MyProject

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 Logger Class Reference

A singleton logger class for logging messages to a file.

`#include <Logger.hpp>`

Collaboration diagram for Logger:

Public Member Functions

- Logger (const Logger &)=delete
- Logger & operator= (const Logger &)=delete
- void log (const LogLevel level, const string &message)

  Log a message with a specific log level.
- Logger & operator<< (const string &message)

  Overload the << operator to log string messages.
- Logger & operator<< (const LogLevel level)

  Overload the << operator to set the current log level.
- Logger & operator<< (const int value)

  Overload the << operator to log integer values.
- Logger & operator<< (const char *message)

  Overload the << operator to log C-style string messages.
- Logger & operator<< (std::ostream &(*manip)(std::ostream &))

  Overload the << operator to handle manipulators (like std::endl).
- Logger & operator<< (std::ios_base &(*manip)(std::ios_base &))

  Overload the << operator to handle other iostream manipulators.

Static Public Member Functions

- static Logger & getInstance (const string &filename="default.log")

  Get the singleton instance of Logger.

3.1.1 Detailed Description

A singleton logger class for logging messages to a file.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Logger()

```
Logger::Logger (
             const Logger &  )  [delete]
```

3.1.3 Member Function Documentation

3.1.3.1 getInstance()

```
static Logger & Logger::getInstance (
             const string & filename = "default.log" )  [inline], [static]
```

Get the singleton instance of Logger.

Parameters

filename	The name of the log file (default is "default.log").

Returns

Reference to the Logger instance.

Here is the caller graph for this function:

3.1.3.2 log()

```
void Logger::log (
             const LogLevel level,
             const string & message )  [inline]
```

Log a message with a specific log level.

Parameters

level	The log level of the message.
message	The message to log.

Here is the caller graph for this function:

3.1.3.3 operator<<() [1/6]

```
Logger & Logger::operator<< (
            const char * message )  [inline]
```

Overload the << operator to log C-style string messages.

Parameters

message	The C-style string message to log.

Returns

Reference to the Logger instance.

Here is the call graph for this function:

3.1.3.4 operator<<() [2/6]

```
Logger & Logger::operator<< (
            const int value )  [inline]
```

Overload the << operator to log integer values.

Parameters

value	The integer value to log.

Returns

Reference to the Logger instance.

Here is the call graph for this function:

3.1.3.5 operator<<() [3/6]

```
Logger & Logger::operator<< (
            const LogLevel level )  [inline]
```

Overload the << operator to set the current log level.

Parameters

level	The log level to set.

Returns

Reference to the Logger instance.

3.1.3.6 operator<<() [4/6]

```
Logger & Logger::operator<< (
             const string & message )  [inline]
```

Overload the << operator to log string messages.

Parameters

message	The string message to log.

Returns

Reference to the Logger instance.

Here is the call graph for this function:

3.1.3.7 operator<<() [5/6]

```
Logger & Logger::operator<< (
             std::ios_base &(*)(std::ios_base &) manip )  [inline]
```

Overload the << operator to handle other iostream manipulators.

Parameters

manip	The manipulator to apply.

Returns

Reference to the Logger instance.

3.1.3.8 operator<<() [6/6]

```
Logger & Logger::operator<< (
             std::ostream &(*)(std::ostream &) manip )  [inline]
```

Overload the << operator to handle manipulators (like std::endl).

Parameters

manip	The manipulator to apply.

Returns

Reference to the Logger instance.

Here is the call graph for this function:

3.1.3.9 operator=()

```
Logger & Logger::operator= (
            const Logger & )  [delete]
```

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

- include/Logger.hpp

3.2 TcpServer Class Reference

A class to handle TCP server operations such as accepting connections and receiving data.

```
#include <TcpServer.hpp>
```

Collaboration diagram for TcpServer:

Public Member Functions

- TcpServer ()
  Default constructor for TcpServer.
- TcpServer (int port)
  Constructor for TcpServer with a specified port.
- virtual ∼TcpServer ()
  Destructor for TcpServer.
- void shutdown ()
  Shuts down the server and closes connections.
- void init ()
  Initializes the server by setting up sockets and binding.
- void loop ()
  Main loop for the server to handle incoming connections and data.
- void onConnected (void(∗ncc)(SOCKET_DISCRIPTOR_ fd))
  Sets the callback function to be called when a new connection is established.
- void onReceivedData (void(∗rc)(SOCKET_DISCRIPTOR_ fd, const char ∗buffer))
  Sets the callback function to be called when data is received.
- void onDisconnected (void(∗dc)(SOCKET_DISCRIPTOR_ fd))
  Sets the callback function to be called when a connection is disconnected.
- int sendMessage (SOCKET_DISCRIPTOR_ fd, const char ∗messageBuffer) const
  Sends a message to a specified socket descriptor.
- int sendMessage (SOCKET_DISCRIPTOR_ fd, char ∗messageBuffer) const
  Overloaded function to send a message using a char pointer.
- int closeConnection (SOCKET_DISCRIPTOR_ fd)
  Closes the connection for a specified socket descriptor.

3.2.1 Detailed Description

A class to handle TCP server operations such as accepting connections and receiving data.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 TcpServer() [1/2]

```
TcpServer::TcpServer ( )
```

Default constructor for TcpServer.

Constructor for TcpServer. Initializes the logger and sets up the server with a default port. Here is the call graph for this function:

3.2.2.2 TcpServer() [2/2]

```
TcpServer::TcpServer (
            int port )  [explicit]
```

Constructor for TcpServer with a specified port.

Parameters

port	The port number to use for the server.

Here is the call graph for this function:

3.2.2.3 ∼TcpServer()

```
TcpServer::∼TcpServer ( )  [virtual]
```

Destructor for TcpServer.

Destructor for TcpServer. Closes the server and cleans up resources. Here is the call graph for this function:

3.2.3 Member Function Documentation

3.2.3.1 closeConnection()

```
int TcpServer::closeConnection (
            SOCKET_DISCRIPTOR_ fd )
```

Closes the connection for a specified socket descriptor.

Parameters

fd	The socket descriptor to close.

Returns

The result of the close operation.

Here is the caller graph for this function:

3.2.3.2 init()

```
void TcpServer::init ( )
```

Initializes the server by setting up sockets and binding.

Initializes the TcpServer by setting up the socket, binding, and starting to listen. Here is the caller graph for this function:

3.2.3.3 loop()

```
void TcpServer::loop ( )
```

Main loop for the server to handle incoming connections and data.

Main loop for the TcpServer, handling incoming connections and data. Here is the call graph for this function: Here is the caller graph for this function:

3.2.3.4 onConnected()

```
void TcpServer::onConnected (
            void(*)(SOCKET_DISCRIPTOR_ fd) ncc )
```

Sets the callback function to be called when a new connection is established.

Parameters

ncc	Pointer to the callback function.

Here is the caller graph for this function:

3.2.3.5 onDisconnected()

```
void TcpServer::onDisconnected (
            void(*)(SOCKET_DISCRIPTOR_ fd) dc )
```

Sets the callback function to be called when a connection is disconnected.

Parameters

dc	Pointer to the callback function.

Here is the caller graph for this function:

3.2.3.6 onReceivedData()

```
void TcpServer::onReceivedData (
            void(*)(SOCKET_DISCRIPTOR_ fd, const char *buffer) rc )
```

Sets the callback function to be called when data is received.

Parameters

rc	Pointer to the callback function.

Here is the caller graph for this function:

3.2.3.7 sendMessage() [1/2]

```
int TcpServer::sendMessage (
            SOCKET_DISCRIPTOR_ fd,
            char * messageBuffer ) const
```

Overloaded function to send a message using a char pointer.

Parameters

fd	The socket descriptor to send the message to.
messageBuffer	The message to send.

Returns

The number of bytes sent.

3.2.3.8 sendMessage() [2/2]

```
int TcpServer::sendMessage (
            SOCKET_DISCRIPTOR_ fd,
            const char * messageBuffer ) const
```

Sends a message to a specified socket descriptor.

Parameters

fd	The socket descriptor to send the message to.
messageBuffer	The message to send.

Returns

The number of bytes sent.

Here is the caller graph for this function:

3.2.3.9 shutdown()

```
void TcpServer::shutdown ( )
```

Shuts down the server and closes connections.

Shuts down the TcpServer by closing the connection. Here is the call graph for this function: Here is the caller graph for this function:

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

- include/TcpServer.hpp
- src/TcpServer.cpp

Chapter 4

File Documentation

4.1 include/Logger.hpp File Reference

```
#include <ctime>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <mutex>
```

Include dependency graph for Logger.hpp:

4.2 Logger.hpp

Go to the documentation of this file.

```
#ifndef LOGGER_  // Include guard to prevent multiple inclusions of this header file
#define LOGGER_

#include <ctime>      // For time-related functions
#include <fstream>    // For file stream operations
#include <iostream>   // For standard input/output stream
#include <sstream>    // For string stream operations
#include <string>     // For using the string class
#include <mutex>      // For mutexes to handle thread safety

using namespace std;

enum LogLevel
{
    DEBUG,
    INFO,
    WARNING,
    ERROR,
    CRITICAL
};

class Logger
{
public:
    static Logger& getInstance(const string& filename = "default.log")
    {
        static Logger instance(filename);  // Static instance of Logger
        return instance;
    }

    // Delete copy constructor and assignment operator to prevent copying
    Logger(const Logger&) = delete;
    Logger& operator=(const Logger&) = delete;

    void log(const LogLevel level, const string& message)
    {
        std::lock_guard<std::mutex> lock(mutex_);  // Lock mutex for thread safety
        if (logFile.is_open())  // Check if the log file is open
        {
            logFile « formatLogEntry(level, message);  // Write formatted log entry to file
            logFile.flush();  // Flush the stream to ensure the message is written
        }
    }

    Logger& operator«(const string& message)
    {
        log(currentLevel, message);  // Log the message with the current log level
        return *this;
    }

    Logger& operator«(const LogLevel level)
    {
        currentLevel = level;  // Set the current log level
        return *this;
    }

    Logger& operator«(const int value)
    {
        log(currentLevel, to_string(value));  // Convert integer to string and log it
        return *this;
    }

    Logger& operator«(const char* message)
    {
        log(currentLevel, message);  // Log the C-style string message
        return *this;
    }

    Logger& operator«(std::ostream& (*manip)(std::ostream&))
    {
        if (manip == static_cast<std::ostream& (*)(std::ostream&)>(std::endl))
        {
            log(currentLevel, "\n");  // Log a newline
            logFile.flush();  // Flush the stream
            firstCall = true;  // Reset firstCall flag
        }
        return *this;
    }

    Logger& operator«(std::ios_base& (*manip)(std::ios_base&))
    {
        manip(std::cout);  // Apply manipulator to std::cout
        return *this;
    }

private:
    explicit Logger(const string& filename)
        : currentLevel(INFO), firstCall(true)  // Initialize log level and first call flag
    {
        logFile.open(filename, ios::app);  // Open the log file in append mode
        if (!logFile.is_open())  // Check if the log file was opened successfully
        {
            cerr « "Error while opening protocol file." « endl;  // Print error message to stderr
        }
    }

    ~Logger()
    {
        logFile.close();  // Close the log file upon destruction of the Logger instance
    }

    ofstream logFile;
    LogLevel currentLevel;
    bool firstCall;
    std::mutex mutex_;

    string formatLogEntry(const LogLevel level, const string& message)
    {
        const time_t now = time(nullptr);  // Get the current time
        const tm* timeinfo = localtime(&now);  // Convert to local time
        char timetxt[20];
        strftime(timetxt, sizeof(timetxt), "%Y-%m-%d %H:%M:%S", timeinfo);  // Format the time

        ostringstream logEntry;  // String stream to build the log entry
        if (firstCall)
        {
            logEntry « "[" « timetxt « "] " « levelToString(level) « ": " « message;  // First log entry format
            firstCall = false;  // Set firstCall to false after the first entry
        }
        else
        {
            logEntry « message;  // Subsequent log entries
        }
        logEntry.flush();  // Flush the string stream
        return logEntry.str();  // Return the formatted log entry as a string
    }

    static string levelToString(const LogLevel level)
    {
        switch (level)
        {
        case DEBUG: return "DEBUG";
        case INFO: return "INFO";
        case WARNING: return "WARNING";
        case ERROR: return "ERROR";
        case CRITICAL: return "CRITICAL";
        default: return "UNKNOWN";
        }
    }
};

#endif  // End of include guard
```

4.3 include/OCEngine.hpp File Reference

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

Macros

- #define OC_ENGINE_VERSION "0.1 pre 1"
- #define OC_ENGINE_STD_PORT 12455
- #define OC_LOG_FILE "OClogfile.txt"

4.3.1 Macro Definition Documentation

4.3.1.1 OC_ENGINE_STD_PORT

```
#define OC_ENGINE_STD_PORT 12455
```

4.3.1.2 OC_ENGINE_VERSION

```
#define OC_ENGINE_VERSION "0.1 pre 1"
```

4.3.1.3 OC_LOG_FILE

```
#define OC_LOG_FILE "OClogfile.txt"
```

4.4 OCEngine.hpp

Go to the documentation of this file.

```
#ifndef OC_ENGINE_
#define OC_ENGINE_

#define OC_ENGINE_VERSION "0.1 pre 1"
#define OC_ENGINE_STD_PORT 12455
#define OC_LOG_FILE "OClogfile.txt"

#endif
```

4.5 include/TcpServer.hpp File Reference

```
#include <iostream>
#include "TcpUtilities.hpp"
#include "Logger.hpp"
```

Include dependency graph for TcpServer.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class TcpServer

  A class to handle TCP server operations such as accepting connections and receiving data.

Macros

- #define INPUT_BUFFER_SIZE 1024
- #define DEFAULT_PORT 38233
- #define SERVER_DEBUG true

4.5.1 Macro Definition Documentation

4.5.1.1 DEFAULT_PORT

```
#define DEFAULT_PORT 38233
```

4.5.1.2 INPUT_BUFFER_SIZE

```
#define INPUT_BUFFER_SIZE 1024
```

4.5.1.3 SERVER_DEBUG

```
#define SERVER_DEBUG true
```

4.6 TcpServer.hpp

Go to the documentation of this file.

```
#ifndef TCP_SERVER_  // Include guard to prevent multiple inclusions of this header file
#define TCP_SERVER_

#include <iostream>
#include "TcpUtilities.hpp" // Include custom TCP utility functions
#include "Logger.hpp" // Include logger for logging messages

// Platform-specific includes and definitions
#if defined (__linux__) || defined (__APPLE__)
    #include <unistd.h> // For POSIX operating system API
    #include <sys/select.h> // For select function
    #include <sys/types.h> // For data types used in system calls
    #include <netinet/in.h> // For internet address family
    #include <arpa/inet.h> // For inet_ntop and inet_pton functions
    #include <sys/socket.h> // For socket functions
    #define SOCKET_DISCRIPTOR_ int // Define SOCKET_DISCRIPTOR_ as int for socket descriptors
#elif _WIN32
```

```
    #include <cstdint> // For fixed-width integer types
    #include <WinSock2.h> // For Windows Sockets API
    #include <WS2tcpip.h> // For Windows TCP/IP functions
    #define SOCKET_DISCRIPTOR_ SOCKET // Define SOCKET_DISCRIPTOR_ as SOCKET for Windows
#endif

#define INPUT_BUFFER_SIZE 1024 // Size of the input buffer
#define DEFAULT_PORT 38233 // Default port for the server

#define SERVER_DEBUG true // Enable server debugging

class TcpServer {
public:
    TcpServer();

    explicit TcpServer(int port);

    virtual ~TcpServer();

#ifdef _WIN32
    WSADATA wsaData; // Structure to hold Windows Sockets data
#endif

    void shutdown();

    void init();

    void loop();

    void onConnected(void (*ncc)(SOCKET_DISCRIPTOR_ fd));

    void onReceivedData(void (*rc)(SOCKET_DISCRIPTOR_ fd, const char* buffer));

    void onDisconnected(void (*dc)(SOCKET_DISCRIPTOR_ fd));

    int sendMessage(SOCKET_DISCRIPTOR_ fd, const char* messageBuffer) const;

    int sendMessage(SOCKET_DISCRIPTOR_ fd, char* messageBuffer) const;

    int closeConnection(SOCKET_DISCRIPTOR_ fd);

private:
    Logger *logger; // Pointer to the logger instance

    fd_set masterfds{}; // Master file descriptor set
    fd_set tempfds{}; // Temporary file descriptor set

    SOCKET_DISCRIPTOR_ maxfd{}; // Maximum file descriptor

    SOCKET_DISCRIPTOR_ mastersocket_fd{}; // Master socket file descriptor
    SOCKET_DISCRIPTOR_ tempsocket_fd{}; // Temporary socket file descriptor

    struct sockaddr_storage client_addr{}; // Structure to hold client address
    struct sockaddr_storage servaddr{}; // Structure to hold server address

    char input_buffer[INPUT_BUFFER_SIZE]{}; // Buffer for incoming data
    char remote_ip[INET6_ADDRSTRLEN]{}; // Buffer for remote IP address

    void (*connectedCallback) (SOCKET_DISCRIPTOR_ fd){}; // Callback for new connections
    void (*receivedCallback) (SOCKET_DISCRIPTOR_ fd, const char* buffer){}; // Callback for received data
    void (*disconnectedCallback) (SOCKET_DISCRIPTOR_ fd){}; // Callback for disconnections

    void setup(int port);

    void initializeSocket();

    void bindSocket();

    void startListen() const;

    void handleNewConnection();

    void recvInputFromExisting(SOCKET_DISCRIPTOR_ fd);
};

#endif // End of include guard
```

4.7 include/TcpUtilities.hpp File Reference

```
#include <iostream>
#include <string>
```

```
#include <iomanip>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
```

Include dependency graph for TcpUtilities.hpp: This graph shows which files directly or indirectly include this file:

Functions

- void printHexlStringStream (const string &buffer)

  Prints the hexadecimal representation of a given string buffer.
- char ∗ getIPbyFD (const int fd)

  Retrieves the IP address of a client connected to a socket.

4.7.1 Function Documentation

4.7.1.1 getIPbyFD()

```
char * getIPbyFD (
          const int fd )
```

Retrieves the IP address of a client connected to a socket.

This function uses the socket file descriptor to get the client's address and converts it to a string format.

Parameters

fd	The socket file descriptor of the client.

Returns

A pointer to a string containing the IP address, or nullptr on failure.

Here is the caller graph for this function:

4.7.1.2 printHexlStringStream()

```
void printHexlStringStream (
          const std::string & buffer )
```

Prints the hexadecimal representation of a given string buffer.

This function formats the buffer into a hexadecimal string and prints it to the standard output.

Parameters

buffer	The string buffer to be printed in hex format.

This function formats the buffer into a hexadecimal string and prints it to the logger. It also prints the ASCII representation of the data alongside the hex values.

Parameters

buffer	The string buffer to be printed in hex format.

Here is the call graph for this function: Here is the caller graph for this function:

4.8 TcpUtilities.hpp

Go to the documentation of this file.

```
#ifndef TCP_UTILITIES_HPP  // Include guard to prevent multiple inclusions of this header file
#define TCP_UTILITIES_HPP

#include <iostream> // Include for input/output stream operations
#include <string>   // Include for using std::string
#include <iomanip>  // Include for input/output manipulators

#include <sys/socket.h> // Include for socket-related functions and definitions
#include <netinet/in.h> // Include for internet address family
#include <cstring>      // Include for string manipulation functions
#include <arpa/inet.h>  // Include for inet_ntop and inet_pton functions
#include <unistd.h>     // Include for POSIX operating system API

using namespace std;

void printHexlStringStream(const string &buffer);

char* getIPbyFD(const int fd);

#endif // End of include guard
```

4.9 src/OCEngine.cpp File Reference

```
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include "Logger.hpp"
#include "OCEngine.hpp"
#include <regex>
#include "TcpServer.hpp"
#include "TcpUtilities.hpp"
```

Include dependency graph for OCEngine.cpp:

Functions

- void onConnect (SOCKET_DISCRIPTOR_ fd)
- void onInput (SOCKET_DISCRIPTOR_ fd, const char ∗buffer)
- void onDisconnect (SOCKET_DISCRIPTOR_ fd)
- int main (int argc, char ∗argv[ ])

Variables

- TcpServer server
- string inSockStr
- Logger ∗ logger

4.9.1 Function Documentation

4.9.1.1 main()

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

Here is the call graph for this function:

4.9.1.2 onConnect()

```
void onConnect (
            SOCKET_DISCRIPTOR_ fd )
```

Here is the call graph for this function: Here is the caller graph for this function:

4.9.1.3 onDisconnect()

```
void onDisconnect (
            SOCKET_DISCRIPTOR_ fd )
```

Here is the call graph for this function: Here is the caller graph for this function:

4.9.1.4 onInput()

```
void onInput (
            SOCKET_DISCRIPTOR_ fd,
            const char * buffer )
```

Here is the call graph for this function: Here is the caller graph for this function:

4.9.2 Variable Documentation

4.9.2.1 inSockStr

```
string inSockStr
```

4.9.2.2 logger

```
Logger* logger
```

4.9.2.3 server

```
TcpServer server
```

4.10 src/TcpServer.cpp File Reference

```
#include "TcpServer.hpp"
#include "OCEngine.hpp"
#include <string>
#include <cstring>
#include <iostream>
#include <arpa/inet.h>
```

Include dependency graph for TcpServer.cpp:

4.11 src/TcpUtilities.cpp File Reference

```
#include "TcpUtilities.hpp"
#include "Logger.hpp"
```

Include dependency graph for TcpUtilities.cpp:

Functions

- void printHexlStringStream (const std::string &buffer)

  Prints the hexadecimal representation of a given string buffer.
- char ∗ getIPbyFD (const int fd)

  Retrieves the IP address of a client connected to a socket.

4.11.1 Function Documentation

4.11.1.1 getIPbyFD()

```
char * getIPbyFD (
            const int fd )
```

Retrieves the IP address of a client connected to a socket.

This function uses the socket file descriptor to get the client's address and converts it to a string format.

Parameters

fd	The socket file descriptor of the client.

Returns

A pointer to a string containing the IP address, or nullptr on failure.

Here is the caller graph for this function:

4.11.1.2 printHexlStringStream()

```
void printHexlStringStream (
            const std::string & buffer )
```

Prints the hexadecimal representation of a given string buffer.

This function formats the buffer into a hexadecimal string and prints it to the logger. It also prints the ASCII representation of the data alongside the hex values.

Parameters

buffer	The string buffer to be printed in hex format.

Here is the call graph for this function: Here is the caller graph for this function:

Index