат

Аргументы

in	number	Число полученное в ходе работы метода <code>handle_data</code>
----	--------	--

Предупреждения

В случае переполнения возвращается максимум или минимум типа данных

Возвращает

Функция возвращает число, возведенное в квадрат

4.4.3.2 `handle_data()` `int data_handler::handle_data ()`

Обработка данных векторов

Производит прием данных векторов от клиента и отправка результата вычислений по этим данным

Предупреждения

В случае приема неверного типа данных соединение с клиентом обрывается

4.4.4 Данные класса

4.4.4.1 `server` `communicator&` `data_handler::server` [private]

Объект модуля `communicator`.

Используется для доступа к сокету клиента, чтобы отправлять результаты вычислений

Объявления и описания членов классов находятся в файлах:

- `communicator.h`
- `data_handler.cpp`

4.5 Класс `logger`

Класс логгера

```
#include <logger.h>
```

Открытые члены

- `int write_log (std::string log_loc, std::string message)`
Метод записи сообщения в лог файл

Открытые атрибуты

- `std::ofstream` [log](#)
Объект `ofstream` для открытия файла для записи

4.5.1 Подробное описание

Класс логгера

Запись сообщений в лог файла осуществляется в методе `write_log`. Отсутствует конструктор.

4.5.2 Методы

4.5.2.1 `write_log()` `int logger::write_log (`
`std::string log_loc,`
`std::string message)`

Метод записи сообщения в лог файл

С помощью библиотеки `chrono` получается текущее время, затем записывается сообщение в файл в формате время/сообщение. В начале сеанса записи файл открывается, в конце закрывается.

Аргументы

in	<code>log_loc</code>	Расположение лог файла
in	<code>message</code>	Сообщение для записи в лог файл

Исключения

critical_error , если	<code>log_loc</code> указывает путь к несуществующему файлу
---------------------------------------	---

Объявления и описания членов классов находятся в файлах:

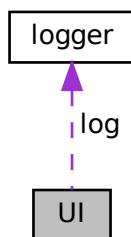
- [logger.h](#)
- [logger.cpp](#)

4.6 Класс UI

Класс пользовательского интерфейса

```
#include <ui.h>
```

Граф связей класса UI:



Открытые члены

- [UI](#) (int argc, char *argv[])
Конструктор "начала работы".
- uint [get_port](#) ()
Метод получения порта
- std::string [get_base_loc](#) ()
Метод получения расоложения базы данных
- std::string [get_log_loc](#) ()
Метод получения расоложения лог файла

Открытые атрибуты

- [logger log](#)
Объект модуля logger для записи сообщений в лог файл
- po::options_description [desc](#)
Описание параметров комстроки
- po::variables_map [vm](#)
Хранит значение параметров командной строки и их опций
- std::string [log_loc](#)
Расположение лог файла

Закрытые данные

- uint [port](#)
Порт для привязки сервера
- std::string [base_loc](#)
Расположение базы данных

4.6.1 Подробное описание

Класс пользовательского интерфейса

Разбор комстроки осуществляется в конструкторе
Происходит старт работы сервера

4.6.2 Конструктор(ы)

4.6.2.1 UI() UI::UI (
int argc,
char * argv[])

Конструктор "начала работы".

Разбирается комстрока и запускается сервер

Аргументы

in	argc	Количество аргументов комстроки
in	argv	Значения аргументов комстроки

Предупреждения

Просиходит "отлов" ошибок типа [critical_error](#), возникших в ходе работы модулей

4.6.3 Методы

4.6.3.1 get_base_loc() std::string UI::get_base_loc ()

Метод получения расоложения базы данных

Поддерживается ввод нескольких значений. Возвращено будет последнее введенное

Возвращает

Из разобранной комстроки возвращается строка с расположением базы данных, введенная пользователем

4.6.3.2 get_log_loc() std::string UI::get_log_loc ()

Метод получения расоложения лог файла

Поддерживается ввод нескольких значений. Возвращено будет последнее введенное

Возвращает

Из разобранной комстроки возвращается строка с расположением лог файла, введенная пользователем

4.6.3.3 `get_port()` `uint UI::get_port ()`

Метод получения порта

Поддерживается ввод нескольких значений. Возвращено будет последнее введенное

Возвращает

Из разобранный строки возвращается значение порта, введенное пользователем

Исключения

<code>critical_error</code> ,если	введен системный порт
-----------------------------------	-----------------------

Объявления и описания членов классов находятся в файлах:

- `ui.h`
- `ui.cpp`

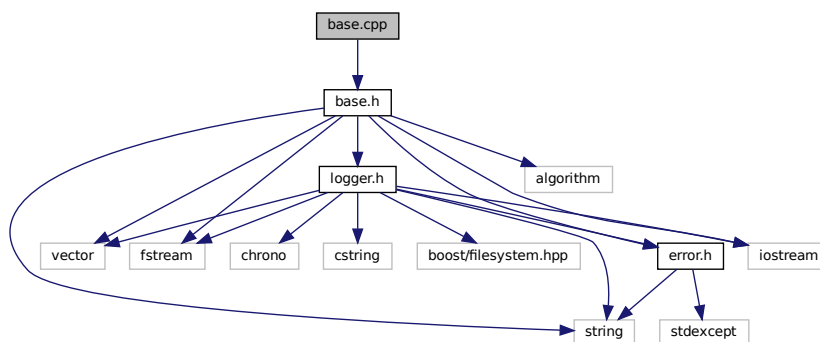
5 Файлы

5.1 Файл `base.cpp`

Исполняемый файл для модуля `base`.

```
#include "base.h"
```

Граф включаемых заголовочных файлов для `base.cpp`:



5.1.1 Подробное описание

Исполняемый файл для модуля `base`.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

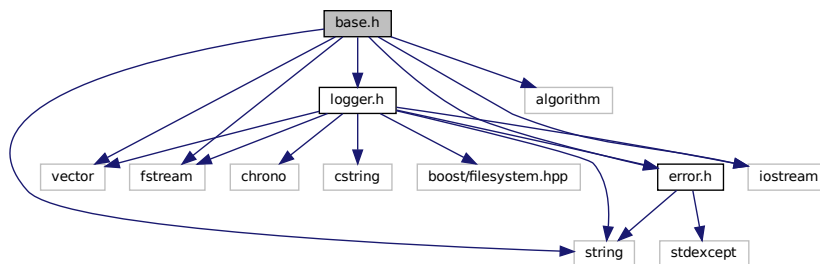
ИБСТ ПГУ

5.2 Файл base.h

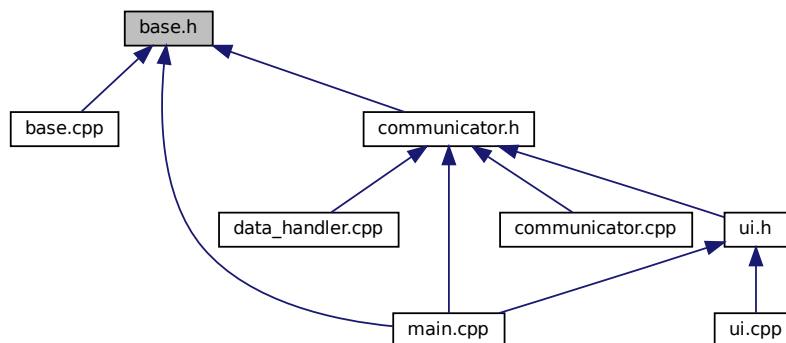
Заголовочный файл для модуля base.

```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <algorithm>
#include "logger.h"
#include "error.h"
```

Граф включаемых заголовочных файлов для base.h:



Граф файлов, в которые включается этот файл:



Классы

- class `base`
Класс базы данных

5.2.1 Подробное описание

Заголовочный файл для модуля base.

Автор

Стригин А.В.

Версия

1.0

Data

23.12.2023

Авторство

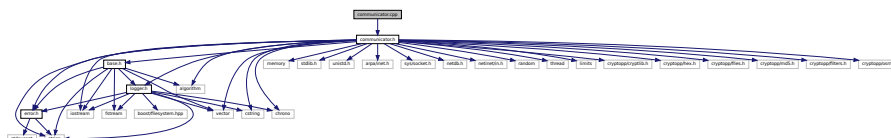
ИБСТ ПГУ

5.3 Файл communicator.cpp

Исполняемый файл для модуля communicator.

```
#include "communicator.h"
```

Граф включаемых заголовочных файлов для communicator.cpp:



5.3.1 Подробное описание

Исполняемый файл для модуля communicator.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

5.4 Файл communicator.h

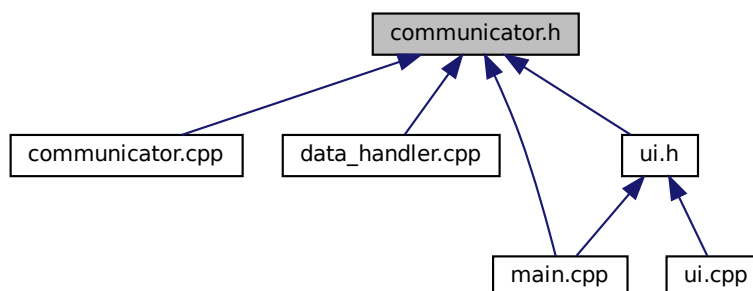
Заголовочный файл для модулей communicator и [data_handler](#).

```
#include <iostream>
#include <string>
#include <vector>
#include <cstring>
#include <algorithm>
#include <memory>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <random>
#include <chrono>
#include <thread>
#include <limits>
#include "base.h"
#include "logger.h"
#include "error.h"
#include <cryptopp/cryptlib.h>
#include <cryptopp/hex.h>
#include <cryptopp/files.h>
#include <cryptopp/md5.h>
#include <cryptopp/filters.h>
#include <cryptopp/osrng.h>
```

Граф включаемых заголовочных файлов для communicator.h:



Граф файлов, в которые включается этот файл:



Классы

- class **communicator**
Класс коммуникатор
- class **data_handler**
Класс обработчик данных

Макросы

- `#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1`

5.4.1 Подробное описание

Заголовочный файл для модулей communicator и [data_handler](#).

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

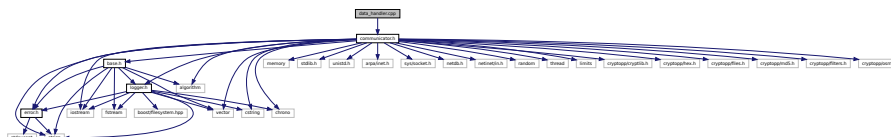
ИБСТ ПГУ

5.5 Файл data_handler.cpp

Исполняемый файл для модуля `data_handler`.

```
#include "communicator.h"
```

Граф включаемых заголовочных файлов для `data_handler.cpp`:



5.5.1 Подробное описание

Исполняемый файл для модуля [data_handler](#).

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

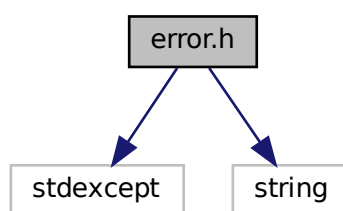
5.6 Файл error.h

Заголовочный файл для модуля error.

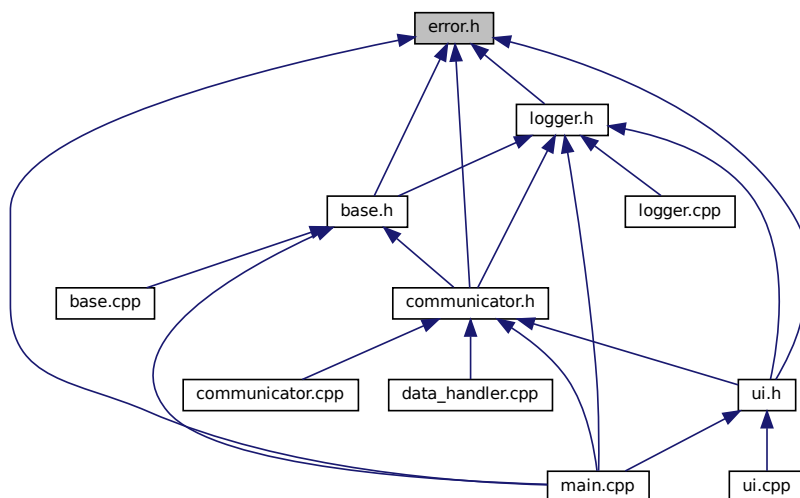
```
#include <stdexcept>
```

```
#include <string>
```

Граф включаемых заголовочных файлов для error.h:



Граф файлов, в которые включается этот файл:



Классы

- class `critical_error`
Класс ошибок

5.6.1 Подробное описание

Заголовочный файл для модуля error.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

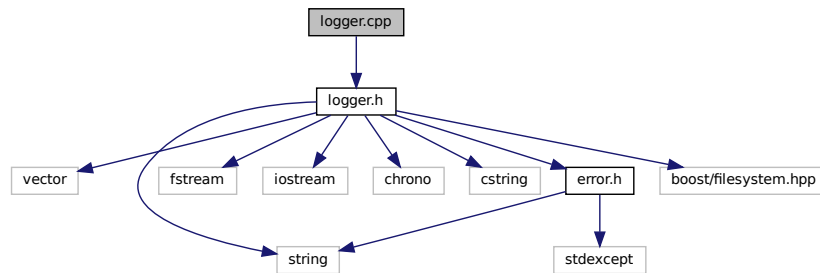
ИБСТ ПГУ

5.7 Файл logger.cpp

Исполняемый файл для модуля logger.

```
#include "logger.h"
```

Граф включаемых заголовочных файлов для logger.cpp:



5.7.1 Подробное описание

Исполняемый файл для модуля logger.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

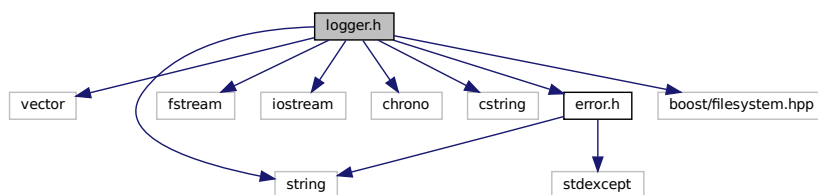
5.8 Файл logger.h

Заголовочный файл для модуля logger.

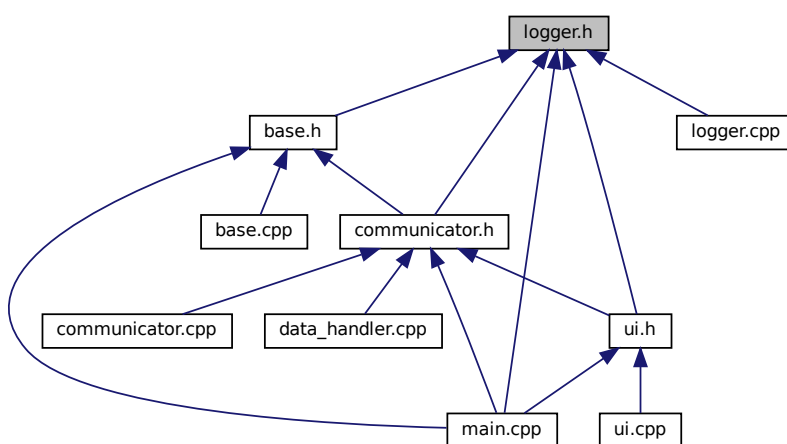
```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <chrono>
#include <cstring>
#include "error.h"
```

```
#include <boost/filesystem.hpp>
```

Граф включаемых заголовочных файлов для logger.h:



Граф файлов, в которые включается этот файл:



Классы

- class `logger`
Класс логгера

5.8.1 Подробное описание

Заголовочный файл для модуля `logger`.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

5.9 Файл main.cpp

Главный файл проекта

```
#include "ui.h"  
#include "base.h"  
#include "communicator.h"  
#include "logger.h"  
#include "error.h"
```

Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, char *argv[])

5.9.1 Подробное описание

Главный файл проекта

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

5.10 Файл ui.cpp

Исполняемый файл для модуля ui.

```
#include "ui.h"
#include <boost/program_options.hpp>
Граф включаемых заголовочных файлов для ui.cpp:
```



5.10.1 Подробное описание

Исполняемый файл для модуля ui.

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

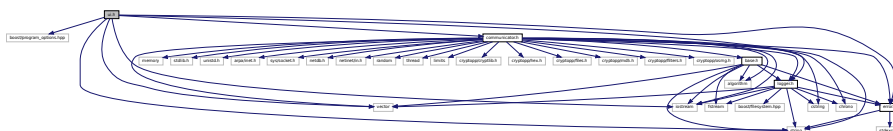
ИБСТ ПГУ

5.11 Файл ui.h

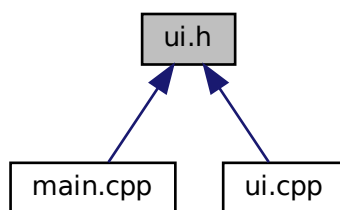
Заголовочный файл для модуля [UI](#).

```
#include <boost/program_options.hpp>
#include <iostream>
#include <string>
#include <vector>
#include "communicator.h"
#include "error.h"
#include "logger.h"
```

Граф включаемых заголовочных файлов для ui.h:



Граф файлов, в которые включается этот файл:



Классы

- class [UI](#)

Класс пользовательского интерфейса

5.11.1 Подробное описание

Заголовочный файл для модуля [UI](#).

Автор

Стригин А.В.

Версия

1.0

Дата

23.12.2023

Авторство

ИБСТ ПГУ

Предметный указатель

~base
 base, [4](#)

base, [2](#)
 ~base, [4](#)
 base, [4](#)
 id_s, [4](#)
 password_s, [5](#)
 read_base, [5](#)

base.cpp, [19](#)
base.h, [20](#)

calculation
 data_handler, [13](#)

client_auth
 communicator, [8](#)

close_sock
 communicator, [8](#)

communicator, [5](#)
 client_auth, [8](#)
 close_sock, [8](#)
 communicator, [7](#)
 connect_to_cl, [8](#)
 convert_to_hex, [8](#)
 hash_gen, [9](#)
 recv_data, [9](#)
 SALT_generate, [9](#)
 send_data, [10](#)
 start, [10](#)

communicator.cpp, [21](#)
communicator.h, [22](#)

connect_to_cl
 communicator, [8](#)

convert_to_hex
 communicator, [8](#)

critical_error, [10](#)
 critical_error, [11](#)

data_handler, [12](#)
 calculation, [13](#)
 data_handler, [13](#)
 handle_data, [14](#)
 server, [14](#)

data_handler.cpp, [23](#)

error.h, [24](#)

get_base_loc
 UI, [17](#)

get_log_loc
 UI, [17](#)

get_port
 UI, [17](#)

handle_data
 data_handler, [14](#)

hash_gen
 communicator, [9](#)

id_s
 base, [4](#)

logger, [14](#)
 write_log, [15](#)

logger.cpp, [26](#)
logger.h, [26](#)

main.cpp, [28](#)

password_s
 base, [5](#)

read_base
 base, [5](#)

recv_data
 communicator, [9](#)

SALT_generate
 communicator, [9](#)

send_data
 communicator, [10](#)

server
 data_handler, [14](#)

start
 communicator, [10](#)

UI, [15](#)
 get_base_loc, [17](#)
 get_log_loc, [17](#)
 get_port, [17](#)
 UI, [17](#)

ui.cpp, [29](#)
ui.h, [29](#)

write_log
 logger, [15](#)