Logger C++ library

v1.2.4

Table of contents

Overview

Logger C++ library provides logging functions: printing in terminal and(or) printing in file. File **Logger.h** contains declaration of class **Logger** which provides logging functions. **Logger** class provide thread safe method to print info file or terminal. The user can create multiple **Logger** class objects and the library will provide thread-safe output to the terminal and thread-safe writing to the file (all data will be written to the file in sequence). The **Logger** class also provides color output to the terminal. When writing the log to a text file, the **Logger** class controls the size of the file as well as the total size of the files in the directory and deletes old files if the directory size is exceeded. It uses C++17 standard. The library is licensed under the **Apache 2.0** license.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	13.02.2023	First version.

Version	Release date	What's new
1.1.0	24.03.2023	- Added new print option DISABLE. - Example updated. - Documentation updated.
1.2.0	08.05.2023	- Added macro to print file name.
1.2.1	20.06.2023	- Required CMake version updated to 3.13.
1.2.2	06.07.2023	- Documentation updated. - Example updated. - License added. - Repository made public.
1.2.3	11.08.2023	- Code clean up. - Updated example. Fixed the same name for log folder as name of executable file.
1.2.4	20.03.2024	- Documentation updated.

Library files

The library is supplied only by source code. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ---------- Main CMake file of the library.
3rdparty ---------------- Folder with 3rdparty libraries.
    CMakeLists.txt ------ CMake file for to include 3rdparty libraries.
    Frame --------------- Folder with Frame library source code.
src --------------------- Folder with library source code.
    Logger.cpp ---------- C++ implementation file.
    Logger.h ------------ Library main header file.
    LoggerVersion.h ----- Header file with library version.
    LoggerVersion.h.in -- Service CMake file to generate version file.
example ----------------- Example.
    CMakeLists.txt ------ CMake file of the example.
    main.cpp ------------ Source C++ file of the example.
```

Logger class description

Logger class declaration

Logger.h file contains **Logger** class declaration. Class declaration:

```
class Logger
{
public:
```

```
    /// Get string of current version of library.
    static std::string getVersion();

    /// Set saving params.
    static bool setSaveLogParams(std::string folder,
                                 std::string filePrefix,
                                 int maxFolderSizeMb,
                                 int maxFileSizeMb);

    /// Print message through the operator "<<".
    ColourPrint print(PrintColor color, PrintFlag flags = PrintFlag::CONSOLE);
};
```

getVersion method

The **getVersion()** method return string of current version of **Logger** class. Method declaration:

```
static std::string getVersion();
```

Method can be used without **Logger** class instance. Example:

```
std::cout << "Logger class version: " << cr::utils::Logger::getVersion() << std::endl;
```

Console output:

```
Logger class version: 1.2.4
```

setSaveLogParams method

The **setSaveLogParams(...)** static method intended to set global parameter for writing logs into text files. The method sets parameters for all objects of the **Logger** class. If a method is called in more than one place in the program, the last parameters will be applied to all objects of the **Logger** class. If the method is not called, all objects of the **Logger** class will only be able to output information to the console. The method used without **Logger** class instance. Method declaration:

```
static bool setSaveLogParams(std::string folder, std::string filePrefix, int
maxFolderSizeMb, int maxFileSizeMb);
```

Parameter	Description
folder	Path to folder. Example: /home/pi/Log
filePrefix	File prefix. The library creates files with names: PREFIX_DATE_TIME.txt. Example: LOG_2023.03.24_15.55.59.txt.
maxFolderSizeMb	Maximum folder size. the library controls size of all files in folder. If overall size of files in folder > maxFolderSizeMb the library deletes oldest files in folder so the overall size of files < maxFolderSizeMb.

Parameter	Description
maxFileSizeMb	Maximum file size, MB. Library creates files and controls their size. If size of file > maxFileSizeMb the library creates new file.

Returns: TRUE if parameters are set or FALSE if not.

Example:

```
string folder = "LoggerExample";
string filePrefix = "LOG";
int maxFolderSizeMb = 3;
int maxFileSizeMb = 1;
Logger::setSaveLogParams(folder, filePrefix, maxFolderSizeMb, maxFileSizeMb);
```

print method

The **print(...)** method intended to print data in console and(or) file. The method provides the same interface as **std::cout** method and supports all **std::cout** formats. The method allows the user to define the colour of the output to the console and the direction of the output (console, file or file and console). When the library prints in file it adds additional information about date and time (e.g.: **2023-03-24 15:56:52.127716 Something to print**) and moved to new line after printing a string. Method declaration:

```
ColourPrint print(PrintColor color, PrintFlag flags = PrintFlag::CONSOLE);
```

Parameter	Description
color	Print color. Colors defined in PrintColor enum.
flags	Print flag to define print direction: console, file, console and file or disabled. Print flags defines in PrintFlag enum.

Returns: TRUE if parameters are set or FALSE if not.

PrintColor enum

PrintColor enum declared in **Logger.h** file and defines . Enum declaration:

```
enum class PrintColor
{
    NORMAL,
    RED,
    GREEN,
    YELLOW,
    BLUE,
    MAGENTA,
    CYAN,
    WHITE
};
```

PrintFlag enum

PrintFlag enum defines output options: print to console, print to file, print to file and console or disable printing. Enum declaration:

```
enum class PrintFlag
{
    CONSOLE = 1,
    FILE = 2,
    CONSOLE_AND_FILE = CONSOLE | FILE,
    DISABLE
};
```

Example

Sample application creates three thread. Each thread print information to file, console or both.

```
#include <iostream>
#include <thread>
#include "Logger.h"

// Link namespaces.
using namespace std;
using namespace cr::utils;

// Print thread function.
void printThreadFunction(PrintColor colour, PrintFlag flag)
{
    // Init local logger.
    Logger log;

    // Thread function.
    int counter = 0;
    while (true)
    {
        // Print something in console.
        log.print(colour, flag) << "Print thread output " << counter++ << endl;
    }
}

// Entry point.
int main(void)
{
    cout << "================================================" << endl;
    cout << "Logger v" << Logger::getVersion() << " example"    << endl;
    cout << "================================================" << endl;
    cout << endl;

    // Init logger.
    Logger log;
```

```
    // Set Logger global parameters to write log in files.
    string folder = "Logs";
    string filePrefix = "LOG";
    int maxFolderSizeMb = 3;
    int maxFileSizeMb = 1;
    Logger::setSaveLogParams(folder, filePrefix, maxFolderSizeMb, maxFileSizeMb);

    // Run print threads.
    thread printThread1(&printThreadFunction, PrintColor::GREEN,
                        PrintFlag::CONSOLE_AND_FILE);
    thread printThread2(&printThreadFunction, PrintColor::BLUE,
                        PrintFlag::CONSOLE);

    // Main pring loop.
    int counter = 0;
    while (true)
    {
        // Print something in console and file.
        log.print(PrintColor::RED, PrintFlag::CONSOLE_AND_FILE) <<
        "Main thread output " << counter++ << endl;

        // Print something in console only.
        log.print(PrintColor::CYAN, PrintFlag::CONSOLE) <<
        "[" << __LOGFILENAME__ << "][" << __LINE__ << "] " <<
        "Main thread output " << counter++ << endl;
    }

    return 1;
}
```

Build and connect to your project

Typical commands to build **Logger** library:

```
git clone https://github.com/ConstantRobotics-Ltd/Logger.git
cd Logger
mkdir build
cd build
cmake ..
make
```

If you want connect Logger library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **Logger** as submodule by command:

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/Logger.git 3rdparty/Logger
```

In you repository folder will be created folder **3rdparty/Logger** which contains files of **Logger** repository. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    Logger
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

###############################################################################
## 3RD-PARTY
## dependencies for the project
###############################################################################
project(3rdparty LANGUAGES CXX)

###############################################################################
## SETTINGS
## basic 3rd-party settings before use
###############################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

###############################################################################
## CONFIGURATION
## 3rd-party submodules configuration
###############################################################################
SET(${PARENT}_SUBMODULE_LOGGER                          ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_LOGGER)
    SET(${PARENT}_LOGGER                                ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_LOGGER_EXAMPLE                        OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
if (${PARENT}_SUBMODULE_LOGGER)
    add_subdirectory(Logger)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **Logger** to your project and excludes example (Logger class example) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    Logger
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include Logger library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} Logger)
```

Done!