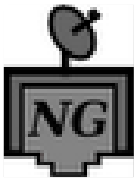
nglogc

Flexible C Logging API



Version: 1.0.0 2010-03-07

Author: Dennis Krzyzaniak

Introduction

Logging is a powerful mechanism to obtain runtime information from software components (programs). These information can be of different types: transaction logs for audit purposes, logs for error detection or detailed debug information for error analysis. Also, the more the source code is growing the harder it gets to have an idea on exactly which place an error occurs or if the program has the correct chain of activity without any log for verification.

There are already tons of examples running everywhere, which make use of logging and also many logging packages are available in the public. However, it can still be found very often, that developers start to use a debugger or add some printf-kind statements to their code in case they run into trouble with software, which is not working as expected.

The difference between an audit trace and a debug log is that the first might be wanted in a productional environment while the latter might be required only during error analysis and should not influence normal operation. It might also be wanted to ouput the one information into a file while the other should go onto the display or to some other device.

But one is equal for all kind of information: they are different levels of information from a software component which may be routed to different destinations.

The intention of the nglogc library is to provide an easy to use and powerful logging API with mechanism which allows to lumbering source codes with log statements at the start of implementation and decide at the level of building or at runtime which statements should be processed.

Therefore the log statements could be controlled by various log levels and define switches are available to completely remove the call of the functions at pre-processor time. So it is possible to switch on the logging only if it is necessary or only print selected messages without any changes in the source code. Different publishers are provided also as different formats of outputs to fulfil the requirements for software development.

This documentation will start with a simple example to use nglogc and goes over the configuration functions thru the logging functions and define switches and ends with more examples for the usage.

Simple example

This simple example shows a short way to use nglogc for logging.
At first a logger which is identified by an uint16_t type must be registered with a associated log level and publisher. Three different basic log functions are available, error logging, info logging and data array logging. Because a logger uses memory of the heap a remove function should be called at the end of the programm.

```
#include "nglogc/src/nglogc/log.h"

/* loggers are identified by an uint16_t type */
#define MAIN_LOGGER 0x0001

/* example error code */
#define EXAMPLE_ERR 0x00000001


int main(int argc, char *argv[])
{
   int i = 0;
   unsigned char data[16] = {0};

   /* register a logger with the stdout publisher and
      LOG_BASIC log level */
   logc_registerLogger(MAIN_LOGGER, STDOUT, LOG_BASIC);

   /* fill some test data */
   for (i=0; i<sizeof(data); i++) {
      data[i] = 'A' + i;
   }

   /* log an error message with the log level LOG_BASIC */
   logc_logError(MAIN_LOGGER, LOG_BASIC, EXAMPLE_ERR,
      "This is a error message");

   /* log an info message with the log level LOG_BASIC */
   logc_log(MAIN_LOGGER, LOG_BASIC,
      "This is a log message");

   /* log an data array in hex with description */
   logc_logArray(MAIN_LOGGER, LOG_BASIC,
      "Data array", data, sizeof(data));

   /* remove the logger */
   logc_removeLogger(MAIN_LOGGER);

   return 0;
}
```

The library and the header file are in the directory nglogc/src/nglogc so this example could be build with gcc -o main main.c -L nglogc/src/nglogc -lnglogc.

The output of this example is:

```
ERR : This is a error message
This is a log message
Data array : 41424344454647484 94A4B4C4D4E4F50
```

Log Configuration

The basic idea of nglogc is to have a logger which is used by the log functions. A logger is a private type which stores information and could be configured by function calls or can be used with the default settings. To use the various logging functions at least one logger must be registered first. A logger holds information about the log level, the publisher and the log format of the output. A logger is stored internally and should be removed if it is not used any more.

Logger

To use a log function a logger must be registered.

```
logc_error_t
logc_registerLogger(
      uint16_t ident,
      logc_loggerType_t type,
      logc_logLevel_t level
     )
```

Description: Register a logger.

Parameters:

ident	(I) identifier for the logger.
type	(I) Type of publisher. Could be STDOUT, STDERROUT or FILEOUT See section publisher for detailed information.
Level	(I) Log level see section Log Level for detailed information

Returns:

LOG_ERR_OK for success.
LOG_ERR_PARAM for invalid parameters.
LOG_ERR_MEM if no memory could be allocated.

A logger is allocated on the heap so the logger should be removed if it is not needed anymore.

```
logc_error_t
logc_removeLogger(
      uint16_t ident
      )
```

Description: Remove a logger.

Parameters:

ident	(I) identifier of the logger to remove.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NOT_FOUND for invalid logger id.

Log Level

Each logger has its own log level which is set by registering a logger or can be changed with a function call.

```
logc_error_t
logc_changeLogLevel(
      uint16_t ident,
      logc_logLevel_t level
      )
```

Description: Change the log level.

Parameters:

ident	(I) identifier of the logger.
level	(I) new log level.

Returns:

LOG_ERR_OK for success.
LOG_ERR_PARAM for invalid log level.
LOG_ERR_NOT_FOUND for invalid logger id.

The type `logc_logLevel_t` is defined as an enumeration with the following levels in order of priority:

LOG_BASIC
`severe log message`

LOG_WARNING
`warning log message`

LOG_INFO
`informal log message`

LOG_FINE
`debug log message`

LOG_FINEST
`detailed debug message`

LOG_SILENT
`no logging`

A log messages will only be processed if the log message has the same or a higher log level as the logger. This could be used also at runtime to decide which messages should be printed.

Log Publisher

A logger is associated to a publisher which prints out the log messages. This publisher is set by the registration call of the logger by the value `logc_loggerType_t type.`

`logc_loggerType_t` is defined as a enumeration with the entries:

STDOUT

STDERROUT

FILEOUT

STDOUT prints out messages to standard out, STDERROUT prints out messages to standard err and FILEOUT prints out messages to a file. To use a file for logging output it must be defined first with a function call.

```
logc_error_t
logc_setLogFile(
      uint16_t ident,
      const char* const filename
      )
```

Description: Set the file for logging output.

Parameters:

ident	(I) identifier of the logger.
filename	(I) name of the file. The file is open with the append option.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if filename is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_OPEN_FILE if the file could not be opened.

It will have no effect to set a file for a logger which has not the FILEOUT publisher.

Log Format

A logger has two different types of log formats, one for error logging and one for info logging. Info logging in this context means the logc_log… and the logc_logArray… functions. The default values of the log format is ERR for error messages and CLEAN for info messages.

```
logc_error_t
logc_setLogFormat(
      uint16_t ident,
      logc_errRecordType_t errForm,
      logc_logRecordTye_t logFrom
      )
```

Description: Set the log format for error and info logging.

Parameters:

ident	(I) identifier of the logger.
errForm	(I) format for error logging
logForm	(I) format for info logging

Returns:

LOG_ERR_OK for success.
LOG_ERR_PARAM for invalid format types.
LOG_ERR_NOT_FOUND for invalid logger id.

`logc_errRecordType_t` is used for error logging and is defined as an enumeration with the entries:

ERR
```
ERR : error message
```

ERR_TAG
```
ERR 0xYYYYYYYY : error message
```

ERR_TAG_TIMESTAMP
```
ERR 0xYYYYYYYY day mon dd hh:mm:ss YYYY : error message
```

ERR_TIMESTAMP_TAG
```
ERR day mon dd hh:mm:ss YYYY : error message
```

TIMESTAMP_ERR_TAG
```
day mon dd hh:mm:ss YYYY ERR 0xYYYYYYYY : error message
```

`logc_logRecordType_t` is used for info logging and is defined as an enumeration with the entries:

CLEAN
```
message
```

TIMESTAMP
```
day mon dd hh:mm:ss YYYY : message
```

Log Functions

Four base types of log functions are provided by nglogc, error logging, info logging, array logging and trace logging. Except trace logging, which is just a enter- and leave-function message, each type has a call with a log level as parameter and one call per log level. All functions with log levels in their names could be controlled (besides the log levels) with define switches. So it is intended to use these functions to be able to use this powerful feature and define at pre-processor time which objects should be linked and have an influence of the source code size. The functions with the log level LOG_FINE and LOG_FINEST are not linked per default and must be enabled with the define LOGC_ENABLE_LOW_LEVEL. See section define switches for detailed information.

Error Logging

```
logc_error_t
logc_logError(
      uint16_t ident,
      logc_logLevel_t level,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a logger with a given log level.

Parameters:

ident	(I) identifier of the logger.
level	(I) log level of this log statement.
err	(I) error to log.
formatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logErrorBasic(
      uint16_t ident,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a given logger, the log level is LOG_BASIC.

Parameters:

ident	(I) identifier of the logger.
err	(I) error to log.
fromatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logErrorWarning(
      uint16_t ident,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a given logger, the log level is LOG_WARNING.

Parameters:

ident	(I) identifier of the logger.
err	(I) error to log.
fromatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logErrorInfo(
      uint16_t ident,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a given logger, the log level is LOG_INFO.

Parameters:

ident	(I) identifier of the logger.
err	(I) error to log.
fromatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logErrorFine(
      uint16_t ident,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a given logger, the log level is LOG_FINE.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
err	(I) error to log.
fromatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logErrorFinest(
      uint16_t ident,
      logc_error_t err,
      const char* formatStr,
      ...
      )
```

Description: Prints error messages to a given logger, the log level is LOG_FINEST.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
err	(I) error to log.
fromatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

Info Logging

```
logc_error_t
logc_log(
      uint32_t ident,
      logc_logLevel_t level,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a logger with a given log level.

Parameters:

ident	(I) identifier of the logger.
level	(I) log level of this log statement.
formatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logBasic(
      uint32_t ident,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a given logger, the log level is LOG_BASIC.

Parameters:

ident	(I) identifier of the logger.
formatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logWarning(
      uint32_t ident,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a given logger, the log level is LOG_WARNING.

Parameters:

ident (I) identifier of the logger.
formatStr (I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logInfo(
      uint32_t ident,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a given logger, the log level is LOG_INFO.

Parameters:

ident (I) identifier of the logger.
formatStr (I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logFine(
      uint32_t ident,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a given logger, the log level is LOG_FINE.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
formatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logFinest(
      uint32_t ident,
      const char* formatStr,
      ...
      )
```

Description: Prints log messages to a given logger, the log level is LOG_FINEST.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
formatStr	(I) log message.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if formatStr is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

Array Logging

```
logc_error_t
logc_logArray(
      uint16_t ident,
      logc_logLevel_t level,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger and log level.

Parameters:

ident	(I) identifier of the logger.
level	(I) log level of this log statement.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logArrayBasic(
      uint16_t ident,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger, the log level is LOG_BASIC.

Parameters:

ident	(I) identifier of the logger.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logArrayWarning(
      uint16_t ident,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger, the log level is LOG_WARNING.

Parameters:

ident	(I) identifier of the logger.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logArrayInfo(
      uint16_t ident,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger, the log level is LOG_INFO.

Parameters:

ident	(I) identifier of the logger.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logArrayFine(
      uint16_t ident,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger, the log level is LOG_FINE. Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logArrayFinest(
      uint16_t ident,
      const char* desc,
      const uint8_t* array,
      size_t len
      )
```

Description: Prints data array described by a descriptor to a given logger, the log level is LOG_FINEST. Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.

Parameters:

ident	(I) identifier of the logger.
desc	(I) description of the data array.
array	(I) data array.
len	(I) size of data array.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if descriptor or array is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

Trace Logging

```
logc_error_t
logc_logEnter(
      uint16_t ident,
      const char* function
      )
```

Description: Prints entering of a function, log level is LOG_FINEST.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.
If the define HAVE_FLF is set the function name is ignored.

Parameters:

ident	(I) identifier of the logger.
function	(I) function name which is entered.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if function name is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

```
logc_error_t
logc_logLeave(
      uint16_t ident,
      const char* function
      )
```

Description: Prints leaving of a function, log level is LOG_FINEST.
Must be enabled with the define LOGC_ENABLE_LOW_LEVEL.
If the define HAVE_FLF is set the function name is ignored.

Parameters:

ident	(I) identifier of the logger.
function	(I) function name which is leaved.

Returns:

LOG_ERR_OK for success.
LOG_ERR_NULL if function name is NULL.
LOG_ERR_NOT_FOUND for invalid logger id.
LOG_ERR_LEVEL message is not printed because of the log level of the logger.

Define Switches

nglogc provides two types of define switches to control the behaviour of logging at pre-processor time.

The first type is for the FILE, LINE and FUNCTION macros to have more detailed logging output. These macros are available for each log function. With enabled macro the information `filename:linenumber fuction -` is put in front of the log message.

LOGC_HAVE_FLF
`Enables the FILE, LINE and FUNCTION macros for each log function.`

LOGC_HAVE_FLF_BASIC
`Enables the FILE, LINE and FUNCTION macros for all log functions dedicated to the log level LOG_BASIC.`

LOGC_HAVE_FLF_WARNING
`Enables the FILE, LINE and FUNCTION macros for all log functions dedicated to the log level LOG_WARNING.`

LOGC_HAVE_FLF_INFO
`Enables the FILE, LINE and FUNCTION macros for all log functions dedicated to the log level LOG_INFO.`

LOGC_HAVE_FLF_FINE
`Enables the FILE, LINE and FUNCTION macros for all log functions dedicated to the log level LOG_FINE.`

LOGC_HAVE_FLF_FINEST
`Enables the FILE, LINE and FUNCTION macros for all log functions dedicated to the log level LOG_FINEST.`

The second type of defines is to enable or disable complete log messages that they are not linked. With these defines log messages could completely removed without any changes in the source code. Only the functions with the log level in their names are affected, functions with a log level as parameter could not be undefined. It is recommended to use the first type of log functions to be able to use this feature. Also all log and error log functions with the log level LOG_BASIC could not be undefined. *Note* that the log and error log functions with the log level LOG_FINE and LOG_FINEST are not enabled per default. Because these functions are intended for debug logging and in this way no define is needed for productive software to disable these logs.

LOGC_DISABLE_WARN
`Disables all log and error log functions with the log level LOG_WARNING and LOG_INFO`

LOGC_DISABLE_WARN_ERRS
`Disables all error log functions with the log level LOG_WARNING and LOG_INFO`

LOGC_DISABLE_WARN_LOGS
Disables all log functions with the log level LOG_WARNING and LOG_INFO

LOGC_ENABLE_LOW_LEVEL
Enables all log and error log functions with the log level LOG_FINE and LOG_FINEST

LOGC_ENABLE_LOW_LEVEL_ERRS
Enables all error log functions with the log level LOG_FINE and LOG_FINEST

LOGC_ENABLE_LOW_LEVEL_LOGS
Enables all log functions with the log level LOG_FINE and LOG_FINEST

Examples

In the end my friend some examples, because this is the best way to explain functionality.

Log Level Example

```
#include "../nglogc/log.h"

/* loggers are identified by an uint16_t type */
#define MAIN_LOGGER          0x0001


static void
printErrorLogs(
      void
      )
{
   /* log an error message with the log level LOG_BASIC */
   logc_logErrorBasic(MAIN_LOGGER, 0,
      "This is a LOG_BASIC error message");
   /* log an error message with the log level LOG_WARNING */
   logc_logErrorWarning(MAIN_LOGGER, 0,
      "This is a LOG_WARNING error message");
   /* log an error message with the log level LOG_INFO */
   logc_logErrorInfo(MAIN_LOGGER, 0,
      "This is a LOG_INFO error message");
   /* log an error message with the log level LOG_FINE */
   logc_logErrorFine(MAIN_LOGGER, 0,
      "This is a LOG_FINE error message");
   /* log an error message with the log level LOG_FINEST */
   logc_logErrorFinest(MAIN_LOGGER, 0,
      "This is a LOG_FINEST error message");
}


int main(int argc, char *argv[])
{
   /* register a logger with the stdout publisher and
    LOG_SILENT log level */
   logc_registerLogger(MAIN_LOGGER, STDOUT, LOG_SILENT);

   /* change log level to LOG_BASIC */
   logc_changeLogLevel(MAIN_LOGGER, LOG_BASIC);
   logc_logBasic(MAIN_LOGGER, "\nlogLevel is set to LOG_BASIC");
   printErrorLogs();

   /* change log level to LOG_WARNING */
   logc_changeLogLevel(MAIN_LOGGER, LOG_WARNING);
   logc_logBasic(MAIN_LOGGER, "\nlogLevel is set to LOG_WARNING");
   printErrorLogs();

   /* change log level to LOG_INFO */
   logc_changeLogLevel(MAIN_LOGGER, LOG_INFO);
   logc_logBasic(MAIN_LOGGER, "\nlogLevel is set to LOG_INFO");
   printErrorLogs();

   /* change log level to LOG_FINE */
   logc_changeLogLevel(MAIN_LOGGER, LOG_FINE);
   logc_logBasic(MAIN_LOGGER, "\nlogLevel is set to LOG_FINE");
```

```
    printErrorLogs();

    /* change log level to LOG_FINEST */
    logc_changeLogLevel(MAIN_LOGGER, LOG_FINEST);
    logc_logBasic(MAIN_LOGGER, "\nlogLevel is set to LOG_FINEST");
    printErrorLogs();

    /* remove the logger */
    logc_removeLogger(MAIN_LOGGER);

    return 0;
}
```

Output

```
logLevel is set to LOG_BASIC
ERR : This is a LOG_BASIC error message

logLevel is set to LOG_WARNING
ERR : This is a LOG_BASIC error message
ERR : This is a LOG_WARNING error message

logLevel is set to LOG_INFO
ERR : This is a LOG_BASIC error message
ERR : This is a LOG_WARNING error message
ERR : This is a LOG_INFO error message

logLevel is set to LOG_FINE
ERR : This is a LOG_BASIC error message
ERR : This is a LOG_WARNING error message
ERR : This is a LOG_INFO error message
ERR : This is a LOG_FINE error message

logLevel is set to LOG_FINEST
ERR : This is a LOG_BASIC error message
ERR : This is a LOG_WARNING error message
ERR : This is a LOG_INFO error message
ERR : This is a LOG_FINE error message
ERR : This is a LOG_FINEST error message
```

Record Type Example

```
#include "../nglogc/log.h"

/* loggers are identified by an uint16_t type */
#define MAIN_LOGGER   0x0001

/* example error code */
#define ERR_TEST      0x00000001


int main(int argc, char *argv[])
{
   /* register a logger with the stdout publisher and
    LOG_BASIC log level. The errRecordType_t is set to ERR
    the logRecordType_t is set to CLEAN per default */
   logc_registerLogger(MAIN_LOGGER, STDOUT, LOG_BASIC);

   /* log an error message with the default error log format ERR */
   logc_logError(MAIN_LOGGER, LOG_BASIC, ERR_TEST,
         "Error message with ERR format");

   /* set error record type to ERR_TAG */
   logc_setLogFormat(MAIN_LOGGER, ERR_TAG, CLEAN);
   logc_logError(MAIN_LOGGER, LOG_BASIC, ERR_TEST,
         "Error message with ERR_TAG format");

   /* set error record type to ERR_TAG_TIMESTAMP */
   logc_setLogFormat(MAIN_LOGGER, ERR_TAG_TIMESTAMP, CLEAN);
   logc_logError(MAIN_LOGGER, LOG_BASIC, ERR_TEST,
         "Error message with ERR_TAG_TIMESTAMP format");

   /* set error record type to ERR_TIMESTAMP_TAG */
   logc_setLogFormat(MAIN_LOGGER, ERR_TIMESTAMP_TAG, CLEAN);
   logc_logError(MAIN_LOGGER, LOG_BASIC, ERR_TEST,
         "Error message with ERR_TIMESTAMP_TAG format");

   /* set error record type to TIMESTAMP_ERR_TAG */
   logc_setLogFormat(MAIN_LOGGER, TIMESTAMP_ERR_TAG, CLEAN);
   logc_logError(MAIN_LOGGER, LOG_BASIC, ERR_TEST,
         "Error message with TIMESTAMP_ERR_TAG format\n");


   /* log an log message with the default log format CLEAN */
   logc_log(MAIN_LOGGER, LOG_BASIC,
         "Log message with CLEAN format");

   /* set log record type to TIMESTAMP */
   logc_setLogFormat(MAIN_LOGGER, TIMESTAMP_ERR_TAG, TIMESTAMP);
   logc_log(MAIN_LOGGER, LOG_BASIC,
         "Log message with TIMESTAMP format");

   /* remove the logger */
   logc_removeLogger(MAIN_LOGGER);

   return 0;
}
```

Output

```
ERR : Error message with ERR format
ERR 0x00000001 : Error message with ERR_TAG format
ERR 0x00000001 Sat Mar  6 13:51:23 2010 : Error message with
ERR_TAG_TIMESTAMP format
ERR Sat Mar  6 13:51:23 2010 0x00000001 : Error message with
ERR_TIMESTAMP_TAG format
Sat Mar  6 13:51:23 2010 ERR 0x00000001 : Error message with
TIMESTAMP_ERR_TAG format

Log message with CLEAN format
Sat Mar  6 13:51:23 2010 : Log message with TIMESTAMP format
```

Define Switches Example

```
#include "../nglogc/log.h"

/* loggers are identified by an uint16_t type */
#define MAIN_LOGGER          0x0001


static void
runLogs(
      void
      )
{
   /* trace the function call */
   logc_logEnter(MAIN_LOGGER, "runLogs");

   /* log an error message with the log level LOG_BASIC */
   logc_logErrorBasic(MAIN_LOGGER, 0, "This is a LOG_BASIC error message");
   /* log an error message with the log level LOG_WARNING */
   logc_logErrorWarning(MAIN_LOGGER, 0,
      "This is a LOG_WARNING error message");
   /* log an error message with the log level LOG_INFO */
   logc_logErrorInfo(MAIN_LOGGER, 0, "This is a LOG_INFO error message");
   /* log an error message with the log level LOG_FINE */
   logc_logErrorFine(MAIN_LOGGER, 0, "This is a LOG_FINE error message");
   /* log an error message with the log level LOG_FINEST */
   logc_logErrorFinest(MAIN_LOGGER, 0,
      "This is a LOG_FINEST error message\n");

   /* log an log message with the log level LOG_BASIC */
   logc_logBasic(MAIN_LOGGER, "This is a LOG_BASIC log message");
   /* log an log message with the log level LOG_WARNING */
   logc_logWarning(MAIN_LOGGER, "This is a LOG_WARNING log message");
   /* log an log message with the log level LOG_INFO */
   logc_logInfo(MAIN_LOGGER, "This is a LOG_INFO log message");
   /* log an log message with the log level LOG_FINE */
   logc_logFine(MAIN_LOGGER, "This is a LOG_FINE log message");
   /* log an log message with the log level LOG_FINEST */
   logc_logFinest(MAIN_LOGGER, "This is a LOG_FINEST log message");
```

```
   /* trace the function call */
   logc_logLeave(MAIN_LOGGER, "runLogs");
}


int main(int argc, char *argv[])
{
   /* register a logger with the stdout publisher and
    LOG_BASIC log level */
   logc_registerLogger(MAIN_LOGGER, STDOUT, LOG_FINEST);

   /* change format for logs and error logs */
   logc_setLogFormat(MAIN_LOGGER, ERR_TAG, TIMESTAMP);

   runLogs();

   /* remove the logger */
   logc_removeLogger(MAIN_LOGGER);

   return 0;
}
```

Output

Built with LOGC_ENABLE_LOW_LEVEL and LOGC_HAVE_FLF defines:

```
Enter > defines.c:15 runLogs
ERR 0x00000000 : defines.c:18 runLogs - This is a LOG_BASIC error message
ERR 0x00000000 : defines.c:20 runLogs - This is a LOG_WARNING error message
ERR 0x00000000 : defines.c:22 runLogs - This is a LOG_INFO error message
ERR 0x00000000 : defines.c:24 runLogs - This is a LOG_FINE error message
ERR 0x00000000 : defines.c:26 runLogs - This is a LOG_FINEST error message

Sat Mar  6 14:25:21 2010 : defines.c:29 runLogs - This is a LOG_BASIC log
message
Sat Mar  6 14:25:21 2010 : defines.c:31 runLogs - This is a LOG_WARNING log
message
Sat Mar  6 14:25:21 2010 : defines.c:33 runLogs - This is a LOG_INFO log
message
Sat Mar  6 14:25:21 2010 : defines.c:35 runLogs - This is a LOG_FINE log
message
Sat Mar  6 14:25:21 2010 : defines.c:37 runLogs - This is a LOG_FINEST log
message
Leave < defines.c:40 runLogs
```

Built with LOGC_ENABLE_LOW_LEVEL_ERRS define:

```
ERR 0x00000000 : This is a LOG_BASIC error message
ERR 0x00000000 : This is a LOG_WARNING error message
ERR 0x00000000 : This is a LOG_INFO error message
ERR 0x00000000 : This is a LOG_FINE error message
ERR 0x00000000 : This is a LOG_FINEST error message

Sat Mar  6 14:22:19 2010 : This is a LOG_BASIC log message
Sat Mar  6 14:22:19 2010 : This is a LOG_WARNING log message
Sat Mar  6 14:22:19 2010 : This is a LOG_INFO log message
```

Built without defines:

```
ERR 0x00000000 : This is a LOG_BASIC error message
ERR 0x00000000 : This is a LOG_WARNING error message
ERR 0x00000000 : This is a LOG_INFO error message

Sat Mar  6 14:23:20 2010 : This is a LOG_BASIC log message
Sat Mar  6 14:23:20 2010 : This is a LOG_WARNING log message
Sat Mar  6 14:23:20 2010 : This is a LOG_INFO log message
```

Built with LOGC_DISABLE_WARN_LOGS define:

```
ERR 0x00000000 : This is a LOG_BASIC error message
ERR 0x00000000 : This is a LOG_WARNING error message
ERR 0x00000000 : This is a LOG_INFO error message
Sat Mar  6 14:24:07 2010 : This is a LOG_BASIC log message
```

Built with LOGC_DISABLE_WARN define:

```
ERR 0x00000000 : This is a LOG_BASIC error message
Sat Mar  6 14:23:52 2010 : This is a LOG_BASIC log message
```