Logger

Generated by Doxygen 1.10.0

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:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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

FileChannel	
ConsoleChannel	

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 specifiesd order to the specified channel .

Parameters

channel	the channel to wwrite the log to, file or console
Order	the log order, debug, info, error
logMessage	the message to be logged in c-string format
args	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 specifiesd order to the specified channel .

Parameters

channel	the channel to wwrite the log to, file or console
Order	the log order, debug, info, error
logMessage	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

new_filepath	the filepath
micros	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.

```

/*********************************************************************
 * @file  Logger.h
 *
 * @brief Implementation of the class Logger.
 *********************************************************************/


#pragma once
#include <string>
#include <cstdio>
#include <mutex>
#include <ctime>


//needed to use fopen if  using visual studio
#if defined(_MSC_VER)
#define _CRT_SECURE_NO_WARNINGS
#endif


class Logger{
public:
    enum LogChannel {
        FileChannel , ConsoleChannel
    };

    enum LogOrder {
        LogDebug, LogInfo , LogError
    };


    static bool setFilepath(const char* new_filepath) {
        return getInstance().init(new_filepath);
    }

    static void Log(LogChannel channel, LogOrder order, std::string logMessage) {
        getInstance().LOG(channel, order, logMessage.c_str());
    }

    template<typename... Args>
    static void Log(LogChannel channel, LogOrder order, const char* logMessage, Args... args) {
        getInstance().LOG(channel, order, logMessage, args...);
    }

private:

    std::mutex logMutex;
    char _time[80];
    const char* timestampFormat = "%d-%m-%Y : %T";
    std::FILE* file = 0;
    const char* filepath = "log.txt";

    Logger() {}

    Logger(const Logger&) = delete;
    Logger& operator= (const Logger&) = delete;

    ~Logger()
    {
        deleteFile();
    }

    static Logger& getInstance()
    {
        static Logger instance;

        return instance;
    }

    const char* orderString(LogOrder order) {
        if (order == LogDebug) {
            return "[Debug]     ";
        }
        else if (order == LogInfo) {
            return "[Info]      ";
        }
        else if (order == LogError) {
            return "[Error]     ";
        }
        else {
            return "[****]      ";
        }
    }

    bool init(){
        deleteFile();
        file = std::fopen(filepath, "a");

        if (file == 0)
        {
            return false;
        }
        return true;
    }

    bool init(const char* new_filepath){
        deleteFile();
        file = std::fopen(new_filepath, "a");

        if (file == 0)
        {
            return false;
        }
        return true;
    }

    void deleteFile()
    {
        if (file)
        {
            std::fclose(file);
            file = 0;
        }
    }

    template<typename... Args>
    void fileLog(const char* time, const char* messageOrderStr, const char* logMessage, Args... args) {
        if (file)
        {
            std::fprintf(file, "%s    ", time);
            std::fprintf(file, messageOrderStr);
            std::fprintf(file, logMessage, args...);
            std::fprintf(file, "\n");
        }
        else {
            if (getInstance().init())
                getInstance().fileLog(time, messageOrderStr, logMessage, args...);
        }
    }

    template<typename... Args>
    void consoleLog(const char* time, const char* messageOrderStr, const char* logMessage, Args... args) {
        std::printf("%s    ", time);
        std::printf(messageOrderStr);
        std::printf(logMessage, args...);
        std::printf("\n");
    }

    template<typename... Args>
    void LOG(LogChannel channel, LogOrder order, const char* message, Args... args) {
        std::time_t current_time = std::time(0);
        std::tm* timestamp = std::localtime(&current_time);

        std::scoped_lock lock(logMutex);
        std::strftime(_time, 80, timestampFormat, timestamp);

        if (channel == FileChannel) {
            getInstance().fileLog(_time, orderString(order), message, args...);
        }
        else if (channel == ConsoleChannel) {
            getInstance().consoleLog(_time, orderString(order), message, args...);
        }
    }
};

//https://charlescookey.com/
```

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

Index