

MyProject

Generated by Doxygen 1.9.8



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 Logger Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 Logger()	6
3.1.3 Member Function Documentation	6
3.1.3.1 getInstance()	6
3.1.3.2 log()	6
3.1.3.3 operator<<() [1/6]	7
3.1.3.4 operator<<() [2/6]	7
3.1.3.5 operator<<() [3/6]	7
3.1.3.6 operator<<() [4/6]	8
3.1.3.7 operator<<() [5/6]	8
3.1.3.8 operator<<() [6/6]	8
3.1.3.9 operator=()	9
3.2 TcpServer Class Reference	9
3.2.1 Detailed Description	9
3.2.2 Constructor & Destructor Documentation	10
3.2.2.1 TcpServer() [1/2]	10
3.2.2.2 TcpServer() [2/2]	10
3.2.2.3 ~TcpServer()	10
3.2.3 Member Function Documentation	10
3.2.3.1 closeConnection()	10
3.2.3.2 init()	11
3.2.3.3 loop()	11
3.2.3.4 onConnected()	11
3.2.3.5 onDisconnected()	11
3.2.3.6 onReceivedData()	12
3.2.3.7 sendMessage() [1/2]	12
3.2.3.8 sendMessage() [2/2]	12
3.2.3.9 shutdown()	13
<b>4 File Documentation</b>	<b>15</b>
4.1 include/Logger.hpp File Reference	15
4.2 Logger.hpp	15
4.3 include/OCEngine.hpp File Reference	17
4.3.1 Macro Definition Documentation	17

---

4.3.1.1 OC_ENGINE_STD_PORT . . . . .	17
4.3.1.2 OC_ENGINE_VERSION . . . . .	17
4.3.1.3 OC_LOG_FILE . . . . .	17
4.4 OCEngine.hpp . . . . .	17
4.5 include/TcpServer.hpp File Reference . . . . .	18
4.5.1 Macro Definition Documentation . . . . .	18
4.5.1.1 DEFAULT_PORT . . . . .	18
4.5.1.2 INPUT_BUFFER_SIZE . . . . .	18
4.5.1.3 SERVER_DEBUG . . . . .	18
4.6 TcpServer.hpp . . . . .	18
4.7 include/TcpUtilities.hpp File Reference . . . . .	19
4.7.1 Function Documentation . . . . .	20
4.7.1.1 getIPbyFD() . . . . .	20
4.7.1.2 printHexlStringStream() . . . . .	20
4.8 TcpUtilities.hpp . . . . .	21
4.9 src/OCEngine.cpp File Reference . . . . .	21
4.9.1 Function Documentation . . . . .	22
4.9.1.1 main() . . . . .	22
4.9.1.2 onConnect() . . . . .	22
4.9.1.3 onDisconnect() . . . . .	22
4.9.1.4 onInput() . . . . .	22
4.9.2 Variable Documentation . . . . .	22
4.9.2.1 inSockStr . . . . .	22
4.9.2.2 logger . . . . .	22
4.9.2.3 server . . . . .	22
4.10 src/TcpServer.cpp File Reference . . . . .	23
4.11 src/TcpUtilities.cpp File Reference . . . . .	23
4.11.1 Function Documentation . . . . .	23
4.11.1.1 getIPbyFD() . . . . .	23
4.11.1.2 printHexlStringStream() . . . . .	23
<b>Index</b>	<b>25</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Logger</a>	A singleton logger class for logging messages to a file . . . . .	<a href="#">5</a>
<a href="#">TcpServer</a>	A class to handle TCP server operations such as accepting connections and receiving data . .	<a href="#">9</a>



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">include/Logger.hpp</a>	15
<a href="#">include/OCEngine.hpp</a>	17
<a href="#">include/TcpServer.hpp</a>	18
<a href="#">include/TcpUtilities.hpp</a>	19
<a href="#">src/OCEngine.cpp</a>	21
<a href="#">src/TcpServer.cpp</a>	23
<a href="#">src/TcpUtilities.cpp</a>	23





## 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

<i>filename</i>	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

<i>level</i>	The log level of the message.
<i>message</i>	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

<i>message</i>	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

<i>value</i>	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

<i>level</i>	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**

<i>message</i>	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**

<i>manip</i>	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**

<i>manip</i>	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\_DESCRIPTOR\_ fd))  
*Sets the callback function to be called when a new connection is established.*
- void [onReceivedData](#) (void(\*rc)(SOCKET\_DESCRIPTOR\_ fd, const char \*buffer))  
*Sets the callback function to be called when data is received.*
- void [onDisconnected](#) (void(\*dc)(SOCKET\_DESCRIPTOR\_ fd))  
*Sets the callback function to be called when a connection is disconnected.*
- int [sendMessage](#) (SOCKET\_DESCRIPTOR\_ fd, const char \*messageBuffer) const  
*Sends a message to a specified socket descriptor.*
- int [sendMessage](#) (SOCKET\_DESCRIPTOR\_ fd, char \*messageBuffer) const  
*Overloaded function to send a message using a char pointer.*
- int [closeConnection](#) (SOCKET\_DESCRIPTOR\_ 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

<i>port</i>	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_DESCRIPTOR_ fd )
```

Closes the connection for a specified socket descriptor.

#### Parameters

<i>fd</i>	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_DESCRIPTOR_ fd) ncc )
```

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

**Parameters**

<i>ncc</i>	Pointer to the callback function.
------------	-----------------------------------

Here is the caller graph for this function:

**3.2.3.5 onDisconnected()**

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

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

**Parameters**

<i>dc</i>	Pointer to the callback function.
-----------	-----------------------------------

Here is the caller graph for this function:

### 3.2.3.6 onReceivedData()

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

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

#### Parameters

<i>rc</i>	Pointer to the callback function.
-----------	-----------------------------------

Here is the caller graph for this function:

### 3.2.3.7 sendMessage() [1/2]

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

Overloaded function to send a message using a char pointer.

#### Parameters

<i>fd</i>	The socket descriptor to send the message to.
<i>messageBuffer</i>	The message to send.

#### Returns

The number of bytes sent.

### 3.2.3.8 sendMessage() [2/2]

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

Sends a message to a specified socket descriptor.

#### Parameters

<i>fd</i>	The socket descriptor to send the message to.
<i>messageBuffer</i>	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.](#)

```
00001 #ifndef LOGGER_ // Include guard to prevent multiple inclusions of this header file
00002 #define LOGGER_
00003
00004 #include <ctime> // For time-related functions
00005 #include <fstream> // For file stream operations
00006 #include <iostream> // For standard input/output stream
00007 #include <sstream> // For string stream operations
00008 #include <string> // For using the string class
00009 #include <mutex> // For mutexes to handle thread safety
00010
00011 using namespace std;
00012
00013 enum LogLevel
00014 {
00015     DEBUG,
00016     INFO,
00017     WARNING,
00018     ERROR,
00019     CRITICAL
00020 };
00021
00022 class Logger
00023 {
00024 public:
00025     static Logger& getInstance(const string& filename = "default.log")
00026     {
00027         static Logger instance(filename); // Static instance of Logger
00028         return instance;
00029     }
00030
00031     // Delete copy constructor and assignment operator to prevent copying
00032     Logger(const Logger&) = delete;
00033     Logger& operator=(const Logger&) = delete;
00034
00035     void log(const LogLevel level, const string& message)
00036     {
```

```

00057         std::lock_guard<std::mutex> lock(mutex_); // Lock mutex for thread safety
00058         if (logFile.is_open()) // Check if the log file is open
00059         {
00060             logFile << formatLogEntry(level, message); // Write formatted log entry to file
00061             logFile.flush(); // Flush the stream to ensure the message is written
00062         }
00063     }
00064
00071     Logger& operator<<(const string& message)
00072     {
00073         log(currentLevel, message); // Log the message with the current log level
00074         return *this;
00075     }
00076
00083     Logger& operator<<(const LogLevel level)
00084     {
00085         currentLevel = level; // Set the current log level
00086         return *this;
00087     }
00088
00095     Logger& operator<<(const int value)
00096     {
00097         log(currentLevel, to_string(value)); // Convert integer to string and log it
00098         return *this;
00099     }
00100
00107     Logger& operator<<(const char* message)
00108     {
00109         log(currentLevel, message); // Log the C-style string message
00110         return *this;
00111     }
00112
00119     Logger& operator<<(std::ostream& (*manip)(std::ostream&))
00120     {
00121         if (manip == static_cast<std::ostream& (*)>(std::ostream&)>(std::endl))
00122         {
00123             log(currentLevel, "\n"); // Log a newline
00124             logFile.flush(); // Flush the stream
00125             firstCall = true; // Reset firstCall flag
00126         }
00127         return *this;
00128     }
00129
00136     Logger& operator<<(std::ios_base& (*manip)(std::ios_base&))
00137     {
00138         manip(std::cout); // Apply manipulator to std::cout
00139         return *this;
00140     }
00141
00142 private:
00148     explicit Logger(const string& filename)
00149         : currentLevel(INFO), firstCall(true) // Initialize log level and first call flag
00150     {
00151         logFile.open(filename, ios::app); // Open the log file in append mode
00152         if (!logFile.is_open()) // Check if the log file was opened successfully
00153         {
00154             cerr << "Error while opening protocol file." << endl; // Print error message to stderr
00155         }
00156     }
00157
00161     ~Logger()
00162     {
00163         logFile.close(); // Close the log file upon destruction of the Logger instance
00164     }
00165
00166     ofstream logFile;
00167     LogLevel currentLevel;
00168     bool firstCall;
00169     std::mutex mutex_;
00170
00178     string formatLogEntry(const LogLevel level, const string& message)
00179     {
00180         const time_t now = time(nullptr); // Get the current time
00181         const tm* timeinfo = localtime(&now); // Convert to local time
00182         char timetxt[20];
00183         strftime(timetxt, sizeof(timetxt), "%Y-%m-%d %H:%M:%S", timeinfo); // Format the time
00184
00185         ostringstream logEntry; // String stream to build the log entry
00186         if (firstCall)
00187         {
00188             logEntry << "[" << timetxt << "]" << levelToString(level) << ": " << message; // First log
entry format
00189             firstCall = false; // Set firstCall to false after the first entry
00190         }
00191         else
00192         {
00193             logEntry << message; // Subsequent log entries

```

```

00194     }
00195     logEntry.flush(); // Flush the string stream
00196     return logEntry.str(); // Return the formatted log entry as a string
00197 }
00198
00205 static string levelToString(const LogLevel level)
00206 {
00207     switch (level)
00208     {
00209         case DEBUG: return "DEBUG";
00210         case INFO: return "INFO";
00211         case WARNING: return "WARNING";
00212         case ERROR: return "ERROR";
00213         case CRITICAL: return "CRITICAL";
00214         default: return "UNKNOWN";
00215     }
00216 }
00217 };
00218
00219 #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.](#)

```

00001 #ifndef OC_ENGINE_
00002 #define OC_ENGINE_
00003
00004 #define OC_ENGINE_VERSION "0.1 pre 1"
00005 #define OC_ENGINE_STD_PORT 12455
00006 #define OC_LOG_FILE "OClogfile.txt"
00007
00008 #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.](#)

```
00001 #ifndef TCP_SERVER_ // Include guard to prevent multiple inclusions of this header file
00002 #define TCP_SERVER_
00003
00004 #include <iostream>
00005 #include "TcpUtilities.hpp" // Include custom TCP utility functions
00006 #include "Logger.hpp" // Include logger for logging messages
00007
00008 // Platform-specific includes and definitions
00009 #if defined (__linux__) || defined (__APPLE__)
00010     #include <unistd.h> // For POSIX operating system API
00011     #include <sys/select.h> // For select function
00012     #include <sys/types.h> // For data types used in system calls
00013     #include <netinet/in.h> // For internet address family
00014     #include <arpa/inet.h> // For inet_ntop and inet_pton functions
00015     #include <sys/socket.h> // For socket functions
00016     #define SOCKET_DESCRIPTOR_ int // Define SOCKET_DESCRIPTOR_ as int for socket descriptors
00017 #elif _WIN32
```

```

00018     #include <cstdint> // For fixed-width integer types
00019     #include <WinSock2.h> // For Windows Sockets API
00020     #include <WS2tcpip.h> // For Windows TCP/IP functions
00021     #define SOCKET_DESCRIPTOR_ SOCKET // Define SOCKET_DESCRIPTOR_ as SOCKET for Windows
00022 #endif
00023
00024 #define INPUT_BUFFER_SIZE 1024 // Size of the input buffer
00025 #define DEFAULT_PORT 38233 // Default port for the server
00026
00027 #define SERVER_DEBUG true // Enable server debugging
00028
00033 class TcpServer {
00034 public:
00038     TcpServer();
00039
00045     explicit TcpServer(int port);
00046
00050     virtual ~TcpServer();
00051
00052 #ifdef _WIN32
00053     WSADATA wsaData; // Structure to hold Windows Sockets data
00054 #endif
00055
00059     void shutdown();
00060
00064     void init();
00065
00069     void loop();
00070
00076     void onConnected(void (*ncc)(SOCKET_DESCRIPTOR_ fd));
00077
00083     void onReceivedData(void (*rc)(SOCKET_DESCRIPTOR_ fd, const char* buffer));
00084
00090     void onDisconnected(void (*dc)(SOCKET_DESCRIPTOR_ fd));
00091
00099     int sendMessage(SOCKET_DESCRIPTOR_ fd, const char* messageBuffer) const;
00100
00108     int sendMessage(SOCKET_DESCRIPTOR_ fd, char* messageBuffer) const;
00109
00116     int closeConnection(SOCKET_DESCRIPTOR_ fd);
00117
00118 private:
00119     Logger *logger; // Pointer to the logger instance
00120
00121     fd_set masterfds{}; // Master file descriptor set
00122     fd_set tempfds{}; // Temporary file descriptor set
00123
00124     SOCKET_DESCRIPTOR_ maxfd{}; // Maximum file descriptor
00125
00126     SOCKET_DESCRIPTOR_ mastersocket_fd{}; // Master socket file descriptor
00127     SOCKET_DESCRIPTOR_ tempsocket_fd{}; // Temporary socket file descriptor
00128
00129     struct sockaddr_storage client_addr{}; // Structure to hold client address
00130     struct sockaddr_storage servaddr{}; // Structure to hold server address
00131
00132     char input_buffer[INPUT_BUFFER_SIZE]{}; // Buffer for incoming data
00133     char remote_ip[INET6_ADDRSTRLEN]{}; // Buffer for remote IP address
00134
00135     void (*connectedCallback)(SOCKET_DESCRIPTOR_ fd)(); // Callback for new connections
00136     void (*receivedCallback)(SOCKET_DESCRIPTOR_ fd, const char* buffer)(); // Callback for received
data
00137     void (*disconnectedCallback)(SOCKET_DESCRIPTOR_ fd)(); // Callback for disconnections
00138
00144     void setup(int port);
00145
00149     void initializeSocket();
00150
00154     void bindSocket();
00155
00159     void startListen() const;
00160
00164     void handleNewConnection();
00165
00171     void recvInputFromExisting(SOCKET_DESCRIPTOR_ fd);
00172 };
00173
00174 #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

<i>fd</i>	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

<i>buffer</i>	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

<i>buffer</i>	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.](#)

```
00001 #ifndef TCP_UTILITIES_HPP // Include guard to prevent multiple inclusions of this header file
00002 #define TCP_UTILITIES_HPP
00003
00004 #include <iostream> // Include for input/output stream operations
00005 #include <string> // Include for using std::string
00006 #include <iomanip> // Include for input/output manipulators
00007
00008 #include <sys/socket.h> // Include for socket-related functions and definitions
00009 #include <netinet/in.h> // Include for internet address family
00010 #include <cstring> // Include for string manipulation functions
00011 #include <arpa/inet.h> // Include for inet_ntop and inet_pton functions
00012 #include <unistd.h> // Include for POSIX operating system API
00013
00014 using namespace std;
00015
00023 void printHexlStringStream(const string &buffer);
00024
00034 char* getIPbyFD(const int fd);
00035
00036 #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\_DESCRIPTOR\_ fd)
- void [onInput](#) (SOCKET\_DESCRIPTOR\_ fd, const char \*buffer)
- void [onDisconnect](#) (SOCKET\_DESCRIPTOR\_ 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_DESCRIPTOR_ 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_DESCRIPTOR_ 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_DESCRIPTOR_ 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

<i>fd</i>	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

<i>buffer</i>	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

- ~TcpServer
  - TcpServer, [10](#)
- closeConnection
  - TcpServer, [10](#)
- DEFAULT\_PORT
  - TcpServer.hpp, [18](#)
- getInstance
  - Logger, [6](#)
- getIPbyFD
  - TcpUtilities.cpp, [23](#)
  - TcpUtilities.hpp, [20](#)
- include/Logger.hpp, [15](#)
- include/OCEngine.hpp, [17](#)
- include/TcpServer.hpp, [18](#)
- include/TcpUtilities.hpp, [19](#), [21](#)
- init
  - TcpServer, [11](#)
- INPUT\_BUFFER\_SIZE
  - TcpServer.hpp, [18](#)
- inSockStr
  - OCEngine.cpp, [22](#)
- log
  - Logger, [6](#)
- Logger, [5](#)
  - getInstance, [6](#)
  - log, [6](#)
  - Logger, [6](#)
  - operator<<, [6–8](#)
  - operator=, [8](#)
- logger
  - OCEngine.cpp, [22](#)
- loop
  - TcpServer, [11](#)
- main
  - OCEngine.cpp, [22](#)
- OC\_ENGINE\_STD\_PORT
  - OCEngine.hpp, [17](#)
- OC\_ENGINE\_VERSION
  - OCEngine.hpp, [17](#)
- OC\_LOG\_FILE
  - OCEngine.hpp, [17](#)
- OCEngine.cpp
  - inSockStr, [22](#)
  - logger, [22](#)
  - main, [22](#)
  - onConnect, [22](#)
  - onDisconnect, [22](#)
  - onInput, [22](#)
  - server, [22](#)
- OCEngine.hpp
  - OC\_ENGINE\_STD\_PORT, [17](#)
  - OC\_ENGINE\_VERSION, [17](#)
  - OC\_LOG\_FILE, [17](#)
- onConnect
  - OCEngine.cpp, [22](#)
- onConnected
  - TcpServer, [11](#)
- onDisconnect
  - OCEngine.cpp, [22](#)
- onDisconnected
  - TcpServer, [11](#)
- onInput
  - OCEngine.cpp, [22](#)
- onReceivedData
  - TcpServer, [12](#)
- operator<<
  - Logger, [6–8](#)
- operator=
  - Logger, [8](#)
- printHexIStringStream
  - TcpUtilities.cpp, [23](#)
  - TcpUtilities.hpp, [20](#)
- sendMessage
  - TcpServer, [12](#)
- server
  - OCEngine.cpp, [22](#)
- SERVER\_DEBUG
  - TcpServer.hpp, [18](#)
- shutdown
  - TcpServer, [13](#)
- src/OCEngine.cpp, [21](#)
- src/TcpServer.cpp, [23](#)
- src/TcpUtilities.cpp, [23](#)
- TcpServer, [9](#)
  - ~TcpServer, [10](#)
  - closeConnection, [10](#)
  - init, [11](#)
  - loop, [11](#)
  - onConnected, [11](#)
  - onDisconnected, [11](#)
  - onReceivedData, [12](#)

- sendMessage, [12](#)
- shutdown, [13](#)
- TcpServer, [10](#)
- TcpServer.hpp
  - DEFAULT\_PORT, [18](#)
  - INPUT\_BUFFER\_SIZE, [18](#)
  - SERVER\_DEBUG, [18](#)
- TcpUtilities.cpp
  - getIPbyFD, [23](#)
  - printHexlStringStream, [23](#)
- TcpUtilities.hpp
  - getIPbyFD, [20](#)
  - printHexlStringStream, [20](#)