Log Facility for C

|  |  |  |  |
| --- | --- | --- | --- |
| 2016/03/12 | 0.1 | Initial Version | Muhammad Z |
| 2016/03/18 | 0.2 | High Level Macros removed | Robert P |
| 2016/03/25 | 0.9 | Implementation details | Muhammad Z |
| 2016/03/29 | 0.99 | Final Draft | Muhammad Z |
| 2016/04/14 | 1.0 | Version 1 | Muhammad Z |
| 2018/02/20 | 2.0 | Version 2.0 | Muhammad Z |

# 

# Summary

The purpose of this project is to create a log facility for use from C programs.

You will gain experience in:

1. Module and API Design
2. Advanced C programming
3. Putting Data Structures to actual usage.
4. Macros

# What is a log?

A log can be a file or a stream where we can store log entries. Log entry contains information that depicts an error message or a trace message. Naturally, a program composed of different modules will have different requirements for logging. While some modules will be writing only critical errors other modules might be configured to output very detailed trace messages to help understand a situation.

Basically any module can be specifically configured to write to a specific log file only entries of certain level or higher.

# What is a log entry?

A log entry is basically a record (terminated with a new line) containing a detailed information written into the log file. The record contains information provided by the developer and information gathered automatically. Each entry has a given severity. If the log entry severity is lower than the current log level it will be ignored otherwise it will be written to the log file.

# Log Entry Detailed Requirements

A log entry written to the log file will look differently depending on the build type:

**Debug builds:**

If NDEBUG is not defined when building the code using the log facility, then an entry will look like:

+- year  
 | +- month +- process id  
 | | +- day | +- thread id   
 | | | | |   
**2016-4-29 22:43:20.244 40059 1299 I query dequeue@reader.c:9 Queued tasks: 4**   
 | | | | | | +-- message w/ params  
 +- time | | | | +--- line number  
 | | | +- source file name  
 | | +- function name   
 | +- tag - Module name  
 +- log level (first letter)

Where the message parts in blue are provided by the programmer and the rest generated automatically.

**Release Builds**

If NDEBUG is defined then the log will not have: file name, function name or line number.

# 

# Requirements

### R1. Efficiency

The Log will be efficient for use from single threaded applications. Nevertheless, there will be minimal performance overhead penalty for calling log functions that will not actually write a log entry due to current configuration.

### R2. Log Levels

The Log will support various levels of logging. It will support these levels with an option to extend to more fine grained levels:

LOG\_**T**RACE Trace message usually very detailed

LOG\_**D**EBUG A message useful for debugging

LOG\_**I**NFO Informative message

LOG\_**W**ARNING Warning

LOG\_**E**RROR An error occurred

LOG\_**C**RITICAL Critical error

LOG\_**S**EVERE Server error requiring immediate intervention

LOG\_**F**ATAL Fatal error signaling an emergency

LOG\_NONE Used only in configuration file

If a the current log level configured for a module is higher than the log level associated with a log entry, then that log entry will not be written to the log file. The bold letter (blue) is what appears in the log entry in the log file.

Each log level has an associated priority. LOG\_NONE is highest level and will be used only in configuration file to silence log entries from a given module.

LOG\_TRACE is lowest priority and effectively will allow all log entries to written to the log file.

### 

### 

### R3. Log Configuration File

The Log facility will be configured by a simple configuration file. The configuration file will have general setting:

**Level =** *default\_log\_level*

**File =** *log\_base\_directory****/****default\_log\_file\_name*

Default log level, only log messages with equal or higher level will actually be saved to the log file.

Directory path specifying where all log files will be placed.

File name to be used for logging entries.

# Usage Example

### Log Configuration File

A sample log configuration file:

**Level =** *ERROR*

**File =** */var/log/dbServer/db.log*

### Initial setup of log facility

The log facility should be initialized during the application startup. Usually in main() :

#include "zlog4c.h"

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

zlog\_init("app.log.config");

...

}

### 

### Usage from a module

The use of the log facility from a module: you use ZLOG macro to send messages to the log file.

For instance:

#include "zlog4c.h"

#define MODULE "net" /\* network module \*/

int net\_start\_listening(int port){

(1) ZLOG(MODULE, LOG\_TRACE, "will start listening on port %d", port);

...

if( r != 0 ){

(2) ZLOG(MODULE, LOG\_ERROR, "listen on port:%d failed: %d", port, r);

return r;

}

...

1. - This log entry will **not** be recorded since it’s level is lower than the configured level for this module ( LOG\_TRACE < LOG\_ERROR). See sample configuration file above.
2. - This log entry will be recorded according to the above configuration file.

# Deliverables

1. Source file: zlog4c.h, zlog4c.c, makefile, testapp and a static library.
2. The zlog4c.h header file should be fully documented

# Constraints

* The log library will handle error in the logging process gracefully. It will report internal errors (of it’s own to standard error stream or to the general log file if available)
* Log entries will be appended to the log file if it exists, it’s not supposed to truncate existing files.
* Make sure the shutdown process is graceful. (closing file)