ServerMultiply

Создано системой Doxygen 1.9.4

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список файлов.

Глава 4

Классы

4.1 Класс BaseConnector

Класс для подключения к базе данных через конфигурационный файл.

#include <DataBaseHandler.h>

Открытые члены

- int baseConnection (string baseFile="/etc/vcalc.conf")
  Подключение к базе данных.
- map< string, string > get_data ()
  Получить загруженные данные.

Закрытые данные

- map< string, string > dataBase
  Хранит данные, загруженные из конфигурационного файла.

4.1.1 Подробное описание

Класс для подключения к базе данных через конфигурационный файл.

Этот класс отвечает за загрузку данных из указанного конфигурационного файла и предоставляет доступ к загруженным данным.

4.1.2 Методы

4.1.2.1 baseConnection()

```
int BaseConnector::baseConnection (
            string baseFile = "/etc/vcalc.conf" )
```

Подключение к базе данных.

Загружает данные из указанного конфигурационного файла. По умолчанию используется файл "/etc/vcalc.conf".

Аргументы

baseFile	Путь к конфигурационному файлу (по умолчанию "/etc/vcalc.conf").

Возвращает

Возвращает 0 в случае успешного подключения, или код ошибки в случае неудачи.

```
9                                       {
10   ifstream readFile(baseFile);
11   if (!readFile.is_open()) {
12     throw criticalDataBaseErr("Ошибка: База данных не найдена: " + baseFile);
13   }
14
15   string line;
16   while (getline(readFile, line)) {
17     auto separator = line.find(':');
18     if (separator != string::npos) {
19       string tempLogin = line.substr(0, separator);
20       string tempPass = line.substr(separator + 1);
21
22       if (tempLogin.empty() || tempPass.empty()) {
23         throw dataBaseErr(
24             "Внимание: Обнаружена некорректная пара логин:пароль: " + line);
25       }
26       dataBase[tempLogin] = tempPass;
27     }
28   }
29
30   if (dataBase.empty()) {
31     throw criticalDataBaseErr("Ошибка: База данных пустая");
32   }
33
34   return 0;
35 }
```

4.1.2.2 get_data()

map< string, string > BaseConnector::get_data ( ) [inline]

Получить загруженные данные.

Возвращает карту, содержащую данные, загруженные из конфигурационного файла.

Возвращает

map<string, string> Содержит ключи и значения загруженных данных.

```
59 { return dataBase; }
```

4.1.3 Данные класса

4.1.3.1 dataBase

map<string, string> BaseConnector::dataBase [private]

Хранит данные, загруженные из конфигурационного файла.

Объявления и описания членов классов находятся в файлах:

- /home/stud/Загрузки/TiMP/TiMP_K/headers/DataBaseHandler.h
- /home/stud/Загрузки/TiMP/TiMP_K/DataBaseHandler.cpp

4.2 Класс Calculator

Класс для выполнения расчетов с вектором целых чисел.

#include <Calculator.h>

Открытые члены

- Calculator (vector< int64_t > inputVector)
  Конструктор класса Calculator.
- int64_t sendResult () const
  Получить конечный результат вычислений.

Закрытые данные

- int64_t finalResult
  Хранит конечный результат вычислений.

4.2.1 Подробное описание

Класс для выполнения расчетов с вектором целых чисел.

Этот класс принимает вектор целых чисел в качестве входных данных, выполняет необходимые вычисления и предоставляет метод для получения конечного результата.

4.2.2 Конструктор(ы)

4.2.2.1 Calculator()

```
Calculator::Calculator (
            vector< int64_t > inputVector )
```

Конструктор класса Calculator.

Принимает вектор целых чисел и выполняет необходимые вычисления, чтобы установить значение finalResult.

Аргументы

inputVector	Вектор целых чисел для обработки.

Исключения

std::invalid_argument	Если вектор пуст или содержит недопустимые значения.

```
8                                       {
9   try {
10    int64_t result = 1;
11    for (auto item : inputVector) {
12      if (item == 0) {
13        finalResult = 0;
14        return;
15      }
16
17      if (result > 0 && item > 0 &&
18          result > std::numeric_limits<int64_t>::max() / item) {
19        throw boost::numeric::positive_overflow();
20      }
21      if (result > 0 && item < 0 &&
22          item < std::numeric_limits<int64_t>::min() / result) {
23        throw boost::numeric::negative_overflow();
24      }
25      if (result < 0 && item > 0 &&
26          result < std::numeric_limits<int64_t>::min() / item) {
27        throw boost::numeric::negative_overflow();
28      }
29      if (result < 0 && item < 0 &&
30          result < std::numeric_limits<int64_t>::max() / item) {
31          throw boost::numeric::positive_overflow();
32        }
33
34      result *= item;
35    }
36    finalResult = result;
37  } catch (const boost::numeric::positive_overflow &) {
38    finalResult = std::numeric_limits<int64_t>::max();
39  } catch (const boost::numeric::negative_overflow &) {
40    finalResult = std::numeric_limits<int64_t>::min();
41  }
42 }
```

4.2.3 Методы

4.2.3.1 sendResult()

int64_t Calculator::sendResult ( ) const

Получить конечный результат вычислений.

Возвращает

Возвращает значение finalResult.

```
44 { return finalResult; }
```

4.2.4 Данные класса

4.2.4.1 finalResult

int64_t Calculator::finalResult [private]

Хранит конечный результат вычислений.

Объявления и описания членов классов находятся в файлах:

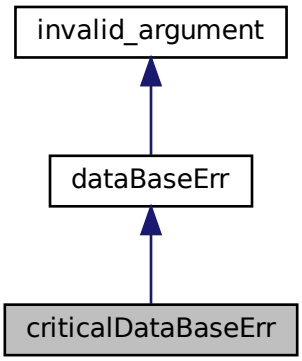
- /home/stud/Загрузки/TiMP/TiMP_K/headers/Calculator.h
- /home/stud/Загрузки/TiMP/TiMP_K/Calculator.cpp

4.3 Класс criticalDataBaseErr

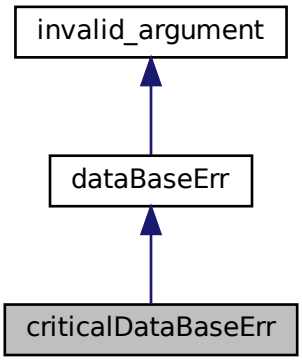
Класс для обработки критических ошибок базы данных.

#include <Errors.h>

Граф наследования:criticalDataBaseErr:



Граф связей класса criticalDataBaseErr:



Открытые члены

- criticalDataBaseErr (const string &s)

  Конструктор для создания объекта criticalDataBaseErr.

4.3.1 Подробное описание

Класс для обработки критических ошибок базы данных.

Этот класс наследуется от dataBaseErr и используется для представления критических ошибок, которые могут привести к серьезным сбоям в работе приложения.

4.3.2 Конструктор(ы)

4.3.2.1 criticalDataBaseErr()

criticalDataBaseErr::criticalDataBaseErr (
const string & s) [inline]

Конструктор для создания объекта criticalDataBaseErr.

Аргументы

s	Сообщение об ошибке.

53 : dataBaseErr(s) {}

Объявления и описания членов класса находятся в файле:

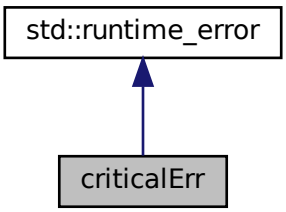
- /home/stud/Загрузки/TiMP/TiMP_K/headers/Errors.h

4.4 Класс criticalErr

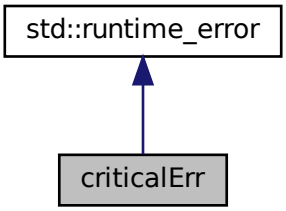
Класс для обработки критических ошибок.

#include <Errors.h>

Граф наследования:criticalErr:



Граф связей класса criticalErr:



Открытые члены

- criticalErr (const std::string &message)
  Конструктор для создания объекта criticalErr.

4.4.1 Подробное описание

Класс для обработки критических ошибок.

Этот класс наследуется от std::runtime_error и используется для представления критических ошибок, которые могут повлиять на выполнение приложения.

4.4.2 Конструктор(ы)

4.4.2.1 criticalErr()

criticalErr::criticalErr (
const std::string & message) [inline], [explicit]

Конструктор для создания объекта criticalErr.

Аргументы

message	Сообщение об ошибке.

```
89        : std::runtime_error(message) {}
```

Объявления и описания членов класса находятся в файле:

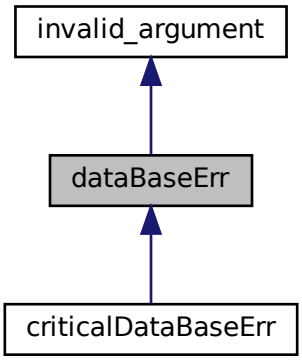
- /home/stud/Загрузки/TiMP/TiMP_K/headers/Errors.h

4.5 Класс dataBaseErr

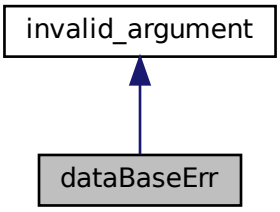
Класс для обработки ошибок базы данных.

#include <Errors.h>

Граф наследования:dataBaseErr:



Граф связей класса dataBaseErr:



Открытые члены

- dataBaseErr (const string &s)

  Конструктор для создания объекта dataBaseErr.

4.5.1 Подробное описание

Класс для обработки ошибок базы данных.

Этот класс наследуется от std::invalid_argument и используется для представления ошибок, связанных с неправильными аргументами при работе с базой данных.

4.5.2 Конструктор(ы)

4.5.2.1 dataBaseErr()

```
dataBaseErr::dataBaseErr (
                const string & s )   [inline]
```

Конструктор для создания объекта dataBaseErr.

Аргументы

s	Сообщение об ошибке.

```
35 : invalid_argument(s) {}
```

Объявления и описания членов класса находятся в файле:

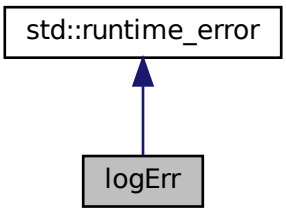
- /home/stud/Загрузки/TiMP/TiMP_K/headers/Errors.h

4.6 Класс logErr

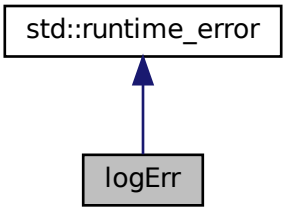
Класс для обработки ошибок логирования.

#include <Errors.h>

Граф наследования:logErr:



Граф связей класса logErr:



Открытые члены

- logErr (const std::string &message)
  Конструктор для создания объекта logErr.

4.6.1 Подробное описание

Класс для обработки ошибок логирования.

Этот класс наследуется от std::runtime_error и используется для представления ошибок, связанных с процессом логирования в приложении.

4.6.2 Конструктор(ы)

4.6.2.1 logErr()

logErr::logErr (
const std::string & message) [inline], [explicit]

Конструктор для создания объекта logErr.

Аргументы

message	Сообщение об ошибке.

```
70 : std::runtime_error(message) {}
```

Объявления и описания членов класса находятся в файле:

- /home/stud/Загрузки/TiMP/TiMP_K/headers/Errors.h

4.7 Класс Logger

Класс для ведения логов приложения.

#include <Logger.h>

Открытые члены

- int writelog (string s)
  Запись сообщения в лог-файл.
- int setLogPath (string pathFile)
  Установка пути к файлу лога.
- Logger ()
  Конструктор по умолчанию.
- Logger (string s)
  Конструктор с заданным путем к файлу лога.

Закрытые статические члены

- static string getLocalDateTime (string s)
  Получение текущей локальной даты и времени в строковом формате.

Закрытые данные

- string pathLogFile
  Путь к файлу лога.

4.7.1 Подробное описание

Класс для ведения логов приложения.

Класс Logger предоставляет функциональность для записи сообщений в лог-файл с временными метками.

4.7.2 Конструктор(ы)

4.7.2.1 Logger() [1/2]

Logger::Logger ( ) [inline]

Конструктор по умолчанию.

Конструктор инициализирует путь к файлу лога пустой строкой.

```
73 { pathLogFile = " "; };
```

4.7.2.2 Logger() [2/2]

Logger::Logger (
string s) [inline]

Конструктор с заданным путем к файлу лога.

Этот конструктор инициализирует путь к файлу лога заданной строкой.

Аргументы

s	Путь к файлу лога.

```
82 { pathLogFile = s; };
```

4.7.3 Методы

4.7.3.1 getLocalDateTime()

```
string Logger::getLocalDateTime (
                string s )   [static], [private]
```

Получение текущей локальной даты и времени в строковом формате.

Этот метод возвращает строку с текущей локальной датой и временем.

Аргументы

s	Формат строки для даты и времени.

Возвращает

Строка с текущей датой и временем.

```
29                                  {
30     time_t now = time(0);
31     struct tm tstruct;
32     char buf[80];
33     tstruct = *localtime(&now);
34
35     if (s == "now")
36         strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);
37     else if (s == "date")
38         strftime(buf, sizeof(buf), "%Y-%m-%d", &tstruct);
39
40     return string(buf);
41 }
```

4.7.3.2 setLogPath()

```
int Logger::setLogPath (
                string pathFile )
```

Установка пути к файлу лога.

Этот метод устанавливает путь к файлу, в который будут записываться логи.

Аргументы

pathFile	Путь к файлу лога.

Возвращает

Возвращает 0 при успешном установлении пути, иначе -1.

```
10                                  {
11     ofstream filelog;
12     filelog.open(pathFile, ios_base::out | ios_base::app);
13
14     if (filelog.is_open()) {
15         pathLogFile = pathFile;
16         writelog("Журнал установлен: " + pathLogFile);
17         return 0;
18     } else if (pathFile == "/var/log/vcalc.log") {
19         pathLogFile = "/tmp/vcalc.log";
20         filelog.open(pathLogFile, ios_base::out | ios_base::app);
21         if (filelog.is_open()) {
22             writelog("Журнал установлен: " + pathLogFile);
23             return 0;
24         }
25     }
26     throw logErr("Не удалось создать журнал: " + pathLogFile);
27 }
```

4.7.3.3 writelog()

```
int Logger::writelog (
                string s )
```

Запись сообщения в лог-файл.

Этот метод записывает переданное сообщение в лог-файл, добавляя к нему временную метку.

Аргументы

s	Сообщение для записи в лог.

Возвращает

Возвращает 0 при успешной записи, иначе -1.

```
43                          {
44     ofstream filelog(pathLogFile, ios_base::out | ios_base::app);
45
46     if (!filelog.is_open()) {
47         cerr « "Внимание: Не удалось создать журнал в " « pathLogFile
48             « ". Создание журнала в /tmp/vcalc.log" « endl;
49
50         pathLogFile = "/tmp/vcalc.log";
51         filelog.open(pathLogFile, ios_base::out | ios_base::app);
52
53         if (!filelog.is_open()) {
54             throw logErr("Не удалось создать журнал: " + pathLogFile);
55         }
56     }
57
58     string time = getLocalDateTime("now");
59     filelog « time « ' ' « s « endl;
60
61     cout « time « ' ' « s « endl;
62
63     return 0;
64 }
```

4.7.4 Данные класса

4.7.4.1 pathLogFile

string Logger::pathLogFile [private]

Путь к файлу лога.

Объявления и описания членов классов находятся в файлах:

- /home/stud/Загрузки/TiMP/TiMP_K/headers/Logger.h
- /home/stud/Загрузки/TiMP/TiMP_K/Logger.cpp

4.8 Структура UserInterface::Params

Структура для хранения параметров приложения.

Открытые члены

- bool p_default () const
  Проверка значения порта по умолчанию.
- bool d_default () const
  Проверка значения пути к файлу конфигурации по умолчанию.
- bool l_default () const
  Проверка значения пути к файлу логирования по умолчанию.

Открытые атрибуты

- string d
  Путь к файлу конфигурации.
- string l
  Путь к файлу логирования.
- int p
  Порт для подключения к серверу.

4.8.1 Подробное описание

Структура для хранения параметров приложения.

Эта структура хранит параметры, переданные через командную строку, и предоставляет методы для проверки значений по умолчанию.

4.8.2 Методы

4.8.2.1 d_default()

bool UserInterface::Params::d_default ( ) const [inline]

Проверка значения пути к файлу конфигурации по умолчанию.

Возвращает

true, если путь равен "/etc/vcalc.conf"; иначе false.

```
75 { return (d == "/etc/vcalc.conf"); }
```

4.8.2.2 l_default()

bool UserInterface::Params::l_default ( ) const [inline]

Проверка значения пути к файлу логирования по умолчанию.

Возвращает

true, если путь равен "/var/log/vcalc.log"; иначе false.

```
82 { return (l == "/var/log/vcalc.log"); }
```

4.8.2.3 p_default()

bool UserInterface::Params::p_default ( ) const [inline]

Проверка значения порта по умолчанию.

Возвращает

true, если порт равен 33333; иначе false.

```
68 { return (p == 33333); }
```

4.8.3 Данные класса

4.8.3.1 d

string UserInterface::Params::d

Путь к файлу конфигурации.

4.8.3.2 l

string UserInterface::Params::l

Путь к файлу логирования.

4.8.3.3 p

int UserInterface::Params::p

Порт для подключения к серверу.

Объявления и описания членов структуры находятся в файле:

- /home/stud/Загрузки/TiMP/TiMP_K/headers/Interface.h

4.9 Класс Server

Класс для работы с серверными функциями.

#include <Server.h>

Открытые члены

- std::string sha224 (const std::string &input_str)
  Вычисление SHA224 хеша для заданной строки.
- int connection (int port, std::map< std::string, std::string > dataFileName, Logger *log)
  Установка соединения на заданном порту.

4.9.1 Подробное описание

Класс для работы с серверными функциями.

Класс Server предоставляет методы для вычисления SHA224 хешей и обработки сетевых соединений.

4.9.2 Методы

4.9.2.1 connection()

```
int Server::connection (
            int port,
            std::map< std::string, std::string > dataFileName,
            Logger * log )
```

Установка соединения на заданном порту.

Этот метод устанавливает соединение на указанном порту и обрабатывает данные из файла, используя переданный логгер для ведения логов.

Аргументы

port	Порт для установки соединения.
dataFileName	Карта, содержащая данные файлов с именами.
log	Указатель на объект Logger для ведения логов.

Возвращает

Возвращает 0 при успешном соединении, иначе -1.

```
18                                {
19   try {
20     int queue_len = 100;
21     sockaddr_in addr;
22     addr.sin_family = AF_INET;
23     addr.sin_port = htons(port);
24     inet_aton("127.0.0.1", &addr.sin_addr);
25
26     int s = socket(AF_INET, SOCK_STREAM, 0);
27     if (s < 0) {
28       throw criticalErr("Ошибка создания сокета");
29     }
30     log->writelog("Сокет прослушивания создан");
31
32     if (bind(s, (const sockaddr *)&addr, sizeof(sockaddr_in)) < 0) {
33       string error_message = strerror(errno);
34       throw criticalErr("Ошибка связывания сокетов: " + error_message);
35     }
36
37     if (listen(s, queue_len) < 0) {
38       throw criticalErr("Ошибка прослушивания сокета");
39     }
40
41     while (true) {
42       sockaddr_in client_addr;
43       socklen_t len = sizeof(sockaddr_in);
44       int handler_socket = accept(s, (sockaddr *)&client_addr, &len);
45       if (handler_socket < 0) {
46         log->writelog("Ошибка сокета клиента");
47         continue;
48       }
49
50       char buff[1024] = {};
51       ssize_t rc = recv(handler_socket, buff, sizeof(buff) - 1, 0);
52       if (rc <= 0) {
53         close(handler_socket);
54         continue;
55       }
56
57       if (rc < 72) {
58         send(handler_socket, "ERR", 3, 0);
59         close(handler_socket);
60         continue;
61       }
62
63       string hashReceived(buff + rc - 56, 56);
64       string salt(buff + rc - 72, 16);
65       string login(buff, rc - 72);
66
67       log->writelog("Полученный логин: " + login);
68       log->writelog("Полученный соль: " + salt);
69       log->writelog("Полученный хэш: " + hashReceived);
70
71       auto it = dataFileName.find(login);
72       if (it == dataFileName.end()) {
73         send(handler_socket, "ERR", 3, 0);
74         log->writelog("Ошибка: Клиент не прошёл аутентификацию");
75         close(handler_socket);
76         continue;
77       }
78
79       string expectedHash = sha224(salt + it->second);
80       log->writelog("Ожидаемый хэш: " + expectedHash);
81       if (expectedHash != hashReceived) {
82         send(handler_socket, "ERR", 3, 0);
83         log->writelog("Ошибка: Клиент не прошёл аутентификацию");
84         close(handler_socket);
85         continue;
86       } else {
87         send(handler_socket, "OK", 2, 0);
```

```
88        log->writelog("Аутентификация клиента произошла успешно: " + login);
89      }
90
91      uint32_t vector_count;
92      if (recv(handler_socket, &vector_count, sizeof(vector_count), 0) <= 0) {
93        log->writelog("Ошибка: Ошибка получения количества векторов");
94        close(handler_socket);
95        break;
96      }
97
98      for (uint32_t i = 0; i < vector_count; i++) {
99        uint32_t vector_size;
100       if (recv(handler_socket, &vector_size, sizeof(vector_size), 0) <= 0) {
101         log->writelog("Ошибка: Ошибка получения размера вектора");
102         close(handler_socket);
103         break;
104       }
105
106       vector<int64_t> vector_values(vector_size);
107       ssize_t received_bytes = recv(handler_socket, vector_values.data(),
108                                     vector_size * sizeof(int64_t), 0);
109       if (received_bytes <= 0 ||
110           received_bytes !=
111               static_cast<ssize_t>(vector_size * sizeof(int64_t))) {
112         log->writelog("Ошибка: Ошибка получения данных вектора");
113         close(handler_socket);
114         break;
115       }
116
117       Calculator calculate(vector_values);
118       int64_t multiply = calculate.sendResult();
119       send(handler_socket, &multiply, sizeof(multiply), 0);
120     }
121
122     close(handler_socket);
123   }
124 } catch (const criticalErr &e) {
125   log->writelog("Критическая ошибка: " + string(e.what()));
126   return -1;
127 }
128 return 0;
129 }
```

4.9.2.2 sha224()

```
string Server::sha224 (
                const std::string & input_str )
```

Вычисление SHA224 хеша для заданной строки.

Этот метод принимает строку, вычисляет ее SHA224 хеш и возвращает его в виде шестнадцатеричной строки.

Аргументы

input_str	Входная строка для вычисления хеша.

Возвращает

Шестнадцатеричная строка, представляющая SHA224 хеш.

```
8                                   {
9   using namespace CryptoPP;
10  SHA224 hash;
11  string new_hash;
12  StringSource(input_str, true,
13               new HashFilter(hash, new HexEncoder(new StringSink(new_hash))));
14  return new_hash;
15 }
```

Объявления и описания членов классов находятся в файлах:

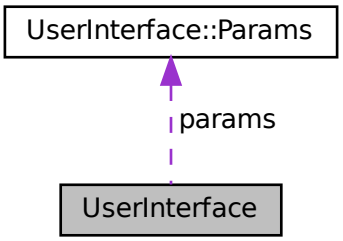
- /home/stud/Загрузки/TiMP/TiMP_K/headers/Server.h
- /home/stud/Загрузки/TiMP/TiMP_K/Server.cpp

4.10 Класс UserInterface

Класс для обработки пользовательского интерфейса.

#include <Interface.h>

Граф связей класса UserInterface:



Классы

- struct Params

  Структура для хранения параметров приложения.

Открытые члены

- int interface (int argc, const char **argv)

  Основной метод интерфейса.

Закрытые данные

- struct UserInterface::Params params

  Экземпляр структуры Params для хранения параметров.

4.10.1 Подробное описание

Класс для обработки пользовательского интерфейса.

Класс UserInterface отвечает за взаимодействие с пользователем, включая обработку аргументов командной строки и настройку параметров приложения.

4.10.2 Методы

4.10.2.1 interface()

```
int UserInterface::interface (
                int argc,
                const char ** argv )
```

Основной метод интерфейса.

Этот метод принимает аргументы командной строки и обрабатывает их, настраивая параметры приложения.

Аргументы

argc	Количество аргументов командной строки.
argv	Массив аргументов командной строки.

Возвращает

Возвращает код завершения (0 - успех, другой - ошибка).

```
11                                          {
12  po::options_description desc("Доступные опции");
13  desc.add_options()("help,h", "Выдать справку")(
14     "database,d",
15     po::value<string>(&params.d)->default_value("/etc/vcalc.conf"),
16     "Установить путь к базе данных")(
17     "log,l",
18     po::value<string>(&params.l)->default_value("/var/log/vcalc.log"),
19     "Установить путь к журналу")(
20     "port,p", po::value<int>(&params.p)->default_value(33333),
21     "Установить порт");
22
23  po::variables_map vm;
24  try {
25    po::store(po::parse_command_line(argc, argv, desc), vm);
26    po::notify(vm);
27
28    if (vm.count("help")) {
29      cout « desc « endl;
30      cout « "Пример: " « argv[0]
31           « " -d[atabase] data.txt -l[og] log.txt -p[ort] 7777" « endl;
32      return 0;
33    }
34    string dataFileName = vm["database"].as<string>();
35    string logFileName = vm["log"].as<string>();
36    int port = vm["port"].as<int>();
37
38    if (port < 1 || port > 65535) {
39      throw po::validation_error(po::validation_error::invalid_option_value,
40                                 "port");
41    }
42    if (params.p_default() && params.d_default() && params.l_default()) {
43      std::cout « desc « std::endl;
44    }
45
46    Logger log(logFileName);
47    BaseConnector bc;
48    bc.baseConnection(dataFileName);
49
50    log.writelog((logFileName != "/var/log/vcalc.log")
51                 ? "Путь к журналу: " + logFileName
52                 : "Путь к журналу по умолчанию: " + logFileName);
53
54    log.writelog((dataFileName != "/etc/vcalc.conf")
55                 ? "Путь к базе данных: " + dataFileName
56                 : "Путь к базе данных по умолчанию: " + dataFileName);
```

```
57
58    log.writelog((port != 33333) ? "Порт: " + to_string(port)
59                                 : "Порт по умолчанию: " + to_string(port));
60
61    Server connect;
62    connect.connection(port, bc.get_data(), &log);
63
64  } catch (const po::error &e) {
65    cerr « "Ошибка Boost: " « e.what() « endl;
66    return 1;
67  } catch (const exception &e) {
68    cerr « "Ошибка: " « e.what() « endl;
69    return 1;
70  } catch (...) {
71    cerr « "Неизвестная ошибка" « endl;
72    return 1;
73  }
74
75  return 0;
76 }
```

4.10.3 Данные класса

4.10.3.1 params

struct UserInterface::Params UserInterface::params [private]

Экземпляр структуры Params для хранения параметров.

Объявления и описания членов классов находятся в файлах:

- /home/stud/Загрузки/TiMP/TiMP_K/headers/Interface.h
- /home/stud/Загрузки/TiMP/TiMP_K/Interface.cpp

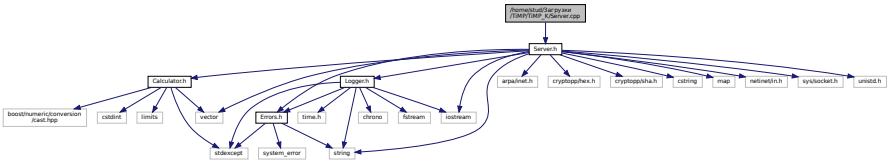
Глава 5

Файлы

5.1 Файл /home/stud/Загрузки/TiMP/TiMP_K/Server.cpp

Определение класса Server для работы с серверными функциями.

#include "Server.h"
Граф включаемых заголовочных файлов для Server.cpp:

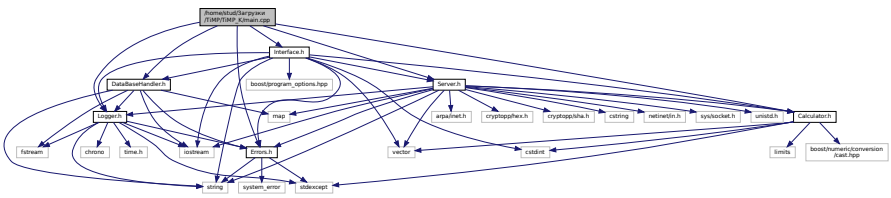


5.1.1 Подробное описание

Определение класса Server для работы с серверными функциями.

5.2 Файл /home/stud/Загрузки/TiMP/TiMP_K/main.cpp

#include "Calculator.h"
#include "DataBaseHandler.h"
#include "Errors.h"
#include "Interface.h"
#include "Logger.h"
#include "Server.h"
Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, const char **argv)

5.2.1 Функции

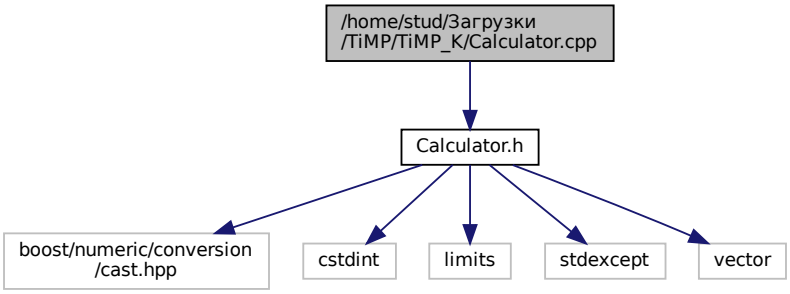
5.2.1.1 main()

```
int main (
                int argc,
                const char ** argv )
8                               {
9   UserInterface interface;
10   return interface.interface(argc, argv);
11   return 0;
12 }
```

5.3 Файл /home/stud/Загрузки/TiMP/TiMP_K/Calculator.cpp

Определение класса Calculator для выполнения расчетов.

#include "Calculator.h"
Граф включаемых заголовочных файлов для Calculator.cpp:



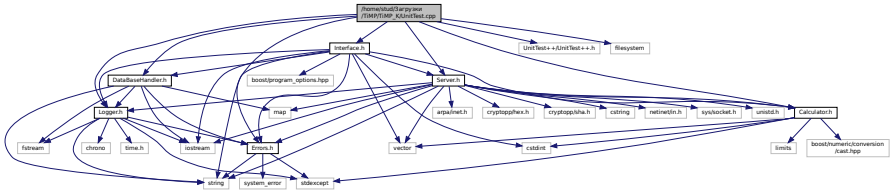
5.3.1 Подробное описание

Определение класса Calculator для выполнения расчетов.

5.4 Файл /home/stud/Загрузки/TiMP/TiMP_K/UnitTest.cpp

```
#include "Calculator.h"
#include "DataBaseHandler.h"
#include "Errors.h"
#include "Interface.h"
#include "Logger.h"
#include "Server.h"
#include <UnitTest++/UnitTest++.h>
#include <filesystem>
```

Граф включаемых заголовочных файлов для UnitTest.cpp:



Функции

- SUITE (SetDataBaseTests)
- SUITE (SetLoggerTests)
- SUITE (CommunicatorTests)
- SUITE (CalculatorTests)
- int main (int argc, const char *argv[])

5.4.1 Функции

5.4.1.1 main()

```
int main (
                int argc,
                const char * argv[] )
131 { return UnitTest::RunAllTests(); }
```

5.4.1.2 SUITE() [1/4]

```
SUITE (
                CalculatorTests  )
93                  {
94   TEST(MultiplyPositiveNumbers) {
95     std::vector<int64_t> input = {9, 18, 27, 36};
96     Calculator calculate(input);
97     CHECK_EQUAL(157464, calculate.sendResult());
98   }
99   TEST(MultiplyNegativeNumbers) {
100     std::vector<int64_t> input = {-9, 18, 27, 36};
101     Calculator calculate(input);
102     CHECK_EQUAL(-157464, calculate.sendResult());
103   }
104   TEST(PositiveOverflow) {
105     std::vector<int64_t> input = {numeric_limits<int64_t>::max(), 2};
106     Calculator calculate(input);
107     CHECK_EQUAL(9223372036854775807, calculate.sendResult());
108   }
109   TEST(NegativeOverflow) {
110     std::vector<int64_t> input = {numeric_limits<int64_t>::min(), 2};
111     Calculator calculate(input);
112     CHECK_EQUAL(numeric_limits<int64_t>::min(), calculate.sendResult());
113   }
114   TEST(SingleElement) {
115     std::vector<int64_t> input = {100};
116     Calculator calculate(input);
117     CHECK_EQUAL(100, calculate.sendResult());
118   }
119   TEST(EmptyVector) {
120     std::vector<int64_t> input = {};
121     Calculator calculate(input);
122     CHECK_EQUAL(1, calculate.sendResult());
123   }
124   TEST(ZeroElement) {
125     std::vector<int64_t> input = {0, 50, 100};
126     Calculator calculate(input);
127     CHECK_EQUAL(0, calculate.sendResult());
128   }
129 }
```

5.4.1.3 SUITE() [2/4]

```
SUITE (
                CommunicatorTests  )
55                  {
56   TEST(Server_sha224_Test) {
57     Server server;
58     std::string input = "test";
59     std::string expectedHash =
60         "90A3ED9E32B2AAF4C61C410EB925426119E1A9DC53D4286ADE99A809";
61     CHECK_EQUAL(server.sha224(input), expectedHash);
62   }
63   TEST(EmptyStringHash) {
64     Server server;
65     std::string expected_empty_hash =
66         "D14A028C2A3A2BC9476102BB288234C415A2B01F828EA62AC5B3E42F";
67     CHECK_EQUAL(expected_empty_hash, server.sha224(""));
68   }
69   TEST(HashWithSalt) {
70     Server server;
71     std::string salt = "random_salt";
72     std::string password = "password";
73     std::string combined_input = salt + password;
74     std::string expectedHash =
75         "F881117367A08C1BF98EE45F7F728650D91C479A3E0D8D06E9D90B6A";
76     CHECK_EQUAL(server.sha224(combined_input), expectedHash);
77   }
78   TEST(HashWithSpecialCharacters) {
79     Server server;
80     std::string input = "test@123!";
81     std::string expectedHash =
82         "9AF50091DEDA5C56499A0FFAEED67B1A1EE19206BA78F78924443208";
83     CHECK_EQUAL(server.sha224(input), expectedHash);
84   }
85   TEST(HashWithWhiteSpace) {
```

```
86     Server server;
87     std::string input = "test 123";
88     std::string expectedHash =
89         "36BACDB6F72F16A6D00674B09F49FA70B8894DC614C5028E3E412517";
90     CHECK_EQUAL(server.sha224(input), expectedHash);
91   }
92 }
```

5.4.1.4 SUITE() [3/4]

```
SUITE (
                SetDataBaseTests  )
11                   {
12   TEST(ValidDatabaseFile) {
13     BaseConnector connector;
14     CHECK_EQUAL(0, connector.baseConnection("test/data.txt"));
15   }
16   TEST(InvalidDataBaseFile) {
17     BaseConnector connector;
18     CHECK_THROW(connector.baseConnection("test/Null.txt"), criticalDataBaseErr);
19   }
20   TEST(EmptyDataBaseFile) {
21     BaseConnector connector;
22     CHECK_THROW(connector.baseConnection("test/emptyBase.txt"), criticalDataBaseErr);
23   }
24   TEST(EmptyLoginDataBase) {
25     BaseConnector connector;
26     CHECK_THROW(connector.baseConnection("test/dataNoLogin.txt"), dataBaseErr);
27   }
28   TEST(EmptyPassDataBase) {
29     BaseConnector connector;
30     CHECK_THROW(connector.baseConnection("test/dataNoPass.txt"), dataBaseErr);
31   }
32   TEST(DataBaseReconnection) {
33     BaseConnector connector;
34     CHECK_EQUAL(0, connector.baseConnection("test/data.txt"));
35     CHECK_EQUAL(0, connector.baseConnection("test/data.txt"));
36   }
37 }
```

5.4.1.5 SUITE() [4/4]

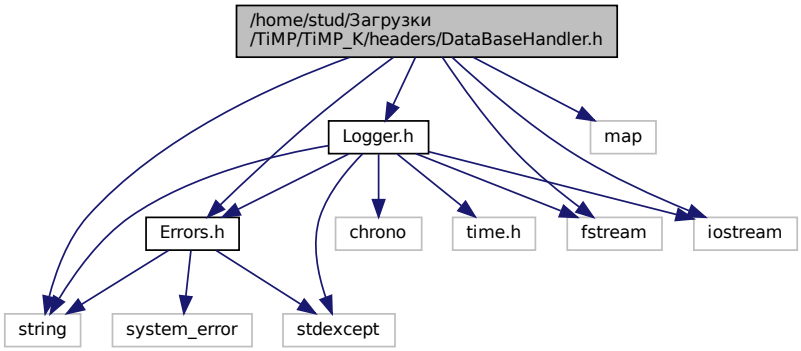
```
SUITE (
                SetLoggerTests  )
39                   {
40   TEST(SetLogPath_Success) {
41     Logger logger;
42     CHECK_EQUAL(logger.setLogPath("test/log.txt"), 0);
43   }
44   TEST(WriteLog_Success) {
45     Logger logger("test/log.txt");
46     CHECK_EQUAL(0, logger.writelog("Это тест логирования журнала"));
47   }
48   TEST(MultipleWriteLog_Success) {
49     Logger logger("test/log.txt");
50     CHECK_EQUAL(0, logger.writelog("Это тест логирования журнала 1"));
51     CHECK_EQUAL(0, logger.writelog("Это тест логирования журнала 2"));
52   }
53 }
```

5.5 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/DataBaseHandler.h

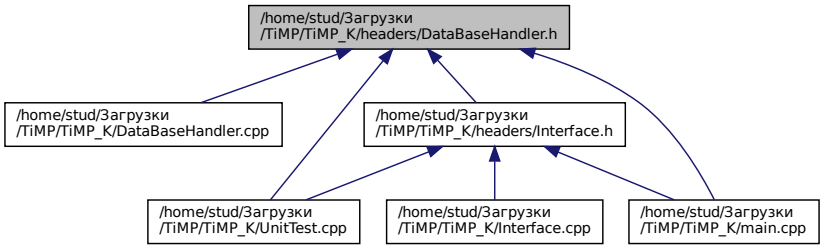
Определение класса BaseConnector для работы с конфигурационными файлами.

```
#include "Errors.h"
#include "Logger.h"
#include <fstream>
#include <iostream>
#include <map>
#include <string>
```

Граф включаемых заголовочных файлов для DataBaseHandler.h:



Граф файлов, в которые включается этот файл:



Классы

- class BaseConnector

  Класс для подключения к базе данных через конфигурационный файл.

5.5.1 Подробное описание

Определение класса BaseConnector для работы с конфигурационными файлами.

Данный класс предназначен для подключения к базе данных, загружая данные из конфигурационного файла.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.6 DataBaseHandler.h

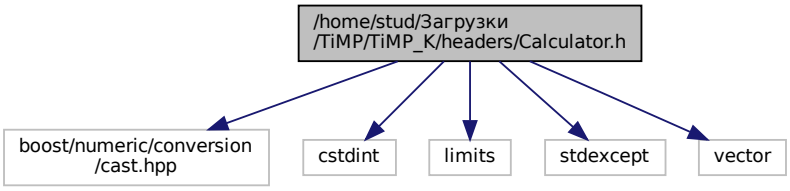
См. документацию.

```
1
13 #pragma once
14
15 #include "Errors.h"
16 #include "Logger.h"
17
18 #include <fstream>
19 #include <iostream>
20 #include <map>
21 #include <string>
22
23 using namespace std;
24
32 class BaseConnector {
33 private:
34   map<string, string>
35       dataBase;
36
37 public:
49   int baseConnection(string baseFile = "/etc/vcalc.conf");
50
59   map<string, string> get_data() { return dataBase; }
60 };
```

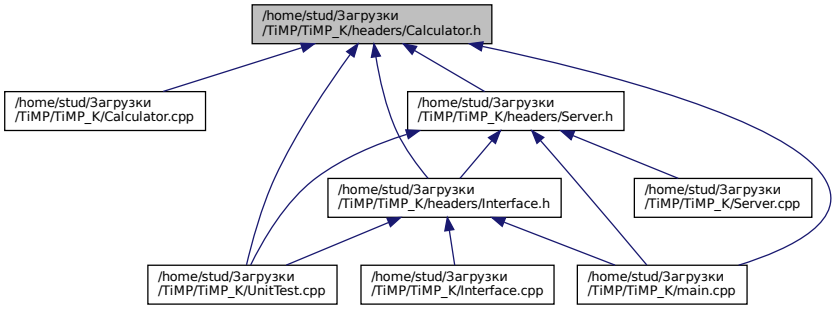
5.7 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/Calculator.h

Определение класса Calculator для выполнения расчетов.

#include <boost/numeric/conversion/cast.hpp>
#include <cstdint>
#include <limits>
#include <stdexcept>
#include <vector>
Граф включаемых заголовочных файлов для Calculator.h:



Граф файлов, в которые включается этот файл:



Классы

- class Calculator

  Класс для выполнения расчетов с вектором целых чисел.

5.7.1 Подробное описание

Определение класса Calculator для выполнения расчетов.

Данный класс предназначен для выполнения арифметических операций над вектором целых чисел (int64_t) и хранения конечного результата.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.8 Calculator.h

См. документацию.

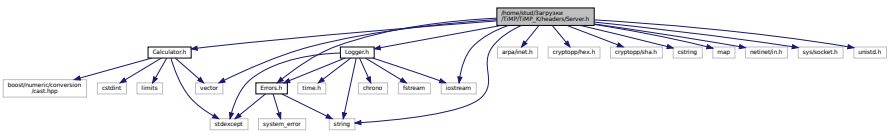
```
1
13 #pragma once
14
15 #include <boost/numeric/conversion/cast.hpp>
16 #include <cstdint>
17 #include <limits>
18 #include <stdexcept>
19 #include <vector>
20
21 using namespace std;
22
31 class Calculator {
32     int64_t finalResult;
33
34 public:
44     Calculator(vector<int64_t> inputVector);
45
51     int64_t sendResult() const;
52 };
```

5.9 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/Server.h

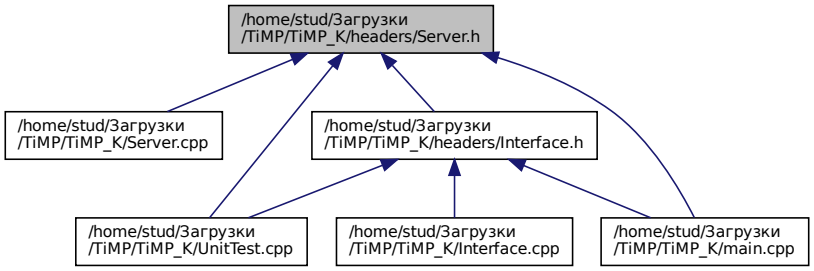
Определение класса Server для работы с серверными функциями.

#include "Calculator.h"
#include "Errors.h"
#include "Logger.h"
#include <arpa/inet.h>
#include <cryptopp/hex.h>
#include <cryptopp/sha.h>
#include <cstring>
#include <iostream>
#include <map>
#include <netinet/in.h>
#include <string>
#include <sys/socket.h>
#include <unistd.h>
#include <vector>

Граф включаемых заголовочных файлов для Server.h:



Граф файлов, в которые включается этот файл:



Классы

- class Server

  Класс для работы с серверными функциями.

5.9.1 Подробное описание

Определение класса Server для работы с серверными функциями.

Этот файл содержит определение класса Server, который предоставляет функциональность для обработки соединений и вычисления SHA224 хешей.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.10 Server.h

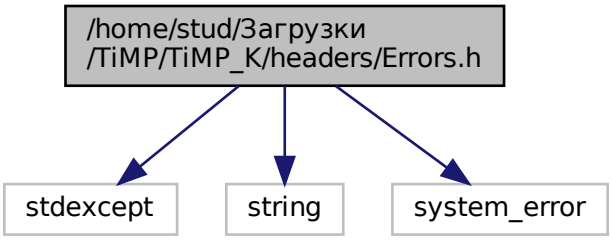
См. документацию.

```
1
12 #pragma once
13
14 #include "Calculator.h"
15 #include "Errors.h"
16 #include "Logger.h"
17
18 #include <arpa/inet.h>
19 #include <cryptopp/hex.h>
20 #include <cryptopp/sha.h>
21 #include <cstring>
22 #include <iostream>
23 #include <map>
24 #include <netinet/in.h>
25 #include <string>
26 #include <sys/socket.h>
27 #include <unistd.h>
28 #include <vector>
29
37 class Server {
38 public:
48     std::string sha224(const std::string &input_str);
49
61     int connection(int port, std::map<std::string, std::string> dataFileName,
62                    Logger *log);
63 };
```

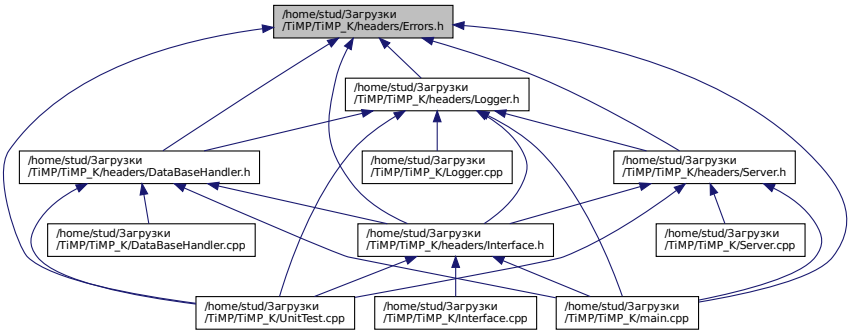
5.11 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/Errors.h

Определение классов ошибок для обработки исключений в приложении.

#include <stdexcept>
#include <string>
#include <system_error>
Граф включаемых заголовочных файлов для Errors.h:



Граф файлов, в которые включается этот файл:



Классы

- class dataBaseErr

  Класс для обработки ошибок базы данных.
- class criticalDataBaseErr

  Класс для обработки критических ошибок базы данных.
- class logErr

  Класс для обработки ошибок логирования.
- class criticalErr

  Класс для обработки критических ошибок.

5.11.1 Подробное описание

Определение классов ошибок для обработки исключений в приложении.

Данный файл содержит определения классов ошибок, которые используются для обработки различных типов исключительных ситуаций в приложении.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.12 Errors.h

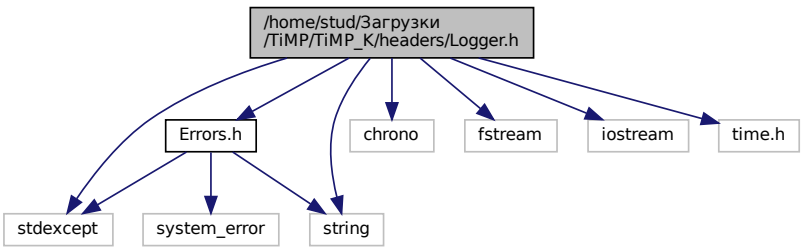
См. документацию.

```
1
12 #pragma once
13
14 #include <stdexcept>
15 #include <string>
16 #include <system_error>
17
18 using namespace std;
19
28 class dataBaseErr : public invalid_argument {
29 public:
35     dataBaseErr(const string &s) : invalid_argument(s) {}
36 };
37
46 class criticalDataBaseErr : public dataBaseErr {
47 public:
53     criticalDataBaseErr(const string &s) : dataBaseErr(s) {}
54 };
55
63 class logErr : public std::runtime_error {
64 public:
70     explicit logErr(const std::string &message) : std::runtime_error(message) {}
71 };
72
81 class criticalErr : public std::runtime_error {
82 public:
88     explicit criticalErr(const std::string &message)
89         : std::runtime_error(message) {}
90 };
```

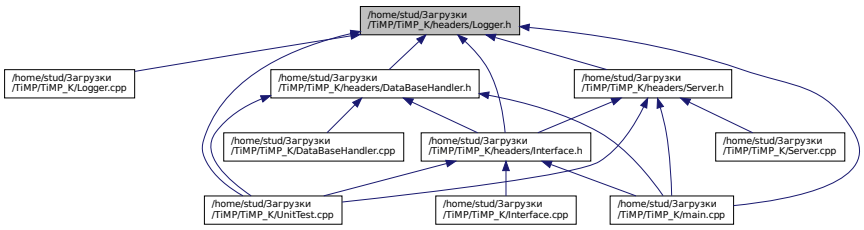
5.13 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/Logger.h

Определение класса Logger для ведения логов приложения.

#include "Errors.h"
#include <chrono>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <time.h>
Граф включаемых заголовочных файлов для Logger.h:



Граф файлов, в которые включается этот файл:



Классы

- class Logger

  Класс для ведения логов приложения.

5.13.1 Подробное описание

Определение класса Logger для ведения логов приложения.

Этот файл содержит определение класса Logger, который отвечает за запись логов в указанный файл.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.14 Logger.h

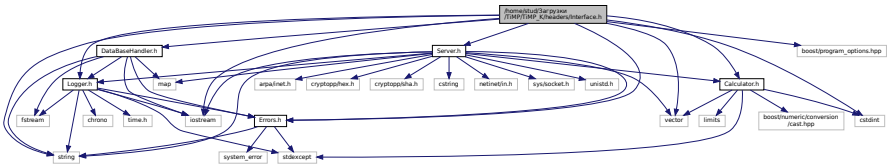
См. документацию.

```
1
12 #pragma once
13
14 #include "Errors.h"
15
16 #include <chrono>
17 #include <fstream>
18 #include <iostream>
19 #include <stdexcept>
20 #include <string>
21 #include <time.h>
22
23 using namespace std;
24
32 class Logger {
33 private:
42   static string getLocalDateTime(string s);
43
44   string pathLogFile;
45
46 public:
56   int writelog(string s);
57
66   int setLogPath(string pathFile);
67
73   Logger() { pathLogFile = " "; };
74
82   Logger(string s) { pathLogFile = s; };
83 };
```

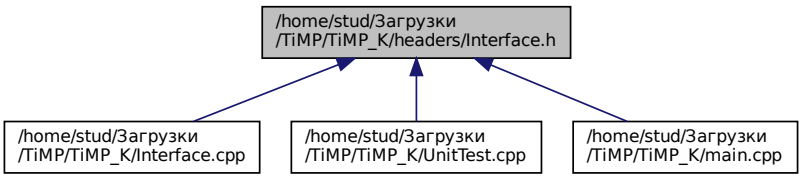
5.15 Файл /home/stud/Загрузки/TiMP/TiMP_K/headers/Interface.h

Определение класса UserInterface для работы с пользовательским интерфейсом приложения.

#include "Calculator.h"
#include "DataBaseHandler.h"
#include "Errors.h"
#include "Logger.h"
#include "Server.h"
#include <boost/program_options.hpp>
#include <cstdint>
#include <iostream>
#include <string>
#include <vector>
Граф включаемых заголовочных файлов для Interface.h:



Граф файлов, в которые включается этот файл:



Классы

- class UserInterface

  Класс для обработки пользовательского интерфейса.
- struct UserInterface::Params

  Структура для хранения параметров приложения.

5.15.1 Подробное описание

Определение класса UserInterface для работы с пользовательским интерфейсом приложения.

Этот файл содержит определение класса UserInterface, который отвечает за взаимодействие с пользователем и обработку командной строки.

Автор

Мочалов Ю.А.

Версия

1.0

Дата

24.12.2024

Авторство

ИБСТ ПГУ

5.16 Interface.h

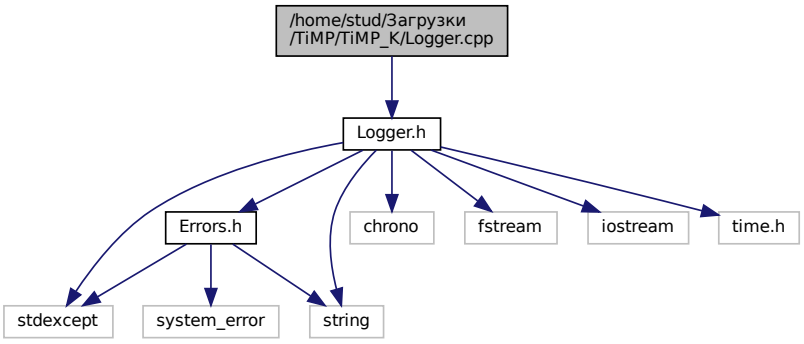
См. документацию.

```
1
12 #pragma once
13
14 #include "Calculator.h"
15 #include "DataBaseHandler.h"
16 #include "Errors.h"
17 #include "Logger.h"
18 #include "Server.h"
19
20 #include <boost/program_options.hpp>
21 #include <cstdint>
22 #include <iostream>
23 #include <string>
24 #include <vector>
25
26 using namespace std;
27
36 class UserInterface {
37 public:
48     int interface(int argc, const char **argv);
49
50 private:
58     struct Params {
59         string d;
60         string l;
61         int p;
62
68         bool p_default()const { return (p == 33333); }
69
75         bool d_default()const { return (d == "/etc/vcalc.conf"); }
76
82         bool l_default()const { return (l == "/var/log/vcalc.log"); }
83     } params;
84 };
```

5.17 Файл /home/stud/Загрузки/TiMP/TiMP_K/Logger.cpp

Определение класса Logger для ведения логов приложения.

#include "Logger.h"
Граф включаемых заголовочных файлов для Logger.cpp:



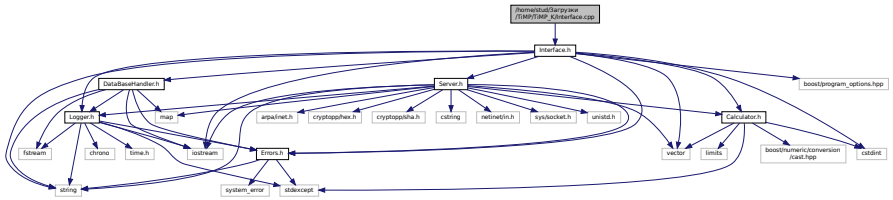
5.17.1 Подробное описание

Определение класса Logger для ведения логов приложения.

5.18 Файл /home/stud/Загрузки/TiMP/TiMP_K/Interface.cpp

Определение класса UserInterface для работы с пользовательским интерфейсом приложения.

#include "Interface.h"
Граф включаемых заголовочных файлов для Interface.cpp:



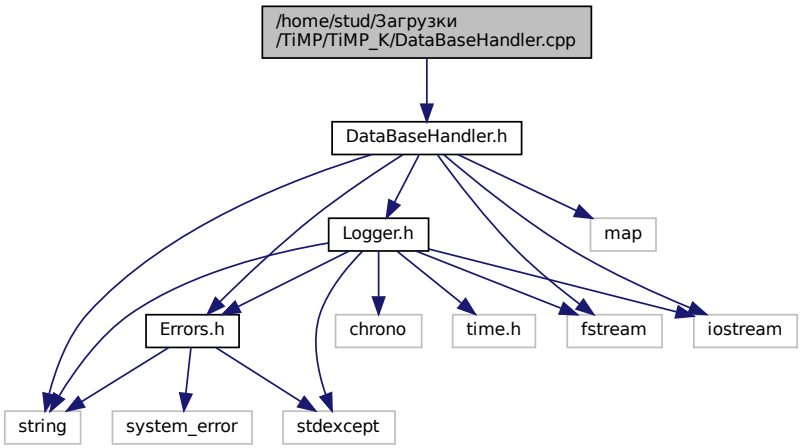
5.18.1 Подробное описание

Определение класса UserInterface для работы с пользовательским интерфейсом приложения.

5.19 Файл /home/stud/Загрузки/TiMP/TiMP_K/DataBaseHandler.cpp

Определение класса BaseConnector для работы с конфигурационными файлами.

#include "DataBaseHandler.h"
Граф включаемых заголовочных файлов для DataBaseHandler.cpp:



5.19.1 Подробное описание

Определение класса BaseConnector для работы с конфигурационными файлами.

Предметный указатель