

## Logger

Generated by Doxygen 1.10.0



<b>1 Logger</b>	<b>1</b>
1.1 Basic Logger.	1
1.1.1 How to Use	1
1.1.2 How to log	2
1.1.3 Ease if use	2
1.1.4 Log Channels	2
1.1.5 Log Orders	2
1.1.6 File Output	2
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Logger Class Reference	7
4.1.1 Member Enumeration Documentation	7
4.1.1.1 LogChannel	7
4.1.1.2 LogOrder	8
4.1.2 Member Function Documentation	8
4.1.2.1 Log() [1/2]	8
4.1.2.2 Log() [2/2]	8
4.1.2.3 setFilepath()	9
<b>5 File Documentation</b>	<b>11</b>
5.1 C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/Logger.h File Reference	11
5.2 Logger.h	11
5.3 C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/README.md File Reference	13
<b>Index</b>	<b>15</b>



# Chapter 1

## Logger

- ME

### 1.1 Basic Logger.

Task Description:

Create a universal trace library for application event logging to different channels. The library should support logging to at least one of the following channels: TCP, serial, console, file, etc. Additionally, it should be designed in a non-blocking manner to ensure it does not impede the normal operation of the applications utilizing it.

Requirements:

1. Implement a C++ library that enables application event logging.
2. The library should support logging to at least one of the following channels: TCP, serial, console, file, etc. You may choose the channel that best suits your expertise or demonstrate versatility by implementing logging to multiple channels.
3. Ensure the library is non-blocking to prevent it from interfering with the normal execution of applications.
4. Provide clear documentation on how to use the library and its different features.
5. Optionally, you may include unit tests to ensure the reliability and robustness of your implementation.

#### 1.1.1 How to Use

This is a header-only library. Copy `Logger.h` file and `#include` it in your project.

### 1.1.2 How to log

Here is an example

```
#include "Logger.h"

int main()
{
    std::string a = "string";
    std::string str = "Test Log with std::string";
    std::string result;

    Logger::Log(Logger::ConsoleChannel , Logger::LogDebug, "Test Log with number %d" , 5);
    Logger::Log(Logger::ConsoleChannel , Logger::LogInfo, "Test Log with c-string %s , %i" , a.c_str(), 15);
    Logger::Log(Logger::ConsoleChannel , Logger::LogError, str);

    Logger::setFilepath("output.txt");

    Logger::Log(Logger::FileChannel , Logger::LogDebug, str);
    Logger::Log(Logger::FileChannel, Logger::LogInfo, "Test Log with c-string %s , %i" , a.c_str(), 15);
}
```

Output:

```
03-05-2024 : 11:46:30 [Debug] Test Log with number 5 03-05-2024 : 11:46:30 [Info] Test Log with
c-string string , 15 03-05-2024 : 11:46:30 [Error] Test Log with std::string
```

and file output

```
03-05-2024 : 11:46:30 [Debug] Test Log with std::string 03-05-2024 : 11:46:30 [Info] Test Log with
c-string string , 15
```

### 1.1.3 Ease if use

No initialization, just use log functions

### 1.1.4 Log Channels

Theses are the channels currently supported

```
Logger::FileChannel
Logger::ConsoleChannel
```

### 1.1.5 Log Orders

Theses are the orders currently supported

```
Logger::LogDebug
Logger::LogInfo
Logger::LogError
```

### 1.1.6 File Output

To set the file path file output, call

```
Logger::setFilepath("output.txt");
```

if no file path is set the default file location is "log.txt".

## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Logger</a>	.....	<a href="#">7</a>
------------------------	-------	-------------------





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/[Logger.h](#) . . . . . 11



# Chapter 4

## Class Documentation

### 4.1 Logger Class Reference

```
#include <Logger.h>
```

#### Public Types

- enum [LogChannel](#) { [FileChannel](#) , [ConsoleChannel](#) }
- enum [LogOrder](#) { [LogDebug](#) , [LogInfo](#) , [LogError](#) }

#### Static Public Member Functions

- static bool [setFilepath](#) (const char \*new\_filepath)
- static void [Log](#) ([LogChannel](#) channel, [LogOrder](#) order, std::string logMessage)
- template<typename... Args>  
static void [Log](#) ([LogChannel](#) channel, [LogOrder](#) order, const char \*logMessage, Args... args)

#### 4.1.1 Member Enumeration Documentation

##### 4.1.1.1 LogChannel

```
enum Logger::LogChannel
```

The different channels currently supported.

#### Enumerator

<a href="#">FileChannel</a>	
<a href="#">ConsoleChannel</a>	

#### 4.1.1.2 LogOrder

```
enum Logger::LogOrder
```

The different orders currently supported.

Enumerator

LogDebug	
LogInfo	
LogError	

### 4.1.2 Member Function Documentation

#### 4.1.2.1 Log() [1/2]

```
template<typename... Args>
static void Logger::Log (
    LogChannel channel,
    LogOrder order,
    const char * logMessage,
    Args... args ) [inline], [static]
```

Writes the log messge for the specifiess order to the specified channel .

Parameters

<i>channel</i>	the channel to wwrite the log to, file or console
<i>Order</i>	the log order, debug, info, error
<i>logMessage</i>	the message to be logged in c-string format
<i>args</i>	the arguments that can be passed into the c-string

#### 4.1.2.2 Log() [2/2]

```
static void Logger::Log (
    LogChannel channel,
    LogOrder order,
    std::string logMessage ) [inline], [static]
```

Writes the log messge for the specifiess order to the specified channel .

Parameters

<i>channel</i>	the channel to wwrite the log to, file or console
<i>Order</i>	the log order, debug, info, error
<i>logMessage</i>	the message to be logged in string format

#### 4.1.2.3 setFilepath()

```
static bool Logger::setFilepath (
    const char * new_filepath )    [inline], [static]
```

Set the file path and name to which the log output would be written.

##### Parameters

<i>new_filepath</i>	the filepath
<i>micros</i>	the microseconds fraction

##### Returns

true if the file exists or was created otherwise it returns false

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

- C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/[Logger.h](#)



# Chapter 5

## File Documentation

### 5.1 C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/Logger.h File Reference

```
#include <string>
#include <cstdio>
#include <mutex>
#include <ctime>
```

#### Classes

- class [Logger](#)

### 5.2 Logger.h

[Go to the documentation of this file.](#)

```
00001
00002 /*****
00003  * @file  Logger.h
00004  *
00005  * @brief Implementation of the class Logger.
00006  *****/
00007
00008
00009 #pragma once
00010 #include <string>
00011 #include <cstdio>
00012 #include <mutex>
00013 #include <ctime>
00014
00015
00016 //needed to use fopen if using visual studio
00017 #if defined(_MSC_VER)
00018 #define _CRT_SECURE_NO_WARNINGS
00019 #endif
00020
00021
00022 class Logger{
00023 public:
00024     enum LogChannel {
00025         FileChannel , ConsoleChannel
00026     };
00027
00028     enum LogOrder {
00029         LogDebug, LogInfo , LogError
00030     }
```

```

00038     };
00039
00040
00041     static bool setFilepath(const char* new_filepath) {
00042         return getInstance().init(new_filepath);
00043     }
00044
00045     static void Log(LogChannel channel, LogOrder order, std::string logMessage) {
00046         getInstance().LOG(channel, order, logMessage.c_str());
00047     }
00048
00049     template<typename... Args>
00050     static void Log(LogChannel channel, LogOrder order, const char* logMessage, Args... args) {
00051         getInstance().LOG(channel, order, logMessage, args...);
00052     }
00053
00054 private:
00055
00056     std::mutex logMutex;
00057     char _time[80];
00058     const char* timestampFormat = "%d-%m-%Y : %T";
00059     std::FILE* file = 0;
00060     const char* filepath = "log.txt";
00061
00062     Logger() {}
00063
00064     Logger(const Logger&) = delete;
00065     Logger& operator= (const Logger&) = delete;
00066
00067     ~Logger()
00068     {
00069         deleteFile();
00070     }
00071
00072     static Logger& getInstance()
00073     {
00074         static Logger instance;
00075
00076         return instance;
00077     }
00078
00079     const char* orderString(LogOrder order) {
00080         if (order == LogDebug) {
00081             return "[Debug]";
00082         }
00083         else if (order == LogInfo) {
00084             return "[Info]";
00085         }
00086         else if (order == LogError) {
00087             return "[Error]";
00088         }
00089         else {
00090             return "[***]";
00091         }
00092     }
00093
00094     bool init(){
00095         deleteFile();
00096         file = std::fopen(filepath, "a");
00097
00098         if (file == 0)
00099         {
00100             return false;
00101         }
00102         return true;
00103     }
00104
00105     bool init(const char* new_filepath){
00106         deleteFile();
00107         file = std::fopen(new_filepath, "a");
00108
00109         if (file == 0)
00110         {
00111             return false;
00112         }
00113         return true;
00114     }
00115
00116     void deleteFile()
00117     {
00118         if (file)
00119         {
00120             std::fclose(file);
00121             file = 0;
00122         }
00123     }
00124
00125
00126

```



```

00162     template<typename... Args>
00163     void fileLog(const char* time, const char* messageOrderStr, const char* logMessage, Args... args)
00164     {
00165         if (file)
00166         {
00167             std::fprintf(file, "%s ", time);
00168             std::fprintf(file, messageOrderStr);
00169             std::fprintf(file, logMessage, args...);
00170             std::fprintf(file, "\n");
00171         }
00172         else {
00173             if (getInstance().init())
00174                 getInstance().fileLog(time, messageOrderStr, logMessage, args...);
00175         }
00176     }
00177     template<typename... Args>
00178     void consoleLog(const char* time, const char* messageOrderStr, const char* logMessage, Args...
00179     args) {
00180         std::printf("%s ", time);
00181         std::printf(messageOrderStr);
00182         std::printf(logMessage, args...);
00183         std::printf("\n");
00184     }
00185     template<typename... Args>
00186     void LOG(LogChannel channel, LogOrder order, const char* message, Args... args) {
00187         std::time_t current_time = std::time(0);
00188         std::tm* timestamp = std::localtime(&current_time);
00189
00190         std::scoped_lock lock(logMutex);
00191         std::strftime(_time, 80, timestampFormat, timestamp);
00192
00193         if (channel == FileChannel) {
00194             getInstance().fileLog(_time, orderString(order), message, args...);
00195         }
00196         else if (channel == ConsoleChannel) {
00197             getInstance().consoleLog(_time, orderString(order), message, args...);
00198         }
00199     }
00200 };
00201
00202 //https://charlescookekey.com/

```

### 5.3 C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/README.md File Reference



# Index

C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/Logger.h, [11](#)

C:/Users/charl/OneDrive/Documents/OpenSource Attempts/DHLo/Answer/README.md, [13](#)

ConsoleChannel  
Logger, [7](#)

FileChannel  
Logger, [7](#)

Log  
Logger, [8](#)

LogChannel  
Logger, [7](#)

LogDebug  
Logger, [8](#)

LogError  
Logger, [8](#)

Logger, [1](#), [7](#)  
ConsoleChannel, [7](#)  
FileChannel, [7](#)  
Log, [8](#)  
LogChannel, [7](#)  
LogDebug, [8](#)  
LogError, [8](#)  
LogInfo, [8](#)  
LogOrder, [7](#)  
setFilepath, [8](#)

LogInfo  
Logger, [8](#)

LogOrder  
Logger, [7](#)

setFilepath  
Logger, [8](#)