

**Documentation for v9.97.1**

**Quick Links**

→ [Latest Release](https://github.com/abumq/easyloggingpp/releases/latest)

→ [Changelog](https://file+.vscode-resource.vscode-cdn.net/CHANGELOG.md)

→ [Samples](https://file+.vscode-resource.vscode-cdn.net/samples)

**Table of Contents**

[Overview](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#overview)

[Why yet another library](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#why-yet-another-library)

[Features at a glance](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#features-at-a-glance)

[Getting Started](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#getting-started)

[Download](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#download)

[Quick Start](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#quick-start)

[Install (Optional)](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#install-optional)

[Setting Application Arguments](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#setting-application-arguments)

[Configuration](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#configuration)

[Level](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#level)

[Configure](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#configure)

[Using Configuration File](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#using-configuration-file)

[Using el::Configurations Class](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#using-elconfigurations-class)

[Using In line Configurations](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#using-in-line-configurations)

[Default Configurations](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#default-configurations)

[Global Configurations](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#global-configurations)

[Logging Format Specifiers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-format-specifiers)

[Date/Time Format Specifiers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#datetime-format-specifiers)

[Custom Format Specifiers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#custom-format-specifiers)

[Logging Flags](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-flags)

[Application Arguments](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#application-arguments)

[Configuration Macros](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#configuration-macros)

[Reading Configurations](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#reading-configurations)

[Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging)

[Basic](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#basic)

[Conditional Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#conditional-logging)

[Occasional Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#occasional-logging)

[printf Like Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#printf-like-logging)

[Network Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#network-logging)

[Verbose Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#verbose-logging)

[Basic](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#basic-1)

[Conditional and Occasional](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#conditional-and-occasional-logging)

[Verbose Level](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#verbose-level)

[Check If Verbose Logging Is On](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#check-if-verbose-logging-is-on)

[VModule](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#vmodule)

[Registering New Loggers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#registering-new-loggers)

[Unregister Loggers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#unregister-loggers)

[Populating Existing Logger IDs](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#populating-existing-logger-ids)

[Sharing Logging Repository](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#sharing-logging-repository)

[Extra Features](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#extra-features)

[Performance Tracking](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#performance-tracking)

[Conditional Performance Tracking](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#conditional-performance-tracking)

[Make Use of Performance Tracking Data](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#make-use-of-performance-tracking-data)

[Log File Rotating](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#log-file-rotating)

[Crash Handling](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#crash-handling)

[Installing Custom Crash Handlers](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#installing-custom-crash-handlers)

[Stacktrace](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#stacktrace)

[Multi-threading](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#multi-threading)

[CHECK Macros](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#check-macros)

[Logging perror()](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-perror)

[Using Syslog](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#syslog)

[STL Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#stl-logging)

[Supported Templates](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#supported-templates)

[Qt Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#qt-logging)

[Boost Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#boost-logging)

[wxWidgets Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#wxwidgets-logging)

[Extending Library](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#extending-library)

[Logging Your Own Class](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-your-own-class)

[Logging Third-party Class](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-third-party-class)

[Manually Flushing and Rolling Log Files](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#manually-flushing-and-rolling-log-files)

[Log Dispatch Callback](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#log-dispatch-callback)

[Logger Registration Callback](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logger-registration-callback)

[Asynchronous Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#asynchronous-logging)

[Helper Classes](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#helper-classes)

[Contribution](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#contribution)

[Submitting Patches](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#submitting-patches)

[Reporting a Bug](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#reporting-a-bug)

[Compatibility](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#compatibility)

[Build Matrix](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#build-matrix)

[Licence](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#licence)

[Disclaimer](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#disclaimer)

**Overview**

Easylogging++ is single header efficient logging library for C++ applications. It is extremely powerful, highly extendable and configurable to user's requirements. It provides ability to [write your own *sinks*](https://file+.vscode-resource.vscode-cdn.net/samples/send-to-network) (via featured referred as LogDispatchCallback). This library is currently used by [hundreds of open-source projects on github](https://github.com/search?q=%22easylogging%2B%2B.h%22&type=Code&utf8=%E2%9C%93) and other open-source source control management sites.

This manual is for Easylogging++ v9.97.1. For other versions please refer to corresponding [release](https://github.com/abumq/easyloggingpp/releases) on github.

You may also be interested in [Residue](https://github.com/abumq/residue/) logging server.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Why yet another library**

If you are working on a small utility or large project in C++, this library can be handy. Its based on single header and only requires to link to single source file. (Originally it was header-only and was changed to use source file in [issue #445](https://github.com/abumq/easyloggingpp/issues/445). You can still use header-only in [v9.89](https://github.com/abumq/easyloggingpp/releases/tag/9.89)).

This library has been designed with various thoughts in mind (i.e, portability, performance, usability, features and easy to setup).

Why yet another library? Well, answer is pretty straight forward, use it as you wrote it so you can fix issues (if any) as you go or raise them on github. In addition to that, I personally have not seen any logging library based on single-header with such a design where you can configure on the go, extend it to your needs and get fast performance. I have seen other single-header logging libraries for C++ but either they use external libraries, e.g, boost or Qt to support certain features like threading, regular expression or date etc. This library has everything built-in to prevent usage of external libraries, not that I don't like those libraries, in fact I love them, but because not all projects use these libraries, I couldn't take risk of depending on them.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Features at a glance**

Easylogging++ is feature-rich containing many features that both typical and advanced developer will require while writing a software;

* [Highly configurable](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#configuration)
* [Extendable](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#extending-library)
* Extremely fast
* [Thread](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#multi-threading) and type safe
* [Cross-platform](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#compatibility)
* [Custom log patterns](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-format-specifiers)
* [Conditional and occasional logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#conditional-logging)
* [Performance tracking](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#performance-tracking)
* [Verbose logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#verbose-logging)
* [Crash handling](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#crash-handling)
* [Helper CHECK macros](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#check-macros)
* [STL logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#stl-logging)
* [Send to Syslog](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#syslog)
* [Third-party library logging (Qt, boost, wxWidgets etc)](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-third-party-class)
* [Extensible (Logging your own class or third-party class)](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging-your-own-class)
* [And many more...](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#extra-features)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Getting Started**

**Download**

Download latest version from [Latest Release](https://github.com/abumq/easyloggingpp/releases/latest)

For other releases, please visit [releases page](https://github.com/abumq/easyloggingpp/releases). If you application does not support C++11, please consider using [v8.91](https://github.com/abumq/easyloggingpp/tree/v8.91). This is stable version for C++98 and C++03, just lack some features.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Quick Start**

In order to get started with Easylogging++, you can follow three easy steps:

* Download latest version
* Include into your project (easylogging++.h and easylogging++.cc)
* Initialize using single macro... and off you go!

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

int main(int argc, char\* argv[]) {

LOG(INFO) << "My first info log using default logger";

return 0;

}

Now compile using

g++ main.cc easylogging++.cc -o prog -std=c++11

That simple! Please note that INITIALIZE\_EASYLOGGINGPP should be used once and once-only otherwise you will end up getting compilation errors. This is the definition of several extern variables. This means it can be defined only once per application. Best place to put this initialization statement is in file where int main(int, char\*\*) function is defined, right after last include statement.

**Install (Optional)**

If you want to install this header system-wide, you can do so via:

mkdir build

cd build

cmake -Dtest=ON ../

make

make test

make install

Following options are supported by Easylogging++ cmake and you can turn these options on using -D<option>=ON

* lib\_utc\_datetime - Defines ELPP\_UTC\_DATETIME
* build\_static\_lib - Builds static library for Easylogging++

With that said, you will still need easylogging++.cc file in order to compile. For header only, please check [v9.89](https://github.com/abumq/easyloggingpp/releases/tag/9.89) and lower.

Alternatively, you can download and install easyloggingpp using the [vcpkg](https://github.com/Microsoft/vcpkg/" \o "https://github.com/Microsoft/vcpkg/) dependency manager:

git clone https://github.com/Microsoft/vcpkg.git

cd vcpkg

./bootstrap-vcpkg.sh

./vcpkg integrate install

./vcpkg install easyloggingpp

The easyloggingpp port in vcpkg is kept up to date by Microsoft team members and community contributors. If the version is out of date, please [create an issue or pull request](https://github.com/Microsoft/vcpkg) on the vcpkg repository.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Setting Application Arguments**

It is always recommended to pass application arguments to Easylogging++. Some features of Easylogging++ require you to set application arguments, e.g, verbose logging to set verbose level or vmodules (explained later). In order to do that you can use helper macro or helper class;

int main(int argc, char\* argv[]) {

START\_EASYLOGGINGPP(argc, argv);

...

}

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Configuration**

**Level**

In order to start configuring your logging library, you must understand severity levels. Easylogging++ deliberately does not use hierarchical logging in order to fully control what's enabled and what's not. That being said, there is still option to use hierarchical logging using LoggingFlag::HierarchicalLogging. Easylogging++ has following levels (ordered for hierarchical levels)

| **Level** | **Description** |
| --- | --- |
| Global | Generic level that represents all levels. Useful when setting global configuration for all levels. |
| Trace | Information that can be useful to back-trace certain events - mostly useful than debug logs. |
| Debug | Informational events most useful for developers to debug application. Only applicable if NDEBUG is not defined (for non-VC++) or \_DEBUG is defined (for VC++). |
| Fatal | Very severe error event that will presumably lead the application to abort. |
| Error | Error information but will continue application to keep running. |
| Warning | Information representing errors in application but application will keep running. |
| Info | Mainly useful to represent current progress of application. |
| Verbose | Information that can be highly useful and vary with verbose logging level. Verbose logging is not applicable to hierarchical logging. |
| Unknown | Only applicable to hierarchical logging and is used to turn off logging completely. |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Configure**

Easylogging++ is easy to configure. There are three possible ways to do so,

* Using configuration file
* Using el::Configurations class
* Using inline configuration

**Using Configuration File**

Configuration can be done by file that is loaded at runtime by Configurations class. This file has following format;

\* LEVEL:

CONFIGURATION NAME = "VALUE" ## Comment

ANOTHER CONFIG NAME = "VALUE"

Level name starts with a star (\*) and ends with colon (:). It is highly recommended to start your configuration file with Global level so that any configuration not specified in the file will automatically use configuration from Global. For example, if you set Filename in Global and you want all the levels to use same filename, do not set it explicitly for each level, library will use configuration value from Global automatically. Following table contains configurations supported by configuration file.

| **Configuration Name** | **Type** | **Description** |
| --- | --- | --- |
| Enabled | bool | Determines whether or not corresponding level for logger is enabled. You may disable all logs by using el::Level::Global |
| To\_File | bool | Whether or not to write corresponding log to log file |
| To\_Standard\_Output | bool | Whether or not to write logs to standard output e.g, terminal or command prompt |
| Format | char\* | Determines format/pattern of logging for corresponding level and logger. |
| Filename | char\* | Determines log file (full path) to write logs to for corresponding level and logger |
| Subsecond\_Precision | uint | Specifies subsecond precision (previously called 'milliseconds width'). Width can be within range (1-6) |
| Performance\_Tracking | bool | Determines whether or not performance tracking is enabled. This does not depend on logger or level. Performance tracking always uses 'performance' logger unless specified |
| Max\_Log\_File\_Size | size\_t | If log file size of corresponding level is >= specified size, log file will be truncated. |
| Log\_Flush\_Threshold | size\_t | Specifies number of log entries to hold until we flush pending log data |

Please do not use double-quotes anywhere in comment, you might end up in unexpected behaviour.

Sample Configuration File

\* GLOBAL:

FORMAT = "%datetime %msg"

FILENAME = "/tmp/logs/my.log"

ENABLED = true

TO\_FILE = true

TO\_STANDARD\_OUTPUT = true

SUBSECOND\_PRECISION = 6

PERFORMANCE\_TRACKING = true

MAX\_LOG\_FILE\_SIZE = 2097152 ## 2MB - Comment starts with two hashes (##)

LOG\_FLUSH\_THRESHOLD = 100 ## Flush after every 100 logs

\* DEBUG:

FORMAT = "%datetime{%d/%M} %func %msg"

**Explanation**

Configuration file contents in above sample is straightforward. We start with GLOBAL level in order to override all the levels. Any explicitly defined subsequent level will override configuration from GLOBAL. For example, all the levels except for DEBUG have the same format, i.e, datetime and log message. For DEBUG level, we have only date (with day and month), source function and log message. The rest of configurations for DEBUG are used from GLOBAL. Also, notice {%d/%M} in DEBUG format above, if you do not specify date format, default format is used. Default values of date/time is %d/%M/%Y %h:%m:%s,%g For more information on these format specifiers, please refer to [Date/Time Format Specifier](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#datetime-format-specifiers) section below

**Usage**

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

int main(int argc, const char\*\* argv) {

*// Load configuration from file*

el::Configurations conf("/path/to/my-conf.conf");

*// Reconfigure single logger*

el::Loggers::reconfigureLogger("default", conf);

*// Actually reconfigure all loggers instead*

el::Loggers::reconfigureAllLoggers(conf);

*// Now all the loggers will use configuration from file*

}

Your configuration file can be converted to el::Configurations object (using constructor) that can be used where ever it is needed (like in above example).

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Using el::Configurations Class**

You can set configurations or reset configurations;

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

int main(int argc, const char\*\* argv) {

el::Configurations defaultConf;

defaultConf.setToDefault();

*// Values are always std::string*

defaultConf.set(el::Level::Info,

el::ConfigurationType::Format, "%datetime %level %msg");

*// default logger uses default configurations*

el::Loggers::reconfigureLogger("default", defaultConf);

LOG(INFO) << "Log using default file";

*// To set GLOBAL configurations you may use*

defaultConf.setGlobally(

el::ConfigurationType::Format, "%date %msg");

el::Loggers::reconfigureLogger("default", defaultConf);

return 0;

}

Configuration just needs to be set once. If you are happy with default configuration, you may use it as well.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Using In line Configurations**

Inline configuration means you can set configurations in std::string but make sure you add all the new line characters etc. This is not recommended because it's always messy.

el::Configurations c;

c.setToDefault();

c.parseFromText("\*GLOBAL:\n FORMAT = %level %msg");

Above code only sets Configurations object, you still need to re-configure logger/s using this configurations.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Default Configurations**

If you wish to have a configuration for existing and future loggers, you can use el::Loggers::setDefaultConfigurations(el::Configurations& configurations, bool configureExistingLoggers = false). This is useful when you are working on fairly large scale, or using a third-party library that is already using Easylogging++. Any newly created logger will use default configurations. If you wish to configure existing loggers as well, you can set second argument to true (it defaults to false).

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Global Configurations**

Level::Global is nothing to do with global configurations, it is concept where you can register configurations for all/or some loggers and even register new loggers using configuration file. Syntax of configuration file is:

-- LOGGER ID ## Case sensitive

## Everything else is same as configuration file

-- ANOTHER LOGGER ID

## Configuration for this logger

Logger ID starts with two dashes. Once you have written your global configuration file you can configure your all loggers (and register new ones) using single function;

int main(void) {

*// Registers new and configures it or*

*// configures existing logger - everything in global.conf*

el::Loggers::configureFromGlobal("global.conf");

*// .. Your prog*

return 0;

}

Please note, it is not possible to register new logger using global configuration without defining its configuration. You must define at least single configuration. Other ways to register loggers are discussed in [Logging](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#logging) section below.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logging Format Specifiers**

You can customize format of logging using following specifiers:

| **Specifier** | **Replaced By** |
| --- | --- |
| %logger | Logger ID |
| %thread | Thread ID - Uses std::thread if available, otherwise GetCurrentThreadId() on windows |
| %thread\_name | Use Helpers::setThreadName to set name of current thread (where you run setThreadName from). See [Thread Names sample](https://file+.vscode-resource.vscode-cdn.net/samples/STL/thread-names.cpp) |
| %level | Severity level (Info, Debug, Error, Warning, Fatal, Verbose, Trace) |
| %levshort | Severity level (Short version i.e, I for Info and respectively D, E, W, F, V, T) |
| %vlevel | Verbosity level (Applicable to verbose logging) |
| %datetime | Date and/or time - Pattern is customizable - see Date/Time Format Specifiers below |
| %user | User currently running application |
| %host | Computer name application is running on |
| %file\* | File name of source file (Full path) - This feature is subject to availability of \_\_FILE\_\_ macro of compiler |
| %fbase\* | File name of source file (Only base name) |
| %line\* | Source line number - This feature is subject to availability of \_\_LINE\_\_ macro of compile |
| %func\* | Logging function |
| %loc\* | Source filename and line number of logging (separated by colon) |
| %msg | Actual log message |
| % | Escape character (e.g, %%level will write %level) |

* Subject to compiler's availability of certain macros, e.g, \_\_LINE\_\_, \_\_FILE\_\_ etc

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Date/Time Format Specifiers**

You can customize date/time format using following specifiers

| **Specifier** | **Replaced By** |
| --- | --- |
| %d | Day of month (zero-padded) |
| %a | Day of the week - short (Mon, Tue, Wed, Thu, Fri, Sat, Sun) |
| %A | Day of the week - long (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday) |
| %M | Month (zero-padded) |
| %b | Month - short (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) |
| %B | Month - Long (January, February, March, April, May, June, July, August, September, October, November, December) |
| %y | Year - Two digit (13, 14 etc) |
| %Y | Year - Four digit (2013, 2014 etc) |
| %h | Hour (12-hour format) |
| %H | Hour (24-hour format) |
| %m | Minute (zero-padded) |
| %s | Second (zero-padded) |
| %g | Subsecond part (precision is configured by ConfigurationType::SubsecondPrecision) |
| %F | AM/PM designation |
| % | Escape character |

Please note, date/time is limited to 30 characters at most.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Custom Format Specifiers**

You can also specify your own format specifiers. In order to do that you can use el::Helpers::installCustomFormatSpecifier. A perfect example is %ip\_addr for TCP server application;

const char\* getIp(const el::LogMessage\*) {

return "192.168.1.1";

}

int main(void) {

el::Helpers::installCustomFormatSpecifier(el::CustomFormatSpecifier("%ip\_addr", getIp));

el::Loggers::reconfigureAllLoggers(el::ConfigurationType::Format, "%datetime %level %ip\_addr : %msg");

LOG(INFO) << "This is request from client";

return 0;

}

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logging Flags**

Form some parts of logging you can set logging flags; here are flags supported:

| **Flag** | **Description** |
| --- | --- |
| NewLineForContainer (1) | Makes sure we have new line for each container log entry |
| AllowVerboseIfModuleNotSpecified (2) | Makes sure if -vmodule is used and does not specifies a module, then verbose logging is allowed via that module. Say param was -vmodule=main\*=3 and a verbose log is being written from a file called something.cpp then if this flag is enabled, log will be written otherwise it will be disallowed. Note: having this defeats purpose of -vmodule |
| LogDetailedCrashReason (4) | When handling crashes by default, detailed crash reason will be logged as well (Disabled by default) ([issue #90](https://github.com/abumq/easyloggingpp/issues/90)) |
| DisableApplicationAbortOnFatalLog (8) | Allows to disable application abortion when logged using FATAL level. Note that this does not apply to default crash handlers as application should be aborted after crash signal is handled. (Not added by default) ([issue #119](https://github.com/abumq/easyloggingpp/issues/119)) |
| ImmediateFlush (16) | Flushes log with every log-entry (performance sensitive) - Disabled by default |
| StrictLogFileSizeCheck (32) | Makes sure log file size is checked with every log |
| ColoredTerminalOutput (64) | Terminal output will be colorful if supported by terminal. |
| MultiLoggerSupport (128) | Enables support for using multiple loggers to log single message. (E.g, CLOG(INFO, "default", "network") << This will be logged using default and network loggers;) |
| DisablePerformanceTrackingCheckpointComparison (256) | Disables checkpoint comparison |
| DisableVModules (512) | Disables usage of vmodules |
| DisableVModulesExtensions (1024) | Disables vmodules extension. This means if you have a vmodule -vmodule=main\*=4 it will cover everything starting with main, where as if you do not have this defined you will be covered for any file starting with main and ending with one of the following extensions; .h .c .cpp .cc .cxx .-inl-.h .hxx .hpp. Please note following vmodule is not correct -vmodule=main.=4 with this macro not defined because this will check for main..c, notice double dots. If you want this to be valid, have a look at logging flag above: AllowVerboseIfModuleNotSpecified '?' and '' wildcards are supported |
| HierarchicalLogging (2048) | Enables hierarchical logging. This is not applicable to verbose logging. |
| CreateLoggerAutomatically (4096) | Creates logger automatically when not available. |
| AutoSpacing (8192) | Automatically adds spaces. E.g, LOG(INFO) << "DODGE" << "THIS!"; will output "DODGE THIS!" |
| FixedTimeFormat (16384) | Applicable to performance tracking only - this prevents formatting time. E.g, 1001 ms will be logged as is, instead of formatting it as 1.01 sec |
| IgnoreSigInt (32768) | When application crashes ignore Interruption signal |

You can set/unset these flags by using static el::Loggers::addFlag and el::Loggers::removeFlag. You can check to see if certain flag is available by using el::Loggers::hasFlag, all these functions take strongly-typed enum el::LoggingFlag

You can set these flags by using --logging-flags command line arg. You need to enable this functionality by defining macro ELPP\_LOGGING\_FLAGS\_FROM\_ARG (You will need to make sure to use START\_EASYLOGGINGPP(argc, argv) to configure arguments).

You can also set default (initial) flags using ELPP\_DEFAULT\_LOGGING\_FLAGS and set numerical value for initial flags

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Application Arguments**

Following table will explain all command line arguments that you may use to define certain behaviour; You will need to initialize application arguments by using START\_EASYLOGGINGPP(argc, argv) in your main(int, char\*\*) function.

| **Argument** | **Description** |
| --- | --- |
| -v | Activates maximum verbosity |
| --v=2 | Activates verbosity upto verbose level 2 (valid range: 0-9) |
| --verbose | Activates maximum verbosity |
| -vmodule=MODULE\_NAME | Activates verbosity for files starting with main to level 1, the rest of the files depend on logging flag AllowVerboseIfModuleNotSpecified Please see Logging Flags section above. Two modules can be separated by comma. Please note vmodules are last in order of precedence of checking arguments for verbose logging, e.g, if we have -v in application arguments before vmodules, vmodules will be ignored. |
| --logging-flags=3 | Sets logging flag. In example i.e, 3, it sets logging flag to NewLineForContainer and AllowVerboseIfModuleNotSpecified. See logging flags section above for further details and values. See macros section to disable this function. |
| --default-log-file=FILE | Sets default log file for existing and future loggers. You may want to consider defining ELPP\_NO\_DEFAULT\_LOG\_FILE to prevent creation of default empty log file during pre-processing. See macros section to disable this function. |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Configuration Macros**

Some of logging options can be set by macros, this is a thoughtful decision, for example if we have ELPP\_THREAD\_SAFE defined, all the thread-safe functionalities are enabled otherwise disabled (making sure over-head of thread-safety goes with it). To make it easy to remember and prevent possible conflicts, all the macros start with ELPP\_

**NOTE:** All the macros can be defined in one of the following ways:

1. Define macros using -D option of compiler, for example in case of g++ you will do g++ source.cpp ... -DELPP\_SYSLOG -DELPP\_THREAD\_SAFE ... (**recommended way**)
2. Define macros inside "easylogging++.h" ([defining macros in other files won't work](https://github.com/abumq/easyloggingpp/issues/590#issuecomment-346753951))

| **Macro Name** | **Description** |
| --- | --- |
| ELPP\_DEBUG\_ASSERT\_FAILURE | Aborts application on first assertion failure. This assertion is due to invalid input e.g, invalid configuration file etc. |
| ELPP\_UNICODE | Enables Unicode support when logging. Requires START\_EASYLOGGINGPP |
| ELPP\_THREAD\_SAFE | Enables thread-safety - make sure -lpthread linking for linux. |
| ELPP\_FORCE\_USE\_STD\_THREAD | Forces to use C++ standard library for threading (Only useful when using ELPP\_THREAD\_SAFE |
| ELPP\_FEATURE\_CRASH\_LOG | Applicable to GCC only. Enables stacktrace on application crash |
| ELPP\_DISABLE\_DEFAULT\_CRASH\_HANDLING | Disables default crash handling. You can use el::Helpers::setCrashHandler to use your own handler. |
| ELPP\_DISABLE\_LOGS | Disables all logs - (preprocessing) |
| ELPP\_DISABLE\_DEBUG\_LOGS | Disables debug logs - (preprocessing) |
| ELPP\_DISABLE\_INFO\_LOGS | Disables info logs - (preprocessing) |
| ELPP\_DISABLE\_WARNING\_LOGS | Disables warning logs - (preprocessing) |
| ELPP\_DISABLE\_ERROR\_LOGS | Disables error logs - (preprocessing) |
| ELPP\_DISABLE\_FATAL\_LOGS | Disables fatal logs - (preprocessing) |
| ELPP\_DISABLE\_VERBOSE\_LOGS | Disables verbose logs - (preprocessing) |
| ELPP\_DISABLE\_TRACE\_LOGS | Disables trace logs - (preprocessing) |
| ELPP\_FORCE\_ENV\_VAR\_FROM\_BASH | If environment variable could not be found, force using alternative bash command to find value, e.g, whoami for username. (DO NOT USE THIS MACRO WITH LD\_PRELOAD FOR LIBRARIES THAT ARE ALREADY USING Easylogging++ OR YOU WILL END UP IN STACK OVERFLOW FOR PROCESSES (popen) (see [issue #87](https://github.com/abumq/easyloggingpp/issues/87) for details)) |
| ELPP\_DEFAULT\_LOG\_FILE | Full filename where you want initial files to be created. You need to embed value of this macro with quotes, e.g, -DELPP\_DEFAULT\_LOG\_FILE='"logs/el.gtest.log"' Note the double quotes inside single quotes, double quotes are the values for const char\* and single quotes specifies value of macro |
| ELPP\_NO\_LOG\_TO\_FILE | Disable logging to file initially |
| ELPP\_NO\_DEFAULT\_LOG\_FILE | If you dont want to initialize library with default log file, define this macro. This will log to null device for unix and windows. In other platforms you may get error and you will need to use ELPP\_DEFAULT\_LOG\_FILE. (PR for other platform's null devices are most welcomed) |
| ELPP\_FRESH\_LOG\_FILE | Never appends log file whenever log file is created (Use with care as it may cause some unexpected result for some users) |
| ELPP\_DEBUG\_ERRORS | If you wish to find out internal errors raised by Easylogging++ that can be because of configuration or something else, you can enable them by defining this macro. You will get your errors on standard output i.e, terminal or command prompt. |
| ELPP\_DISABLE\_CUSTOM\_FORMAT\_SPECIFIERS | Forcefully disables custom format specifiers |
| ELPP\_DISABLE\_LOGGING\_FLAGS\_FROM\_ARG | Forcefully disables ability to set logging flags using command-line arguments |
| ELPP\_DISABLE\_LOG\_FILE\_FROM\_ARG | Forcefully disables ability to set default log file from command-line arguments |
| ELPP\_WINSOCK2 | On windows system force to use winsock2.h instead of winsock.h when WIN32\_LEAN\_AND\_MEAN is defined |
| ELPP\_CUSTOM\_COUT (advanced) | Resolves to a value e.g, #define ELPP\_CUSTOM\_COUT qDebug() or #define ELPP\_CUSTOM\_COUT std::cerr. This will use the value for standard output (instead of using std::cout |
| ELPP\_CUSTOM\_COUT\_LINE (advanced) | Used with ELPP\_CUSTOM\_COUT to define how to write a log line with custom cout. e.g, #define ELPP\_CUSTOM\_COUT\_LINE(msg) QString::fromStdString(msg).trimmed() |
| ELPP\_NO\_CHECK\_MACROS | Do not define the *CHECK* macros |
| ELPP\_NO\_DEBUG\_MACROS | Do not define the *DEBUG* macros |
| ELPP\_UTC\_DATETIME | Uses UTC time instead of local time (essentially uses gmtime instead of localtime and family functions) |
| ELPP\_NO\_GLOBAL\_LOCK | Do not lock the whole storage on dispatch. This should be used with care. See [issue #580](https://github.com/abumq/easyloggingpp/issues/580) |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Reading Configurations**

If you wish to read configurations of certain logger, you can do so by using typedConfigurations() function in Logger class.

el::Logger\* l = el::Loggers::getLogger("default");

bool enabled = l->typedConfigurations()->enabled(el::Level::Info);

*// Or to read log format/pattern*

std::string format =

l->typedConfigurations()->logFormat(el::Level::Info).format();

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logging**

Logging in easylogging++ is done using collection of macros. This is to make it easier for user and to prevent them knowing about unnecessary greater details of how things are done.

**Basic**

You are provided with two basic macros that you can use in order to write logs:

* LOG(LEVEL)
* CLOG(LEVEL, logger ID)

LOG uses 'default' logger while in CLOG (Custom LOG) you specify the logger ID. For LEVELs please refer to Configurations - Levels section above. Different loggers might have different configurations depending on your need, you may as well write custom macro to access custom logger. You also have different macros for verbose logging that is explained in section below. Here is very simple example of using these macros after you have initialized easylogging++.

LOG(INFO) << "This is info log";

CLOG(ERROR, "performance") << "This is info log using performance logger";

There is another way to use same macro i.e, LOG (and associated macros). This is that you define macro ELPP\_DEFAULT\_LOGGER and ELPP\_DEFAULT\_PERFORMANCE\_LOGGER with logger ID that is already registered, and now when you use LOG macro, it automatically will use specified logger instead of default logger. Please note that this should be defined in source file instead of header file. This is so that when we include header we dont accidently use invalid logger.

A quick example is here

#ifndef ELPP\_DEFAULT\_LOGGER

# define ELPP\_DEFAULT\_LOGGER "update\_manager"

#endif

#ifndef ELPP\_DEFAULT\_PERFORMANCE\_LOGGER

# define ELPP\_DEFAULT\_PERFORMANCE\_LOGGER ELPP\_DEFAULT\_LOGGER

#endif

#include "easylogging++.h"

UpdateManager::UpdateManager {

\_TRACE; *// Logs using LOG(TRACE) provided logger is already registered - i.e, update\_manager*

LOG(INFO) << "This will log using update\_manager logger as well";

}

#include "easylogging++.h"

UpdateManager::UpdateManager {

\_TRACE; *// Logs using LOG(TRACE) using default logger because no `ELPP\_DEFAULT\_LOGGER` is defined unless you have it in makefile*

}

You can also write logs by using Logger class directly. This feature is available on compilers that support variadic templates. You can explore more by looking at samples/STL/logger-log-functions.cpp.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Conditional Logging**

Easylogging++ provides certain aspects of logging, one these aspects is conditional logging, i.e, log will be written only if certain condition fulfils. This comes very handy in some situations. Helper macros end with \_IF;

* LOG\_IF(condition, LEVEL)
* CLOG\_IF(condition, LEVEL, logger ID)

**Some examples:**

LOG\_IF(condition, INFO) << "Logged if condition is true";

LOG\_IF(false, WARNING) << "Never logged";

CLOG\_IF(true, INFO, "performance") << "Always logged (performance logger)"

Same macros are available for verbose logging with V in the beginning, i.e, VLOG\_IF and CVLOG\_IF. see verbose logging section below for further information. You may have as complicated conditions as you want depending on your need.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Occasional Logging**

Occasional logging is another useful aspect of logging with Easylogging++. This means a log will be written if it's hit certain times or part of certain times, e.g, every 10th hit or 100th hit or 2nd hit. Helper macros end with \_EVERY\_N;

* LOG\_EVERY\_N(n, LEVEL)
* CLOG\_EVERY\_N(n, LEVEL, logger ID)

**Other Hit Counts Based Logging**

There are some other ways of logging as well based on hit counts. These useful macros are

* LOG\_AFTER\_N(n, LEVEL); Only logs when we have reached hit counts of n
* LOG\_N\_TIMES(n, LEVEL); Logs n times

**Some examples:**

for (int i = 1; i <= 10; ++i) {

LOG\_EVERY\_N(2, INFO) << "Logged every second iter";

}

*// 5 logs written; 2, 4, 6, 7, 10*

for (int i = 1; i <= 10; ++i) {

LOG\_AFTER\_N(2, INFO) << "Log after 2 hits; " << i;

}

*// 8 logs written; 3, 4, 5, 6, 7, 8, 9, 10*

for (int i = 1; i <= 100; ++i) {

LOG\_N\_TIMES(3, INFO) << "Log only 3 times; " << i;

}

*// 3 logs writter; 1, 2, 3*

Same versions of macros are available for DEBUG only mode, these macros start with D (for debug) followed by the same name. e.g, DLOG to log only in debug mode (i.e, when \_DEBUG is defined or NDEBUG is undefined)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**printf Like Logging**

For compilers that support C++11's variadic templates, ability to log like "printf" is available. This is done by using Logger class. This feature is thread and type safe (as we do not use any macros like LOG(INFO) etc)

This is done in two steps:

1. Pulling registered logger using el::Loggers::getLogger(<logger\_id>);
2. Using one of logging functions

The only difference from printf is that logging using these functions require %v for each arg (This is for type-safety); instead of custom format specifiers. You can escape this by %%v

Following are various function signatures:

* info(const char\*, const T&, const Args&...)
* warn(const char\*, const T&, const Args&...)
* error(const char\*, const T&, const Args&...)
* debug(const char\*, const T&, const Args&...)
* fatal(const char\*, const T&, const Args&...)
* trace(const char\*, const T&, const Args&...)
* verbose(int vlevel, const char\*, const T&, const Args&...)

**Simple example:**

*// Use default logger*

el::Logger\* defaultLogger = el::Loggers::getLogger("default");

*// STL logging (`ELPP\_STL\_LOGGING` should be defined)*

std::vector<int> i;

i.push\_back(1);

defaultLogger->warn("My first ultimate log message %v %v %v", 123, 222, i);

*// Escaping*

defaultLogger->info("My first ultimate log message %% %%v %v %v", 123, 222);

%file, %func %line and %loc format specifiers will not work with printf like logging.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Network Logging**

You can send your messages to network. But you will have to implement your own way using log dispatcher API. We have written fully working sample for this purpose. Please see [Send to Network sample](https://file+.vscode-resource.vscode-cdn.net/samples/send-to-network)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Verbose Logging**

**Basic**

Verbose logging is useful in every software to record more information than usual. Very useful for troubleshooting. Following are verbose logging specific macros;

* VLOG(verbose-level)
* CVLOG(verbose-level, logger ID)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Conditional and Occasional Logging**

Verbose logging also has conditional and occasional logging aspects i.e,

* VLOG\_IF(condition, verbose-level)
* CVLOG\_IF(condition, verbose-level, loggerID)
* VLOG\_EVERY\_N(n, verbose-level)
* CVLOG\_EVERY\_N(n, verbose-level, loggerID)
* VLOG\_AFTER\_N(n, verbose-level)
* CVLOG\_AFTER\_N(n, verbose-level, loggerID)
* VLOG\_N\_TIMES(n, verbose-level)
* CVLOG\_N\_TIMES(n, verbose-level, loggerID)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Verbose-Level**

Verbose level is level of verbosity that can have range of 1-9. Verbose level will not be active unless you either set application arguments for it. Please read through [Application Arguments](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#application-arguments) section to understand more about verbose logging.

In order to change verbose level on the fly, please use Loggers::setVerboseLevel(base::type::VerboseLevel) aka Loggers::setVerboseLevel(int) function. (You can check current verbose level by Loggers::verboseLevel()

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Check If Verbose Logging Is On**

You can use a macro VLOG\_IS\_ON(verbose-level) to check to see if certain logging is on for source file for specified verbose level. This returns boolean that you can embed into if condition.

if (VLOG\_IS\_ON(2)) {

*// Verbosity level 2 is on for this file*

}

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**VModule**

VModule is functionality for verbose logging (as mentioned in above table) where you can specify verbosity by modules/source file. Following are some examples with explanation; Any of vmodule below starts with -vmodule= and LoggingFlag::DisableVModulesExtensions flag not set. Vmodule can completely be disabled by adding flag LoggingFlag::DisableVModules

Example with LoggingFlag::AllowVerboseIfModuleNotSpecified flag;

main=3,parser\*=4:

* A bad example but good enough for explanation;
* Verbosity for any following file will be allowed; main{.h, .c, .cpp, .cc, .cxx, -inl.h, .hxx, .hpp} parser{.h, .c, .cpp, .cc, .cxx, -inl.h, .hxx, .hpp}
* No other file will be logged for verbose level

Example with no LoggingFlag::AllowVerboseIfModuleNotSpecified flag;

main=3,parser\*=4: Same explanation but any other file that does not fall under specified modules will have verbose logging enabled.

In order to change vmodules on the fly (instead of via command line args) - use Loggers::setVModules(const char\*) where const char\* represents the modules e.g, main=3,parser\*=4 (as per above example)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Registering New Loggers**

Loggers are unique in logger repository by ID. You can register new logger the same way as you would get logger. Using getLogger(.., ..) from el::Loggers helper class. This function takes two params, first being ID and second being boolean (optional) to whether or not to register new logger if does not already exist and returns pointer to existing (or newly created) el::Logger class. This second param is optional and defaults to true. If you set it to false and logger does not exist already, it will return nullptr.

By default, Easylogging++ registers three loggers (+ an internal logger);

* Default logger (ID: default)
* Performance logger (ID: performance)
* Syslog logger (if ELPP\_SYSLOG macro is defined) (ID: syslog)

If you wish to register a new logger, say e.g, with ID business

el::Logger\* businessLogger = el::Loggers::getLogger("business");

This will register a new logger if it does not already exist otherwise it will get an existing one. But if you have passed in false to the second param and logger does not already exist, businessLogger will be nullptr.

When you register a new logger, default configurations are used (see Default Configurations section above). Also worth noticing, logger IDs are case sensitive.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Unregister Loggers**

You may unregister loggers; any logger except for default. You should be really careful with this function, only unregister loggers that you have created yourself otherwise you may end up in unexpected errors. For example, you dont want to unregister logger that is used or initialized by a third-party library and it may be using it.

To unregister logger, use el::Loggers::unregisterLogger("logger-id")

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Populating Existing Logger IDs**

Although this is a rare situation but if you wish to get list of all the logger IDs currently in repository, you may use el::Loggers::populateAllLoggerIds(std::vector<std::string>&) function to do that. The list passed in is cleared and filled up with all existing logger IDs.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Sharing Logging Repository**

For advance logging, you can share your logging repositories to shared or static libraries, or even from library to application. This is rare case but a very good example is as follows;

Let's say we have an application that uses easylogging++ and has its own configuration, now you are importing library that uses easylogging++ and wants to access logging repository of main application. You can do this using two ways;

* Instead of using INITIALIZE\_EASYLOGGINGPP you use SHARE\_EASYLOGGINGPP(access-function-to-repository)
* Instead of using INITIALIZE\_EASYLOGGINGPP you use INITIALIZE\_NULL\_EASYLOGGINGPP and then el::Helpers::setStorage(el::base::type::StoragePointer)

After you share repository, you can reconfigure the only repository (i.e, the one that is used by application and library both), and use both to write logs.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Extra Features**

Easylogging++ is feature-rich logging library. Apart from features already mentioned above, here are some extra features. If code snippets don't make sense and further sample is needed, there are many samples available at github repository (samples). Feel free to browse around.

Some features require you to define macros (marked as prerequisite in each section) to enable them. This is to reduce compile time. If you want to enable all features you can define ELPP\_FEATURE\_ALL.

**Performance Tracking**

Prerequisite: Define macro ELPP\_FEATURE\_PERFORMANCE\_TRACKING

One of the most notable features of Easylogging++ is its ability to track performance of your function or block of function. Please note, this is not backward compatible as previously we had macros that user must had defined in order to track performance and I am sure many users had avoided in doing so. (Read v8.91 ReadMe for older way of doing it) The new way of tracking performance is much easier and reliable. All you need to do is use one of two macros from where you want to start tracking.

* TIMED\_FUNC(obj-name)
* TIMED\_SCOPE(obj-name, block-name)
* TIMED\_BLOCK(obj-name, block-name)

An example that just uses usleep

void performHeavyTask(int iter) {

TIMED\_FUNC(timerObj);

*// Some initializations*

*// Some more heavy tasks*

usleep(5000);

while (iter-- > 0) {

TIMED\_SCOPE(timerBlkObj, "heavy-iter");

*// Perform some heavy task in each iter*

usleep(10000);

}

}

The result of above execution for iter = 10, is as following

06:22:31,368 INFO Executed [heavy-iter] in [10 ms]

06:22:31,379 INFO Executed [heavy-iter] in [10 ms]

06:22:31,389 INFO Executed [heavy-iter] in [10 ms]

06:22:31,399 INFO Executed [heavy-iter] in [10 ms]

06:22:31,409 INFO Executed [heavy-iter] in [10 ms]

06:22:31,419 INFO Executed [heavy-iter] in [10 ms]

06:22:31,429 INFO Executed [heavy-iter] in [10 ms]

06:22:31,440 INFO Executed [heavy-iter] in [10 ms]

06:22:31,450 INFO Executed [heavy-iter] in [10 ms]

06:22:31,460 INFO Executed [heavy-iter] in [10 ms]

06:22:31,460 INFO Executed [void performHeavyTask(int)] in [106 ms]

In the above example, we have used both the macros. In line-2 we have TIMED\_FUNC with object pointer name timerObj and line-7 we have TIMED\_SCOPE with object pointer name timerBlkObj and block name heavy-iter. Notice how block name is thrown out to the logs with every hit. (Note: TIMED\_FUNC is TIMED\_SCOPE with block name = function name)

You might wonder why do we need object name. Well easylogging++ performance tracking feature takes it further and provides ability to add, what's called checkpoints. Checkpoints have two macros:

* PERFORMANCE\_CHECKPOINT(timed-block-obj-name)
* PERFORMANCE\_CHECKPOINT\_WITH\_ID(timed-block-obj-name, id)

Take a look at following example

void performHeavyTask(int iter) {

TIMED\_FUNC(timerObj);

*// Some initializations*

*// Some more heavy tasks*

usleep(5000);

while (iter-- > 0) {

TIMED\_SCOPE(timerBlkObj, "heavy-iter");

*// Perform some heavy task in each iter*

*// Notice following sleep varies with each iter*

usleep(iter \* 1000);

if (iter % 3) {

PERFORMANCE\_CHECKPOINT(timerBlkObj);

}

}

}

Notice macro on line-11 (also note comment on line-8). It's checkpoint for heavy-iter block. Now notice following output

06:33:07,558 INFO Executed [heavy-iter] in [9 ms]

06:33:07,566 INFO Performance checkpoint for block [heavy-iter] : [8 ms]

06:33:07,566 INFO Executed [heavy-iter] in [8 ms]

06:33:07,573 INFO Performance checkpoint for block [heavy-iter] : [7 ms]

06:33:07,573 INFO Executed [heavy-iter] in [7 ms]

06:33:07,579 INFO Executed [heavy-iter] in [6 ms]

06:33:07,584 INFO Performance checkpoint for block [heavy-iter] : [5 ms]

06:33:07,584 INFO Executed [heavy-iter] in [5 ms]

06:33:07,589 INFO Performance checkpoint for block [heavy-iter] : [4 ms]

06:33:07,589 INFO Executed [heavy-iter] in [4 ms]

06:33:07,592 INFO Executed [heavy-iter] in [3 ms]

06:33:07,594 INFO Performance checkpoint for block [heavy-iter] : [2 ms]

06:33:07,594 INFO Executed [heavy-iter] in [2 ms]

06:33:07,595 INFO Performance checkpoint for block [heavy-iter] : [1 ms]

06:33:07,595 INFO Executed [heavy-iter] in [1 ms]

06:33:07,595 INFO Executed [heavy-iter] in [0 ms]

06:33:07,595 INFO Executed [void performHeavyTask(int)] in [51 ms]

You can also compare two checkpoints if they are in sub-blocks e.g, changing from PERFORMANCE\_CHECKPOINT(timerBlkObj) to PERFORMANCE\_CHECKPOINT(timerObj) will result in following output

06:40:35,522 INFO Performance checkpoint for block [void performHeavyTask(int)] : [51 ms ([1 ms] from last checkpoint)]

If you had used PERFORMANCE\_CHECKPOINT\_WITH\_ID(timerObj, "mychkpnt"); instead, you will get

06:44:37,979 INFO Performance checkpoint [mychkpnt] for block [void performHeavyTask(int)] : [51 ms ([1 ms] from checkpoint 'mychkpnt')]

Following are some useful macros that you can define to change the behaviour

| **Macro Name** | **Description** |
| --- | --- |
| ELPP\_DISABLE\_PERFORMANCE\_TRACKING | Disables performance tracking |
| ELPP\_PERFORMANCE\_MICROSECONDS | Track up-to microseconds (this includes initializing of el::base::PerformanceTracker as well so might time not be 100% accurate) |

Notes:

1. Performance tracking uses performance logger (INFO level) by default unless el::base::PerformanceTracker is constructed manually (not using macro - not recommended). When configuring other loggers, make sure you configure this one as well.
2. In above examples, timerObj and timerBlkObj is of type el::base::type::PerformanceTrackerPtr. The checkpoint() routine of the el::base::PerformanceTracker can be accessed by timerObj->checkpoint() but not recommended as this will override behaviour of using macros, behaviour like location of checkpoint.
3. In order to access el::base::type::PerformanceTrackerPtr while in TIMED\_BLOCK, you can use timerObj.timer
4. TIMED\_BLOCK macro resolves to a single-looped for-loop, so be careful where you define TIMED\_BLOCK, if for-loop is allowed in the line where you use it, you should have no errors.

You may be interested in [python script to parse performance logs](https://github.com/abumq/easyloggingpp/issues/206)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Conditional Performance Tracking**

If you want to enable performance tracking for certain conditions only, e.g. based on a certain verbosity level, you can use the variants TIMED\_FUNC\_IF or TIMED\_SCOPE\_IF.

A verbosity level example is given below

void performHeavyTask(int iter) {

*// enable performance tracking for verbosity level 4 or higher*

TIMED\_FUNC\_IF( timerObj, VLOG\_IS\_ON(4) );

*// Some more heavy tasks*

}

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Make Use of Performance Tracking Data**

If you wish to capture performance tracking data right after it is finished, you can do so by extending el::PerformanceTrackingCallback.

In order to install this handler, use void Helpers::installPerformanceTrackingCallback<T>(const std::string& id). Where T is type of your handler. If you wish to uninstall a callback, you can do so by using Helpers::uninstallPerformanceTrackingCallback<T>(const std::string& id). See samples for details

DO NOT TRACK PERFORMANCE IN THIS HANDLER OR YOU WILL END UP IN INFINITE-LOOP

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Log File Rotating**

Easylogging++ has ability to roll out (or throw away / rotate) log files if they reach certain limit. You can configure this by setting Max\_Log\_File\_Size. See Configuration section above.

Rollout checking happens when Easylogging++ flushes the log file, or, if you have added the flag el::LoggingFlags::StrictLogFileSizeCheck, at each log output.

This feature has its own section in this reference manual because you can do stuffs with the file being thrown away. This is useful, for example if you wish to back this file up etc. This can be done by using el::Helpers::installPreRollOutCallback(const PreRollOutCallback& handler) where PreRollOutCallback is typedef of type std::function<void(const char\*, std::size\_t)>. Please note following if you are using this feature

There is a [sample](https://file+.vscode-resource.vscode-cdn.net/samples/STL/logrotate.cpp) available that you can use as basis.

You should not log anything in this function. This is because logger would already be locked in multi-threaded application and you can run into dead lock conditions. If you are sure that you are not going to log to same file and not using same logger, feel free to give it a try.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Crash Handling**

Prerequisite: Define macro ELPP\_FEATURE\_CRASH\_LOG

Easylogging++ provides ability to handle unexpected crashes for GCC compilers. This is active by default and can be disabled by defining macro ELPP\_DISABLE\_DEFAULT\_CRASH\_HANDLING. By doing so you are telling library not to handle any crashes. Later on if you wish to handle crash yourself, you can assign crash handler of type void func(int) where int is signal caught.

Following signals are handled;

* SIGABRT (If ELPP\_HANDLE\_SIGABRT macro is defined)
* SIGFPE
* SIGILL
* SIGSEGV
* SIGINT

Stacktraces are not printed by default, in order to do so define macro ELPP\_FEATURE\_CRASH\_LOG. Remember, stack trace is only available for GCC compiler.

Default handler and stack trace uses default logger.

Following are some useful macros that you can define to change the behaviour

| **Macro Name** | **Description** |
| --- | --- |
| ELPP\_DISABLE\_DEFAULT\_CRASH\_HANDLING | Disables default crash handling. |
| ELPP\_HANDLE\_SIGABRT | Enables handling SIGABRT. This is disabled by default to prevent annoying CTRL + C behaviour when you wish to abort. |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Installing Custom Crash Handlers**

You can use your own crash handler by using el::Helpers::setCrashHandler(const el::base::debug::CrashHandler::Handler&);.

Make sure to abort application at the end of your crash handler using el::Helpers::crashAbort(int). If you fail to do so, you will get into endless loop of crashes.

Here is a good example of your own handler

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

void myCrashHandler(int sig) {

LOG(ERROR) << "Woops! Crashed!";

*// FOLLOWING LINE IS ABSOLUTELY NEEDED AT THE END IN ORDER TO ABORT APPLICATION*

el::Helpers::crashAbort(sig);

}

int main(void) {

el::Helpers::setCrashHandler(myCrashHandler);

LOG(INFO) << "My crash handler!";

int\* i;

\*i = 0; *// Crash!*

return 0;

}

If you wish to log reason for crash you can do so by using el::Helpers::logCrashReason(int, bool, const el::Level&, const char\*). Following are default parameters for this function:

> bool stackTraceIfAvailable = false

> const el::Level& level = el::Level::Fatal

> const char\* logger = "default"

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Stacktrace**

Prerequisite: Define macro ELPP\_FEATURE\_CRASH\_LOG

Easylogging++ supports stack trace printing for GCC compilers. You can print stack trace at anytime by calling el::base::debug::StackTrace(), formatting will be done automatically. Note, if you are using non-GCC compiler, you will end-up getting empty output.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Multi-threading**

Prerequisite: Define macro ELPP\_THREAD\_SAFE

Easylogging++ is thread-safe. By default thread-safety is disabled. You can enable it by defining ELPP\_THREAD\_SAFE otherwise you will see unexpected results. This is intentional to make library efficient for single threaded application.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**CHECK Macros**

Easylogging++ supports CHECK macros, with these macros you can quickly check whether certain condition fulfills or not. If not Easylogging++ writes FATAL log, causing application to stop (unless defined macro to prevent stopping application on fatal).

| **CHECK Name** | **Notes + Example** |
| --- | --- |
| CHECK(condition) | Checks for condition e.g, CHECK(isLoggedIn()) << "Not logged in"; |
| CHECK\_EQ(a, b) | Equality check e.g, CHECK\_EQ(getId(), getLoggedOnId()) << "Invalid user logged in"; |
| CHECK\_NE(a, b) | Inequality check e.g, CHECK\_NE(isUserBlocked(userId), false) << "User is blocked"; |
| CHECK\_LT(a, b) | Less than e.g, CHECK\_LT(1, 2) << "How 1 is not less than 2"; |
| CHECK\_GT(a, b) | Greater than e.g, CHECK\_GT(2, 1) << "How 2 is not greater than 1?"; |
| CHECK\_LE(a, b) | Less than or equal e.g, CHECK\_LE(1, 1) << "1 is not equal or less than 1"; |
| CHECK\_GE(a, b) | Greater than or equal e.g, CHECK\_GE(1, 1) << "1 is not equal or greater than 1"; |
| CHECK\_NOTNULL(pointer) | Ensures pointer is not null. This function does not return anything |
| CHECK\_STREQ(str1, str2) | C-string equality (case-sensitive) e.g, CHECK\_STREQ(argv[1], "0") << "First arg cannot be 0"; |
| CHECK\_STRNE(str1, str2) | C-string inequality (case-sensitive) e.g, CHECK\_STRNE(username1, username2) << "Usernames cannot be same"; |
| CHECK\_STRCASEEQ(str1, str2) | C-string inequality (*case-insensitive*) e.g, CHECK\_CASESTREQ(argv[1], "Z") << "First arg cannot be 'z' or 'Z'"; |
| CHECK\_STRCASENE(str1, str2) | C-string inequality (*case-insensitive*) e.g, CHECK\_STRCASENE(username1, username2) << "Same username not allowed"; |
| CHECK\_BOUNDS(val, min, max) | Checks that val falls under the min and max range e.g, CHECK\_BOUNDS(i, 0, list.size() - 1) << "Index out of bounds"; |

Same versions of macros are available for DEBUG only mode, these macros start with D (for debug) followed by the same name. e.g, DCHECK to check only in debug mode (i.e, when \_DEBUG is defined or NDEBUG is undefined)

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logging perror()**

Easylogging++ supports perror() styled logging using PLOG(LEVEL), PLOG\_IF(Condition, LEVEL), and PCHECK() using default logger; and for custom logger use CPLOG(LEVEL, LoggerId), CPLOG\_IF(Condition, LEVEL, LoggerId). This will append : log-error [errno] in the end of log line.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Syslog**

Prerequisite: Define macro ELPP\_SYSLOG

Easylogging++ supports syslog for platforms that have syslog.h header. If your platform does not have syslog.h, make sure you do not define this macro or you will end up in errors. Once you are ready to use syslog, you can do so by using one of SYSLOG(LEVEL), SYSLOG\_IF(Condition, LEVEL), SYSLOG\_EVERY\_N(n, LEVEL) and uses logger ID: syslog. If you want to use custom logger you can do so by using CSYSLOG(LEVEL, loggerId) or CSYSLOG\_IF(Condition, LEVEL, loggerId) or CSYSLOG\_EVERY\_N(n, LEVEL, loggerId)

Syslog in Easylogging++ supports C++ styled streams logging, following example;

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

int main(void) {

ELPP\_INITIALIZE\_SYSLOG("my\_proc", LOG\_PID | LOG\_CONS | LOG\_PERROR, LOG\_USER) *// This is optional, you may not add it if you dont want to specify options*

*// Alternatively you may do*

*// el::SysLogInitializer elSyslogInit("my\_proc", LOG\_PID | LOG\_CONS | LOG\_PERROR, LOG\_USER);*

SYSLOG(INFO) << "This is syslog - read it from /var/log/syslog"

return 0;

}

Syslog support for Easylogging++ only supports following levels; each level is corresponded with syslog priority as following

* INFO (LOG\_INFO)
* DEBUG (LOG\_DEBUG)
* WARNING (LOG\_WARNING)
* ERROR (LOG\_ERR)
* FATAL (LOG\_EMERG)

Following levels are not supported and correspond to LOG\_NOTICE: TRACE, whereas VERBOSE level is completely not supported

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**STL Logging**

Prerequisite: Define macro ELPP\_STL\_LOGGING

As mentioned earlier, with easylogging++, you can log your STL templates including most containers. In order to do so you will need to define ELPP\_STL\_LOGGING macro. This enables including all the necessary headers and defines all necessary functions. For performance, containers are limited to log maximum of 100 entries. This behaviour can be changed by changed header file (base::consts::kMaxLogPerContainer) but not recommended as in order to log, writer has to go through each entry causing potential delays. But if you are not really concerned with performance, you may change this value.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Supported Templates**

Following templates are supported as part of STL Logging; note: basic and primitive types e.g, std::string or long are not listed as they is supported anyway, following list only contains non-basic types e.g, containers or bitset etc.

| **\*** | **\*** | **\*** | **\*** |
| --- | --- | --- | --- |
| std::vector | std::list | std::deque | std::queue |
| std::stack | std::priority\_queue | std::set | std::multiset |
| std::pair | std::bitset | std::map | std::multimap |

Some C++11 specific templates are supported by further explicit macro definitions; note these also need ELPP\_STL\_LOGGING

| **Template** | **Macro Needed** |
| --- | --- |
| std::array | ELPP\_LOG\_STD\_ARRAY |
| std::unordered\_map | ELPP\_LOG\_UNORDERED\_MAP |
| std::unordered\_multimap | ELPP\_LOG\_UNORDERED\_MAP |
| std::unordered\_set | ELPP\_LOG\_UNORDERED\_SET |
| std::unordered\_multiset | ELPP\_LOG\_UNORDERED\_SET |

Standard manipulators are also supported, in addition std::stringstream is also supported.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Qt Logging**

Prerequisite: Define macro ELPP\_QT\_LOGGING

Easylogging++ has complete logging support for Qt core library. When enabled, this will include all the headers supported Qt logging. Once you did that, you should be good to go.

Following Qt classes and containers are supported by Easylogging++ v9.0+

| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| --- | --- | --- | --- | --- | --- |
| QString | QByteArray | QLatin | QList | QVector | QQueue |
| QSet | QPair | QMap | QMultiMap | QHash | QMultiHash |
| QLinkedList | QStack | QChar | q[u]int[64] |  |  |

Similar to STL logging, Qt containers are also limit to log 100 entries per log, you can change this behaviour by changing base::consts::kMaxLogPerContainer from header but this is not recommended as this was done for performance purposes.

Also note, if you are logging a container that contains custom class, make sure you have read Extending Library section below.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Boost Logging**

Prerequisite: Define macro ELPP\_BOOST\_LOGGING

Easylogging++ supports some of boost templates. Following table shows the templates supported.

| **\*** | **\*** |
| --- | --- |
| boost::container::vector | boost::container::stable\_vector |
| boost::container::map | boost::container::flat\_map |
| boost::container::set | boost::container::flat\_set |
| boost::container::deque | boost::container::list |
| boost::container::string |  |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**wxWidgets Logging**

Prerequisite: Define macro ELPP\_WXWIDGETS\_LOGGING

Easylogging++ supports some of wxWidgets templates.

Following table shows the templates supported.

| **\*** | **\*** | **\*** | **\*** | **\*** | **\*** |
| --- | --- | --- | --- | --- | --- |
| wxString | wxVector | wxList | wxString | wxHashSet | wxHashMap |

wxWidgets has its own way of declaring and defining some templates e.g, wxList where you use WX\_DECLARE\_LIST macro to declare a list.

In order to setup a container for logging that holds pointers to object, use ELPP\_WX\_PTR\_ENABLED, otherwise if container holds actual object e.g, wxHashSet use ELPP\_WX\_ENABLED. For containers like wxHashMap because it contains value and pair, use ELPP\_WX\_HASH\_MAP\_ENABLED macro.

*// wxList example*

WX\_DECLARE\_LIST(int, MyList);

WX\_DEFINE\_LIST(MyList);

*// Following line does the trick*

ELPP\_WX\_PTR\_ENABLED(MyList);

*// wxHashSet example*

WX\_DECLARE\_HASH\_SET(int, wxIntegerHash, wxIntegerEqual, IntHashSet);

*// Following line does the trick!*

ELPP\_WX\_ENABLED(IntHashSet)

*// wxHashMap example*

WX\_DECLARE\_STRING\_HASH\_MAP(wxString, MyHashMap);

*// Following line does the trick*

ELPP\_WX\_HASH\_MAP\_ENABLED(MyHashMap)

You may also have a look at wxWidgets sample

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Extending Library**

You can extend this library using various callback handlers and inheritable classes.

A perfect example of using these features is the logging server built with this library. It's called [Residue](https://github.com/abumq/residue/) that is feature rich. In fact, you may be interested in using that instead of this library for your medium to large sized projects.

**Logging Your Own Class**

You can log your own classes by extending el::Loggable class and implementing pure-virtual function void log(std::ostream& os) const. Following example shows a good way to extend a class.

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

class Integer : public el::Loggable {

public:

Integer(int i) : m\_underlyingInt(i) {

}

Integer& operator=(const Integer& integer) {

m\_underlyingInt = integer.m\_underlyingInt;

return \*this;

}

*// Following line does the trick!*

*// Note: el::base::type::ostream\_t is either std::wostream or std::ostream depending on unicode enabled or not*

virtual void log(el::base::type::ostream\_t& os) const {

os << m\_underlyingInt;

}

private:

int m\_underlyingInt;

};

int main(void) {

Integer count = 5;

LOG(INFO) << count;

return 0;

}

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logging Third-party Class**

Let's say you have third-party class that you don't have access to make changes to, and it's not yet loggable. In order to make it loggable, you can use MAKE\_LOGGABLE(ClassType, ClassInstance, OutputStreamInstance) to make it Easylogging++ friendly.

Following sample shows a good usage:

#include "easylogging++.h"

INITIALIZE\_EASYLOGGINGPP

class Integer {

public:

Integer(int i) : m\_underlyingInt(i) {

}

Integer& operator=(const Integer& integer) {

m\_underlyingInt = integer.m\_underlyingInt;

return \*this;

}

int getInt(void) const { return m\_underlyingInt; }

private:

int m\_underlyingInt;

};

*// Following line does the trick!*

inline MAKE\_LOGGABLE(Integer, integer, os) {

os << integer.getInt();

return os;

}

int main(void) {

Integer count = 5;

LOG(INFO) << count;

return 0;

}

Another very nice example (to log std::chrono::system\_clock::time\_point)

inline MAKE\_LOGGABLE(std::chrono::system\_clock::time\_point, when, os) {

time\_t t = std::chrono::system\_clock::to\_time\_t(when);

auto tm = std::localtime(&t);

char buf[1024];

strftime(buf,sizeof(buf), "%F %T (%Z)", tm);

os << buf;

return os;

}

This may not be practically best implementation but you get the point.

Just be careful with this as having a time-consuming overloading of log(el::base::type::ostream\_t& os) and MAKE\_LOGGABLE, they get called everytime class is being logged.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Manually Flushing and Rolling Log Files**

You can manually flush log files using el::Logger::flush() (to flush single logger with all referencing log files) or el::Loggers::flushAll() (to flush all log files for all levels).

If you have not set flag LoggingFlag::StrictLogFileSizeCheck for some reason, you can manually check for log files that need rolling; by using el::Helpers::validateFileRolling(el::Logger\*, const el::Level&).

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Log Dispatch Callback**

If you wish to capture log message right after it is dispatched, you can do so by having a class that extends el::LogDispatchCallback and implement the pure-virtual functions, then install it at anytime using el::Helpers::installLogDispatchCallback<T>(const std::string&). If you wish to uninstall a pre-installed handler with same ID, you can do so by using el::Helpers::uninstallLogDispatchCallback<T>(const std::string&)

You can use this feature to send it to custom destinations e.g, log stash or TCP client etc.

You can also look at [send-to-network](https://file+.vscode-resource.vscode-cdn.net/samples/send-to-network) sample for practical usage of this.

*// samples/send-to-network/network-logger.cpp*

#include "easylogging++.h"

#include <boost/asio.hpp>

INITIALIZE\_EASYLOGGINGPP

class Client

{

boost::asio::io\_service\* io\_service;

boost::asio::ip::tcp::socket socket;

public:

Client(boost::asio::io\_service\* svc, const std::string& host, const std::string& port)

: io\_service(svc), socket(\*io\_service)

{

boost::asio::ip::tcp::resolver resolver(\*io\_service);

boost::asio::ip::tcp::resolver::iterator endpoint = resolver.resolve(boost::asio::ip::tcp::resolver::query(host, port));

boost::asio::connect(this->socket, endpoint);

};

void send(std::string const& message) {

socket.send(boost::asio::buffer(message));

}

};

class NetworkDispatcher : public el::LogDispatchCallback

{

public:

void updateServer(const std::string& host, int port) {

m\_client = std::unique\_ptr<Client>(new Client(&m\_svc, host, std::to\_string(port)));

}

protected:

void handle(const el::LogDispatchData\* data) noexcept override {

m\_data = data;

*// Dispatch using default log builder of logger*

dispatch(m\_data->logMessage()->logger()->logBuilder()->build(m\_data->logMessage(),

m\_data->dispatchAction() == el::base::DispatchAction::NormalLog));

}

private:

const el::LogDispatchData\* m\_data;

boost::asio::io\_service m\_svc;

std::unique\_ptr<Client> m\_client;

void dispatch(el::base::type::string\_t&& logLine) noexcept

{

m\_client->send(logLine);

}

};

int main() {

el::Helpers::installLogDispatchCallback<NetworkDispatcher>("NetworkDispatcher");

*// you can uninstall default one by*

*// el::Helpers::uninstallLogDispatchCallback<el::base::DefaultLogDispatchCallback>("DefaultLogDispatchCallback");*

*// Set server params*

NetworkDispatcher\* dispatcher = el::Helpers::logDispatchCallback<NetworkDispatcher>("NetworkDispatcher");

dispatcher->setEnabled(true);

dispatcher->updateServer("127.0.0.1", 9090);

*// Start logging and normal program...*

LOG(INFO) << "First network log";

*// You can even use a different logger, say "network" and send using a different log pattern*

}

DO NOT LOG ANYTHING IN THIS HANDLER OR YOU WILL END UP IN INFINITE-LOOP

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Logger Registration Callback**

If you wish to capture event of logger registration (and potentially want to reconfigure this logger without changing default configuration) you can use el::LoggerRegistrationCallback. The syntax is similar to [other callbacks](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#log-dispatch-callback). You can use [this sample](https://github.com/abumq/easyloggingpp/blob/master/samples/STL/new-logger-registration-callback.cpp) as basis.

DO NOT LOG ANYTHING IN THIS HANDLER

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Asynchronous Logging**

Prerequisite: Define macro ELPP\_EXPERIMENTAL\_ASYNC

Asynchronous logging is in experimental stages and they are not widely promoted. You may enable and test this feature by defining macro ELPP\_EXPERIMENTAL\_ASYNC and if you find some issue with the feature please report in [this issue](https://github.com/abumq/easyloggingpp/issues/202). Reporting issues always help for constant improvements.

Please note:

* Asynchronous will only work with few compilers (it purely uses std::thread)
* Compiler should support std::this\_thread::sleep\_for. This restriction may (or may not) be removed in future (stable) version of asynchronous logging.
* You should not rely on asynchronous logging in production, this is because feature is in experimental stages.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Helper Classes**

There are static helper classes available to make it easy to do stuffs;

* el::Helpers
* el::Loggers

You can do various cool stuffs using functions in these classes, see [this issue](https://github.com/abumq/easyloggingpp/issues/210) for instance.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Contribution**

**Submitting Patches**

You can submit patches to develop branch and we will try and merge them. Since it's based on single header, it can be sometimes difficult to merge without having merge conflicts.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Reporting a Bug**

If you have found a bug and wish to report it, feel free to do so at [github issue tracker](https://github.com/abumq/easyloggingpp/issues?state=open" \o "https://github.com/abumq/easyloggingpp/issues?state=open). I will try to look at it as soon as possible. Some information should be provided to make it easy to reproduce;

* Platform (OS, Compiler)
* Log file location
* Macros defined (on compilation) OR simple compilation
* Please assign issue label.

Try to provide as much information as possible. Any bug with no clear information will be ignored and closed.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Compatibility**

Easylogging++ requires a decent C++0x compliant compiler. Some compilers known to work with v9.0+ are shown in table below, for older versions please refer to readme on corresponding release at github

| **\*\*\*\*\*** | **Compiler/Platform** | **Notes** |
| --- | --- | --- |
|  | GCC 4.6.4+ | Stack trace logging. Very close to support GCC 4.6.0 if it had supported strong enum types casting to underlying type. It causes internal compiler error. |
|  | Clang++ 3.1+ | Stack trace logging only with gcc compliant. |
|  | Intel C++ 13.0+ | Workarounds to support: Use if instead of switch on strong enum type. No final keyword etc. Stack trace logging only with gcc compliant |
|  | Visual C++ 11.0+ | Tested with VS2012, VS2013; Use of argument templates instead of variadic templates. CRT warnings control. No stack trace logging. |
|  | MinGW | (gcc version 4.7+) Workarounds to support: Mutex wrapper, no stack trace logging. No thread ID on windows |
|  | TDM-GCC 4.7.1 | Tested with TDM-GCC 4.7.1 32 and 64 bit compilers |
|  | Cygwin | Tested with gcc version 4.8+ |
|  | Dev C++ 5.4+ | Tested with Dev-C++ 5.4.2 using TDM-GCC 4.7.1 32 & 64-bit compilers |

Operating systems that have been tested are shown in table below. Easylogging++ should work on other major operating systems that are not in the list.

| **\*\*\*\*\*** | **Operating System** | **Notes** |
| --- | --- | --- |
|  | Windows 10 | Tested on 64-bit, should also work on 32-bit |
|  | Windows 8 | Tested on 64-bit, should also work on 32-bit |
|  | Windows 7 | Tested on 64-bit, should also work on 32-bit |
|  | Windows XP | Tested on 32-bit, should also work on 64-bit |
|  | Mac OSX | Clang++ 3.1, g++ (You need -std=c++11 -stdlib=libc++ to successfully compile) |
|  | Scientific Linux 6.2 | Tested using Intel C++ 13.1.3 (gcc version 4.4.6 compatibility) |
|  | Linux Mint 14 | 64-bit, mainly developed on this machine using all compatible linux compilers |
|  | Fedora 19 | 64-bit, using g++ 4.8.1 |
|  | Ubuntu 13.04 | 64-bit, using g++ 4.7.3 (libstdc++6-4.7-dev) |
|  | FreeBSD | (from github user) |
|  | Android | Tested with C4droid (g++) on Galaxy Tab 2 |
|  | RaspberryPi 7.6 | Tested with 7.6.2-1.1 (gcc version 4.9.1 (Raspbian 4.9.1-1)) by contributor |
|  | Solaris i86 | Tested by contributor |
|  | IBM AIX | Support added by contributor |

Easylogging++ has also been tested with following C++ libraries;

| **\*\*\*\*\*** | **Library** | **Notes** |
| --- | --- | --- |
|  | Qt | Tested with Qt 4.6.2, Qt 5 and Qt 5.5 (with C++0x and C++11) |
|  | Boost | Tested with boost 1.51 |
|  | wxWidgets | Tested with wxWidgets 2.9.4 |
|  | gtkmm | Tested with gtkmm 2.4 |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Build Matrix**

| **Branch** | **Platform** | **Build Status** |
| --- | --- | --- |
| develop | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / clang++ |  |
| develop | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-4.9 |  |
| develop | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-5 |  |
| develop | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-6 |  |
| develop | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-7 |  |
| master | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / clang++ |  |
| master | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-4.9 |  |
| master | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-5 |  |
| master | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-6 |  |
| master | GNU/Linux 4.4 / Ubuntu 4.8.4 64-bit / g++-7 |  |

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Licence**

The MIT License (MIT)

Copyright (c) 2012-present @abumq (Majid Q.)

Permission is hereby granted, free of charge, to any person obtaining a copy of

this software and associated documentation files (the "Software"), to deal in

the Software without restriction, including without limitation the rights to

use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of

the Software, and to permit persons to whom the Software is furnished to do so,

subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS

FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR

COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER

IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN

CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)

**Disclaimer**

Icons used in this manual (in compatibility section) are solely for information readability purposes. I do not own these icons. If anyone has issues with usage of these icon, please feel free to contact me via company's email and I will look for an alternative. Company's email address is required so that I can verify the ownership, any other email address for this purpose will be ignored.

"Pencil +" icon is Easylogging++ logo and should only be used where giving credit to Easylogging++ library.

[[top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents) Goto Top](https://file+.vscode-resource.vscode-cdn.net/d%3A/Work/Temp/easyloggingpp-9.97.1/README.md#table-of-contents)