jsonToBatProject

Generated on Mon Feb 26 2024 15:43:45 for jsonToBatProject by Doxygen 1.9.8

Mon Feb 26 2024 15:43:45

Chapter 1

README

`Doxygen Documentation`

`Sonar Cloud`

1.1 README

1.1.1 Current workflows:

- build
  - build and test the application on:
    - windows with cl
    - ubunut with g++
    - ubuntu with clang++
- CodeQl
  - Code security
- Doxygen Action
  - Generate Doxygen documentation
  - Deploys generated documentation to gh-pages
- Microsoft C++ Code Analysis
- pages-build-deployment
- SonarCloud
  - Static code analysis *For Scanning Alerts* -> *Security*

1.1.2 Regarding coding style (?):

- no classes in global namespace
- no "using NAMESPACE"
- 4 space indenting
- ? *setup astyle options?*

1.1.3 Git (?):

- no direct commits onto main (only via pull-requests)

Chapter 2

Todo List

Namespace WIP

Github

- "Dev-Ops"
- Doxygen settings
- Template-Comment
- Template-Header-Comment

Global WIP::exampleEasyLogging ()

Configure easylogging properly

- outsource easylogging config
  - e.g. startup class?

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 el Namespace Reference

Easylogging++ entry namespace.

Namespaces

- namespace base

    Namespace containing base/internal functionality used by Easylogging++.

Data Structures

- class Callback
- class Configuration

    Represents single configuration that has representing level, configuration type and a string based value.
- class Configurations

    Thread-safe Configuration repository.
- struct ConfigurationStringToTypeItem
- class ConfigurationTypeHelper

    Static class that contains helper functions for el::ConfigurationType.
- class CustomFormatSpecifier

    User-provided custom format specifier.
- class Helpers

    Static helpers for developers.
- class LevelHelper

    Static class that contains helper functions for el::Level.
- class LogBuilder
- class LogDispatchCallback
- class LogDispatchData
- class Loggable

    Base of Easylogging++ friendly class.
- class Logger

    Represents a logger holding ID and configurations we need to write logs.
- class LoggerRegistrationCallback
- class Loggers

Static helpers to deal with loggers and their configurations.
- class LogMessage
- class PerformanceTrackingCallback
- struct StringToLevelItem
- class SysLogInitializer

  Initializes syslog with process ID, options and facility. calls closelog() on d'tor.
- class VersionInfo

Typedefs

- typedef std::function< void(const char *, std::size_t)> PreRollOutCallback
- typedef std::function< std::string(const LogMessage *)> FormatSpecifierValueResolver

  Resolving function for format specifier.
- typedef std::shared_ptr< LogBuilder > LogBuilderPtr

Enumerations

- enum class Level : base::type::EnumType {
  Global = 1 , Trace = 2 , Debug = 4 , Fatal = 8 ,
  Error = 16 , Warning = 32 , Verbose = 64 , Info = 128 ,
  Unknown = 1010 }

  Represents enumeration for severity level used to determine level of logging.
- enum class ConfigurationType : base::type::EnumType {
  Enabled = 1 , ToFile = 2 , ToStandardOutput = 4 , Format = 8 ,
  Filename = 16 , SubsecondPrecision = 32 , MillisecondsWidth = SubsecondPrecision , PerformanceTracking = 64 ,
  MaxLogFileSize = 128 , LogFlushThreshold = 256 , Unknown = 1010 }

  Represents enumeration of ConfigurationType used to configure or access certain aspect of logging.
- enum class LoggingFlag : base::type::EnumType {
  NewLineForContainer = 1 , AllowVerboseIfModuleNotSpecified = 2 , LogDetailedCrashReason = 4 , DisableApplicationAbortOnFatalLog = 8 ,
  ImmediateFlush = 16 , StrictLogFileSizeCheck = 32 , ColoredTerminalOutput = 64 , MultiLoggerSupport = 128 ,
  DisablePerformanceTrackingCheckpointComparison = 256 , DisableVModules = 512 , DisableVModulesExtensions = 1024 , HierarchicalLogging = 2048 ,
  CreateLoggerAutomatically = 4096 , AutoSpacing = 8192 , FixedTimeFormat = 16384 , IgnoreSigInt = 32768
  }

  Flags used while writing logs. This flags are set by user.

Variables

- static struct StringToLevelItem stringToLevelMap [ ]
- static struct ConfigurationStringToTypeItem configStringToTypeMap [ ]
- base::debug::CrashHandler elCrashHandler

7.1.1 Detailed Description

Easylogging++ entry namespace.

7.1.2 Typedef Documentation

7.1.2.1 FormatSpecifierValueResolver

```
typedef std::function<std::string(const LogMessage*)> el::FormatSpecifierValueResolver
```

Resolving function for format specifier.

Definition at line 1642 of file easylogging++.h.

7.1.2.2 LogBuilderPtr

```
typedef std::shared_ptr<LogBuilder> el::LogBuilderPtr
```

Definition at line 2209 of file easylogging++.h.

7.1.2.3 PreRollOutCallback

```
typedef std::function<void(const char*, std::size_t)> el::PreRollOutCallback
```

Definition at line 808 of file easylogging++.h.

7.1.3 Enumeration Type Documentation

7.1.3.1 ConfigurationType

```
enum class el::ConfigurationType : base::type::EnumType [strong]
```

Represents enumeration of ConfigurationType used to configure or access certain aspect of logging.

Enumerator

Enabled	Determines whether or not corresponding level and logger of logging is enabled You may disable all logs by using el::Level::Global.
ToFile	Whether or not to write corresponding log to log file.
ToStandardOutput	Whether or not to write corresponding level and logger log to standard output. By standard output meaning termnal, command prompt etc.
Format	Determines format of logging corresponding level and logger.
Filename	Determines log file (full path) to write logs to for corresponding level and logger.
SubsecondPrecision	Specifies precision of the subsecond part. It should be within range (1-6).
MillisecondsWidth	Alias of SubsecondPrecision (for backward compatibility)
PerformanceTracking	Determines whether or not performance tracking is enabled. @detail This does not depend on logger or level. Performance tracking always uses 'performance' logger
MaxLogFileSize	Specifies log file max size. @detail If file size of corresponding log file (for corresponding level) is >= specified size, log file will be truncated and re-initiated.
LogFlushThreshold	Specifies number of log entries to hold until we flush pending log data.
Unknown	Represents unknown configuration.

Definition at line 633 of file easylogging++.h.

7.1.3.2 Level

```
enum class el::Level :  base::type::EnumType  [strong]
```

Represents enumeration for severity level used to determine level of logging.

@detail With Easylogging++, developers may disable or enable any level regardless of what the severity is. Or they can choose to log using hierarchical logging flag

Enumerator

Global	Generic level that represents all the 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.
Fatal	Severe error information that will presumably abort application.
Error	Information representing errors in application but application will keep running.
Warning	Useful when application has potentially harmful situations.
Verbose	Information that can be highly useful and vary with verbose logging level.
Info	Mainly useful to represent current progress of application.
Unknown	Represents unknown level.

Definition at line 573 of file easylogging++.h.

7.1.3.3 LoggingFlag

```
enum class el::LoggingFlag :  base::type::EnumType  [strong]
```

Flags used while writing logs. This flags are set by user.

Enumerator

NewLineForContainer	Makes sure we have new line for each container log entry.
AllowVerboseIfModuleNotSpecified	Makes sure if -vmodule is used and does not specifies a module, then verbose logging is allowed via that module.
LogDetailedCrashReason	When handling crashes by default, detailed crash reason will be logged as well.
DisableApplicationAbortOnFatalLog	Allows to disable application abortion when logged using FATAL level.
ImmediateFlush	Flushes log with every log-entry (performance sensitive) - Disabled by default.
StrictLogFileSizeCheck	Enables strict file rolling.
ColoredTerminalOutput	Make terminal output colorful for supported terminals.
MultiLoggerSupport	Supports use of multiple logging in same macro, e.g, CLOG(INFO, "default", "network")
DisablePerformanceTrackingCheckpointComparison	Disables comparing performance tracker's checkpoints.

Enumerator

DisableVModules	Disable VModules.
DisableVModulesExtensions	Disable VModules extensions.
HierarchicalLogging	Enables hierarchical logging.
CreateLoggerAutomatically	Creates logger automatically when not available.
AutoSpacing	Adds spaces b/w logs that separated by left-shift operator.
FixedTimeFormat	Preserves time format and does not convert it to sec, hour etc (performance tracking only)
IgnoreSigInt	

Definition at line 694 of file easylogging++.h.

7.1.4 Variable Documentation

7.1.4.1 configStringToTypeMap

```
struct ConfigurationStringToTypeItem el::configStringToTypeMap[ ]  [static]
```

Initial value:

```
= {
  { "enabled", ConfigurationType::Enabled },
  { "to_file", ConfigurationType::ToFile },
  { "to_standard_output", ConfigurationType::ToStandardOutput },
  { "format", ConfigurationType::Format },
  { "filename", ConfigurationType::Filename },
  { "subsecond_precision", ConfigurationType::SubsecondPrecision },
  { "milliseconds_width", ConfigurationType::MillisecondsWidth },
  { "performance_tracking", ConfigurationType::PerformanceTracking },
  { "max_log_file_size", ConfigurationType::MaxLogFileSize },
  { "log_flush_threshold", ConfigurationType::LogFlushThreshold },
}
```

Definition at line 201 of file easylogging++.cc.

7.1.4.2 elCrashHandler

```
base::debug::CrashHandler el::elCrashHandler  [extern]
```

7.1.4.3 stringToLevelMap

```
struct StringToLevelItem el::stringToLevelMap[ ]  [static]
```

Initial value:

```
= {
  { "global", Level::Global },
  { "debug", Level::Debug },
  { "info", Level::Info },
  { "warning", Level::Warning },
  { "error", Level::Error },
  { "fatal", Level::Fatal },
  { "verbose", Level::Verbose },
  { "trace", Level::Trace }
}
```

Definition at line 150 of file easylogging++.cc.

7.2 el::base Namespace Reference

Namespace containing base/internal functionality used by Easylogging++.

Namespaces

- namespace consts

    Namespace containing constants used internally.
- namespace debug

    Contains some internal debugging tools like crash handler and stack tracer.
- namespace threading
- namespace type

    Data types used by Easylogging++.
- namespace utils

    Namespace containing utility functions/static classes used internally.

Data Structures

- class DefaultLogBuilder
- class DefaultLogDispatchCallback
- class HitCounter

    Class that keeps record of current line hit for occasional logging.
- class LogDispatcher

    Dispatches log messages.
- class LogFormat

    Represents log format containing flags and date format. This is used internally to start initial log.
- class MessageBuilder
- class NoCopy

    Internal helper class that prevent copy constructor for class.
- class NullWriter

    Writes nothing - Used when certain log is disabled.
- class PErrorWriter
- class RegisteredHitCounters

    Repository for hit counters used across the application.
- class RegisteredLoggers

    Loggers repository.
- class StaticClass

    Internal helper class that makes all default constructors private.
- class Storage

    Easylogging++ management storage.
- class SubsecondPrecision

    A subsecond precision class containing actual width and offset of the subsecond part.
- class TypedConfigurations

    Configurations with data types.
- class VRegistry

    Represents registries for verbose logging.
- class Writer

    Main entry point of each logging.

Typedefs

- typedef SubsecondPrecision MillisecondsWidth

  Type alias of SubsecondPrecision.
- typedef std::shared_ptr< base::type::fstream_t > FileStreamPtr
- typedef std::unordered_map< std::string, FileStreamPtr > LogStreamsReferenceMap
- typedef std::shared_ptr< base::LogStreamsReferenceMap > LogStreamsReferenceMapPtr

Enumerations

- enum class TimestampUnit : base::type::EnumType {
  Microsecond = 0 , Millisecond = 1 , Second = 2 , Minute = 3 ,
  Hour = 4 , Day = 5 }

  Enum to represent timestamp unit.
- enum class FormatFlags : base::type::EnumType {
  DateTime = 1 << 1 , LoggerId = 1 << 2 , File = 1 << 3 , Line = 1 << 4 ,
  Location = 1 << 5 , Function = 1 << 6 , User = 1 << 7 , Host = 1 << 8 ,
  LogMessage = 1 << 9 , VerboseLevel = 1 << 10 , AppName = 1 << 11 , ThreadId = 1 << 12 ,
  Level = 1 << 13 , FileBase = 1 << 14 , LevelShort = 1 << 15 }

  Format flags used to determine specifiers that are active for performance improvements.
- enum class DispatchAction : base::type::EnumType { None = 1 , NormalLog = 2 , SysLog = 4 }

  Action to be taken for dispatching.

Functions

- static void defaultPreRollOutCallback (const char ∗, std::size_t)

Variables

- ELPP_EXPORT base::type::StoragePointer elStorage

7.2.1 Detailed Description

Namespace containing base/internal functionality used by Easylogging++.

7.2.2 Typedef Documentation

7.2.2.1 FileStreamPtr

```
typedef std::shared_ptr<base::type::fstream_t> el::base::FileStreamPtr  [private]
```

Definition at line 1895 of file easylogging++.h.

7.2.2.2 LogStreamsReferenceMap

```
typedef std::unordered_map<std::string, FileStreamPtr> el::base::LogStreamsReferenceMap  [private]
```

Definition at line 1896 of file easylogging++.h.

7.2.2.3 LogStreamsReferenceMapPtr

```
typedef std::shared_ptr<base::LogStreamsReferenceMap> el::base::LogStreamsReferenceMapPtr
[private]
```

Definition at line 1897 of file easylogging++.h.

7.2.2.4 MillisecondsWidth

```
typedef SubsecondPrecision el::base::MillisecondsWidth
```

Type alias of SubsecondPrecision.

Definition at line 851 of file easylogging++.h.

7.2.3 Enumeration Type Documentation

7.2.3.1 DispatchAction

```
enum class el::base::DispatchAction :  base::type::EnumType  [strong], [private]
```

Action to be taken for dispatching.

Enumerator

None	
NormalLog	
SysLog	

Definition at line 2139 of file easylogging++.h.

7.2.3.2 FormatFlags

```
enum class el::base::FormatFlags :  base::type::EnumType  [strong]
```

Format flags used to determine specifiers that are active for performance improvements.

Enumerator

DateTime	
LoggerId	
File	
Line	
Location	
Function	
User	
Host	
LogMessage	

Enumerator

VerboseLevel	
AppName	
ThreadId	
Level	
FileBase	
LevelShort	

Definition at line 816 of file easylogging++.h.

7.2.3.3 TimestampUnit

```
enum class el::base::TimestampUnit : base::type::EnumType [strong]
```

Enum to represent timestamp unit.

Enumerator

Microsecond	
Millisecond	
Second	
Minute	
Hour	
Day	

Definition at line 812 of file easylogging++.h.

7.2.4 Function Documentation

7.2.4.1 defaultPreRollOutCallback()

```
static void el::base::defaultPreRollOutCallback (
             const char * ,
             std::size_t ) [inline], [static]
```

Definition at line 810 of file easylogging++.h.

7.2.5 Variable Documentation

7.2.5.1 elStorage

```
ELPP_EXPORT base::type::StoragePointer el::base::elStorage [extern], [private]
```

7.3 el::base::consts Namespace Reference

Namespace containing constants used internally.

Variables

- static const base::type::char_t ∗ kInfoLevelLogValue = ELPP_LITERAL("INFO")
- static const base::type::char_t ∗ kDebugLevelLogValue = ELPP_LITERAL("DEBUG")
- static const base::type::char_t ∗ kWarningLevelLogValue = ELPP_LITERAL("WARNING")
- static const base::type::char_t ∗ kErrorLevelLogValue = ELPP_LITERAL("ERROR")
- static const base::type::char_t ∗ kFatalLevelLogValue = ELPP_LITERAL("FATAL")
- static const base::type::char_t ∗ kVerboseLevelLogValue
- static const base::type::char_t ∗ kTraceLevelLogValue = ELPP_LITERAL("TRACE")
- static const base::type::char_t ∗ kInfoLevelShortLogValue = ELPP_LITERAL("I")
- static const base::type::char_t ∗ kDebugLevelShortLogValue = ELPP_LITERAL("D")
- static const base::type::char_t ∗ kWarningLevelShortLogValue = ELPP_LITERAL("W")
- static const base::type::char_t ∗ kErrorLevelShortLogValue = ELPP_LITERAL("E")
- static const base::type::char_t ∗ kFatalLevelShortLogValue = ELPP_LITERAL("F")
- static const base::type::char_t ∗ kVerboseLevelShortLogValue = ELPP_LITERAL("V")
- static const base::type::char_t ∗ kTraceLevelShortLogValue = ELPP_LITERAL("T")
- static const base::type::char_t ∗ kAppNameFormatSpecifier = ELPP_LITERAL("%app")
- static const base::type::char_t ∗ kLoggerIdFormatSpecifier = ELPP_LITERAL("%logger")
- static const base::type::char_t ∗ kThreadIdFormatSpecifier = ELPP_LITERAL("%thread")
- static const base::type::char_t ∗ kSeverityLevelFormatSpecifier = ELPP_LITERAL("%level")
- static const base::type::char_t ∗ kSeverityLevelShortFormatSpecifier = ELPP_LITERAL("%levshort")
- static const base::type::char_t ∗ kDateTimeFormatSpecifier = ELPP_LITERAL("%datetime")
- static const base::type::char_t ∗ kLogFileFormatSpecifier = ELPP_LITERAL("%file")
- static const base::type::char_t ∗ kLogFileBaseFormatSpecifier = ELPP_LITERAL("%fbase")
- static const base::type::char_t ∗ kLogLineFormatSpecifier = ELPP_LITERAL("%line")
- static const base::type::char_t ∗ kLogLocationFormatSpecifier = ELPP_LITERAL("%loc")
- static const base::type::char_t ∗ kLogFunctionFormatSpecifier = ELPP_LITERAL("%func")
- static const base::type::char_t ∗ kCurrentUserFormatSpecifier = ELPP_LITERAL("%user")
- static const base::type::char_t ∗ kCurrentHostFormatSpecifier = ELPP_LITERAL("%host")
- static const base::type::char_t ∗ kMessageFormatSpecifier = ELPP_LITERAL("%msg")
- static const base::type::char_t ∗ kVerboseLevelFormatSpecifier = ELPP_LITERAL("%vlevel")
- static const char ∗ kDateTimeFormatSpecifierForFilename = "%datetime"
- static const char ∗ kDays [7] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" }
- static const char ∗ kDaysAbbrev [7] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" }
- static const char ∗ kMonths [12]
- static const char ∗ kMonthsAbbrev [12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" }
- static const char ∗ kDefaultDateTimeFormat = "%Y-%M-%d %H:%m:%s,%g"
- static const char ∗ kDefaultDateTimeFormatInFilename = "%Y-%M-%d_%H-%m"
- static const int kYearBase = 1900
- static const char ∗ kAm = "AM"
- static const char ∗ kPm = "PM"
- static const char ∗ kNullPointer = "nullptr"
- static const base::type::VerboseLevel kMaxVerboseLevel = 9
- static const char ∗ kUnknownUser = "unknown-user"
- static const char ∗ kUnknownHost = "unknown-host"
- static const char ∗ kDefaultLogFile = "myeasylog.log"
- static const char ∗ kDefaultLogFileParam = "--default-log-file"
- static const char ∗ kValidLoggerIdSymbols
- static const char ∗ kConfigurationComment = "##"
- static const char ∗ kConfigurationLevel = "∗"
- static const char ∗ kConfigurationLoggerId = "--"
- static const char kFormatSpecifierCharValue = 'v'
- static const char kFormatSpecifierChar = '%'

- static const unsigned int kMaxLogPerCounter = 100000
- static const unsigned int kMaxLogPerContainer = 100
- static const unsigned int kDefaultSubsecondPrecision = 3
- static const char ∗ kDefaultLoggerId = "default"
- static const char ∗ kFilePathSeparator = "/"
- static const std::size_t kSourceFilenameMaxLength = 100
- static const std::size_t kSourceLineMaxLength = 10
- static const Level kPerformanceTrackerDefaultLevel = Level::Info
- struct {
  double el::base::consts::value
  const base::type::char_t ∗ el::base::consts::unit
  } kTimeFormats [ ]
- static const int kTimeFormatsCount = sizeof(kTimeFormats) / sizeof(kTimeFormats[0])
- struct {
  int el::base::consts::numb
  const char ∗ el::base::consts::name
  const char ∗ el::base::consts::brief
  const char ∗ el::base::consts::detail
  } kCrashSignals [ ]
- static const int kCrashSignalsCount = sizeof(kCrashSignals) / sizeof(kCrashSignals[0])

7.3.1 Detailed Description

Namespace containing constants used internally.

7.3.2 Variable Documentation

7.3.2.1 brief

```
const char* el::base::consts::brief
```

Definition at line 780 of file easylogging++.h.

7.3.2.2 detail

```
const char* el::base::consts::detail
```

Definition at line 781 of file easylogging++.h.

7.3.2.3 kAm

```
const char* el::base::consts::kAm = "AM"  [static]
```

Definition at line 70 of file easylogging++.cc.

7.3.2.4 kAppNameFormatSpecifier

```
const base::type::char_t* el::base::consts::kAppNameFormatSpecifier = ELPP_LITERAL("%app")
[static]
```

Definition at line 44 of file easylogging++.cc.

7.3.2.5 kConfigurationComment

```
const char* el::base::consts::kConfigurationComment = "##"  [static]
```

Definition at line 105 of file easylogging++.cc.

7.3.2.6 kConfigurationLevel

```
const char* el::base::consts::kConfigurationLevel = "*"  [static]
```

Definition at line 106 of file easylogging++.cc.

7.3.2.7 kConfigurationLoggerId

```
const char* el::base::consts::kConfigurationLoggerId = "--"  [static]
```

Definition at line 107 of file easylogging++.cc.

7.3.2.8 [struct]

```
const struct { ...  } el::base::consts::kCrashSignals[]
```

Initial value:

```
= {

  {
    SIGABRT, "SIGABRT", "Abnormal termination",
    "Program was abnormally terminated."
  },
  {
    SIGFPE, "SIGFPE", "Erroneous arithmetic operation",
    "Arithmetic operation issue such as division by zero or operation resulting in overflow."
  },
  {
    SIGILL, "SIGILL", "Illegal instruction",
    "Generally due to a corruption in the code or to an attempt to execute data."
  },
  {
    SIGSEGV, "SIGSEGV", "Invalid access to memory",
    "Program is trying to read an invalid (unallocated, deleted or corrupted) or inaccessible memory."
  },
  {
    SIGINT, "SIGINT", "Interactive attention signal",
    "Interruption generated (generally) by user or operating system."
  },
}
```

7.3.2.9 kCrashSignalsCount

```
const int el::base::consts::kCrashSignalsCount = sizeof(kCrashSignals) / sizeof(kCrashSignals[0])
[static]
```

Definition at line 805 of file easylogging++.h.

7.3.2.10 kCurrentHostFormatSpecifier

```
const base::type::char_t* el::base::consts::kCurrentHostFormatSpecifier = ELPP_LITERAL("%host")
[static]
```

Definition at line 56 of file easylogging++.cc.

7.3.2.11 kCurrentUserFormatSpecifier

```
const base::type::char_t* el::base::consts::kCurrentUserFormatSpecifier = ELPP_LITERAL("%user")
[static]
```

Definition at line 55 of file easylogging++.cc.

7.3.2.12 kDateTimeFormatSpecifier

```
const base::type::char_t* el::base::consts::kDateTimeFormatSpecifier = ELPP_LITERAL("%datetime")
[static]
```

Definition at line 49 of file easylogging++.cc.

7.3.2.13 kDateTimeFormatSpecifierForFilename

```
const char* el::base::consts::kDateTimeFormatSpecifierForFilename = "%datetime"  [static]
```

Definition at line 59 of file easylogging++.cc.

7.3.2.14 kDays

```
const char* el::base::consts::kDays[7] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday" }  [static]
```

Definition at line 61 of file easylogging++.cc.

7.3.2.15 kDaysAbbrev

```
const char* el::base::consts::kDaysAbbrev[7] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" }  [static]
```

Definition at line 62 of file easylogging++.cc.

7.3.2.16 kDebugLevelLogValue

```
const base::type::char_t* el::base::consts::kDebugLevelLogValue = ELPP_LITERAL("DEBUG")  [static]
```

Definition at line 29 of file easylogging++.cc.

7.3.2.17 kDebugLevelShortLogValue

```
const base::type::char_t* el::base::consts::kDebugLevelShortLogValue = ELPP_LITERAL("D")  [static]
```

Definition at line 37 of file easylogging++.cc.

7.3.2.18 kDefaultDateTimeFormat

```
const char* el::base::consts::kDefaultDateTimeFormat = "%Y-%M-%d %H:%m:%s,%g"  [static]
```

Definition at line 67 of file easylogging++.cc.

7.3.2.19 kDefaultDateTimeFormatInFilename

```
const char* el::base::consts::kDefaultDateTimeFormatInFilename = "%Y-%M-%d_%H-%m"  [static]
```

Definition at line 68 of file easylogging++.cc.

7.3.2.20 kDefaultLogFile

```
const char* el::base::consts::kDefaultLogFile = "myeasylog.log"  [static]
```

Definition at line 93 of file easylogging++.cc.

7.3.2.21 kDefaultLogFileParam

```
const char* el::base::consts::kDefaultLogFileParam = "--default-log-file"  [static]
```

Definition at line 98 of file easylogging++.cc.

7.3.2.22 kDefaultLoggerId

```
const char* el::base::consts::kDefaultLoggerId = "default"  [static]
```

Definition at line 741 of file easylogging++.h.

7.3.2.23 kDefaultSubsecondPrecision

```
const unsigned int el::base::consts::kDefaultSubsecondPrecision = 3  [static]
```

Definition at line 736 of file easylogging++.h.

7.3.2.24 kErrorLevelLogValue

```
const base::type::char_t* el::base::consts::kErrorLevelLogValue = ELPP_LITERAL("ERROR")  [static]
```

Definition at line 31 of file easylogging++.cc.

7.3.2.25 kErrorLevelShortLogValue

```
const base::type::char_t* el::base::consts::kErrorLevelShortLogValue = ELPP_LITERAL("E")  [static]
```

Definition at line 39 of file easylogging++.cc.

7.3.2.26 kFatalLevelLogValue

```
const base::type::char_t* el::base::consts::kFatalLevelLogValue = ELPP_LITERAL("FATAL")  [static]
```

Definition at line 32 of file easylogging++.cc.

7.3.2.27 kFatalLevelShortLogValue

```
const base::type::char_t* el::base::consts::kFatalLevelShortLogValue = ELPP_LITERAL("F")  [static]
```

Definition at line 40 of file easylogging++.cc.

7.3.2.28 kFilePathSeparator

```
const char* el::base::consts::kFilePathSeparator = "/"  [static]
```

Definition at line 759 of file easylogging++.h.

7.3.2.29 kFormatSpecifierChar

```
const char el::base::consts::kFormatSpecifierChar = '%'  [static]
```

Definition at line 733 of file easylogging++.h.

7.3.2.30 kFormatSpecifierCharValue

```
const char el::base::consts::kFormatSpecifierCharValue = 'v'  [static]
```

Definition at line 732 of file easylogging++.h.

7.3.2.31 kInfoLevelLogValue

```
const base::type::char_t* el::base::consts::kInfoLevelLogValue = ELPP_LITERAL("INFO")  [static]
```

Definition at line 28 of file easylogging++.cc.

7.3.2.32 kInfoLevelShortLogValue

```
const base::type::char_t* el::base::consts::kInfoLevelShortLogValue = ELPP_LITERAL("I")  [static]
```

Definition at line 36 of file easylogging++.cc.

7.3.2.33 kLogFileBaseFormatSpecifier

```
const base::type::char_t* el::base::consts::kLogFileBaseFormatSpecifier = ELPP_LITERAL("%fbase")
[static]
```

Definition at line 51 of file easylogging++.cc.

7.3.2.34 kLogFileFormatSpecifier

```
const base::type::char_t* el::base::consts::kLogFileFormatSpecifier = ELPP_LITERAL("%file")
[static]
```

Definition at line 50 of file easylogging++.cc.

7.3.2.35 kLogFunctionFormatSpecifier

```
const base::type::char_t* el::base::consts::kLogFunctionFormatSpecifier = ELPP_LITERAL("%func")
[static]
```

Definition at line 54 of file easylogging++.cc.

7.3.2.36 kLoggerIdFormatSpecifier

```
const base::type::char_t* el::base::consts::kLoggerIdFormatSpecifier = ELPP_LITERAL("%logger")
[static]
```

Definition at line 45 of file easylogging++.cc.

7.3.2.37 kLogLineFormatSpecifier

```
const base::type::char_t* el::base::consts::kLogLineFormatSpecifier = ELPP_LITERAL("%line")
[static]
```

Definition at line 52 of file easylogging++.cc.

7.3.2.38 kLogLocationFormatSpecifier

```
const base::type::char_t* el::base::consts::kLogLocationFormatSpecifier = ELPP_LITERAL("%loc")
[static]
```

Definition at line 53 of file easylogging++.cc.

7.3.2.39 kMaxLogPerContainer

```
const unsigned int el::base::consts::kMaxLogPerContainer = 100  [static]
```

Definition at line 735 of file easylogging++.h.

7.3.2.40 kMaxLogPerCounter

```
const unsigned int el::base::consts::kMaxLogPerCounter = 100000  [static]
```

Definition at line 734 of file easylogging++.h.

7.3.2.41 kMaxVerboseLevel

```
const base::type::VerboseLevel el::base::consts::kMaxVerboseLevel = 9  [static]
```

Definition at line 77 of file easylogging++.cc.

7.3.2.42 kMessageFormatSpecifier

```
const base::type::char_t* el::base::consts::kMessageFormatSpecifier = ELPP_LITERAL("%msg")
[static]
```

Definition at line 57 of file easylogging++.cc.

7.3.2.43 kMonths

```
const char* el::base::consts::kMonths[12]  [static]
```

Initial value:

```
=      { "January", "February", "March", "April", "May", "June", "July", "August",
                                                            "September", "October", "November", "December"
                                                          }
```

Definition at line 63 of file easylogging++.cc.

7.3.2.44 kMonthsAbbrev

```
const char* el::base::consts::kMonthsAbbrev[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec" }  [static]
```

Definition at line 66 of file easylogging++.cc.

7.3.2.45 kNullPointer

```
const char* el::base::consts::kNullPointer = "nullptr"  [static]
```

Definition at line 74 of file easylogging++.cc.

7.3.2.46 kPerformanceTrackerDefaultLevel

```
const Level el::base::consts::kPerformanceTrackerDefaultLevel = Level::Info  [static]
```

Definition at line 764 of file easylogging++.h.

7.3.2.47 kPm

```
const char* el::base::consts::kPm = "PM"  [static]
```

Definition at line 71 of file easylogging++.cc.

7.3.2.48 kSeverityLevelFormatSpecifier

```
const base::type::char_t* el::base::consts::kSeverityLevelFormatSpecifier = ELPP_LITERAL("%level")
[static]
```

Definition at line 47 of file easylogging++.cc.

7.3.2.49 kSeverityLevelShortFormatSpecifier

```
const base::type::char_t* el::base::consts::kSeverityLevelShortFormatSpecifier = ELPP_LITERAL("%levshort")
[static]
```

Definition at line 48 of file easylogging++.cc.

7.3.2.50 kSourceFilenameMaxLength

```
const std::size_t el::base::consts::kSourceFilenameMaxLength = 100  [static]
```

Definition at line 762 of file easylogging++.h.

7.3.2.51 kSourceLineMaxLength

```
const std::size_t el::base::consts::kSourceLineMaxLength = 10  [static]
```

Definition at line 763 of file easylogging++.h.

7.3.2.52 kThreadIdFormatSpecifier

```
const base::type::char_t* el::base::consts::kThreadIdFormatSpecifier = ELPP_LITERAL("%thread")
[static]
```

Definition at line 46 of file easylogging++.cc.

7.3.2.53 [struct]

```
const struct { ... } el::base::consts::kTimeFormats[ ]
```

Initial value:

```
= {
  { 1000.0f, ELPP_LITERAL("us") },
  { 1000.0f, ELPP_LITERAL("ms") },
  { 60.0f, ELPP_LITERAL("seconds") },
  { 60.0f, ELPP_LITERAL("minutes") },
  { 24.0f, ELPP_LITERAL("hours") },
  { 7.0f, ELPP_LITERAL("days") }
}
```

7.3.2.54 kTimeFormatsCount

```
const int el::base::consts::kTimeFormatsCount = sizeof(kTimeFormats) / sizeof(kTimeFormats[0])
[static]
```

Definition at line 776 of file easylogging++.h.

7.3.2.55 kTraceLevelLogValue

```
const base::type::char_t* el::base::consts::kTraceLevelLogValue = ELPP_LITERAL("TRACE")  [static]
```

Definition at line 35 of file easylogging++.cc.

7.3.2.56 kTraceLevelShortLogValue

```
const base::type::char_t* el::base::consts::kTraceLevelShortLogValue = ELPP_LITERAL("T")  [static]
```

Definition at line 42 of file easylogging++.cc.

7.3.2.57 kUnknownHost

```
const char* el::base::consts::kUnknownHost = "unknown-host"  [static]
```

Definition at line 79 of file easylogging++.cc.

7.3.2.58 kUnknownUser

```
const char* el::base::consts::kUnknownUser = "unknown-user"  [static]
```

Definition at line 78 of file easylogging++.cc.

7.3.2.59 kValidLoggerIdSymbols

```
const char* el::base::consts::kValidLoggerIdSymbols  [static]
```

Initial value:

```
=
  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._"
```

Definition at line 103 of file easylogging++.cc.

7.3.2.60 kVerboseLevelFormatSpecifier

```
const base::type::char_t* el::base::consts::kVerboseLevelFormatSpecifier = ELPP_LITERAL("%vlevel")
[static]
```

Definition at line 58 of file easylogging++.cc.

7.3.2.61 kVerboseLevelLogValue

```
const base::type::char_t* el::base::consts::kVerboseLevelLogValue  [static]
```

Initial value:

```
=
  ELPP_LITERAL("VERBOSE")
```

Definition at line 33 of file easylogging++.cc.

7.3.2.62 kVerboseLevelShortLogValue

```
const base::type::char_t* el::base::consts::kVerboseLevelShortLogValue = ELPP_LITERAL("V")
[static]
```

Definition at line 41 of file easylogging++.cc.

7.3.2.63 kWarningLevelLogValue

```
const base::type::char_t* el::base::consts::kWarningLevelLogValue = ELPP_LITERAL("WARNING")
[static]
```

Definition at line 30 of file easylogging++.cc.

7.3.2.64 kWarningLevelShortLogValue

```
const base::type::char_t* el::base::consts::kWarningLevelShortLogValue = ELPP_LITERAL("W")
[static]
```

Definition at line 38 of file easylogging++.cc.

7.3.2.65 kYearBase

```
const int el::base::consts::kYearBase = 1900  [static]
```

Definition at line 69 of file easylogging++.cc.

7.3.2.66 name

```
const char* el::base::consts::name
```

Definition at line 779 of file easylogging++.h.

7.3.2.67 numb

```
int el::base::consts::numb
```

Definition at line 778 of file easylogging++.h.

7.3.2.68 unit

```
const base::type::char_t* el::base::consts::unit
```

Definition at line 767 of file easylogging++.h.

7.3.2.69 value

```
double el::base::consts::value
```

Definition at line 766 of file easylogging++.h.

7.4 el::base::debug Namespace Reference

Contains some internal debugging tools like crash handler and stack tracer.

Data Structures

- class CrashHandler

7.4.1 Detailed Description

Contains some internal debugging tools like crash handler and stack tracer.

7.5 el::base::threading Namespace Reference

Namespaces

- namespace internal

Data Structures

- class ThreadSafe

    Base of thread safe class, this class is inheritable-only.

Typedefs

- typedef base::threading::internal::NoMutex Mutex
- typedef base::threading::internal::NoScopedLock< base::threading::Mutex > ScopedLock

Functions

- static std::string getCurrentThreadId (void)

7.5.1 Typedef Documentation

7.5.1.1 Mutex

```
typedef base::threading::internal::NoMutex el::base::threading::Mutex  [private]
```

Definition at line 998 of file easylogging++.h.

7.5.1.2 ScopedLock

```
typedef base::threading::internal::NoScopedLock<base::threading::Mutex> el::base::threading::ScopedLock
[private]
```

Definition at line 999 of file easylogging++.h.

7.5.2 Function Documentation

7.5.2.1 getCurrentThreadId()

```
static std::string el::base::threading::getCurrentThreadId (
            void  )  [inline], [static], [private]
```

Definition at line 1033 of file easylogging++.h.

References getCurrentThreadId().

7.6 el::base::threading::internal Namespace Reference

Data Structures

- class NoMutex

  Mutex wrapper used when multi-threading is disabled.
- class NoScopedLock

  Lock guard wrapper used when multi-threading is disabled.

7.7 el::base::type Namespace Reference

Data types used by Easylogging++.

Typedefs

- typedef char char_t
- typedef std::string string_t
- typedef std::stringstream stringstream_t
- typedef std::fstream fstream_t
- typedef std::ostream ostream_t
- typedef unsigned int EnumType
- typedef unsigned short VerboseLevel
- typedef unsigned long int LineNumber
- typedef std::shared_ptr< base::Storage > StoragePointer
- typedef std::shared_ptr< LogDispatchCallback > LogDispatchCallbackPtr
- typedef std::shared_ptr< PerformanceTrackingCallback > PerformanceTrackingCallbackPtr
- typedef std::shared_ptr< LoggerRegistrationCallback > LoggerRegistrationCallbackPtr
- typedef std::unique_ptr< el::base::PerformanceTracker > PerformanceTrackerPtr

7.7.1 Detailed Description

Data types used by Easylogging++.

7.7.2 Typedef Documentation

7.7.2.1 char_t

```
typedef char el::base::type::char_t
```

Definition at line 528 of file easylogging++.h.

7.7.2.2 EnumType

```
typedef unsigned int el::base::type::EnumType
```

Definition at line 539 of file easylogging++.h.

7.7.2.3 fstream_t

```
typedef std::fstream el::base::type::fstream_t
```

Definition at line 531 of file easylogging++.h.

7.7.2.4 LineNumber

```
typedef unsigned long int el::base::type::LineNumber
```

Definition at line 541 of file easylogging++.h.

7.7.2.5 LogDispatchCallbackPtr

```
typedef std::shared_ptr<LogDispatchCallback> el::base::type::LogDispatchCallbackPtr
```

Definition at line 543 of file easylogging++.h.

7.7.2.6 LoggerRegistrationCallbackPtr

```
typedef std::shared_ptr<LoggerRegistrationCallback> el::base::type::LoggerRegistrationCallbackPtr
```

Definition at line 545 of file easylogging++.h.

7.7.2.7 ostream_t

```
typedef std::ostream el::base::type::ostream_t
```

Definition at line 532 of file easylogging++.h.

7.7.2.8 PerformanceTrackerPtr

```
typedef std::unique_ptr<el::base::PerformanceTracker> el::base::type::PerformanceTrackerPtr
```

Definition at line 546 of file easylogging++.h.

7.7.2.9 PerformanceTrackingCallbackPtr

```
typedef std::shared_ptr<PerformanceTrackingCallback> el::base::type::PerformanceTrackingCallbackPtr
```

Definition at line 544 of file easylogging++.h.

7.7.2.10 StoragePointer

```
typedef std::shared_ptr<base::Storage> el::base::type::StoragePointer
```

Definition at line 542 of file easylogging++.h.

7.7.2.11 string_t

```
typedef std::string el::base::type::string_t
```

Definition at line 529 of file easylogging++.h.

7.7.2.12 stringstream_t

```
typedef std::stringstream el::base::type::stringstream_t
```

Definition at line 530 of file easylogging++.h.

7.7.2.13 VerboseLevel

```
typedef unsigned short el::base::type::VerboseLevel
```

Definition at line 540 of file easylogging++.h.

7.8 el::base::utils Namespace Reference

Namespace containing utility functions/static classes used internally.

Namespaces

- namespace bitwise

  Bitwise operations for C++11 strong enum class. This casts e into Flag_T and returns value after bitwise operation Use these function as.

Data Structures

- class AbstractRegistry

  Abstract registry (aka repository) that provides basic interface for pointer repository specified by T_Ptr type.
- class CommandLineArgs

  Command line arguments for application if specified using el::Helpers::setArgs(..) or START_EASYLOGGINGPP(..)
- class DateTime

  Contains utilities for cross-platform date/time. This class make use of el::base::utils::Str.
- class File
- class OS

  Operating System helper static class used internally. You should not use it.
- class Registry

  A pointer registry mechanism to manage memory and provide search functionalities. (non-predicate version)
- class RegistryWithPred

  A pointer registry mechanism to manage memory and provide search functionalities. (predicate version)
- class Str

  String utilities helper class used internally. You should not use it.
- class Utils

Functions

- static void abort (int status, const std::string &reason)

  Aborts application due with user-defined status.
- base::type::ostream_t & operator<< (base::type::ostream_t &os, const CommandLineArgs &c)
- template<typename T >
  static std::enable_if< std::is_pointer< T ∗ >::value, void >::type safeDelete (T ∗&pointer)

  Deletes memory safely and points to null.
- template<typename Enum >
  static void addFlag (Enum e, base::type::EnumType ∗flag)
- template<typename Enum >
  static void removeFlag (Enum e, base::type::EnumType ∗flag)
- template<typename Enum >
  static bool hasFlag (Enum e, base::type::EnumType flag)

7.8.1 Detailed Description

Namespace containing utility functions/static classes used internally.

7.8.2 Function Documentation

7.8.2.1 abort()

```
static void el::base::utils::abort (
            int status,
            const std::string & reason )  [static]
```

Aborts application due with user-defined status.

Definition at line 113 of file easylogging++.cc.

References abort(), and ELPP_UNUSED.

7.8.2.2 addFlag()

```
template<typename Enum >
static void el::base::utils::addFlag (
            Enum e,
            base::type::EnumType ∗ flag )  [inline], [static], [private]
```

Definition at line 881 of file easylogging++.h.

7.8.2.3 hasFlag()

```
template<typename Enum >
static bool el::base::utils::hasFlag (
            Enum e,
            base::type::EnumType flag )  [inline], [static], [private]
```

Definition at line 889 of file easylogging++.h.

7.8.2.4 operator<<()

```
base::type::ostream_t & el::base::utils::operator<< (
             base::type::ostream_t & os,
             const CommandLineArgs & c )
```

Definition at line 1368 of file easylogging++.cc.

7.8.2.5 removeFlag()

```
template<typename Enum >
static void el::base::utils::removeFlag (
             Enum e,
             base::type::EnumType * flag )  [inline], [static], [private]
```

Definition at line 885 of file easylogging++.h.

7.8.2.6 safeDelete()

```
template<typename T >
static std::enable_if< std::is_pointer< T * >::value, void >::type el::base::utils::safe←
Delete (
             T *& pointer )  [static], [private]
```

Deletes memory safely and points to null.

Definition at line 858 of file easylogging++.h.

7.9 el::base::utils::bitwise Namespace Reference

Bitwise operations for C++11 strong enum class. This casts e into Flag_T and returns value after bitwise operation Use these function as.

Functions

- template<typename Enum >
  static base::type::EnumType And (Enum e, base::type::EnumType flag)
- template<typename Enum >
  static base::type::EnumType Not (Enum e, base::type::EnumType flag)
- template<typename Enum >
  static base::type::EnumType Or (Enum e, base::type::EnumType flag)

7.9.1 Detailed Description

Bitwise operations for C++11 strong enum class. This casts e into Flag_T and returns value after bitwise operation Use these function as.

```
flag = bitwise::Or<MyEnum>(MyEnum::val1, flag);
```

7.9.2 Function Documentation

7.9.2.1 And()

```
template<typename Enum >
static base::type::EnumType el::base::utils::bitwise::And (
            Enum e,
            base::type::EnumType flag )  [inline], [static], [private]
```

Definition at line 868 of file easylogging++.h.

7.9.2.2 Not()

```
template<typename Enum >
static base::type::EnumType el::base::utils::bitwise::Not (
            Enum e,
            base::type::EnumType flag )  [inline], [static], [private]
```

Definition at line 872 of file easylogging++.h.

7.9.2.3 Or()

```
template<typename Enum >
static base::type::EnumType el::base::utils::bitwise::Or (
            Enum e,
            base::type::EnumType flag )  [inline], [static], [private]
```

Definition at line 876 of file easylogging++.h.

7.10 std Namespace Reference

Data Structures

- struct hash< el::Level >

7.11 WIP Namespace Reference

Namespace for work in progress.

Functions

- void exampleEasyLogging ()
  Example of how to use easylogging with a configuration file.

7.11.1 Detailed Description

Namespace for work in progress.

Todo
- Github
- "Dev-Ops"
- Doxygen settings
- Template-Comment
- Template-Header-Comment

- This namespace is for code that is not finished yet
- It is used to keep the main namespace clean

7.11.2 Function Documentation

7.11.2.1 exampleEasyLogging()

```
void WIP::exampleEasyLogging ( )
```

Example of how to use easylogging with a configuration file.

- This function is an example of how to use easylogging
- The configuration file is located in ../conf
- Before proper integration, config has to be done properly

  Todo

Definition at line 53 of file main.cpp.

References LOG, el::Loggers::reconfigureAllLoggers(), and el::Loggers::reconfigureLogger().

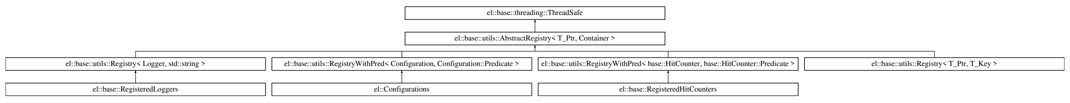
Chapter 8

Data Structure Documentation

8.1 el::base::utils::AbstractRegistry< T_Ptr, Container > Class Template Reference

Abstract registry (aka repository) that provides basic interface for pointer repository specified by T_Ptr type.

Inheritance diagram for el::base::utils::AbstractRegistry< T_Ptr, Container >:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< T_Ptr, Container >
el::base::utils::Registry< Logger, std::string >
el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >
el::base::utils::Registry< T_Ptr, T_Key >
el::base::RegisteredLoggers
el::Configurations
el::base::RegisteredHitCounters

Public Types

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Public Member Functions

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, Container > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, Container > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ∼AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual Container & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const Container & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.
- virtual void unregisterAll (void)=0

  Unregisters all the pointers from current repository.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Protected Member Functions

- virtual void deepCopy (const AbstractRegistry< T_Ptr, Container > &)=0
- void reinitDeepCopy (const AbstractRegistry< T_Ptr, Container > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

Private Attributes

- Container m_list

8.1.1 Detailed Description

template<typename T_Ptr, typename Container>
class el::base::utils::AbstractRegistry< T_Ptr, Container >

Abstract registry (aka repository) that provides basic interface for pointer repository specified by T_Ptr type.

@detail Most of the functions are virtual final methods but anything implementing this abstract class should implement unregisterAll() and deepCopy(const AbstractRegistry<T_Ptr, Container>&) and write registerNew() method according to container and few more methods; get() to find element, unregister() to unregister single entry. Please note that this is thread-unsafe and should also implement thread-safety mechanisms in implementation.

Definition at line 1255 of file easylogging++.h.

8.1.2 Member Typedef Documentation

8.1.2.1 const_iterator

```
template<typename T_Ptr , typename Container >
typedef Container::const_iterator el::base::utils::AbstractRegistry< T_Ptr, Container >←
::const_iterator
```

Definition at line 1258 of file easylogging++.h.

8.1.2.2 iterator

```
template<typename T_Ptr , typename Container >
typedef Container::iterator el::base::utils::AbstractRegistry< T_Ptr, Container >::iterator
```

Definition at line 1257 of file easylogging++.h.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 AbstractRegistry() [1/2]

```
template<typename T_Ptr , typename Container >
el::base::utils::AbstractRegistry< T_Ptr, Container >::AbstractRegistry (
             void  )  [inline]
```

Default constructor.

Definition at line 1261 of file easylogging++.h.

8.1.3.2 AbstractRegistry() [2/2]

```
template<typename T_Ptr , typename Container >
el::base::utils::AbstractRegistry< T_Ptr, Container >::AbstractRegistry (
             AbstractRegistry< T_Ptr, Container > && sr )  [inline]
```

Move constructor that is useful for base classes.

Definition at line 1264 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::m_list.

8.1.3.3 ∼AbstractRegistry()

```
template<typename T_Ptr , typename Container >
virtual el::base::utils::AbstractRegistry< T_Ptr, Container >::∼AbstractRegistry (
             void  )  [inline], [virtual]
```

Definition at line 1306 of file easylogging++.h.

8.1.4 Member Function Documentation

8.1.4.1 begin()

```
template<typename T_Ptr , typename Container >
virtual iterator el::base::utils::AbstractRegistry< T_Ptr, Container >::begin (
             void  )  [inline], [virtual]
```

Returns

Iterator pointer from start of repository

Definition at line 1310 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin().

8.1.4.2 cbegin()

```
template<typename T_Ptr , typename Container >
virtual const_iterator el::base::utils::AbstractRegistry< T_Ptr, Container >::cbegin (
            void  ) const  [inline], [virtual]
```

Returns

Constant iterator pointer from start of repository

Definition at line 1321 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::cbegin().

8.1.4.3 cend()

```
template<typename T_Ptr , typename Container >
virtual const_iterator el::base::utils::AbstractRegistry< T_Ptr, Container >::cend (
            void  ) const  [inline], [virtual]
```

Returns

End of repository

Definition at line 1326 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::cend().

8.1.4.4 deepCopy()

```
template<typename T_Ptr , typename Container >
virtual void el::base::utils::AbstractRegistry< T_Ptr, Container >::deepCopy (
            const AbstractRegistry< T_Ptr, Container > &  )  [protected], [pure virtual]
```

Implemented in el::base::utils::RegistryWithPred< T_Ptr, Pred >.

8.1.4.5 empty()

```
template<typename T_Ptr , typename Container >
virtual bool el::base::utils::AbstractRegistry< T_Ptr, Container >::empty (
            void  ) const  [inline], [virtual]
```

Returns

Whether or not repository is empty

Definition at line 1331 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::empty().

8.1.4.6 end()

```
template<typename T_Ptr , typename Container >
virtual iterator el::base::utils::AbstractRegistry< T_Ptr, Container >::end (
            void ) [inline], [virtual]
```

Returns

Iterator pointer from end of repository

Definition at line 1315 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end().

8.1.4.7 list() [1/2]

```
template<typename T_Ptr , typename Container >
virtual const Container & el::base::utils::AbstractRegistry< T_Ptr, Container >::list (
            void ) const [inline], [virtual]
```

Returns underlying container by constant reference.

Definition at line 1346 of file easylogging++.h.

8.1.4.8 list() [2/2]

```
template<typename T_Ptr , typename Container >
virtual Container & el::base::utils::AbstractRegistry< T_Ptr, Container >::list (
            void ) [inline], [virtual]
```

Returns underlying container by reference.

Definition at line 1341 of file easylogging++.h.

8.1.4.9 operator"!=()

```
template<typename T_Ptr , typename Container >
bool el::base::utils::AbstractRegistry< T_Ptr, Container >::operator!= (
            const AbstractRegistry< T_Ptr, Container > & other ) [inline]
```

Definition at line 1284 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::m_list, and el::base::utils::AbstractRegistry< T_Ptr, Container >::s

8.1.4.10 operator=()

```
template<typename T_Ptr , typename Container >
AbstractRegistry & el::base::utils::AbstractRegistry< T_Ptr, Container >::operator= (
            AbstractRegistry< T_Ptr, Container > && sr ) [inline]
```

Assignment move operator.

Definition at line 1297 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::m_list.

8.1.4.11 operator==()

```
template<typename T_Ptr , typename Container >
bool el::base::utils::AbstractRegistry< T_Ptr, Container >::operator== (
            const AbstractRegistry< T_Ptr, Container > & other )  [inline]
```

Definition at line 1272 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::m_list, and el::base::utils::AbstractRegistry< T_Ptr, Container >::s

8.1.4.12 reinitDeepCopy()

```
template<typename T_Ptr , typename Container >
void el::base::utils::AbstractRegistry< T_Ptr, Container >::reinitDeepCopy (
            const AbstractRegistry< T_Ptr, Container > & sr )  [inline], [protected]
```

Definition at line 1355 of file easylogging++.h.

8.1.4.13 size()

```
template<typename T_Ptr , typename Container >
virtual std::size_t el::base::utils::AbstractRegistry< T_Ptr, Container >::size (
            void  ) const  [inline], [virtual]
```

Returns

Size of repository

Definition at line 1336 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::size().

8.1.4.14 unregisterAll()

```
template<typename T_Ptr , typename Container >
virtual void el::base::utils::AbstractRegistry< T_Ptr, Container >::unregisterAll (
            void  )  [pure virtual]
```

Unregisters all the pointers from current repository.

Implemented in el::base::utils::Registry< T_Ptr, T_Key >, el::base::utils::Registry< Logger, std::string >, el::base::utils::RegistryWithPred< T_Ptr, Pred >, el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >
and el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >.

8.1.5 Field Documentation

8.1.5.1 m_list

```
template<typename T_Ptr , typename Container >
Container el::base::utils::AbstractRegistry< T_Ptr, Container >::m_list  [private]
```

Definition at line 1361 of file easylogging++.h.

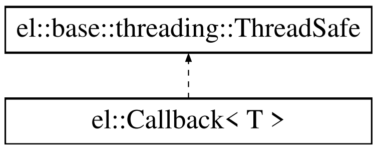
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.2 el::Callback< T > Class Template Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::Callback< T >:



Public Member Functions

- Callback (void)
- bool enabled (void) const
- void setEnabled (bool enabled)

Protected Member Functions

- virtual void handle (const T ∗handlePtr)=0

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Attributes

- bool m_enabled

8.2.1 Detailed Description

template<typename T>
class el::Callback< T >

Definition at line 2144 of file easylogging++.h.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Callback()

```
template<typename T >
el::Callback< T >::Callback (
             void  )  [inline]
```

Definition at line 2146 of file easylogging++.h.

8.2.3 Member Function Documentation

8.2.3.1 enabled()

```
template<typename T >
bool el::Callback< T >::enabled (
             void  ) const  [inline]
```

Definition at line 2147 of file easylogging++.h.

8.2.3.2 handle()

```
template<typename T >
virtual void el::Callback< T >::handle (
             const T * handlePtr )  [protected], [pure virtual]
```

Implemented in el::LogDispatchCallback, and el::base::DefaultLogDispatchCallback.

8.2.3.3 setEnabled()

```
template<typename T >
void el::Callback< T >::setEnabled (
             bool enabled )  [inline]
```

Definition at line 2150 of file easylogging++.h.

8.2.4 Field Documentation

8.2.4.1 m_enabled

```
template<typename T >
bool el::Callback< T >::m_enabled  [private]
```

Definition at line 2157 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.3 el::base::utils::CommandLineArgs Class Reference

Command line arguments for application if specified using el::Helpers::setArgs(..) or START_EASYLOGGINGPP(..)

Public Member Functions

- CommandLineArgs (void)
- CommandLineArgs (int argc, const char **argv)
- CommandLineArgs (int argc, char **argv)
- virtual ∼CommandLineArgs (void)
- void setArgs (int argc, const char **argv)

  Sets arguments and parses them.
- void setArgs (int argc, char **argv)

  Sets arguments and parses them.
- bool hasParamWithValue (const char *paramKey) const

  Returns true if arguments contain paramKey with a value (separated by '=')
- const char * getParamValue (const char *paramKey) const

  Returns value of arguments.
- bool hasParam (const char *paramKey) const

  Return true if arguments has a param (not having a value) i,e without '='.
- bool empty (void) const

  Returns true if no params available. This exclude argv[0].
- std::size_t size (void) const

  Returns total number of arguments. This exclude argv[0].

Private Attributes

- int m_argc
- char ** m_argv
- std::unordered_map< std::string, std::string > m_paramsWithValue
- std::vector< std::string > m_params

Friends

- base::type::ostream_t & operator<< (base::type::ostream_t &os, const CommandLineArgs &c)

8.3.1 Detailed Description

Command line arguments for application if specified using el::Helpers::setArgs(..) or START_EASYLOGGINGPP(..)

Definition at line 1211 of file easylogging++.h.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 CommandLineArgs() [1/3]

```
el::base::utils::CommandLineArgs::CommandLineArgs (
            void ) [inline]
```

Definition at line 1213 of file easylogging++.h.

8.3.2.2 CommandLineArgs() [2/3]

```
el::base::utils::CommandLineArgs::CommandLineArgs (
            int argc,
            const char ** argv )  [inline]
```

Definition at line 1216 of file easylogging++.h.

8.3.2.3 CommandLineArgs() [3/3]

```
el::base::utils::CommandLineArgs::CommandLineArgs (
            int argc,
            char ** argv )  [inline]
```

Definition at line 1219 of file easylogging++.h.

8.3.2.4 ∼CommandLineArgs()

```
virtual el::base::utils::CommandLineArgs::∼CommandLineArgs (
            void  )  [inline], [virtual]
```

Definition at line 1222 of file easylogging++.h.

8.3.3 Member Function Documentation

8.3.3.1 empty()

```
bool el::base::utils::CommandLineArgs::empty (
            void  ) const
```

Returns true if no params available. This exclude argv[0].

Definition at line 1360 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::empty(), m_params, and m_paramsWithValue.

8.3.3.2 getParamValue()

```
const char * el::base::utils::CommandLineArgs::getParamValue (
            const char * paramKey ) const
```

Returns value of arguments.

See also

hasParamWithValue(const char*)

Definition at line 1351 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), and m_paramsWithValue.

8.3.3.3 hasParam()

```
bool el::base::utils::CommandLineArgs::hasParam (
            const char * paramKey ) const
```

Return true if arguments has a param (not having a value) i,e without '='.

Definition at line 1356 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), and m_params.

8.3.3.4 hasParamWithValue()

```
bool el::base::utils::CommandLineArgs::hasParamWithValue (
            const char * paramKey ) const
```

Returns true if arguments contain paramKey with a value (separated by '=')

Definition at line 1347 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), and m_paramsWithValue.

8.3.3.5 setArgs() [1/2]

```
void el::base::utils::CommandLineArgs::setArgs (
            int argc,
            char ** argv )
```

Sets arguments and parses them.

Definition at line 1317 of file easylogging++.cc.

References ELPP_INTERNAL_INFO, getParamValue(), hasParam(), hasParamWithValue(), m_argc, m_argv, m_params, and m_paramsWithValue.

8.3.3.6 setArgs() [2/2]

```
void el::base::utils::CommandLineArgs::setArgs (
            int argc,
            const char ** argv )  [inline]
```

Sets arguments and parses them.

Definition at line 1224 of file easylogging++.h.

8.3.3.7 size()

```
std::size_t el::base::utils::CommandLineArgs::size (
            void  ) const
```

Returns total number of arguments. This exclude argv[0].

Definition at line 1364 of file easylogging++.cc.

References m_params, m_paramsWithValue, and el::base::utils::AbstractRegistry< T_Ptr, Container >::size().

8.3.4 Friends And Related Symbol Documentation

8.3.4.1 operator<<

```
base::type::ostream_t & operator<< (
             base::type::ostream_t & os,
             const CommandLineArgs & c )  [friend]
```

Definition at line 1368 of file easylogging++.cc.

8.3.5 Field Documentation

8.3.5.1 m_argc

```
int el::base::utils::CommandLineArgs::m_argc  [private]
```

Definition at line 1243 of file easylogging++.h.

8.3.5.2 m_argv

```
char** el::base::utils::CommandLineArgs::m_argv  [private]
```

Definition at line 1244 of file easylogging++.h.

8.3.5.3 m_params

```
std::vector<std::string> el::base::utils::CommandLineArgs::m_params  [private]
```

Definition at line 1246 of file easylogging++.h.

8.3.5.4 m_paramsWithValue

```
std::unordered_map<std::string, std::string> el::base::utils::CommandLineArgs::m_paramsWith↩
Value  [private]
```

Definition at line 1245 of file easylogging++.h.

The documentation for this class was generated from the following files:

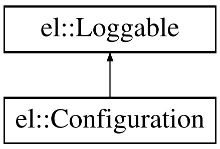
- lib/easylogging++.h
- lib/easylogging++.cc

8.4 el::Configuration Class Reference

Represents single configuration that has representing level, configuration type and a string based value.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Configuration:



Data Structures

- class Predicate

  Used to find configuration from configuration (pointers) repository. Avoid using it.

Public Member Functions

- Configuration (const Configuration &c)
- Configuration & operator= (const Configuration &c)
- virtual ∼Configuration (void)
- Configuration (Level level, ConfigurationType configurationType, const std::string &value)

  Full constructor used to sets value of configuration.
- Level level (void) const

  Gets level of current configuration.
- ConfigurationType configurationType (void) const

  Gets configuration type of current configuration.
- const std::string & value (void) const

  Gets string based configuration value.
- void setValue (const std::string &value)

  Set string based configuration value.
- virtual void log (el::base::type::ostream_t &os) const

Public Member Functions inherited from el::Loggable

- virtual ∼Loggable (void)

Private Attributes

- Level m_level
- ConfigurationType m_configurationType
- std::string m_value

8.4.1 Detailed Description

Represents single configuration that has representing level, configuration type and a string based value.

@detail String based value means any value either its boolean, integer or string itself, it will be embedded inside quotes and will be parsed later.

Consider some examples below:

- el::Configuration confEnabledInfo(el::Level::Info, el::ConfigurationType::Enabled, "true");
- el::Configuration confMaxLogFileSizeInfo(el::Level::Info, el::ConfigurationType::MaxLogFileSize, "2048");
- el::Configuration confFilenameInfo(el::Level::Info, el::ConfigurationType::Filename, "/var/log/my.log");

Definition at line 1673 of file easylogging++.h.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Configuration() `[1/2]`

```
el::Configuration::Configuration (
             const Configuration & c )
```

Definition at line 235 of file easylogging++.cc.

8.4.2.2 ∼Configuration()

```
virtual el::Configuration::∼Configuration (
             void ) [inline], [virtual]
```

Definition at line 1678 of file easylogging++.h.

8.4.2.3 Configuration() `[2/2]`

```
el::Configuration::Configuration (
             Level level,
             ConfigurationType configurationType,
             const std::string & value )
```

Full constructor used to sets value of configuration.

Definition at line 251 of file easylogging++.cc.

8.4.3 Member Function Documentation

8.4.3.1 configurationType()

```
ConfigurationType el::Configuration::configurationType (
             void ) const [inline]
```

Gets configuration type of current configuration.

Definition at line 1690 of file easylogging++.h.

8.4.3.2 level()

```
Level el::Configuration::level (
             void  ) const  [inline]
```

Gets level of current configuration.

Definition at line 1685 of file easylogging++.h.

8.4.3.3 log()

```
void el::Configuration::log (
             el::base::type::ostream_t & os ) const  [virtual]
```

Implements el::Loggable.

Definition at line 257 of file easylogging++.cc.

References el::ConfigurationTypeHelper::convertToString(), el::LevelHelper::convertToString(), ELPP_LITERAL, m_configurationType, m_level, and m_value.

8.4.3.4 operator=()

```
Configuration & el::Configuration::operator= (
             const Configuration & c )
```

Definition at line 241 of file easylogging++.cc.

References m_configurationType, m_level, and m_value.

8.4.3.5 setValue()

```
void el::Configuration::setValue (
             const std::string & value )  [inline]
```

Set string based configuration value.

Parameters

value	Value to set. Values have to be std::string; For boolean values use "true", "false", for any integral values use them in quotes. They will be parsed when configuring

Definition at line 1702 of file easylogging++.h.

8.4.3.6 value()

```
const std::string & el::Configuration::value (
             void  ) const  [inline]
```

Gets string based configuration value.

Definition at line 1695 of file easylogging++.h.

8.4.4 Field Documentation

8.4.4.1 m_configurationType

```
ConfigurationType el::Configuration::m_configurationType  [private]
```

Definition at line 1722 of file easylogging++.h.

8.4.4.2 m_level

```
Level el::Configuration::m_level  [private]
```

Definition at line 1721 of file easylogging++.h.

8.4.4.3 m_value

```
std::string el::Configuration::m_value  [private]
```

Definition at line 1723 of file easylogging++.h.

The documentation for this class was generated from the following files:

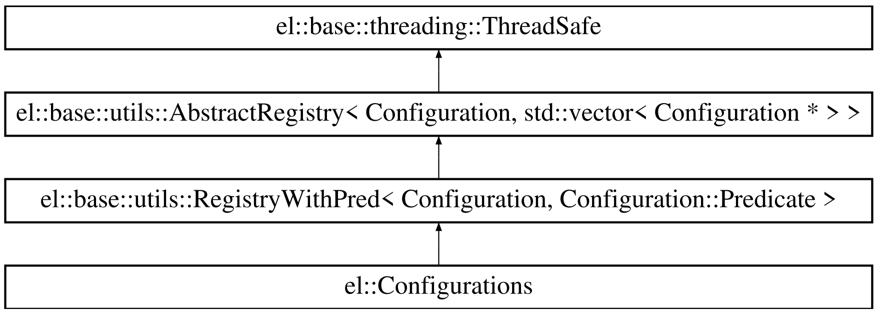
- lib/easylogging++.h
- lib/easylogging++.cc

8.5 el::Configurations Class Reference

Thread-safe Configuration repository.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Configurations:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< Configuration, std::vector< Configuration * > >
el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >
el::Configurations

Data Structures

- class Parser

    Parser used internally to parse configurations from file or text.

Public Member Functions

- Configurations (void)

    Default constructor with empty repository.
- Configurations (const std::string &configurationFile, bool useDefaultsForRemaining=true, Configurations ∗base=nullptr)

    Constructor used to set configurations using configuration file.
- virtual ∼Configurations (void)
- bool parseFromFile (const std::string &configurationFile, Configurations ∗base=nullptr)

    Parses configuration from file.
- bool parseFromText (const std::string &configurationsString, Configurations ∗base=nullptr)

    Parse configurations from configuration string.
- void setFromBase (Configurations ∗base)

    Sets configuration based-off an existing configurations.
- bool hasConfiguration (ConfigurationType configurationType)

    Determines whether or not specified configuration type exists in the repository.
- bool hasConfiguration (Level level, ConfigurationType configurationType)

    Determines whether or not specified configuration type exists for specified level.
- void set (Level level, ConfigurationType configurationType, const std::string &value)

    Sets value of configuration for specified level.
- void set (Configuration ∗conf)

    Sets single configuration based on other single configuration.
- Configuration ∗ get (Level level, ConfigurationType configurationType)
- void setGlobally (ConfigurationType configurationType, const std::string &value)

    Sets configuration for all levels.
- void clear (void)

    Clears repository so that all the configurations are unset.
- const std::string & configurationFile (void) const

    Gets configuration file used in parsing this configurations.
- void setToDefault (void)

    Sets configurations to "factory based" configurations.
- void setRemainingToDefault (void)

    Lets you set the remaining configurations to default.

Public Member Functions inherited from el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >

- RegistryWithPred (void)
- RegistryWithPred (const RegistryWithPred &sr)

    Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.
- virtual ∼RegistryWithPred (void)
- RegistryWithPred & operator= (const RegistryWithPred &sr)

    Assignment operator that unregisters all the existing registries and deeply copies each of repo element.

Public Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, Container > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, Container > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ∼AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual Container & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const Container & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Member Functions

- void unsafeSetIfNotExist (Level level, ConfigurationType configurationType, const std::string &value)

  Unsafely sets configuration if does not already exist.
- void unsafeSet (Level level, ConfigurationType configurationType, const std::string &value)

  Thread unsafe set.
- void setGlobally (ConfigurationType configurationType, const std::string &value, bool includeGlobalLevel)

  Sets configurations for all levels including Level::Global if includeGlobalLevel is true.
- void unsafeSetGlobally (ConfigurationType configurationType, const std::string &value, bool includeGlobalLevel)

  Sets configurations (Unsafely) for all levels including Level::Global if includeGlobalLevel is true.

Private Attributes

- std::string m_configurationFile
- bool m_isFromFile

Friends

- class el::Loggers

Additional Inherited Members

Public Types inherited from el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >

- typedef RegistryWithPred< Configuration, Configuration::Predicate >::iterator iterator
- typedef RegistryWithPred< Configuration, Configuration::Predicate >::const_iterator const_iterator

Public Types inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Protected Member Functions inherited from el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >

- virtual void unregisterAll (void) ELPP_FINAL

  Unregisters all the pointers from current repository.
- virtual void unregister (Configuration ∗&ptr) ELPP_FINAL
- virtual void registerNew (Configuration ∗ptr) ELPP_FINAL
- Configuration ∗ get (const T &arg1, const T2 arg2)

  Gets pointer from repository with specified arguments. Arguments are passed to predicate in order to validate pointer.

Protected Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- virtual void deepCopy (const AbstractRegistry< T_Ptr, Container > &)=0
- void reinitDeepCopy (const AbstractRegistry< T_Ptr, Container > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

8.5.1 Detailed Description

Thread-safe Configuration repository.

@detail This repository represents configurations for all the levels and configuration type mapped to a value.

Definition at line 1729 of file easylogging++.h.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Configurations() [1/2]

```
el::Configurations::Configurations (
            void )
```

Default constructor with empty repository.

Definition at line 275 of file easylogging++.cc.

8.5.2.2 Configurations() [2/2]

```
el::Configurations::Configurations (
            const std::string & configurationFile,
            bool useDefaultsForRemaining = true,
            Configurations * base = nullptr )
```

Constructor used to set configurations using configuration file.

Parameters

configurationFile	Full path to configuration file
useDefaultsForRemaining	Lets you set the remaining configurations to default.
base	If provided, this configuration will be based off existing repository that this argument is pointing to.

See also

parseFromFile(const std::string&, Configurations∗ base)

setRemainingToDefault()

Definition at line 280 of file easylogging++.cc.

References configurationFile(), parseFromFile(), and setRemainingToDefault().

8.5.2.3 ∼Configurations()

```
virtual el::Configurations::∼Configurations (
            void ) [inline], [virtual]
```

Definition at line 1743 of file easylogging++.h.

8.5.3 Member Function Documentation

8.5.3.1 clear()

```
void el::Configurations::clear (
            void ) [inline]
```

Clears repository so that all the configurations are unset.

Definition at line 1811 of file easylogging++.h.

8.5.3.2 configurationFile()

```
const std::string & el::Configurations::configurationFile (
            void ) const [inline]
```

Gets configuration file used in parsing this configurations.

@detail If this repository was set manually or by text this returns empty string.

Definition at line 1819 of file easylogging++.h.

8.5.3.3 get()

```
Configuration * el::Configurations::get (
            Level level,
            ConfigurationType configurationType ) [inline]
```

Definition at line 1797 of file easylogging++.h.

8.5.3.4 hasConfiguration() [1/2]

```
bool el::Configurations::hasConfiguration (
            ConfigurationType configurationType )
```

Determines whether or not specified configuration type exists in the repository.

@detail Returns as soon as first level is found.

Parameters

configurationType	Type of configuration to check existence for.

Definition at line 322 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), el::LevelHelper::forEachLevel(), hasConfiguration(), and el::LevelHelper::kMinValid.

8.5.3.5 hasConfiguration() [2/2]

```
bool el::Configurations::hasConfiguration (
            Level level,
            ConfigurationType configurationType )
```

Determines whether or not specified configuration type exists for specified level.

Parameters

level	Level to check
configurationType	Type of configuration to check existence for.

Definition at line 334 of file easylogging++.cc.

References el::base::threading::ThreadSafe::lock(), and el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >:

8.5.3.6 parseFromFile()

```
bool el::Configurations::parseFromFile (
            const std::string & configurationFile,
            Configurations * base = nullptr )
```

Parses configuration from file.

Parameters

configurationFile	Full path to configuration file
base	Configurations to base new configuration repository off. This value is used when you want to use existing Configurations to base all the values and then set rest of configuration via configuration file.

Returns

True if successfully parsed, false otherwise. You may define 'ELPP_DEBUG_ASSERT_FAILURE' to make sure you do not proceed without successful parse.

Definition at line 290 of file easylogging++.cc.

References configurationFile(), ELPP_ASSERT, m_isFromFile, el::Configurations::Parser::parseFromFile(), and el::base::utils::File::pathExists().

8.5.3.7 parseFromText()

```
bool el::Configurations::parseFromText (
            const std::string & configurationsString,
            Configurations * base = nullptr )
```

Parse configurations from configuration string.

@detail This configuration string has same syntax as configuration file contents. Make sure all the necessary new line characters are provided.

Parameters

base	Configurations to base new configuration repository off. This value is used when you want to use existing Configurations to base all the values and then set rest of configuration via configuration text.

Returns

True if successfully parsed, false otherwise. You may define 'ELPP_DEBUG_ASSERT_FAILURE' to make sure you do not proceed without successful parse.

Definition at line 304 of file easylogging++.cc.

References m_isFromFile, and el::Configurations::Parser::parseFromText().

8.5.3.8 set() [1/2]

```
void el::Configurations::set (
            Configuration * conf )
```

Sets single configuration based on other single configuration.

See also

set(Level level, ConfigurationType configurationType, const std::string& value)

Definition at line 353 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and set().

8.5.3.9 set() [2/2]

```
void el::Configurations::set (
            Level level,
            ConfigurationType configurationType,
            const std::string & value )
```

Sets value of configuration for specified level.

@detail Any existing configuration for specified level will be replaced. Also note that configuration types ConfigurationType::SubsecondPrecision and ConfigurationType::PerformanceTracking will be ignored if not set for Level::Global because these configurations are not dependant on level.

Parameters

level	Level to set configuration for (el::Level).
configurationType	Type of configuration (el::ConfigurationType)
value	A string based value. Regardless of what the data type of configuration is, it will always be string from users' point of view. This is then parsed later to be used internally.

See also

Configuration::setValue(const std::string& value)

el::Level

el::ConfigurationType

Definition at line 345 of file easylogging++.cc.

References el::Global, el::base::threading::ThreadSafe::lock(), el::base::utils::RegistryWithPred< Configuration, Configuration::Predic
unsafeSet(), and unsafeSetGlobally().

8.5.3.10 setFromBase()

```
void el::Configurations::setFromBase (
            Configurations * base )
```

Sets configuration based-off an existing configurations.

Parameters

base	Pointer to existing configurations.

Definition at line 312 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::list(), el::base::threading::ThreadSafe::lock(), and set().

8.5.3.11 setGlobally() [1/2]

```
void el::Configurations::setGlobally (
            ConfigurationType configurationType,
            const std::string & value )  [inline]
```

Sets configuration for all levels.

Parameters

configurationType	Type of configuration
value	String based value

See also

Configurations::set(Level level, ConfigurationType configurationType, const std::string& value)

Definition at line 1806 of file easylogging++.h.

8.5.3.12 setGlobally() [2/2]

```
void el::Configurations::setGlobally (
            ConfigurationType configurationType,
            const std::string & value,
            bool includeGlobalLevel )  [private]
```

Sets configurations for all levels including Level::Global if includeGlobalLevel is true.

See also

Configurations::setGlobally(ConfigurationType configurationType, const std::string& value)

Definition at line 555 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), el::LevelHelper::forEachLevel(), el::Global, el::LevelHelper::kMinValid, el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and set().

8.5.3.13 setRemainingToDefault()

```
void el::Configurations::setRemainingToDefault (
            void )
```

Lets you set the remaining configurations to default.

@detail By remaining, it means that the level/type a configuration does not exist for. This function is useful when you want to minimize chances of failures, e.g, if you have a configuration file that sets configuration for all the configurations except for Enabled or not, we use this so that ENABLED is set to default i.e, true. If you dont do this explicitly (either by calling this function or by using second param in Constructor and try to access a value, an error is thrown

Definition at line 384 of file easylogging++.cc.

References el::Debug, el::Enabled, el::Error, el::Fatal, el::Filename, el::Format, el::Global, el::base::consts::kDefaultLogFile, el::base::threading::ThreadSafe::lock(), el::MaxLogFileSize, el::PerformanceTracking, el::base::utils::RegistryWithPred< Configuratio
el::SubsecondPrecision, el::ToStandardOutput, el::Trace, unsafeSetIfNotExist(), and el::Verbose.

8.5.3.14 setToDefault()

```
void el::Configurations::setToDefault (
            void )
```

Sets configurations to "factory based" configurations.

Definition at line 360 of file easylogging++.cc.

References el::Debug, el::Enabled, el::Error, el::Fatal, el::Filename, el::Format, el::base::consts::kDefaultLogFile, el::LogFlushThreshold, el::MaxLogFileSize, el::PerformanceTracking, set(), setGlobally(), el::SubsecondPrecision, el::ToFile, el::ToStandardOutput, el::Trace, and el::Verbose.

8.5.3.15 unsafeSet()

```
void el::Configurations::unsafeSet (
            Level level,
            ConfigurationType configurationType,
            const std::string & value )  [private]
```

Thread unsafe set.

Definition at line 543 of file easylogging++.cc.

References el::Global, el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::registerNew(), el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and unsafeSetGlobally().

8.5.3.16 unsafeSetGlobally()

```
void el::Configurations::unsafeSetGlobally (
             ConfigurationType configurationType,
             const std::string & value,
             bool includeGlobalLevel )  [private]
```

Sets configurations (Unsafely) for all levels including Level::Global if includeGlobalLevel is true.

See also

Configurations::setGlobally(ConfigurationType configurationType, const std::string& value)

Definition at line 567 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), el::LevelHelper::forEachLevel(), el::Global, el::LevelHelper::kMinValid, el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and unsafeSet().

8.5.3.17 unsafeSetIfNotExist()

```
void el::Configurations::unsafeSetIfNotExist (
             Level level,
             ConfigurationType configurationType,
             const std::string & value )  [private]
```

Unsafely sets configuration if does not already exist.

Definition at line 536 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and unsafeSet().

8.5.4 Friends And Related Symbol Documentation

8.5.4.1 el::Loggers

```
friend class el::Loggers  [friend]
```

Definition at line 1877 of file easylogging++.h.

8.5.5 Field Documentation

8.5.5.1 m_configurationFile

```
std::string el::Configurations::m_configurationFile  [private]
```

Definition at line 1875 of file easylogging++.h.

8.5.5.2 m_isFromFile

```
bool el::Configurations::m_isFromFile [private]
```

Definition at line 1876 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.6 el::ConfigurationStringToTypeItem Struct Reference

Data Fields

- const char ∗ configString
- ConfigurationType configType

8.6.1 Detailed Description

Definition at line 196 of file easylogging++.cc.

8.6.2 Field Documentation

8.6.2.1 configString

```
const char* el::ConfigurationStringToTypeItem::configString
```

Definition at line 197 of file easylogging++.cc.

8.6.2.2 configType

```
ConfigurationType el::ConfigurationStringToTypeItem::configType
```

Definition at line 198 of file easylogging++.cc.

The documentation for this struct was generated from the following file:

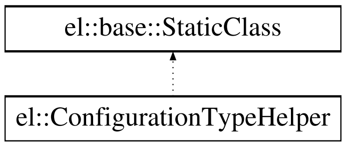
- lib/easylogging++.cc

8.7 el::ConfigurationTypeHelper Class Reference

Static class that contains helper functions for el::ConfigurationType.

```
#include <easylogging++.h>
```

Inheritance diagram for el::ConfigurationTypeHelper:



Static Public Member Functions

- static base::type::EnumType castToInt (ConfigurationType configurationType)

  Casts configuration type to int, useful for iterating through enum.
- static ConfigurationType castFromInt (base::type::EnumType c)

  Casts int(ushort) to configuration type, useful for iterating through enum.
- static const char ∗ convertToString (ConfigurationType configurationType)

  Converts configuration type to associated const char∗.
- static ConfigurationType convertFromString (const char ∗configStr)

  Converts from configStr to ConfigurationType.
- static void forEachConfigType (base::type::EnumType ∗startIndex, const std::function< bool(void)> &fn)

  Applies specified function to each configuration type starting from startIndex.

Static Public Attributes

- static const base::type::EnumType kMinValid = static_cast<base::type::EnumType>(ConfigurationType::Enabled)

  Represents minimum valid configuration type. Useful when iterating through enum.
- static const base::type::EnumType kMaxValid = static_cast<base::type::EnumType>(ConfigurationType::MaxLogFileSize)

  Represents maximum valid configuration type. This is used internally and you should not need it.

8.7.1 Detailed Description

Static class that contains helper functions for el::ConfigurationType.

Definition at line 665 of file easylogging++.h.

8.7.2 Member Function Documentation

8.7.2.1 castFromInt()

```
static ConfigurationType el::ConfigurationTypeHelper::castFromInt (
            base::type::EnumType c )  [inline], [static]
```

Casts int(ushort) to configuration type, useful for iterating through enum.

Definition at line 676 of file easylogging++.h.

8.7.2.2 castToInt()

```
static base::type::EnumType el::ConfigurationTypeHelper::castToInt (
            ConfigurationType configurationType )  [inline], [static]
```

Casts configuration type to int, useful for iterating through enum.

Definition at line 672 of file easylogging++.h.

8.7.2.3 convertFromString()

```
ConfigurationType el::ConfigurationTypeHelper::convertFromString (
             const char * configStr )  [static]
```

Converts from configStr to ConfigurationType.

Parameters

configStr	Upper case string based configuration type. Lower case is also valid but providing upper case is recommended.

Definition at line 214 of file easylogging++.cc.

References el::configStringToTypeMap, el::ConfigurationStringToTypeItem::configType, el::base::utils::Str::cStringCaseEq(), and el::Unknown.

8.7.2.4 convertToString()

```
const char * el::ConfigurationTypeHelper::convertToString (
             ConfigurationType configurationType )  [static]
```

Converts configuration type to associated const char*.

Returns

Upper case string based configuration type.

Definition at line 182 of file easylogging++.cc.

References el::Enabled, el::Filename, el::Format, el::LogFlushThreshold, el::MaxLogFileSize, el::PerformanceTracking, el::SubsecondPrecision, el::ToFile, and el::ToStandardOutput.

8.7.2.5 forEachConfigType()

```
void el::ConfigurationTypeHelper::forEachConfigType (
             base::type::EnumType * startIndex,
             const std::function< bool(void)> & fn )  [inline], [static]
```

Applies specified function to each configuration type starting from startIndex.

Parameters

startIndex	initial value to start the iteration from. This is passed by pointer and is left-shifted so this can be used inside function (fn) to represent current configuration type.
fn	function to apply with each configuration type. This bool represent whether or not to stop iterating through configurations.

Definition at line 223 of file easylogging++.cc.

References kMaxValid.

8.7.3 Field Documentation

8.7.3.1 kMaxValid

```
const base::type::EnumType el::ConfigurationTypeHelper::kMaxValid = static_cast<base::type::EnumType>(Configu
[static]
```

Represents maximum valid configuration type. This is used internally and you should not need it.

Definition at line 670 of file easylogging++.h.

8.7.3.2 kMinValid

```
const base::type::EnumType el::ConfigurationTypeHelper::kMinValid = static_cast<base::type::EnumType>(Configu
[static]
```

Represents minimum valid configuration type. Useful when iterating through enum.

Definition at line 668 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.8 el::base::debug::CrashHandler Class Reference

```
#include <easylogging++.h>
```

Public Member Functions

- CrashHandler (bool)

8.8.1 Detailed Description

Definition at line 3622 of file easylogging++.h.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 CrashHandler()

```
el::base::debug::CrashHandler::CrashHandler (
            bool   )  [inline], [explicit]
```

Definition at line 3624 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.9 el::CustomFormatSpecifier Class Reference

User-provided custom format specifier.

`#include <easylogging++.h>`

Public Member Functions

- CustomFormatSpecifier (const char ∗formatSpecifier, const FormatSpecifierValueResolver &resolver)
- const char ∗ formatSpecifier (void) const
- const FormatSpecifierValueResolver & resolver (void) const
- bool operator== (const char ∗formatSpecifier)

Private Attributes

- const char ∗ m_formatSpecifier
- FormatSpecifierValueResolver m_resolver

8.9.1 Detailed Description

User-provided custom format specifier.

See also

el::Helpers::installCustomFormatSpecifier

FormatSpecifierValueResolver

Definition at line 1646 of file easylogging++.h.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 CustomFormatSpecifier()

```
el::CustomFormatSpecifier::CustomFormatSpecifier (
             const char * formatSpecifier,
             const FormatSpecifierValueResolver & resolver )  [inline]
```

Definition at line 1648 of file easylogging++.h.

8.9.3 Member Function Documentation

8.9.3.1 formatSpecifier()

```
const char * el::CustomFormatSpecifier::formatSpecifier (
             void  ) const  [inline]
```

Definition at line 1650 of file easylogging++.h.

8.9.3.2 operator==()

```
bool el::CustomFormatSpecifier::operator== (
             const char * formatSpecifier )  [inline]
```

Definition at line 1656 of file easylogging++.h.

8.9.3.3 resolver()

```
const FormatSpecifierValueResolver & el::CustomFormatSpecifier::resolver (
             void  ) const  [inline]
```

Definition at line 1653 of file easylogging++.h.

8.9.4 Field Documentation

8.9.4.1 m_formatSpecifier

```
const char* el::CustomFormatSpecifier::m_formatSpecifier  [private]
```

Definition at line 1661 of file easylogging++.h.

8.9.4.2 m_resolver

```
FormatSpecifierValueResolver el::CustomFormatSpecifier::m_resolver  [private]
```

Definition at line 1662 of file easylogging++.h.

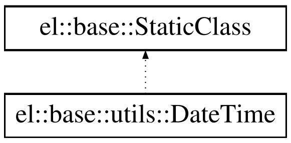
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.10 el::base::utils::DateTime Class Reference

Contains utilities for cross-platform date/time. This class make use of el::base::utils::Str.

Inheritance diagram for el::base::utils::DateTime:



Static Public Member Functions

- static void gettimeofday (struct timeval ∗tv)

  Cross platform gettimeofday for Windows and unix platform. This can be used to determine current microsecond.
- static std::string getDateTime (const char ∗format, const base::SubsecondPrecision ∗ssPrec)

  Gets current date and time with a subsecond part.
- static std::string timevalToString (struct timeval tval, const char ∗format, const el::base::SubsecondPrecision ∗ssPrec)

  Converts timeval (struct from ctime) to string using specified format and subsecond precision.
- static base::type::string_t formatTime (unsigned long long time, base::TimestampUnit timestampUnit)

  Formats time to get unit accordingly, units like second if > 1000 or minutes if > 60000 etc.
- static unsigned long long getTimeDifference (const struct timeval &endTime, const struct timeval &startTime, base::TimestampUnit timestampUnit)

  Gets time difference in milli/micro second depending on timestampUnit.
- static struct::tm ∗ buildTimeInfo (struct timeval ∗currTime, struct ::tm ∗timeInfo)

Static Private Member Functions

- static char ∗ parseFormat (char ∗buf, std::size_t bufSz, const char ∗format, const struct tm ∗tInfo, std::size_t msec, const base::SubsecondPrecision ∗ssPrec)

8.10.1 Detailed Description

Contains utilities for cross-platform date/time. This class make use of el::base::utils::Str.

Definition at line 1179 of file easylogging++.h.

8.10.2 Member Function Documentation

8.10.2.1 buildTimeInfo()

```
struct::tm ∗ el::base::utils::DateTime::buildTimeInfo (
            struct timeval ∗ currTime,
            struct ::tm ∗ timeInfo )  [static]
```

Definition at line 1225 of file easylogging++.cc.

References ELPP_UNUSED, elpptime, elpptime_r, and elpptime_s.

8.10.2.2 formatTime()

```
base::type::string_t el::base::utils::DateTime::formatTime (
            unsigned long long time,
            base::TimestampUnit timestampUnit )  [static]
```

Formats time to get unit accordingly, units like second if $>$ 1000 or minutes if $>$ 60000 etc.

Definition at line 1194 of file easylogging++.cc.

References el::base::consts::kTimeFormats, and el::base::consts::kTimeFormatsCount.

8.10.2.3 getDateTime()

```
std::string el::base::utils::DateTime::getDateTime (
            const char * format,
            const base::SubsecondPrecision * ssPrec )  [static]
```

Gets current date and time with a subsecond part.

Parameters

format	User provided date/time format
ssPrec	A pointer to base::SubsecondPrecision from configuration (non-null)

Returns

string based date time in specified format.

Definition at line 1177 of file easylogging++.cc.

References gettimeofday(), and timevalToString().

8.10.2.4 getTimeDifference()

```
unsigned long long el::base::utils::DateTime::getTimeDifference (
            const struct timeval & endTime,
            const struct timeval & startTime,
            base::TimestampUnit timestampUnit )  [static]
```

Gets time difference in milli/micro second depending on timestampUnit.

Definition at line 1212 of file easylogging++.cc.

References el::base::Microsecond.

8.10.2.5 gettimeofday()

```
void el::base::utils::DateTime::gettimeofday (
            struct timeval * tv )  [static]
```

Cross platform gettimeofday for Windows and unix platform. This can be used to determine current microsecond.

@detail For unix system it uses gettimeofday(timeval∗, timezone∗) and for Windows, a separate implementation is provided

Parameters

in,out	*tv*	Pointer that gets updated

Definition at line 1150 of file easylogging++.cc.

References gettimeofday().

8.10.2.6 parseFormat()

```
char * el::base::utils::DateTime::parseFormat (
            char * buf,
            std::size_t bufSz,
            const char * format,
            const struct tm * tInfo,
            std::size_t msec,
            const base::SubsecondPrecision * ssPrec )  [static], [private]
```

Definition at line 1251 of file easylogging++.cc.

References el::base::utils::Str::addToBuff(), el::base::utils::Str::convertAndAddToBuff(), el::base::consts::kAm, el::base::consts::kDays, el::base::consts::kDaysAbbrev, el::base::consts::kFormatSpecifierChar, el::base::consts::kMonths, el::base::consts::kMonthsAbbrev, el::base::consts::kPm, and el::base::consts::kYearBase.

8.10.2.7 timevalToString()

```
std::string el::base::utils::DateTime::timevalToString (
            struct timeval tval,
            const char * format,
            const el::base::SubsecondPrecision * ssPrec )  [static]
```

Converts timeval (struct from ctime) to string using specified format and subsecond precision.

Definition at line 1183 of file easylogging++.cc.

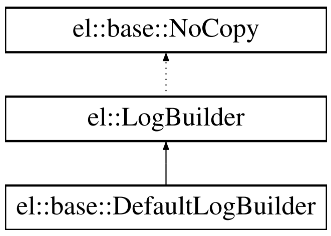
References buildTimeInfo(), and parseFormat().

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.11 el::base::DefaultLogBuilder Class Reference

Inheritance diagram for el::base::DefaultLogBuilder:



Public Member Functions

- base::type::string_t build (const LogMessage ∗logMessage, bool appendNewLine) const

Public Member Functions inherited from el::LogBuilder

- LogBuilder ()
- virtual ∼LogBuilder (void)
- void convertToColoredOutput (base::type::string_t ∗logLine, Level level)

8.11.1 Detailed Description

Definition at line 2765 of file easylogging++.h.

8.11.2 Member Function Documentation

8.11.2.1 build()

```
base::type::string_t el::base::DefaultLogBuilder::build (
            const LogMessage * logMessage,
            bool appendNewLine ) const  [virtual]
```

Implements el::LogBuilder.

Definition at line 2392 of file easylogging++.cc.

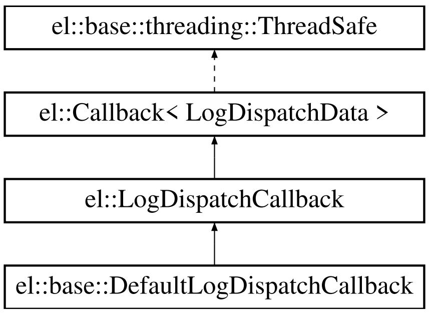
References el::base::utils::Str::addToBuff(), el::base::AppName, el::base::utils::File::buildBaseFilename(), el::base::utils::File::buildStrippedFilename(), el::base::utils::Str::clearBuff(), el::base::utils::Str::convertAndAddToBuff(), el::base::DateTime, el::base::LogFormat::dateTimeFormat(), ELPP, ELPP_LITERAL, ELPP_UNUSED, el::base::File, el::LogMessage::file(), el::base::FileBase, el::base::LogFormat::format(), el::LogMessage::func(), el::base::Function, el::base::threading::getCurrentThreadId(), el::base::utils::DateTime::getDateTime(), el::base::LogFormat::hasFlag(), el::base::consts::kAppNameFormatSpecifier, el::base::consts::kDateTimeFormatSpecifier, el::base::consts::kLogFileBaseFormatSpe
el::base::consts::kLogFileFormatSpecifier, el::base::consts::kLogFunctionFormatSpecifier, el::base::consts::kLogLineFormatSpecifier
el::base::consts::kLogLocationFormatSpecifier, el::base::consts::kMessageFormatSpecifier, el::base::consts::kSourceFilenameMaxL
el::base::consts::kSourceLineMaxLength, el::base::consts::kThreadIdFormatSpecifier, el::base::consts::kVerboseLevelFormatSpecifie
el::LogMessage::level(), el::base::Line, el::LogMessage::line(), el::base::Location, el::base::TypedConfigurations::logFormat(), el::LogMessage::logger(), el::base::LogMessage, el::LogMessage::message(), el::Logger::parentApplicationName(), el::base::utils::Str::replaceFirstWithEscape(), el::base::TypedConfigurations::subsecondPrecision(), el::base::ThreadId, el::Logger::typedConfigurations(), el::Verbose, el::base::VerboseLevel, and el::LogMessage::verboseLevel().

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.12 el::base::DefaultLogDispatchCallback Class Reference

Inheritance diagram for el::base::DefaultLogDispatchCallback:


el::base::threading::ThreadSafe
el::Callback< LogDispatchData >
el::LogDispatchCallback
el::base::DefaultLogDispatchCallback

Protected Member Functions

- void handle (const LogDispatchData ∗data)

Protected Member Functions inherited from el::LogDispatchCallback

- base::threading::Mutex & fileHandle (const LogDispatchData ∗data)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Member Functions

- void dispatch (base::type::string_t &&logLine)

Private Attributes

- const LogDispatchData ∗ m_data

Additional Inherited Members

Public Member Functions inherited from el::Callback< LogDispatchData >

- Callback (void)
- bool enabled (void) const
- void setEnabled (bool enabled)

8.12.1 Detailed Description

Definition at line 2726 of file easylogging++.h.

8.12.2 Member Function Documentation

8.12.2.1 dispatch()

```
void el::base::DefaultLogDispatchCallback::dispatch (
            base::type::string_t && logLine )  [private]
```

Definition at line 2215 of file easylogging++.cc.

References el::ColoredTerminalOutput, el::LogBuilder::convertToColoredOutput(), el::LevelHelper::convertToString(), el::Debug, el::LogDispatchData::dispatchAction(), ELPP, ELPP_COUT, ELPP_COUT_LINE, ELPP_INTERNAL_ERROR, el::Error, el::Fatal, el::base::TypedConfigurations::filename(), el::base::TypedConfigurations::fileStream(), el::Logger::flush(), el::Logger::id(), el::ImmediateFlush, el::Info, el::Logger::isFlushNeeded(), el::LogMessage::level(), el::Logger::logBuilder(), el::LogMessage::logger(), el::LogDispatchData::logMessage(), m_data, el::Logger::m_typedConfigurations, el::base::NormalLog, el::base::SysLog, el::base::TypedConfigurations::toFile(), el::base::TypedConfigurations::toStandardOutput(), el::Warning, and el::base::utils::Str::wcharPtrToCharPtr().

8.12.2.2 handle()

```
void el::base::DefaultLogDispatchCallback::handle (
            const LogDispatchData * data )  [protected], [virtual]
```

Reimplemented from el::LogDispatchCallback.

Definition at line 2205 of file easylogging++.cc.

References el::LogBuilder::build(), dispatch(), el::LogDispatchData::dispatchAction(), el::LogDispatchCallback::fileHandle(), el::LogDispatchCallback::handle(), el::Logger::logBuilder(), el::LogMessage::logger(), el::LogDispatchData::logMessage(), m_data, and el::base::NormalLog.

8.12.3 Field Documentation

8.12.3.1 m_data

```
const LogDispatchData* el::base::DefaultLogDispatchCallback::m_data  [private]
```

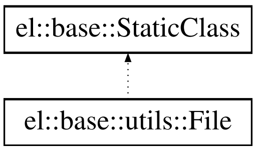
Definition at line 2730 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.13 el::base::utils::File Class Reference

Inheritance diagram for el::base::utils::File:



Static Public Member Functions

- static base::type::fstream_t ∗ newFileStream (const std::string &filename)

    Creates new out file stream for specified filename.
- static std::size_t getSizeOfFile (base::type::fstream_t ∗fs)

    Gets size of file provided in stream.
- static bool pathExists (const char ∗path, bool considerFile=false)

    Determines whether or not provided path exist in current file system.
- static bool createPath (const std::string &path)

    Creates specified path on file system.
- static std::string extractPathFromFilename (const std::string &fullPath, const char ∗separator=base::consts::kFilePathSeparator)

    Extracts path of filename with leading slash.
- static void buildStrippedFilename (const char ∗filename, char buff[ ], std::size_t limit=base::consts::kSourceFilenameMaxLength

    builds stripped filename and puts it in buff
- static void buildBaseFilename (const std::string &fullPath, char buff[ ], std::size_t limit=base::consts::kSourceFilenameMaxLengt
const char ∗separator=base::consts::kFilePathSeparator)

    builds base filename and puts it in buff

8.13.1 Detailed Description

Definition at line 1039 of file easylogging++.h.

8.13.2 Member Function Documentation

8.13.2.1 buildBaseFilename()

```
void el::base::utils::File::buildBaseFilename (
            const std::string & fullPath,
            char buff[],
            std::size_t limit = base::consts::kSourceFilenameMaxLength,
            const char * separator = base::consts::kFilePathSeparator )  [static]
```

builds base filename and puts it in buff

Definition at line 841 of file easylogging++.cc.

References STRCAT.

8.13.2.2 buildStrippedFilename()

```
void el::base::utils::File::buildStrippedFilename (
             const char * filename,
             char buff[],
             std::size_t limit = base::consts::kSourceFilenameMaxLength )  [static]
```

builds stripped filename and puts it in buff

Definition at line 829 of file easylogging++.cc.

References STRCAT.

8.13.2.3 createPath()

```
bool el::base::utils::File::createPath (
             const std::string & path )  [static]
```

Creates specified path on file system.

Parameters

path	Path to create.

Definition at line 778 of file easylogging++.cc.

References ELPP_INTERNAL_ERROR, ELPP_UNUSED, el::base::utils::AbstractRegistry< T_Ptr, Container >::empty(), el::base::consts::kFilePathSeparator, pathExists(), and STRTOK.

8.13.2.4 extractPathFromFilename()

```
std::string el::base::utils::File::extractPathFromFilename (
             const std::string & fullPath,
             const char * separator = base::consts::kFilePathSeparator )  [static]
```

Extracts path of filename with leading slash.

Definition at line 818 of file easylogging++.cc.

8.13.2.5 getSizeOfFile()

```
std::size_t el::base::utils::File::getSizeOfFile (
             base::type::fstream_t * fs )  [static]
```

Gets size of file provided in stream.

Definition at line 751 of file easylogging++.cc.

8.13.2.6 newFileStream()

```
base::type::fstream_t * el::base::utils::File::newFileStream (
            const std::string & filename )  [static]
```

Creates new out file stream for specified filename.

Returns

Pointer to newly created fstream or nullptr

Definition at line 727 of file easylogging++.cc.

References ELPP_INTERNAL_ERROR, and el::base::utils::safeDelete().

8.13.2.7 pathExists()

```
bool el::base::utils::File::pathExists (
            const char * path,
            bool considerFile = false )  [static]
```

Determines whether or not provided path exist in current file system.

Definition at line 761 of file easylogging++.cc.

References ELPP_UNUSED.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.14 std::hash< el::Level > Struct Reference

```
#include <easylogging++.h>
```

Public Member Functions

- std::size_t operator() (const el::Level &l) const

8.14.1 Detailed Description

Definition at line 595 of file easylogging++.h.

8.14.2 Member Function Documentation

8.14.2.1 operator()()

```
std::size_t std::hash< el::Level >::operator() (
            const el::Level & l ) const  [inline]
```

Definition at line 597 of file easylogging++.h.

The documentation for this struct was generated from the following file:

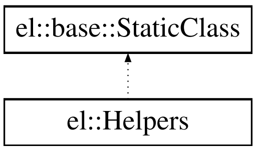
- lib/easylogging++.h

8.15 el::Helpers Class Reference

Static helpers for developers.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Helpers:



Static Public Member Functions

- static void setStorage (base::type::StoragePointer storage)

  Shares logging repository (base::Storage)
- static base::type::StoragePointer storage ()
- static void setArgs (int argc, char ∗∗argv)

  Sets application arguments and figures out whats active for logging and whats not.
- static void setArgs (int argc, const char ∗∗argv)

  Sets application arguments and figures out whats active for logging and whats not.
- static void setThreadName (const std::string &name)

  Sets thread name for current thread. Requires std::thread.
- static std::string getThreadName ()
- static void installPreRollOutCallback (const PreRollOutCallback &callback)

  Installs pre rollout callback, this callback is triggered when log file is about to be rolled out (can be useful for backing up)
- static void uninstallPreRollOutCallback (void)

  Uninstalls pre rollout callback.
- template<typename T >
  static bool installLogDispatchCallback (const std::string &id)

  Installs post log dispatch callback, this callback is triggered when log is dispatched.
- template<typename T >
  static void uninstallLogDispatchCallback (const std::string &id)

  Uninstalls log dispatch callback.

- template<typename T >
  static T ∗ logDispatchCallback (const std::string &id)
- template<typename T >
  static std::string convertTemplateToStdString (const T &templ)

  Converts template to std::string - useful for loggable classes to log containers within log(std::ostream&) const.
- static const el::base::utils::CommandLineArgs ∗ commandLineArgs (void)

  Returns command line arguments (pointer) provided to easylogging++.
- static void reserveCustomFormatSpecifiers (std::size_t size)

  Reserve space for custom format specifiers for performance.
- static void installCustomFormatSpecifier (const CustomFormatSpecifier &customFormatSpecifier)

  Installs user defined format specifier and handler.
- static bool uninstallCustomFormatSpecifier (const char ∗formatSpecifier)

  Uninstalls user defined format specifier and handler.
- static bool hasCustomFormatSpecifier (const char ∗formatSpecifier)

  Returns true if custom format specifier is installed.
- static void validateFileRolling (Logger ∗logger, Level level)

8.15.1 Detailed Description

Static helpers for developers.

Definition at line 3653 of file easylogging++.h.

8.15.2 Member Function Documentation

8.15.2.1 commandLineArgs()

```
static const el::base::utils::CommandLineArgs * el::Helpers::commandLineArgs (
            void  )  [inline], [static]
```

Returns command line arguments (pointer) provided to easylogging++.

Definition at line 3757 of file easylogging++.h.

References ELPP.

8.15.2.2 convertTemplateToStdString()

```
template<typename T >
static std::string el::Helpers::convertTemplateToStdString (
            const T & templ )  [inline], [static]
```

Converts template to std::string - useful for loggable classes to log containers within log(std::ostream&) const.

Definition at line 3737 of file easylogging++.h.

References el::base::threading::ThreadSafe::acquireLock(), ELPP, ELPP_LITERAL, el::base::MessageBuilder::initialize(), el::base::consts::kDefaultLoggerId, el::base::threading::ThreadSafe::releaseLock(), and el::Logger::stream().

8.15.2.3 getThreadName()

```
static std::string el::Helpers::getThreadName ( )  [inline], [static]
```

Definition at line 3675 of file easylogging++.h.

References ELPP.

8.15.2.4 hasCustomFormatSpecifier()

```
static bool el::Helpers::hasCustomFormatSpecifier (
             const char * formatSpecifier )  [inline], [static]
```

Returns true if custom format specifier is installed.

Definition at line 3774 of file easylogging++.h.

References ELPP.

8.15.2.5 installCustomFormatSpecifier()

```
static void el::Helpers::installCustomFormatSpecifier (
             const CustomFormatSpecifier & customFormatSpecifier )  [inline], [static]
```

Installs user defined format specifier and handler.

Definition at line 3766 of file easylogging++.h.

References ELPP.

8.15.2.6 installLogDispatchCallback()

```
template<typename T >
static bool el::Helpers::installLogDispatchCallback (
             const std::string & id )  [inline], [static]
```

Installs post log dispatch callback, this callback is triggered when log is dispatched.

Definition at line 3707 of file easylogging++.h.

References ELPP.

8.15.2.7 installPreRollOutCallback()

```
static void el::Helpers::installPreRollOutCallback (
             const PreRollOutCallback & callback )  [inline], [static]
```

Installs pre rollout callback, this callback is triggered when log file is about to be rolled out (can be useful for backing up)

Definition at line 3698 of file easylogging++.h.

8.15.2.8 logDispatchCallback()

```
template<typename T >
static T * el::Helpers::logDispatchCallback (
             const std::string & id )  [inline], [static]
```

Definition at line 3716 of file easylogging++.h.

References ELPP.

8.15.2.9 reserveCustomFormatSpecifiers()

```
static void el::Helpers::reserveCustomFormatSpecifiers (
             std::size_t size )  [inline], [static]
```

Reserve space for custom format specifiers for performance.

See also

std::vector::reserve

Definition at line 3762 of file easylogging++.h.

References ELPP.

8.15.2.10 setArgs() [1/2]

```
static void el::Helpers::setArgs (
             int argc,
             char ** argv )  [inline], [static]
```

Sets application arguments and figures out whats active for logging and whats not.

Definition at line 3664 of file easylogging++.h.

References ELPP.

8.15.2.11 setArgs() [2/2]

```
static void el::Helpers::setArgs (
             int argc,
             const char ** argv )  [inline], [static]
```

Sets application arguments and figures out whats active for logging and whats not.

Definition at line 3668 of file easylogging++.h.

References ELPP.

8.15.2.12 setStorage()

```
static void el::Helpers::setStorage (
            base::type::StoragePointer storage )  [inline], [static]
```

Shares logging repository (base::Storage)

Definition at line 3656 of file easylogging++.h.

References ELPP.

8.15.2.13 setThreadName()

```
static void el::Helpers::setThreadName (
            const std::string & name )  [inline], [static]
```

Sets thread name for current thread. Requires std::thread.

Definition at line 3672 of file easylogging++.h.

References ELPP.

8.15.2.14 storage()

```
static base::type::StoragePointer el::Helpers::storage ( )  [inline], [static]
```

Returns

Main storage repository

Definition at line 3660 of file easylogging++.h.

References ELPP.

8.15.2.15 uninstallCustomFormatSpecifier()

```
static bool el::Helpers::uninstallCustomFormatSpecifier (
            const char * formatSpecifier )  [inline], [static]
```

Uninstalls user defined format specifier and handler.

Definition at line 3770 of file easylogging++.h.

References ELPP.

8.15.2.16 uninstallLogDispatchCallback()

```
template<typename T >
static void el::Helpers::uninstallLogDispatchCallback (
            const std::string & id )  [inline], [static]
```

Uninstalls log dispatch callback.

Definition at line 3712 of file easylogging++.h.

References ELPP.

8.15.2.17 uninstallPreRollOutCallback()

```
static void el::Helpers::uninstallPreRollOutCallback (
            void  )  [inline], [static]
```

Uninstalls pre rollout callback.

Definition at line 3702 of file easylogging++.h.

References ELPP.

8.15.2.18 validateFileRolling()

```
static void el::Helpers::validateFileRolling (
            Logger * logger,
            Level level )  [inline], [static]
```

Definition at line 3777 of file easylogging++.h.

References ELPP, el::Logger::m_typedConfigurations, and el::base::TypedConfigurations::validateFileRolling().

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.16 el::base::HitCounter Class Reference

Class that keeps record of current line hit for occasional logging.

Data Structures

- class Predicate

Public Member Functions

- HitCounter (void)
- HitCounter (const char ∗filename, base::type::LineNumber lineNumber)
- HitCounter (const HitCounter &hitCounter)
- HitCounter & operator= (const HitCounter &hitCounter)
- virtual ∼HitCounter (void)
- void resetLocation (const char ∗filename, base::type::LineNumber lineNumber)

  Resets location of current hit counter.
- void validateHitCounts (std::size_t n)

  Validates hit counts and resets it if necessary.
- const char ∗ filename (void) const
- base::type::LineNumber lineNumber (void) const
- std::size_t hitCounts (void) const
- void increment (void)

Private Attributes

- const char ∗ m_filename
- base::type::LineNumber m_lineNumber
- std::size_t m_hitCounts

8.16.1 Detailed Description

Class that keeps record of current line hit for occasional logging.

Definition at line 2033 of file easylogging++.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 HitCounter() `[1/3]`

```
el::base::HitCounter::HitCounter (
            void ) [inline]
```

Definition at line 2035 of file easylogging++.h.

8.16.2.2 HitCounter() `[2/3]`

```
el::base::HitCounter::HitCounter (
            const char * filename,
            base::type::LineNumber lineNumber ) [inline]
```

Definition at line 2041 of file easylogging++.h.

8.16.2.3 HitCounter() [3/3]

```
el::base::HitCounter::HitCounter (
            const HitCounter & hitCounter )  [inline]
```

Definition at line 2047 of file easylogging++.h.

8.16.2.4 ∼HitCounter()

```
virtual el::base::HitCounter::∼HitCounter (
            void  )  [inline], [virtual]
```

Definition at line 2062 of file easylogging++.h.

8.16.3 Member Function Documentation

8.16.3.1 filename()

```
const char * el::base::HitCounter::filename (
            void  ) const  [inline]
```

Definition at line 2079 of file easylogging++.h.

8.16.3.2 hitCounts()

```
std::size_t el::base::HitCounter::hitCounts (
            void  ) const  [inline]
```

Definition at line 2087 of file easylogging++.h.

8.16.3.3 increment()

```
void el::base::HitCounter::increment (
            void  )  [inline]
```

Definition at line 2091 of file easylogging++.h.

8.16.3.4 lineNumber()

```
base::type::LineNumber el::base::HitCounter::lineNumber (
            void  ) const  [inline]
```

Definition at line 2083 of file easylogging++.h.

8.16.3.5 operator=()

```
HitCounter & el::base::HitCounter::operator= (
             const HitCounter & hitCounter )  [inline]
```

Definition at line 2053 of file easylogging++.h.

References m_filename, m_hitCounts, and m_lineNumber.

8.16.3.6 resetLocation()

```
void el::base::HitCounter::resetLocation (
             const char * filename,
             base::type::LineNumber lineNumber )  [inline]
```

Resets location of current hit counter.

Definition at line 2066 of file easylogging++.h.

8.16.3.7 validateHitCounts()

```
void el::base::HitCounter::validateHitCounts (
             std::size_t n )  [inline]
```

Validates hit counts and resets it if necessary.

Definition at line 2072 of file easylogging++.h.

8.16.4 Field Documentation

8.16.4.1 m_filename

```
const char* el::base::HitCounter::m_filename  [private]
```

Definition at line 2113 of file easylogging++.h.

8.16.4.2 m_hitCounts

```
std::size_t el::base::HitCounter::m_hitCounts  [private]
```

Definition at line 2115 of file easylogging++.h.

8.16.4.3 m_lineNumber

```
base::type::LineNumber el::base::HitCounter::m_lineNumber  [private]
```

Definition at line 2114 of file easylogging++.h.

The documentation for this class was generated from the following file:

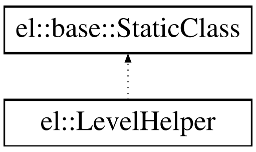
- lib/easylogging++.h

8.17 el::LevelHelper Class Reference

Static class that contains helper functions for el::Level.

```
#include <easylogging++.h>
```

Inheritance diagram for el::LevelHelper:



Static Public Member Functions

- static base::type::EnumType castToInt (Level level)

  Casts level to int, useful for iterating through enum.
- static Level castFromInt (base::type::EnumType l)

  Casts int(ushort) to level, useful for iterating through enum.
- static const char ∗ convertToString (Level level)

  Converts level to associated const char∗.
- static Level convertFromString (const char ∗levelStr)

  Converts from levelStr to Level.
- static void forEachLevel (base::type::EnumType ∗startIndex, const std::function< bool(void)> &fn)

  Applies specified function to each level starting from startIndex.

Static Public Attributes

- static const base::type::EnumType kMinValid = static_cast<base::type::EnumType>(Level::Trace)

  Represents minimum valid level. Useful when iterating through enum.
- static const base::type::EnumType kMaxValid = static_cast<base::type::EnumType>(Level::Info)

  Represents maximum valid level. This is used internally and you should not need it.

8.17.1 Detailed Description

Static class that contains helper functions for el::Level.

Definition at line 604 of file easylogging++.h.

8.17.2 Member Function Documentation

8.17.2.1 castFromInt()

```
static Level el::LevelHelper::castFromInt (
            base::type::EnumType l ) [inline], [static]
```

Casts int(ushort) to level, useful for iterating through enum.

Definition at line 615 of file easylogging++.h.

8.17.2.2 castToInt()

```
static base::type::EnumType el::LevelHelper::castToInt (
            Level level )  [inline], [static]
```

Casts level to int, useful for iterating through enum.

Definition at line 611 of file easylogging++.h.

8.17.2.3 convertFromString()

```
Level el::LevelHelper::convertFromString (
            const char * levelStr )  [static]
```

Converts from levelStr to Level.

Parameters

levelStr	Upper case string based level. Lower case is also valid but providing upper case is recommended.

Definition at line 161 of file easylogging++.cc.

References el::base::utils::Str::cStringCaseEq(), el::StringToLevelItem::level, el::stringToLevelMap, and el::Unknown.

8.17.2.4 convertToString()

```
const char * el::LevelHelper::convertToString (
            Level level )  [static]
```

Converts level to associated const char*.

Returns

Upper case string based level.

Definition at line 132 of file easylogging++.cc.

References el::Debug, el::Error, el::Fatal, el::Global, el::Info, el::Trace, el::Verbose, and el::Warning.

8.17.2.5 forEachLevel()

```
void el::LevelHelper::forEachLevel (
            base::type::EnumType * startIndex,
            const std::function< bool(void)> & fn )  [static]
```

Applies specified function to each level starting from startIndex.

Parameters

startIndex	initial value to start the iteration from. This is passed as pointer and is left-shifted so this can be used inside function (fn) to represent current level.
fn	function to apply with each level. This bool represent whether or not to stop iterating through levels.

Definition at line 170 of file easylogging++.cc.

References kMaxValid.

8.17.3 Field Documentation

8.17.3.1 kMaxValid

```
const base::type::EnumType el::LevelHelper::kMaxValid = static_cast<base::type::EnumType>(Level::Info)
[static]
```

Represents maximum valid level. This is used internally and you should not need it.

Definition at line 609 of file easylogging++.h.

8.17.3.2 kMinValid

```
const base::type::EnumType el::LevelHelper::kMinValid = static_cast<base::type::EnumType>(Level::Trace)
[static]
```

Represents minimum valid level. Useful when iterating through enum.

Definition at line 607 of file easylogging++.h.

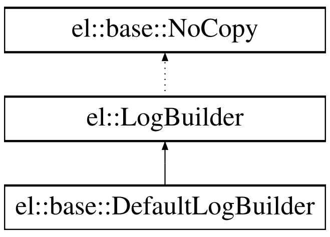
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.18 el::LogBuilder Class Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::LogBuilder:



Public Member Functions

- LogBuilder ()
- virtual ∼LogBuilder (void)
- virtual base::type::string_t build (const LogMessage ∗logMessage, bool appendNewLine) const =0
- void convertToColoredOutput (base::type::string_t ∗logLine, Level level)

Private Attributes

- bool m_termSupportsColor

Friends

- class el::base::DefaultLogDispatchCallback

Additional Inherited Members

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.18.1 Detailed Description

Definition at line 2197 of file easylogging++.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 LogBuilder()

```
el::LogBuilder::LogBuilder ( )  [inline]
```

Definition at line 2199 of file easylogging++.h.

8.18.2.2 ∼LogBuilder()

```
virtual el::LogBuilder::∼LogBuilder (
             void  )  [inline], [virtual]
```

Definition at line 2200 of file easylogging++.h.

References ELPP_INTERNAL_INFO.

8.18.3 Member Function Documentation

8.18.3.1 build()

```
virtual base::type::string_t el::LogBuilder::build (
            const LogMessage * logMessage,
            bool appendNewLine ) const  [pure virtual]
```

Implemented in el::base::DefaultLogBuilder.

8.18.3.2 convertToColoredOutput()

```
void el::LogBuilder::convertToColoredOutput (
            base::type::string_t * logLine,
            Level level )
```

Definition at line 581 of file easylogging++.cc.

References el::Debug, ELPP_LITERAL, el::Error, el::Fatal, el::Info, el::base::utils::RegistryWithPred< Configuration, Configuration::Pr
el::Trace, and el::Warning.

8.18.4 Friends And Related Symbol Documentation

8.18.4.1 el::base::DefaultLogDispatchCallback

```
friend class el::base::DefaultLogDispatchCallback  [friend]
```

Definition at line 2207 of file easylogging++.h.

8.18.5 Field Documentation

8.18.5.1 m_termSupportsColor

```
bool el::LogBuilder::m_termSupportsColor  [private]
```

Definition at line 2206 of file easylogging++.h.

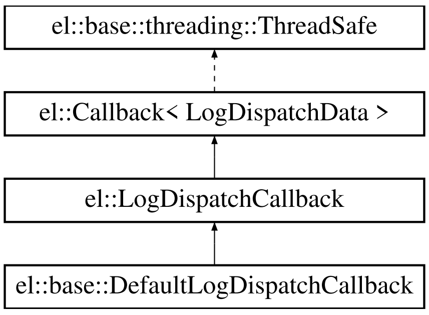
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.19 el::LogDispatchCallback Class Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::LogDispatchCallback:


el::base::threading::ThreadSafe
el::Callback< LogDispatchData >
el::LogDispatchCallback
el::base::DefaultLogDispatchCallback

Protected Member Functions

- virtual void handle (const LogDispatchData ∗data)
- base::threading::Mutex & fileHandle (const LogDispatchData ∗data)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Attributes

- std::unordered_map< std::string, std::unique_ptr< base::threading::Mutex > > m_fileLocks
- base::threading::Mutex m_fileLocksMapLock

Friends

- class base::LogDispatcher

Additional Inherited Members

Public Member Functions inherited from el::Callback< LogDispatchData >

- Callback (void)
- bool enabled (void) const
- void setEnabled (bool enabled)

8.19.1 Detailed Description

Definition at line 2180 of file easylogging++.h.

8.19.2 Member Function Documentation

8.19.2.1 fileHandle()

```
base::threading::Mutex & el::LogDispatchCallback::fileHandle (
            const LogDispatchData * data )  [protected]
```

Definition at line 2197 of file easylogging++.cc.

References el::base::TypedConfigurations::filename(), el::LogMessage::level(), el::LogMessage::logger(), el::LogDispatchData::logMessage(), m_fileLocks, and el::Logger::typedConfigurations().

8.19.2.2 handle()

```
void el::LogDispatchCallback::handle (
            const LogDispatchData * data )  [protected], [virtual]
```

Implements el::Callback< LogDispatchData >.

Reimplemented in el::base::DefaultLogDispatchCallback.

Definition at line 2194 of file easylogging++.cc.

8.19.3 Friends And Related Symbol Documentation

8.19.3.1 base::LogDispatcher

```
friend class base::LogDispatcher  [friend]
```

Definition at line 2185 of file easylogging++.h.

8.19.4 Field Documentation

8.19.4.1 m_fileLocks

```
std::unordered_map<std::string, std::unique_ptr<base::threading::Mutex> > el::LogDispatch↩
Callback::m_fileLocks  [private]
```

Definition at line 2186 of file easylogging++.h.

8.19.4.2 m_fileLocksMapLock

`base::threading::Mutex el::LogDispatchCallback::m_fileLocksMapLock [private]`

Definition at line 2187 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.20 el::LogDispatchData Class Reference

`#include <easylogging++.h>`

Public Member Functions

- LogDispatchData ()
- const LogMessage ∗ logMessage (void) const
- base::DispatchAction dispatchAction (void) const
- void setLogMessage (LogMessage ∗logMessage)
- void setDispatchAction (base::DispatchAction dispatchAction)

Private Attributes

- LogMessage ∗ m_logMessage
- base::DispatchAction m_dispatchAction

Friends

- class base::LogDispatcher

8.20.1 Detailed Description

Definition at line 2159 of file easylogging++.h.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 LogDispatchData()

`el::LogDispatchData::LogDispatchData ( ) [inline]`

Definition at line 2161 of file easylogging++.h.

8.20.3 Member Function Documentation

8.20.3.1 dispatchAction()

```
base::DispatchAction el::LogDispatchData::dispatchAction (
            void ) const [inline]
```

Definition at line 2165 of file easylogging++.h.

8.20.3.2 logMessage()

```
const LogMessage * el::LogDispatchData::logMessage (
            void ) const [inline]
```

Definition at line 2162 of file easylogging++.h.

8.20.3.3 setDispatchAction()

```
void el::LogDispatchData::setDispatchAction (
            base::DispatchAction dispatchAction ) [inline]
```

Definition at line 2171 of file easylogging++.h.

8.20.3.4 setLogMessage()

```
void el::LogDispatchData::setLogMessage (
            LogMessage * logMessage ) [inline]
```

Definition at line 2168 of file easylogging++.h.

8.20.4 Friends And Related Symbol Documentation

8.20.4.1 base::LogDispatcher

```
friend class base::LogDispatcher [friend]
```

Definition at line 2177 of file easylogging++.h.

8.20.5 Field Documentation

8.20.5.1 m_dispatchAction

```
base::DispatchAction el::LogDispatchData::m_dispatchAction [private]
```

Definition at line 2176 of file easylogging++.h.

8.20.5.2 m_logMessage

```
LogMessage* el::LogDispatchData::m_logMessage  [private]
```

Definition at line 2175 of file easylogging++.h.

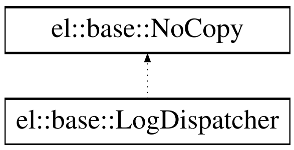
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.21 el::base::LogDispatcher Class Reference

Dispatches log messages.

Inheritance diagram for el::base::LogDispatcher:



Public Member Functions

- LogDispatcher (bool proceed, LogMessage ∗logMessage, base::DispatchAction dispatchAction)
- void dispatch (void)

Private Attributes

- bool m_proceed
- LogMessage ∗ m_logMessage
- base::DispatchAction m_dispatchAction

Additional Inherited Members

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.21.1 Detailed Description

Dispatches log messages.

Definition at line 2770 of file easylogging++.h.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 LogDispatcher()

```
el::base::LogDispatcher::LogDispatcher (
            bool proceed,
            LogMessage * logMessage,
            base::DispatchAction dispatchAction )  [inline]
```

Definition at line 2772 of file easylogging++.h.

8.21.3 Member Function Documentation

8.21.3.1 dispatch()

```
void el::base::LogDispatcher::dispatch (
            void  )
```

Definition at line 2473 of file easylogging++.cc.

References ELPP, el::Callback< T >::enabled(), el::LogDispatchCallback::handle(), el::LogMessage::level(), el::LogMessage::logger(), m_dispatchAction, m_logMessage, m_proceed, el::Logger::m_typedConfigurations, el::base::None, el::LogDispatchData::setDispatchAction(), el::LogDispatchData::setLogMessage(), el::StrictLogFileSizeCheck, and el::base::TypedConfigurations::validateFileRolling().

8.21.4 Field Documentation

8.21.4.1 m_dispatchAction

```
base::DispatchAction el::base::LogDispatcher::m_dispatchAction  [private]
```

Definition at line 2783 of file easylogging++.h.

8.21.4.2 m_logMessage

```
LogMessage* el::base::LogDispatcher::m_logMessage  [private]
```

Definition at line 2782 of file easylogging++.h.

8.21.4.3 m_proceed

```
bool el::base::LogDispatcher::m_proceed  [private]
```

Definition at line 2781 of file easylogging++.h.

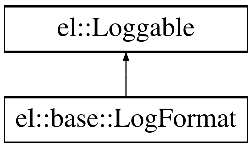
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.22 el::base::LogFormat Class Reference

Represents log format containing flags and date format. This is used internally to start initial log.

Inheritance diagram for el::base::LogFormat:



Public Member Functions

- LogFormat (void)
- LogFormat (Level level, const base::type::string_t &format)
- LogFormat (const LogFormat &logFormat)
- LogFormat (LogFormat &&logFormat)
- LogFormat & operator= (const LogFormat &logFormat)
- virtual ∼LogFormat (void)
- bool operator== (const LogFormat &other)
- void parseFromFormat (const base::type::string_t &userFormat)

  Updates format to be used while logging.
- Level level (void) const
- const base::type::string_t & userFormat (void) const
- const base::type::string_t & format (void) const
- const std::string & dateTimeFormat (void) const
- base::type::EnumType flags (void) const
- bool hasFlag (base::FormatFlags flag) const
- virtual void log (el::base::type::ostream_t &os) const

Public Member Functions inherited from el::Loggable

- virtual ∼Loggable (void)

Protected Member Functions

- virtual void updateDateFormat (std::size_t index, base::type::string_t &currFormat) ELPP_FINAL

  Updates date time format if available in currFormat.
- virtual void updateFormatSpec (void) ELPP_FINAL

  Updates level from format. This is so that we dont have to do it at log-writing-time. It uses m_format and m_level.
- void addFlag (base::FormatFlags flag)

Private Attributes

- Level m_level
- base::type::string_t m_userFormat
- base::type::string_t m_format
- std::string m_dateTimeFormat
- base::type::EnumType m_flags
- std::string m_currentUser
- std::string m_currentHost

Friends

- class el::Logger

8.22.1 Detailed Description

Represents log format containing flags and date format. This is used internally to start initial log.

Definition at line 1575 of file easylogging++.h.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 LogFormat() [1/4]

```
el::base::LogFormat::LogFormat (
             void  )
```

Definition at line 1430 of file easylogging++.cc.

8.22.2.2 LogFormat() [2/4]

```
el::base::LogFormat::LogFormat (
             Level level,
             const base::type::string_t & format )
```

Definition at line 1440 of file easylogging++.cc.

References m_userFormat, and parseFromFormat().

8.22.2.3 LogFormat() [3/4]

```
el::base::LogFormat::LogFormat (
             const LogFormat & logFormat )
```

Definition at line 1446 of file easylogging++.cc.

8.22.2.4 LogFormat() [4/4]

```
el::base::LogFormat::LogFormat (
             LogFormat && logFormat )
```

Definition at line 1456 of file easylogging++.cc.

References m_currentHost, m_currentUser, m_dateTimeFormat, m_flags, m_format, m_level, and m_userFormat.

8.22.2.5 ∼LogFormat()

```
virtual el::base::LogFormat::∼LogFormat (
            void ) [inline], [virtual]
```

Definition at line 1582 of file easylogging++.h.

8.22.3 Member Function Documentation

8.22.3.1 addFlag()

```
void el::base::LogFormat::addFlag (
            base::FormatFlags flag ) [inline], [protected]
```

Definition at line 1626 of file easylogging++.h.

8.22.3.2 dateTimeFormat()

```
const std::string & el::base::LogFormat::dateTimeFormat (
            void ) const [inline]
```

Definition at line 1601 of file easylogging++.h.

8.22.3.3 flags()

```
base::type::EnumType el::base::LogFormat::flags (
            void ) const [inline]
```

Definition at line 1605 of file easylogging++.h.

8.22.3.4 format()

```
const base::type::string_t & el::base::LogFormat::format (
            void ) const [inline]
```

Definition at line 1597 of file easylogging++.h.

8.22.3.5 hasFlag()

```
bool el::base::LogFormat::hasFlag (
            base::FormatFlags flag ) const [inline]
```

Definition at line 1609 of file easylogging++.h.

8.22.3.6 level()

```
Level el::base::LogFormat::level (
             void ) const [inline]
```

Definition at line 1589 of file easylogging++.h.

8.22.3.7 log()

```
virtual void el::base::LogFormat::log (
             el::base::type::ostream_t & os ) const [inline], [virtual]
```

Implements el::Loggable.

Definition at line 1613 of file easylogging++.h.

8.22.3.8 operator=()

```
LogFormat & el::base::LogFormat::operator= (
             const LogFormat & logFormat )
```

Definition at line 1466 of file easylogging++.cc.

References m_currentHost, m_currentUser, m_dateTimeFormat, m_flags, m_level, and m_userFormat.

8.22.3.9 operator==()

```
bool el::base::LogFormat::operator== (
             const LogFormat & other )
```

Definition at line 1478 of file easylogging++.cc.

References m_dateTimeFormat, m_flags, m_format, m_level, and m_userFormat.

8.22.3.10 parseFromFormat()

```
void el::base::LogFormat::parseFromFormat (
             const base::type::string_t & userFormat )
```

Updates format to be used while logging.

Parameters

userFormat	User provided format

Definition at line 1485 of file easylogging++.cc.

References addFlag(), el::base::AppName, el::base::DateTime, el::base::File, el::base::FileBase, el::base::Function, hasFlag(), el::base::Host, el::base::consts::kAppNameFormatSpecifier, el::base::consts::kCurrentHostFormatSpecifier, el::base::consts::kCurrentUserFormatSpecifier, el::base::consts::kDateTimeFormatSpecifier, el::base::consts::kFormatSpecifierChar, el::base::consts::kLogFileBaseFormatSpecifier, el::base::consts::kLogFileFormatSpecifier, el::base::consts::kLogFunctionFormatSpe el::base::consts::kLoggerIdFormatSpecifier, el::base::consts::kLogLineFormatSpecifier, el::base::consts::kLogLocationFormatSpecifie el::base::consts::kMessageFormatSpecifier, el::base::consts::kSeverityLevelFormatSpecifier, el::base::consts::kSeverityLevelShortFo el::base::consts::kThreadIdFormatSpecifier, el::base::consts::kVerboseLevelFormatSpecifier, el::base::Level, el::base::LevelShort, el::base::Line, el::base::Location, el::base::LoggerId, el::base::LogMessage, m_flags, m_format, el::base::ThreadId, updateDateFormat(), updateFormatSpec(), el::base::User, userFormat(), and el::base::VerboseLevel.

8.22.3.11 updateDateFormat()

```
void el::base::LogFormat::updateDateFormat (
            std::size_t index,
            base::type::string_t & currFormat )  [protected], [virtual]
```

Updates date time format if available in currFormat.

Parameters

	index	Index where datetime, date or time was found
in,out	*currFormat*	current format that is being used to format

Definition at line 1535 of file easylogging++.cc.

References el::base::DateTime, ELPP_STRLEN, hasFlag(), el::base::consts::kDateTimeFormatSpecifier, el::base::consts::kDefaultDateTimeFormat, and m_dateTimeFormat.

8.22.3.12 updateFormatSpec()

```
void el::base::LogFormat::updateFormatSpec (
            void )  [protected], [virtual]
```

Updates level from format. This is so that we dont have to do it at log-writing-time. It uses m_format and m_level.

Definition at line 1562 of file easylogging++.cc.

References el::Debug, el::Error, el::Fatal, hasFlag(), el::base::Host, el::Info, el::base::consts::kCurrentHostFormatSpecifier, el::base::consts::kCurrentUserFormatSpecifier, el::base::consts::kDebugLevelLogValue, el::base::consts::kDebugLevelShortLogValue el::base::consts::kErrorLevelLogValue, el::base::consts::kErrorLevelShortLogValue, el::base::consts::kFatalLevelLogValue, el::base::consts::kFatalLevelShortLogValue, el::base::consts::kInfoLevelLogValue, el::base::consts::kInfoLevelShortLogValue, el::base::consts::kSeverityLevelFormatSpecifier, el::base::consts::kSeverityLevelShortFormatSpecifier, el::base::consts::kTraceLevelL el::base::consts::kTraceLevelShortLogValue, el::base::consts::kVerboseLevelLogValue, el::base::consts::kVerboseLevelShortLogValu el::base::consts::kWarningLevelLogValue, el::base::consts::kWarningLevelShortLogValue, m_currentHost, m_currentUser, m_format, m_level, el::base::utils::Str::replaceFirstWithEscape(), el::Trace, el::base::User, el::Verbose, and el::Warning.

8.22.3.13 userFormat()

```
const base::type::string_t & el::base::LogFormat::userFormat (
            void ) const  [inline]
```

Definition at line 1593 of file easylogging++.h.

8.22.4 Friends And Related Symbol Documentation

8.22.4.1 el::Logger

```
friend class el::Logger  [friend]
```

Definition at line 1638 of file easylogging++.h.

8.22.5 Field Documentation

8.22.5.1 m_currentHost

```
std::string el::base::LogFormat::m_currentHost  [private]
```

Definition at line 1637 of file easylogging++.h.

8.22.5.2 m_currentUser

```
std::string el::base::LogFormat::m_currentUser  [private]
```

Definition at line 1636 of file easylogging++.h.

8.22.5.3 m_dateTimeFormat

```
std::string el::base::LogFormat::m_dateTimeFormat  [private]
```

Definition at line 1634 of file easylogging++.h.

8.22.5.4 m_flags

```
base::type::EnumType el::base::LogFormat::m_flags  [private]
```

Definition at line 1635 of file easylogging++.h.

8.22.5.5 m_format

```
base::type::string_t el::base::LogFormat::m_format  [private]
```

Definition at line 1633 of file easylogging++.h.

8.22.5.6 m_level

```
Level el::base::LogFormat::m_level  [private]
```

Definition at line 1631 of file easylogging++.h.

8.22.5.7 m_userFormat

```
base::type::string_t el::base::LogFormat::m_userFormat  [private]
```

Definition at line 1632 of file easylogging++.h.

The documentation for this class was generated from the following files:

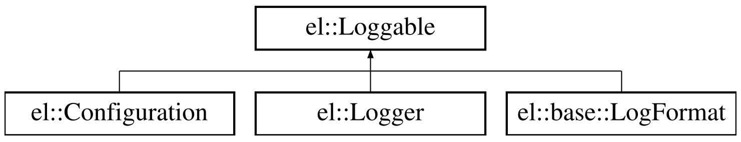
- lib/easylogging++.h
- lib/easylogging++.cc

8.23 el::Loggable Class Reference

Base of Easylogging++ friendly class.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Loggable:



Public Member Functions

- virtual ∼Loggable (void)
- virtual void log (el::base::type::ostream_t &) const =0

Friends

- el::base::type::ostream_t & operator<< (el::base::type::ostream_t &os, const Loggable &loggable)

8.23.1 Detailed Description

Base of Easylogging++ friendly class.

@detail After inheriting this class publicly, implement pure-virtual function `void log(std::ostream&) const`

Definition at line 1563 of file easylogging++.h.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 ∼Loggable()

```
virtual el::Loggable::∼Loggable (
            void  )  [inline], [virtual]
```

Definition at line 1565 of file easylogging++.h.

8.23.3 Member Function Documentation

8.23.3.1 log()

```
virtual void el::Loggable::log (
            el::base::type::ostream_t &  ) const  [pure virtual]
```

Implemented in el::base::LogFormat, el::Configuration, and el::Logger.

8.23.4 Friends And Related Symbol Documentation

8.23.4.1 operator<<

```
el::base::type::ostream_t & operator<< (
            el::base::type::ostream_t & os,
            const Loggable & loggable )  [friend]
```

Definition at line 1568 of file easylogging++.h.

The documentation for this class was generated from the following file:

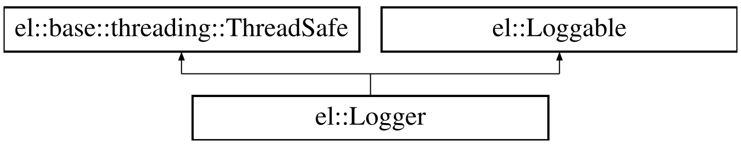
- lib/easylogging++.h

8.24 el::Logger Class Reference

Represents a logger holding ID and configurations we need to write logs.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Logger:



Public Member Functions

- Logger (const std::string &id, base::LogStreamsReferenceMapPtr logStreamsReference)
- Logger (const std::string &id, const Configurations &configurations, base::LogStreamsReferenceMapPtr logStreamsReference)
- Logger (const Logger &logger)
- Logger & operator= (const Logger &logger)
- virtual ∼Logger (void)
- virtual void log (el::base::type::ostream_t &os) const
- void configure (const Configurations &configurations)

    Configures the logger using specified configurations.
- void reconfigure (void)

    Reconfigures logger using existing configurations.
- const std::string & id (void) const
- const std::string & parentApplicationName (void) const
- void setParentApplicationName (const std::string &parentApplicationName)
- Configurations ∗ configurations (void)
- base::TypedConfigurations ∗ typedConfigurations (void)
- void flush (void)

    Flushes logger to sync all log files for all levels.
- void flush (Level level, base::type::fstream_t ∗fs)
- bool isFlushNeeded (Level level)
- LogBuilder ∗ logBuilder (void) const
- void setLogBuilder (const LogBuilderPtr &logBuilder)
- bool enabled (Level level) const

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Public Member Functions inherited from el::Loggable

- virtual ∼Loggable (void)

Static Public Member Functions

- static bool isValidId (const std::string &id)

Private Member Functions

- Logger (void)
- void initUnflushedCount (void)
- base::type::stringstream_t & stream (void)
- void resolveLoggerFormatSpec (void) const

Private Attributes

- std::string m_id
- base::TypedConfigurations ∗ m_typedConfigurations
- base::type::stringstream_t m_stream
- std::string m_parentApplicationName
- bool m_isConfigured
- Configurations m_configurations
- std::unordered_map< Level, unsigned int > m_unflushedCount
- base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr
- LogBuilderPtr m_logBuilder

Friends

- class el::LogMessage
- class el::Loggers
- class el::Helpers
- class el::base::RegisteredLoggers
- class el::base::DefaultLogDispatchCallback
- class el::base::MessageBuilder
- class el::base::Writer
- class el::base::PErrorWriter
- class el::base::Storage
- class el::base::PerformanceTracker
- class el::base::LogDispatcher

Additional Inherited Members

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

8.24.1 Detailed Description

Represents a logger holding ID and configurations we need to write logs.

@detail This class does not write logs itself instead its used by writer to read configurations from.

Definition at line 2213 of file easylogging++.h.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 Logger() [1/4]

```
el::Logger::Logger (
            const std::string & id,
            base::LogStreamsReferenceMapPtr logStreamsReference )
```

Definition at line 598 of file easylogging++.cc.

References initUnflushedCount().

8.24.2.2 Logger() [2/4]

```
el::Logger::Logger (
            const std::string & id,
            const Configurations & configurations,
            base::LogStreamsReferenceMapPtr logStreamsReference )
```

Definition at line 607 of file easylogging++.cc.

References configurations(), configure(), and initUnflushedCount().

8.24.2.3 Logger() [3/4]

```
el::Logger::Logger (
            const Logger & logger )
```

Definition at line 618 of file easylogging++.cc.

References m_configurations, m_id, m_isConfigured, m_logStreamsReference, m_parentApplicationName, m_typedConfigurations, m_unflushedCount, and el::base::utils::safeDelete().

8.24.2.4 ∼Logger()

```
virtual el::Logger::∼Logger (
            void ) [inline], [virtual]
```

Definition at line 2220 of file easylogging++.h.

8.24.2.5 Logger() [4/4]

```
el::Logger::Logger (
            void ) [private]
```

8.24.3 Member Function Documentation

8.24.3.1 configurations()

```
Configurations * el::Logger::configurations (
            void ) [inline]
```

Definition at line 2246 of file easylogging++.h.

8.24.3.2 configure()

```
void el::Logger::configure (
            const Configurations & configurations )
```

Configures the logger using specified configurations.

Definition at line 643 of file easylogging++.cc.

References configurations(), el::base::TypedConfigurations::configurations(), el::Filename, flush(), el::Global, el::Configurations::hasConfiguration(), initUnflushedCount(), el::base::threading::ThreadSafe::lock(), m_configurations, m_isConfigured, m_logStreamsReference, m_typedConfigurations, resolveLoggerFormatSpec(), el::base::utils::safeDelete(), and el::Configurations::setFromBase().

8.24.3.3 enabled()

```
bool el::Logger::enabled (
            Level level ) const  [inline]
```

Definition at line 2273 of file easylogging++.h.

8.24.3.4 flush() [1/2]

```
void el::Logger::flush (
            Level level,
            base::type::fstream_t * fs )
```

Definition at line 686 of file easylogging++.cc.

References el::base::TypedConfigurations::fileStream(), m_typedConfigurations, m_unflushedCount, el::base::TypedConfigurations::t
and el::Helpers::validateFileRolling().

8.24.3.5 flush() [2/2]

```
void el::Logger::flush (
            void  )
```

Flushes logger to sync all log files for all levels.

Definition at line 676 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), ELPP_INTERNAL_INFO, flush(), el::LevelHelper::forEachLevel(), el::LevelHelper::kMinValid, el::base::threading::ThreadSafe::lock(), and m_id.

8.24.3.6 id()

```
const std::string & el::Logger::id (
            void  ) const  [inline]
```

Definition at line 2234 of file easylogging++.h.

8.24.3.7 initUnflushedCount()

```
void el::Logger::initUnflushedCount (
            void  )  [private]
```

Definition at line 700 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), el::LevelHelper::forEachLevel(), el::LevelHelper::kMinValid, and m_unflushedCount.

8.24.3.8 isFlushNeeded()

```
bool el::Logger::isFlushNeeded (
            Level level )  [inline]
```

Definition at line 2261 of file easylogging++.h.

8.24.3.9 isValidId()

```
bool el::Logger::isValidId (
            const std::string & id )  [static]
```

Definition at line 667 of file easylogging++.cc.

References el::base::utils::Str::contains(), and el::base::consts::kValidLoggerIdSymbols.

8.24.3.10 log()

```
virtual void el::Logger::log (
            el::base::type::ostream_t & os ) const  [inline], [virtual]
```

Implements el::Loggable.

Definition at line 2224 of file easylogging++.h.

8.24.3.11 logBuilder()

```
LogBuilder * el::Logger::logBuilder (
            void  ) const  [inline]
```

Definition at line 2265 of file easylogging++.h.

8.24.3.12 operator=()

```
Logger & el::Logger::operator= (
            const Logger & logger )
```

Definition at line 629 of file easylogging++.cc.

References m_configurations, m_id, m_isConfigured, m_logStreamsReference, m_parentApplicationName, m_typedConfigurations, m_unflushedCount, and el::base::utils::safeDelete().

8.24.3.13 parentApplicationName()

```
const std::string & el::Logger::parentApplicationName (
            void  ) const  [inline]
```

Definition at line 2238 of file easylogging++.h.

8.24.3.14 reconfigure()

```
void el::Logger::reconfigure (
             void  )
```

Reconfigures logger using existing configurations.

Definition at line 662 of file easylogging++.cc.

References configure(), ELPP_INTERNAL_INFO, m_configurations, and m_id.

8.24.3.15 resolveLoggerFormatSpec()

```
void el::Logger::resolveLoggerFormatSpec (
             void  ) const  [private]
```

Definition at line 709 of file easylogging++.cc.

References el::LevelHelper::castFromInt(), el::LevelHelper::forEachLevel(), el::base::consts::kLoggerIdFormatSpecifier, el::LevelHelper::kMinValid, el::base::TypedConfigurations::logFormat(), el::base::LogFormat::m_format, m_id, m_typedConfigurations, and el::base::utils::Str::replaceFirstWithEscape().

8.24.3.16 setLogBuilder()

```
void el::Logger::setLogBuilder (
             const LogBuilderPtr & logBuilder )  [inline]
```

Definition at line 2269 of file easylogging++.h.

8.24.3.17 setParentApplicationName()

```
void el::Logger::setParentApplicationName (
             const std::string & parentApplicationName )  [inline]
```

Definition at line 2242 of file easylogging++.h.

8.24.3.18 stream()

```
base::type::stringstream_t & el::Logger::stream (
             void  )  [inline], [private]
```

Definition at line 2339 of file easylogging++.h.

8.24.3.19 typedConfigurations()

```
base::TypedConfigurations * el::Logger::typedConfigurations (
             void  )  [inline]
```

Definition at line 2250 of file easylogging++.h.

8.24.4 Friends And Related Symbol Documentation

8.24.4.1 el::base::DefaultLogDispatchCallback

```
friend class el::base::DefaultLogDispatchCallback  [friend]
```

Definition at line 2313 of file easylogging++.h.

8.24.4.2 el::base::LogDispatcher

```
friend class el::base::LogDispatcher  [friend]
```

Definition at line 2319 of file easylogging++.h.

8.24.4.3 el::base::MessageBuilder

```
friend class el::base::MessageBuilder  [friend]
```

Definition at line 2314 of file easylogging++.h.

8.24.4.4 el::base::PerformanceTracker

```
friend class el::base::PerformanceTracker  [friend]
```

Definition at line 2318 of file easylogging++.h.

8.24.4.5 el::base::PErrorWriter

```
friend class el::base::PErrorWriter  [friend]
```

Definition at line 2316 of file easylogging++.h.

8.24.4.6 el::base::RegisteredLoggers

```
friend class el::base::RegisteredLoggers  [friend]
```

Definition at line 2312 of file easylogging++.h.

8.24.4.7 el::base::Storage

```
friend class el::base::Storage  [friend]
```

Definition at line 2317 of file easylogging++.h.

8.24.4.8 el::base::Writer

```
friend class el::base::Writer  [friend]
```

Definition at line 2315 of file easylogging++.h.

8.24.4.9 el::Helpers

```
friend class el::Helpers  [friend]
```

Definition at line 2311 of file easylogging++.h.

8.24.4.10 el::Loggers

```
friend class el::Loggers  [friend]
```

Definition at line 2310 of file easylogging++.h.

8.24.4.11 el::LogMessage

```
friend class el::LogMessage  [friend]
```

Definition at line 2309 of file easylogging++.h.

8.24.5 Field Documentation

8.24.5.1 m_configurations

```
Configurations el::Logger::m_configurations  [private]
```

Definition at line 2304 of file easylogging++.h.

8.24.5.2 m_id

```
std::string el::Logger::m_id  [private]
```

Definition at line 2299 of file easylogging++.h.

8.24.5.3 m_isConfigured

```
bool el::Logger::m_isConfigured  [private]
```

Definition at line 2303 of file easylogging++.h.

8.24.5.4 m_logBuilder

```
LogBuilderPtr el::Logger::m_logBuilder  [private]
```

Definition at line 2307 of file easylogging++.h.

8.24.5.5 m_logStreamsReference

```
base::LogStreamsReferenceMapPtr el::Logger::m_logStreamsReference = nullptr  [private]
```

Definition at line 2306 of file easylogging++.h.

8.24.5.6 m_parentApplicationName

```
std::string el::Logger::m_parentApplicationName  [private]
```

Definition at line 2302 of file easylogging++.h.

8.24.5.7 m_stream

```
base::type::stringstream_t el::Logger::m_stream  [private]
```

Definition at line 2301 of file easylogging++.h.

8.24.5.8 m_typedConfigurations

```
base::TypedConfigurations* el::Logger::m_typedConfigurations  [private]
```

Definition at line 2300 of file easylogging++.h.

8.24.5.9 m_unflushedCount

```
std::unordered_map<Level, unsigned int> el::Logger::m_unflushedCount  [private]
```

Definition at line 2305 of file easylogging++.h.

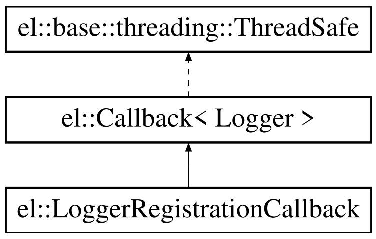
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.25 el::LoggerRegistrationCallback Class Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::LoggerRegistrationCallback:


el::base::threading::ThreadSafe
el::Callback< Logger >
el::LoggerRegistrationCallback

Friends

- class base::RegisteredLoggers

Additional Inherited Members

Public Member Functions inherited from el::Callback< Logger >

- Callback (void)
- bool enabled (void) const
- void setEnabled (bool enabled)

Protected Member Functions inherited from el::Callback< Logger >

- virtual void handle (const Logger ∗handlePtr)=0

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

8.25.1 Detailed Description

Definition at line 2193 of file easylogging++.h.

8.25.2 Friends And Related Symbol Documentation

8.25.2.1 base::RegisteredLoggers

```
friend class base::RegisteredLoggers [friend]
```

Definition at line 2195 of file easylogging++.h.

The documentation for this class was generated from the following file:

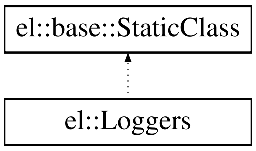
- lib/easylogging++.h

8.26 el::Loggers Class Reference

Static helpers to deal with loggers and their configurations.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Loggers:



Data Structures

- class ScopedAddFlag

  Adds flag and removes it when scope goes out.
- class ScopedRemoveFlag

  Removes flag and add it when scope goes out.

Static Public Member Functions

- static Logger ∗ getLogger (const std::string &identity, bool registerIfNotAvailable=true)

  Gets existing or registers new logger.
- static void setDefaultLogBuilder (el::LogBuilderPtr &logBuilderPtr)

  Changes default log builder for future loggers.
- template<typename T >
  static bool installLoggerRegistrationCallback (const std::string &id)

  Installs logger registration callback, this callback is triggered when new logger is registered.
- template<typename T >
  static void uninstallLoggerRegistrationCallback (const std::string &id)

  Uninstalls log dispatch callback.
- template<typename T >
  static T ∗ loggerRegistrationCallback (const std::string &id)
- static bool unregisterLogger (const std::string &identity)

Unregisters logger - use it only when you know what you are doing, you may unregister loggers initialized / used by third-party libs.
- static bool hasLogger (const std::string &identity)

    Whether or not logger with id is registered.
- static Logger ∗ reconfigureLogger (Logger ∗logger, const Configurations &configurations)

    Reconfigures specified logger with new configurations.
- static Logger ∗ reconfigureLogger (const std::string &identity, const Configurations &configurations)

    Reconfigures logger with new configurations after looking it up using identity.
- static Logger ∗ reconfigureLogger (const std::string &identity, ConfigurationType configurationType, const std::string &value)

    Reconfigures logger's single configuration.
- static void reconfigureAllLoggers (const Configurations &configurations)

    Reconfigures all the existing loggers with new configurations.
- static void reconfigureAllLoggers (ConfigurationType configurationType, const std::string &value)

    Reconfigures single configuration for all the loggers.
- static void reconfigureAllLoggers (Level level, ConfigurationType configurationType, const std::string &value)

    Reconfigures single configuration for all the loggers for specified level.
- static void setDefaultConfigurations (const Configurations &configurations, bool reconfigureExisting↩ Loggers=false)

    Sets default configurations. This configuration is used for future (and conditionally for existing) loggers.
- static const Configurations ∗ defaultConfigurations (void)

    Returns current default.
- static const base::LogStreamsReferenceMapPtr logStreamsReference (void)

    Returns log stream reference pointer if needed by user.
- static base::TypedConfigurations defaultTypedConfigurations (void)

    Default typed configuration based on existing defaultConf.
- static std::vector< std::string > ∗ populateAllLoggerIds (std::vector< std::string > ∗targetList)

    Populates all logger IDs in current repository.
- static void configureFromGlobal (const char ∗globalConfigurationFilePath)

    Sets configurations from global configuration file.
- static bool configureFromArg (const char ∗argKey)

    Configures loggers using command line arg. Ensure you have already set command line args,.
- static void flushAll (void)

    Flushes all loggers for all levels - Be careful if you dont know how many loggers are registered.
- static void addFlag (LoggingFlag flag)

    Adds logging flag used internally.
- static void removeFlag (LoggingFlag flag)

    Removes logging flag used internally.
- static bool hasFlag (LoggingFlag flag)

    Determines whether or not certain flag is active.
- static void setLoggingLevel (Level level)

    Sets hierarchy for logging. Needs to enable logging flag (HierarchicalLogging)
- static void setVerboseLevel (base::type::VerboseLevel level)

    Sets verbose level on the fly.
- static base::type::VerboseLevel verboseLevel (void)

    Gets current verbose level.
- static void setVModules (const char ∗modules)

    Sets vmodules as specified (on the fly)
- static void clearVModules (void)

    Clears vmodules.

8.26.1 Detailed Description

Static helpers to deal with loggers and their configurations.

Definition at line 3783 of file easylogging++.h.

8.26.2 Member Function Documentation

8.26.2.1 addFlag()

```
static void el::Loggers::addFlag (
             LoggingFlag flag )  [inline], [static]
```

Adds logging flag used internally.

Definition at line 3846 of file easylogging++.h.

References ELPP.

8.26.2.2 clearVModules()

```
void el::Loggers::clearVModules (
             void  )  [static]
```

Clears vmodules.

Definition at line 3102 of file easylogging++.cc.

References ELPP.

8.26.2.3 configureFromArg()

```
bool el::Loggers::configureFromArg (
             const char * argKey )  [static]
```

Configures loggers using command line arg. Ensure you have already set command line args,.

Returns

> False if invalid argument or argument with no value provided, true if attempted to configure logger. If true is returned that does not mean it has been configured successfully, it only means that it has attempted to configure logger using configuration file provided in argument

Definition at line 3072 of file easylogging++.cc.

References el::Helpers::commandLineArgs(), configureFromGlobal(), and ELPP_UNUSED.

8.26.2.4 configureFromGlobal()

```
void el::Loggers::configureFromGlobal (
            const char * globalConfigurationFilePath )  [static]
```

Sets configurations from global configuration file.

Definition at line 3031 of file easylogging++.cc.

References el::Logger::configure(), ELPP_ASSERT, ELPP_INTERNAL_INFO, getLogger(), el::Logger::id(), el::Configurations::Parser::ignoreComments(), el::Configurations::Parser::isComment(), el::base::consts::kConfigurationLoggerId, el::Configurations::parseFromText(), el::base::utils::Str::startsWith(), and el::base::utils::Str::trim().

8.26.2.5 defaultConfigurations()

```
const Configurations * el::Loggers::defaultConfigurations (
            void )  [static]
```

Returns current default.

Definition at line 3008 of file easylogging++.cc.

References ELPP.

8.26.2.6 defaultTypedConfigurations()

```
base::TypedConfigurations el::Loggers::defaultTypedConfigurations (
            void )  [static]
```

Default typed configuration based on existing defaultConf.

Definition at line 3016 of file easylogging++.cc.

References ELPP.

8.26.2.7 flushAll()

```
void el::Loggers::flushAll (
            void )  [static]
```

Flushes all loggers for all levels - Be careful if you dont know how many loggers are registered.

Definition at line 3084 of file easylogging++.cc.

References ELPP.

8.26.2.8 getLogger()

```
Logger * el::Loggers::getLogger (
             const std::string & identity,
             bool registerIfNotAvailable = true )  [static]
```

Gets existing or registers new logger.

Definition at line 2947 of file easylogging++.cc.

References ELPP.

8.26.2.9 hasFlag()

```
static bool el::Loggers::hasFlag (
             LoggingFlag flag )  [inline], [static]
```

Determines whether or not certain flag is active.

Definition at line 3854 of file easylogging++.h.

References ELPP.

8.26.2.10 hasLogger()

```
bool el::Loggers::hasLogger (
             const std::string & identity )  [static]
```

Whether or not logger with id is registered.

Definition at line 2959 of file easylogging++.cc.

References ELPP.

8.26.2.11 installLoggerRegistrationCallback()

```
template<typename T >
static bool el::Loggers::installLoggerRegistrationCallback (
             const std::string & id )  [inline], [static]
```

Installs logger registration callback, this callback is triggered when new logger is registered.

Definition at line 3791 of file easylogging++.h.

References ELPP.

8.26.2.12 loggerRegistrationCallback()

```
template<typename T >
static T * el::Loggers::loggerRegistrationCallback (
             const std::string & id )  [inline], [static]
```

Definition at line 3800 of file easylogging++.h.

References ELPP.

8.26.2.13 logStreamsReference()

```
const base::LogStreamsReferenceMapPtr el::Loggers::logStreamsReference (
            void ) [static]
```

Returns log stream reference pointer if needed by user.

Definition at line 3012 of file easylogging++.cc.

References ELPP.

8.26.2.14 populateAllLoggerIds()

```
std::vector< std::string > * el::Loggers::populateAllLoggerIds (
            std::vector< std::string > * targetList ) [static]
```

Populates all logger IDs in current repository.

Parameters

out	*targetList*	List of fill up.

Definition at line 3022 of file easylogging++.cc.

References ELPP.

8.26.2.15 reconfigureAllLoggers() [1/3]

```
static void el::Loggers::reconfigureAllLoggers (
            ConfigurationType configurationType,
            const std::string & value ) [inline], [static]
```

Reconfigures single configuration for all the loggers.

Definition at line 3818 of file easylogging++.h.

8.26.2.16 reconfigureAllLoggers() [2/3]

```
void el::Loggers::reconfigureAllLoggers (
            const Configurations & configurations ) [static]
```

Reconfigures all the existing loggers with new configurations.

Definition at line 2984 of file easylogging++.cc.

References ELPP, and reconfigureLogger().

8.26.2.17 reconfigureAllLoggers() [3/3]

```
void el::Loggers::reconfigureAllLoggers (
            Level level,
            ConfigurationType configurationType,
            const std::string & value )  [static]
```

Reconfigures single configuration for all the loggers for specified level.

Definition at line 2991 of file easylogging++.cc.

References el::Logger::configurations(), ELPP, el::Logger::reconfigure(), and el::Configurations::set().

8.26.2.18 reconfigureLogger() [1/3]

```
Logger * el::Loggers::reconfigureLogger (
            const std::string & identity,
            ConfigurationType configurationType,
            const std::string & value )  [static]
```

Reconfigures logger's single configuration.

Definition at line 2973 of file easylogging++.cc.

References el::Logger::configurations(), getLogger(), el::Global, el::Logger::reconfigure(), and el::Configurations::set().

8.26.2.19 reconfigureLogger() [2/3]

```
Logger * el::Loggers::reconfigureLogger (
            const std::string & identity,
            const Configurations & configurations )  [static]
```

Reconfigures logger with new configurations after looking it up using identity.

Definition at line 2969 of file easylogging++.cc.

References getLogger(), and reconfigureLogger().

8.26.2.20 reconfigureLogger() [3/3]

```
Logger * el::Loggers::reconfigureLogger (
            Logger * logger,
            const Configurations & configurations )  [static]
```

Reconfigures specified logger with new configurations.

Definition at line 2963 of file easylogging++.cc.

References el::Logger::configure().

8.26.2.21 removeFlag()

```
static void el::Loggers::removeFlag (
             LoggingFlag flag )  [inline], [static]
```

Removes logging flag used internally.

Definition at line 3850 of file easylogging++.h.

References ELPP.

8.26.2.22 setDefaultConfigurations()

```
void el::Loggers::setDefaultConfigurations (
             const Configurations & configurations,
             bool reconfigureExistingLoggers = false )  [static]
```

Sets default configurations. This configuration is used for future (and conditionally for existing) loggers.

Definition at line 3001 of file easylogging++.cc.

References ELPP, and reconfigureAllLoggers().

8.26.2.23 setDefaultLogBuilder()

```
void el::Loggers::setDefaultLogBuilder (
             el::LogBuilderPtr & logBuilderPtr )  [static]
```

Changes default log builder for future loggers.

Definition at line 2951 of file easylogging++.cc.

References ELPP.

8.26.2.24 setLoggingLevel()

```
static void el::Loggers::setLoggingLevel (
             Level level )  [inline], [static]
```

Sets hierarchy for logging. Needs to enable logging flag (HierarchicalLogging)

Definition at line 3882 of file easylogging++.h.

References ELPP.

8.26.2.25 setVerboseLevel()

```
void el::Loggers::setVerboseLevel (
             base::type::VerboseLevel level )  [static]
```

Sets verbose level on the fly.

Definition at line 3088 of file easylogging++.cc.

References ELPP.

8.26.2.26 setVModules()

```
void el::Loggers::setVModules (
            const char * modules )  [static]
```

Sets vmodules as specified (on the fly)

Definition at line 3096 of file easylogging++.cc.

References ELPP.

8.26.2.27 uninstallLoggerRegistrationCallback()

```
template<typename T >
static void el::Loggers::uninstallLoggerRegistrationCallback (
            const std::string & id )  [inline], [static]
```

Uninstalls log dispatch callback.

Definition at line 3796 of file easylogging++.h.

References ELPP.

8.26.2.28 unregisterLogger()

```
bool el::Loggers::unregisterLogger (
            const std::string & identity )  [static]
```

Unregisters logger - use it only when you know what you are doing, you may unregister loggers initialized / used by third-party libs.

Definition at line 2955 of file easylogging++.cc.

References ELPP.

8.26.2.29 verboseLevel()

```
base::type::VerboseLevel el::Loggers::verboseLevel (
            void  )  [static]
```

Gets current verbose level.

Definition at line 3092 of file easylogging++.cc.

References ELPP.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.27 el::LogMessage Class Reference

```
#include <easylogging++.h>
```

Public Member Functions

- LogMessage (Level level, const std::string &file, base::type::LineNumber line, const std::string &func, base::type::VerboseLevel verboseLevel, Logger ∗logger)
- Level level (void) const
- const std::string & file (void) const
- base::type::LineNumber line (void) const
- const std::string & func (void) const
- base::type::VerboseLevel verboseLevel (void) const
- Logger ∗ logger (void) const
- const base::type::string_t & message (void) const

Private Attributes

- Level m_level
- std::string m_file
- base::type::LineNumber m_line
- std::string m_func
- base::type::VerboseLevel m_verboseLevel
- Logger ∗ m_logger
- base::type::string_t m_message

8.27.1 Detailed Description

Definition at line 2454 of file easylogging++.h.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 LogMessage()

```
el::LogMessage::LogMessage (
            Level level,
            const std::string & file,
            base::type::LineNumber line,
            const std::string & func,
            base::type::VerboseLevel verboseLevel,
            Logger ∗ logger )  [inline]
```

Definition at line 2456 of file easylogging++.h.

8.27.3 Member Function Documentation

8.27.3.1 file()

```
const std::string & el::LogMessage::file (
            void ) const [inline]
```

Definition at line 2464 of file easylogging++.h.

8.27.3.2 func()

```
const std::string & el::LogMessage::func (
            void ) const [inline]
```

Definition at line 2470 of file easylogging++.h.

8.27.3.3 level()

```
Level el::LogMessage::level (
            void ) const [inline]
```

Definition at line 2461 of file easylogging++.h.

8.27.3.4 line()

```
base::type::LineNumber el::LogMessage::line (
            void ) const [inline]
```

Definition at line 2467 of file easylogging++.h.

8.27.3.5 logger()

```
Logger * el::LogMessage::logger (
            void ) const [inline]
```

Definition at line 2476 of file easylogging++.h.

8.27.3.6 message()

```
const base::type::string_t & el::LogMessage::message (
            void ) const [inline]
```

Definition at line 2479 of file easylogging++.h.

8.27.3.7 verboseLevel()

```
base::type::VerboseLevel el::LogMessage::verboseLevel (
            void ) const [inline]
```

Definition at line 2473 of file easylogging++.h.

8.27.4 Field Documentation

8.27.4.1 m_file

```
std::string el::LogMessage::m_file [private]
```

Definition at line 2484 of file easylogging++.h.

8.27.4.2 m_func

```
std::string el::LogMessage::m_func [private]
```

Definition at line 2486 of file easylogging++.h.

8.27.4.3 m_level

```
Level el::LogMessage::m_level [private]
```

Definition at line 2483 of file easylogging++.h.

8.27.4.4 m_line

```
base::type::LineNumber el::LogMessage::m_line [private]
```

Definition at line 2485 of file easylogging++.h.

8.27.4.5 m_logger

```
Logger* el::LogMessage::m_logger [private]
```

Definition at line 2488 of file easylogging++.h.

8.27.4.6 m_message

```
base::type::string_t el::LogMessage::m_message [private]
```

Definition at line 2489 of file easylogging++.h.

8.27.4.7 m_verboseLevel

```
base::type::VerboseLevel el::LogMessage::m_verboseLevel  [private]
```

Definition at line 2487 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.28 el::base::MessageBuilder Class Reference

Public Member Functions

- MessageBuilder (void)
- void initialize (Logger ∗logger)
- MessageBuilder & operator<< (const std::string &msg)
- MessageBuilder & operator<< (const std::wstring &msg)
- MessageBuilder & operator<< (const wchar_t ∗msg)
- MessageBuilder & operator<< (std::ostream &(∗OStreamMani)(std::ostream &))

Private Member Functions

- template<class Iterator >
  MessageBuilder & writeIterator (Iterator begin_, Iterator end_, std::size_t size_)

Private Attributes

- template<class Class >
  Logger ∗ m_logger
- const base::type::char_t ∗ m_containerLogSeparator

8.28.1 Detailed Description

Definition at line 2862 of file easylogging++.h.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 MessageBuilder()

```
el::base::MessageBuilder::MessageBuilder (
             void  )  [inline]
```

Definition at line 2864 of file easylogging++.h.

8.28.3 Member Function Documentation

8.28.3.1 initialize()

```
void el::base::MessageBuilder::initialize (
            Logger * logger )
```

Definition at line 2505 of file easylogging++.cc.

References ELPP, ELPP_LITERAL, m_containerLogSeparator, m_logger, and el::NewLineForContainer.

8.28.3.2 operator<<() [1/4]

```
MessageBuilder & el::base::MessageBuilder::operator<< (
            const std::string & msg )  [inline]
```

Definition at line 2876 of file easylogging++.h.

8.28.3.3 operator<<() [2/4]

```
MessageBuilder & el::base::MessageBuilder::operator<< (
            const std::wstring & msg )  [inline]
```

Definition at line 2893 of file easylogging++.h.

8.28.3.4 operator<<() [3/4]

```
MessageBuilder & el::base::MessageBuilder::operator<< (
            const wchar_t * msg )
```

Definition at line 2511 of file easylogging++.cc.

References el::AutoSpacing, ELPP, el::base::consts::kNullPointer, m_logger, el::Logger::stream(), and el::base::utils::Str::wcharPtrToCharPtr().

8.28.3.5 operator<<() [4/4]

```
MessageBuilder & el::base::MessageBuilder::operator<< (
            std::ostream &(*)(std::ostream &) OStreamMani )  [inline]
```

Definition at line 2898 of file easylogging++.h.

8.28.3.6 writeIterator()

```
template<class Iterator >
MessageBuilder & el::base::MessageBuilder::writeIterator (
            Iterator begin_,
            Iterator end_,
            std::size_t size_ )  [inline], [private]
```

Definition at line 3154 of file easylogging++.h.

References ELPP, ELPP_LITERAL, and el::Logger::stream().

8.28.4 Field Documentation

8.28.4.1 m_containerLogSeparator

```
const base::type::char_t* el::base::MessageBuilder::m_containerLogSeparator  [private]
```

Definition at line 3151 of file easylogging++.h.

8.28.4.2 m_logger

```
template<class Class >
Logger* el::base::MessageBuilder::m_logger  [private]
```

Definition at line 3150 of file easylogging++.h.

The documentation for this class was generated from the following files:

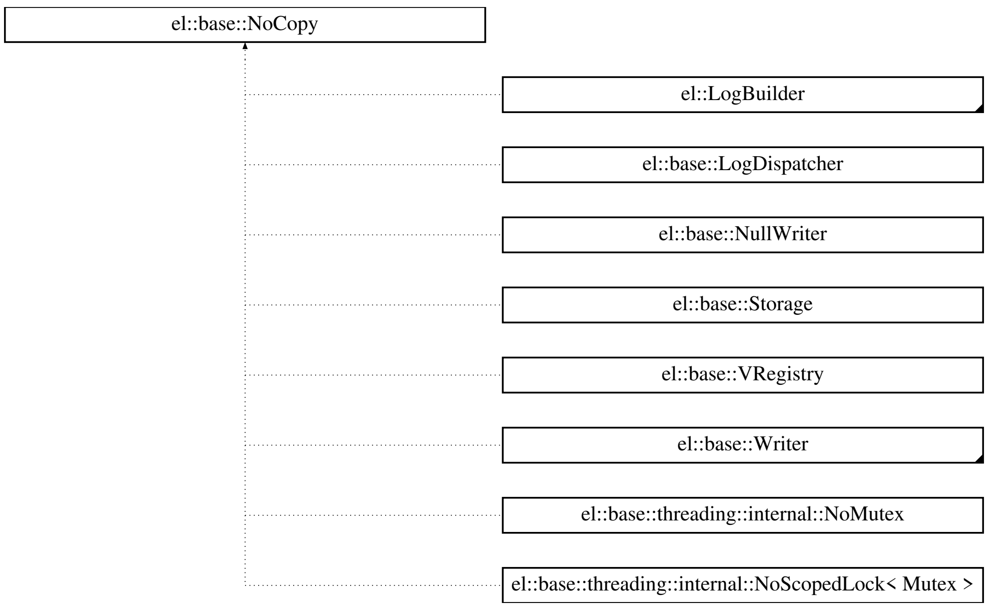
- lib/easylogging++.h
- lib/easylogging++.cc

8.29 el::base::NoCopy Class Reference

Internal helper class that prevent copy constructor for class.

```
#include <easylogging++.h>
```

Inheritance diagram for el::base::NoCopy:



Protected Member Functions

- NoCopy (void)

Private Member Functions

- NoCopy (const NoCopy &)
- NoCopy & operator= (const NoCopy &)

8.29.1 Detailed Description

Internal helper class that prevent copy constructor for class.

@detail When using this class simply inherit it privately

Definition at line 551 of file easylogging++.h.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 NoCopy() [1/2]

```
el::base::NoCopy::NoCopy (
            void  )  [inline], [protected]
```

Definition at line 553 of file easylogging++.h.

8.29.2.2 NoCopy() [2/2]

```
el::base::NoCopy::NoCopy (
            const NoCopy &  )  [private]
```

8.29.3 Member Function Documentation

8.29.3.1 operator=()

```
NoCopy & el::base::NoCopy::operator= (
            const NoCopy &  )  [private]
```

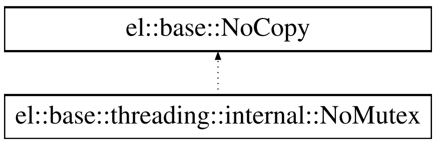
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.30 el::base::threading::internal::NoMutex Class Reference

Mutex wrapper used when multi-threading is disabled.

Inheritance diagram for el::base::threading::internal::NoMutex:



Public Member Functions

- NoMutex (void)
- void lock (void)
- bool try_lock (void)
- void unlock (void)

Additional Inherited Members

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.30.1 Detailed Description

Mutex wrapper used when multi-threading is disabled.

Definition at line 977 of file easylogging++.h.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 NoMutex()

```
el::base::threading::internal::NoMutex::NoMutex (
             void  )  [inline]
```

Definition at line 979 of file easylogging++.h.

8.30.3 Member Function Documentation

8.30.3.1 lock()

```
void el::base::threading::internal::NoMutex::lock (
             void  )  [inline]
```

Definition at line 980 of file easylogging++.h.

8.30.3.2 try_lock()

```
bool el::base::threading::internal::NoMutex::try_lock (
             void  )  [inline]
```

Definition at line 981 of file easylogging++.h.

8.30.3.3 unlock()

```
void el::base::threading::internal::NoMutex::unlock (
             void  )  [inline]
```

Definition at line 984 of file easylogging++.h.

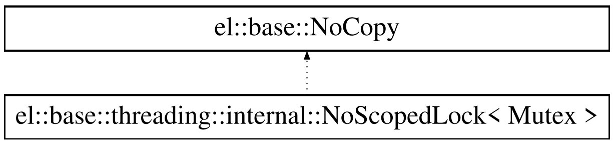
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.31 el::base::threading::internal::NoScopedLock< Mutex > Class Template Reference

Lock guard wrapper used when multi-threading is disabled.

Inheritance diagram for el::base::threading::internal::NoScopedLock< Mutex >:



Public Member Functions

- NoScopedLock (Mutex &)
- virtual ∼NoScopedLock (void)

Private Member Functions

- NoScopedLock (void)

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.31.1 Detailed Description

template<typename Mutex>
class el::base::threading::internal::NoScopedLock< Mutex >

Lock guard wrapper used when multi-threading is disabled.

Definition at line 988 of file easylogging++.h.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 NoScopedLock() [1/2]

```
template<typename Mutex >
el::base::threading::internal::NoScopedLock< Mutex >::NoScopedLock (
            Mutex &  )  [inline], [explicit]
```

Definition at line 990 of file easylogging++.h.

8.31.2.2 ∼NoScopedLock()

```
template<typename Mutex >
virtual el::base::threading::internal::NoScopedLock< Mutex >::∼NoScopedLock (
            void  )  [inline], [virtual]
```

Definition at line 992 of file easylogging++.h.

8.31.2.3 NoScopedLock() [2/2]

```
template<typename Mutex >
el::base::threading::internal::NoScopedLock< Mutex >::NoScopedLock (
            void  )  [private]
```

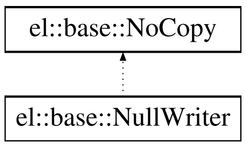
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.32 el::base::NullWriter Class Reference

Writes nothing - Used when certain log is disabled.

Inheritance diagram for el::base::NullWriter:



Public Member Functions

- NullWriter (void)
- NullWriter & operator<< (std::ostream &(∗)(std::ostream &))
- template<typename T >
  NullWriter & operator<< (const T &)
- operator bool ()

Additional Inherited Members

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.32.1 Detailed Description

Writes nothing - Used when certain log is disabled.

Definition at line 3171 of file easylogging++.h.

8.32.2 Constructor & Destructor Documentation

8.32.2.1 NullWriter()

```
el::base::NullWriter::NullWriter (
             void  )  [inline]
```

Definition at line 3173 of file easylogging++.h.

8.32.3 Member Function Documentation

8.32.3.1 operator bool()

```
el::base::NullWriter::operator bool ( )  [inline]
```

Definition at line 3185 of file easylogging++.h.

8.32.3.2 operator<<() [1/2]

```
template<typename T >
NullWriter & el::base::NullWriter::operator<< (
             const T &  )  [inline]
```

Definition at line 3181 of file easylogging++.h.

8.32.3.3 operator<<() [2/2]

```
NullWriter & el::base::NullWriter::operator<< (
             std::ostream & *)(std::ostream & )  [inline]
```

Definition at line 3176 of file easylogging++.h.

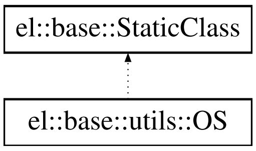
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.33 el::base::utils::OS Class Reference

Operating System helper static class used internally. You should not use it.

Inheritance diagram for el::base::utils::OS:



Static Public Member Functions

- static const std::string getBashOutput (const char ∗command)

  Runs command on terminal and returns the output.
- static std::string getEnvironmentVariable (const char ∗variableName, const char ∗defaultVal, const char ∗alternativeBashCommand=nullptr)

  Gets environment variable. This is cross-platform and CRT safe (for VC++)
- static std::string currentUser (void)

  Gets current username.
- static std::string currentHost (void)

  Gets current host name or computer name.
- static bool termSupportsColor (void)

  Whether or not terminal supports colors.

8.33.1 Detailed Description

Operating System helper static class used internally. You should not use it.

Definition at line 1137 of file easylogging++.h.

8.33.2 Member Function Documentation

8.33.2.1 currentHost()

```
std::string el::base::utils::OS::currentHost (
            void ) [static]
```

Gets current host name or computer name.

@detail For android systems this is device name with its manufacturer and model separated by hyphen

Definition at line 1128 of file easylogging++.cc.

References ELPP_UNUSED, getEnvironmentVariable(), and el::base::consts::kUnknownHost.

8.33.2.2 currentUser()

```
std::string el::base::utils::OS::currentUser (
             void  )  [static]
```

Gets current username.

Definition at line 1115 of file easylogging++.cc.

References ELPP_UNUSED, getEnvironmentVariable(), and el::base::consts::kUnknownUser.

8.33.2.3 getBashOutput()

```
const std::string el::base::utils::OS::getBashOutput (
             const char * command )  [static]
```

Runs command on terminal and returns the output.

@detail This is applicable only on unix based systems, for all other OS, an empty string is returned.

Parameters

command	Bash command

Returns

Result of bash output or empty string if no result found.

Definition at line 1063 of file easylogging++.cc.

References ELPP_INTERNAL_ERROR, and ELPP_UNUSED.

8.33.2.4 getEnvironmentVariable()

```
std::string el::base::utils::OS::getEnvironmentVariable (
             const char * variableName,
             const char * defaultVal,
             const char * alternativeBashCommand = nullptr )  [static]
```

Gets environment variable. This is cross-platform and CRT safe (for VC++)

Parameters

variableName	Environment variable name
defaultVal	If no environment variable or value found the value to return by default
alternativeBashCommand	If environment variable not found what would be alternative bash command in order to look for value user is looking for. E.g, for 'user' alternative command will 'whoami'

Definition at line 1091 of file easylogging++.cc.

References ELPP_UNUSED, el::base::utils::AbstractRegistry< T_Ptr, Container >::empty(), and getBashOutput().

8.33.2.5 termSupportsColor()

```
bool el::base::utils::OS::termSupportsColor (
            void ) [static]
```

Whether or not terminal supports colors.

Definition at line 1141 of file easylogging++.cc.

References getEnvironmentVariable().

The documentation for this class was generated from the following files:

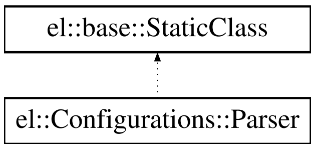
- lib/easylogging++.h
- lib/easylogging++.cc

8.34 el::Configurations::Parser Class Reference

Parser used internally to parse configurations from file or text.

```
#include <easylogging++.h>
```

Inheritance diagram for el::Configurations::Parser:



Static Public Member Functions

- static bool parseFromFile (const std::string &configurationFile, Configurations ∗sender, Configurations ∗base=nullptr)

    Parses configuration from file.
- static bool parseFromText (const std::string &configurationsString, Configurations ∗sender, Configurations ∗base=nullptr)

    Parse configurations from configuration string.

Static Private Member Functions

- static void ignoreComments (std::string ∗line)
- static bool isLevel (const std::string &line)
- static bool isComment (const std::string &line)
- static bool isConfig (const std::string &line)
- static bool parseLine (std::string ∗line, std::string ∗currConfigStr, std::string ∗currLevelStr, Level ∗currLevel, Configurations ∗conf)

Friends

- class el::Loggers

8.34.1 Detailed Description

Parser used internally to parse configurations from file or text.

@detail This class makes use of base::utils::Str. You should not need this unless you are working on some tool for Easylogging++

Definition at line 1839 of file easylogging++.h.

8.34.2 Member Function Documentation

8.34.2.1 ignoreComments()

```
void el::Configurations::Parser::ignoreComments (
            std::string * line ) [static], [private]
```

Definition at line 441 of file easylogging++.cc.

References el::base::consts::kConfigurationComment, and el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate

8.34.2.2 isComment()

```
bool el::Configurations::Parser::isComment (
            const std::string & line ) [static], [private]
```

Definition at line 464 of file easylogging++.cc.

References el::base::consts::kConfigurationComment, and el::base::utils::Str::startsWith().

8.34.2.3 isConfig()

```
bool el::Configurations::Parser::isConfig (
            const std::string & line ) [inline], [static], [private]
```

Definition at line 468 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred().

8.34.2.4 isLevel()

```
bool el::Configurations::Parser::isLevel (
            const std::string & line ) [static], [private]
```

Definition at line 460 of file easylogging++.cc.

References el::base::consts::kConfigurationLevel, and el::base::utils::Str::startsWith().

8.34.2.5 parseFromFile()

```
bool el::Configurations::Parser::parseFromFile (
            const std::string & configurationFile,
            Configurations * sender,
            Configurations * base = nullptr ) [static]
```

Parses configuration from file.

Parameters

configurationFile	Full path to configuration file
sender	Sender configurations pointer. Usually 'this' is used from calling class
base	Configurations to base new configuration repository off. This value is used when you want to use existing Configurations to base all the values and then set rest of configuration via configuration file.

Returns

True if successfully parsed, false otherwise. You may define '_STOP_ON_FIRSTELPP_ASSERTION' to make sure you do not proceed without successful parse.

Definition at line 407 of file easylogging++.cc.

References el::Configurations::configurationFile(), ELPP_ASSERT, parseLine(), el::base::utils::RegistryWithPred< Configuration, Co
and el::Unknown.

8.34.2.6 parseFromText()

```
bool el::Configurations::Parser::parseFromText (
            const std::string & configurationsString,
            Configurations * sender,
            Configurations * base = nullptr )  [static]
```

Parse configurations from configuration string.

@detail This configuration string has same syntax as configuration file contents. Make sure all the necessary new line characters are provided. You may define '_STOP_ON_FIRSTELPP_ASSERTION' to make sure you do not proceed without successful parse (This is recommended)

Parameters

configurationsString	the configuration in plain text format
sender	Sender configurations pointer. Usually 'this' is used from calling class
base	Configurations to base new configuration repository off. This value is used when you want to use existing Configurations to base all the values and then set rest of configuration via configuration text.

Returns

True if successfully parsed, false otherwise.

Definition at line 425 of file easylogging++.cc.

References ELPP_ASSERT, el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), and el::Unknown.

8.34.2.7 parseLine()

```
bool el::Configurations::Parser::parseLine (
            std::string * line,
            std::string * currConfigStr,
            std::string * currLevelStr,
            Level * currLevel,
            Configurations * conf )  [static], [private]
```

Definition at line 476 of file easylogging++.cc.

References el::ConfigurationTypeHelper::convertFromString(), el::LevelHelper::convertFromString(), ELPP_ASSERT, el::base::utils::RegistryWithPred< Configuration, Configuration::Predicate >::RegistryWithPred(), el::base::utils::Str::toUpper(), el::base::utils::Str::trim(), and el::Unknown.

8.34.3 Friends And Related Symbol Documentation

8.34.3.1 el::Loggers

```
friend class el::Loggers  [friend]
```

Definition at line 1865 of file easylogging++.h.

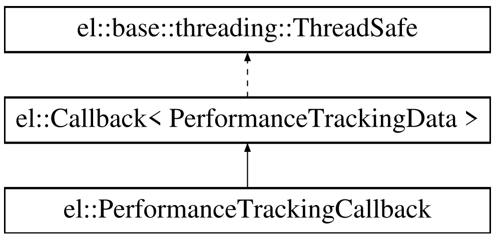
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.35 el::PerformanceTrackingCallback Class Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::PerformanceTrackingCallback:


el::base::threading::ThreadSafe
el::Callback< PerformanceTrackingData >
el::PerformanceTrackingCallback

Friends

- class base::PerformanceTracker

Additional Inherited Members

Public Member Functions inherited from el::Callback< PerformanceTrackingData >

- Callback (void)
- bool enabled (void) const
- void setEnabled (bool enabled)

Protected Member Functions inherited from el::Callback< PerformanceTrackingData >

- virtual void handle (const PerformanceTrackingData ∗handlePtr)=0

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

8.35.1 Detailed Description

Definition at line 2189 of file easylogging++.h.

8.35.2 Friends And Related Symbol Documentation

8.35.2.1 base::PerformanceTracker

```
friend class base::PerformanceTracker  [friend]
```

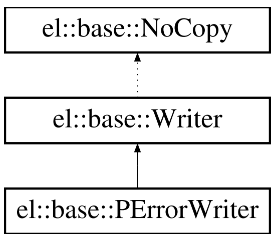
Definition at line 2191 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.36 el::base::PErrorWriter Class Reference

Inheritance diagram for el::base::PErrorWriter:



Public Member Functions

- PErrorWriter (Level level, const char ∗file, base::type::LineNumber line, const char ∗func, base::DispatchAction dispatchAction=base::DispatchAction::NormalLog, base::type::VerboseLevel verboseLevel=0)
- virtual ∼PErrorWriter (void)

Public Member Functions inherited from el::base::Writer

- Writer (Level level, const char ∗file, base::type::LineNumber line, const char ∗func, base::DispatchAction dispatchAction=base::DispatchAction::NormalLog, base::type::VerboseLevel verboseLevel=0)
- Writer (LogMessage ∗msg, base::DispatchAction dispatchAction=base::DispatchAction::NormalLog)
- virtual ∼Writer (void)
- template<typename T >
  Writer & operator<< (const T &log)
- Writer & operator<< (std::ostream &(∗log)(std::ostream &))
- operator bool ()
- Writer & construct (Logger ∗logger, bool needLock=true)
- Writer & construct (int count, const char ∗loggerIds,...)

Additional Inherited Members

Protected Member Functions inherited from el::base::Writer

- void initializeLogger (const std::string &loggerId, bool lookup=true, bool needLock=true)
- void processDispatch ()
- void triggerDispatch (void)

Protected Attributes inherited from el::base::Writer

- LogMessage ∗ m_msg
- Level m_level
- const char ∗ m_file
- const base::type::LineNumber m_line
- const char ∗ m_func
- base::type::VerboseLevel m_verboseLevel
- Logger ∗ m_logger
- bool m_proceed
- base::MessageBuilder m_messageBuilder
- base::DispatchAction m_dispatchAction
- std::vector< std::string > m_loggerIds

8.36.1 Detailed Description

Definition at line 3251 of file easylogging++.h.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 PErrorWriter()

```
el::base::PErrorWriter::PErrorWriter (
            Level level,
            const char * file,
            base::type::LineNumber line,
            const char * func,
            base::DispatchAction dispatchAction = base::DispatchAction::NormalLog,
            base::type::VerboseLevel verboseLevel = 0 )  [inline]
```

Definition at line 3253 of file easylogging++.h.

8.36.2.2 ∼PErrorWriter()

```
el::base::PErrorWriter::∼PErrorWriter (
            void  )  [virtual]
```

Definition at line 2660 of file easylogging++.cc.

References el::base::Writer::m_logger, el::base::Writer::m_proceed, and el::Logger::stream().

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.37 el::base::HitCounter::Predicate Class Reference

```
#include <easylogging++.h>
```

Public Member Functions

- Predicate (const char ∗filename, base::type::LineNumber lineNumber)
- bool operator() (const HitCounter ∗counter)

Private Attributes

- const char ∗ m_filename
- base::type::LineNumber m_lineNumber

8.37.1 Detailed Description

Definition at line 2095 of file easylogging++.h.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 Predicate()

```
el::base::HitCounter::Predicate::Predicate (
             const char * filename,
             base::type::LineNumber lineNumber )  [inline]
```

Definition at line 2097 of file easylogging++.h.

8.37.3 Member Function Documentation

8.37.3.1 operator()()

```
bool el::base::HitCounter::Predicate::operator() (
             const HitCounter * counter )  [inline]
```

Definition at line 2101 of file easylogging++.h.

References el::base::HitCounter::m_filename, and el::base::HitCounter::m_lineNumber.

8.37.4 Field Documentation

8.37.4.1 m_filename

```
const char* el::base::HitCounter::Predicate::m_filename  [private]
```

Definition at line 2108 of file easylogging++.h.

8.37.4.2 m_lineNumber

```
base::type::LineNumber el::base::HitCounter::Predicate::m_lineNumber  [private]
```

Definition at line 2109 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.38 el::Configuration::Predicate Class Reference

Used to find configuration from configuration (pointers) repository. Avoid using it.

```
#include <easylogging++.h>
```

Public Member Functions

- Predicate (Level level, ConfigurationType configurationType)

  Used to find configuration from configuration (pointers) repository. Avoid using it.
- bool operator() (const Configuration ∗conf) const

Private Attributes

- Level m_level
- ConfigurationType m_configurationType

8.38.1 Detailed Description

Used to find configuration from configuration (pointers) repository. Avoid using it.

Definition at line 1709 of file easylogging++.h.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 Predicate()

```
el::Configuration::Predicate::Predicate (
            Level level,
            ConfigurationType configurationType )
```

Used to find configuration from configuration (pointers) repository. Avoid using it.

Definition at line 264 of file easylogging++.cc.

8.38.3 Member Function Documentation

8.38.3.1 operator()()

```
bool el::Configuration::Predicate::operator() (
            const Configuration * conf ) const
```

Definition at line 269 of file easylogging++.cc.

References el::Configuration::configurationType(), el::Configuration::level(), el::Configuration::m_configurationType, and el::Configuration::m_level.

8.38.4 Field Documentation

8.38.4.1 m_configurationType

```
ConfigurationType el::Configuration::Predicate::m_configurationType  [private]
```

Definition at line 1717 of file easylogging++.h.

8.38.4.2 m_level

```
Level el::Configuration::Predicate::m_level  [private]
```

Definition at line 1716 of file easylogging++.h.

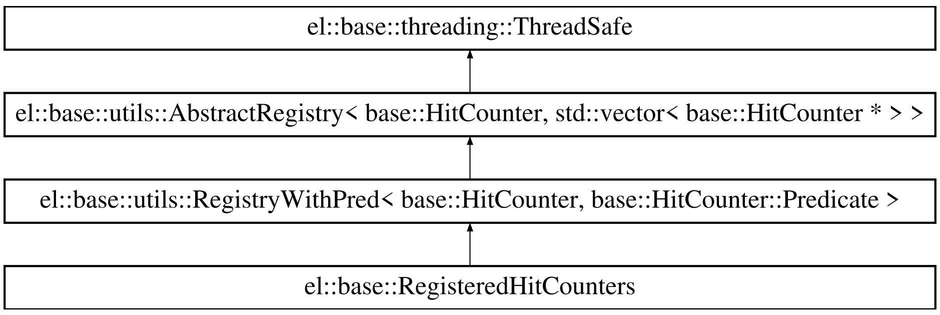
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.39 el::base::RegisteredHitCounters Class Reference

Repository for hit counters used across the application.

Inheritance diagram for el::base::RegisteredHitCounters:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< base::HitCounter, std::vector< base::HitCounter * > >
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >
el::base::RegisteredHitCounters

Public Member Functions

- bool validateEveryN (const char ∗filename, base::type::LineNumber lineNumber, std::size_t n)

  Validates counter for every N, i.e, registers new if does not exist otherwise updates original one.
- bool validateAfterN (const char ∗filename, base::type::LineNumber lineNumber, std::size_t n)

  Validates counter for hits >= N, i.e, registers new if does not exist otherwise updates original one.
- bool validateNTimes (const char ∗filename, base::type::LineNumber lineNumber, std::size_t n)

  Validates counter for hits are <= n, i.e, registers new if does not exist otherwise updates original one.
- const base::HitCounter ∗ getCounter (const char ∗filename, base::type::LineNumber lineNumber)

  Gets hit counter registered at specified position.

Public Member Functions inherited from el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >

- RegistryWithPred (void)
- RegistryWithPred (const RegistryWithPred &sr)

  Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.
- virtual ∼RegistryWithPred (void)
- RegistryWithPred & operator= (const RegistryWithPred &sr)

  Assignment operator that unregisters all the existing registries and deeply copies each of repo element.

Public Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, Container > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, Container > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ∼AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual Container & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const Container & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Additional Inherited Members

Public Types inherited from el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >

- typedef RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::iterator iterator
- typedef RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::const_iterator const_iterator

Public Types inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Protected Member Functions inherited from el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >

- virtual void unregisterAll (void) ELPP_FINAL

  Unregisters all the pointers from current repository.
- virtual void unregister (base::HitCounter ∗&ptr) ELPP_FINAL
- virtual void registerNew (base::HitCounter ∗ptr) ELPP_FINAL
- base::HitCounter ∗ get (const T &arg1, const T2 arg2)

  Gets pointer from repository with specified arguments. Arguments are passed to predicate in order to validate pointer.

Protected Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- virtual void deepCopy (const AbstractRegistry< T_Ptr, Container > &)=0
- void reinitDeepCopy (const AbstractRegistry< T_Ptr, Container > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

8.39.1 Detailed Description

Repository for hit counters used across the application.

Definition at line 2118 of file easylogging++.h.

8.39.2 Member Function Documentation

8.39.2.1 getCounter()

```
const base::HitCounter * el::base::RegisteredHitCounters::getCounter (
             const char * filename,
             base::type::LineNumber lineNumber )  [inline]
```

Gets hit counter registered at specified position.

Definition at line 2133 of file easylogging++.h.

8.39.2.2 validateAfterN()

```
bool el::base::RegisteredHitCounters::validateAfterN (
             const char * filename,
             base::type::LineNumber lineNumber,
             std::size_t n )
```

Validates counter for hits >= N, i.e, registers new if does not exist otherwise updates original one.

Returns

True if validation resulted in triggering hit. Meaning logs should be written everytime true is returned

Definition at line 1849 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::get(), el::base::threading::ThreadSa
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::registerNew(), and el::base::utils::RegistryWithP

8.39.2.3 validateEveryN()

```
bool el::base::RegisteredHitCounters::validateEveryN (
            const char * filename,
            base::type::LineNumber lineNumber,
            std::size_t n )
```

Validates counter for every N, i.e, registers new if does not exist otherwise updates original one.

Returns

True if validation resulted in triggering hit. Meaning logs should be written everytime true is returned

Definition at line 1836 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::get(), el::base::threading::ThreadSa
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::registerNew(), and el::base::utils::RegistryWithP

8.39.2.4 validateNTimes()

```
bool el::base::RegisteredHitCounters::validateNTimes (
            const char * filename,
            base::type::LineNumber lineNumber,
            std::size_t n )
```

Validates counter for hits are <= n, i.e, registers new if does not exist otherwise updates original one.

Returns

True if validation resulted in triggering hit. Meaning logs should be written everytime true is returned

Definition at line 1866 of file easylogging++.cc.

References el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::get(), el::base::threading::ThreadSa
el::base::utils::RegistryWithPred< base::HitCounter, base::HitCounter::Predicate >::registerNew(), and el::base::utils::RegistryWithP

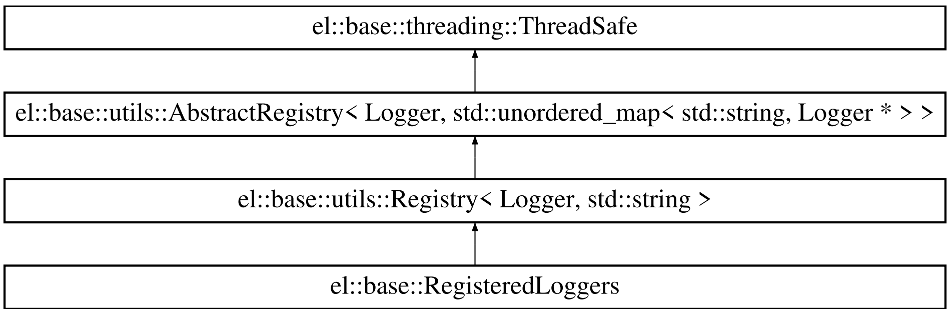
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.40 el::base::RegisteredLoggers Class Reference

Loggers repository.

Inheritance diagram for el::base::RegisteredLoggers:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< Logger, std::unordered_map< std::string, Logger * > >
el::base::utils::Registry< Logger, std::string >
el::base::RegisteredLoggers

Public Member Functions

- RegisteredLoggers (const LogBuilderPtr &defaultLogBuilder)
- virtual ∼RegisteredLoggers (void)
- void setDefaultConfigurations (const Configurations &configurations)
- Configurations ∗ defaultConfigurations (void)
- Logger ∗ get (const std::string &id, bool forceCreation=true)
- template<typename T >
  bool installLoggerRegistrationCallback (const std::string &id)
- template<typename T >
  void uninstallLoggerRegistrationCallback (const std::string &id)
- template<typename T >
  T ∗ loggerRegistrationCallback (const std::string &id)
- bool remove (const std::string &id)
- bool has (const std::string &id)
- void unregister (Logger ∗&logger)
- LogStreamsReferenceMapPtr logStreamsReference (void)
- void flushAll (void)
- void setDefaultLogBuilder (LogBuilderPtr &logBuilderPtr)

Public Member Functions inherited from el::base::utils::Registry< Logger, std::string >

- Registry (void)
- Registry (const Registry &sr)

  Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.
- Registry & operator= (const Registry &sr)

  Assignment operator that unregisters all the existing registries and deeply copies each of repo element.
- virtual ∼Registry (void)

Public Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, Container > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, Container > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ∼AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual Container & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const Container & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Member Functions

- void unsafeFlushAll (void)

Private Attributes

- LogBuilderPtr m_defaultLogBuilder
- Configurations m_defaultConfigurations
- base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr
- std::unordered_map< std::string, base::type::LoggerRegistrationCallbackPtr > m_loggerRegistrationCallbacks

Friends

- class el::base::Storage

Additional Inherited Members

Public Types inherited from el::base::utils::Registry< Logger, std::string >

- typedef Registry< Logger, std::string >::iterator iterator
- typedef Registry< Logger, std::string >::const_iterator const_iterator

Public Types inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Protected Member Functions inherited from el::base::utils::Registry< Logger, std::string >

- virtual void unregisterAll (void) ELPP_FINAL

  Unregisters all the pointers from current repository.
- virtual void registerNew (const std::string &uniqKey, Logger ∗ptr) ELPP_FINAL

  Registers new registry to repository.
- void unregister (const std::string &uniqKey)

  Unregisters single entry mapped to specified unique key.
- Logger ∗ get (const std::string &uniqKey)

  Gets pointer from repository. If none found, nullptr is returned.

Protected Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- virtual void deepCopy (const AbstractRegistry< T_Ptr, Container > &)=0
- void reinitDeepCopy (const AbstractRegistry< T_Ptr, Container > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

8.40.1 Detailed Description

Loggers repository.

Definition at line 2347 of file easylogging++.h.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 RegisteredLoggers()

```
el::base::RegisteredLoggers::RegisteredLoggers (
            const LogBuilderPtr & defaultLogBuilder )  [explicit]
```

Definition at line 1881 of file easylogging++.cc.

References m_defaultConfigurations, m_logStreamsReference, and el::Configurations::setToDefault().

8.40.2.2 ∼RegisteredLoggers()

```
virtual el::base::RegisteredLoggers::∼RegisteredLoggers (
            void  )  [inline], [virtual]
```

Definition at line 2351 of file easylogging++.h.

8.40.3 Member Function Documentation

8.40.3.1 defaultConfigurations()

```
Configurations * el::base::RegisteredLoggers::defaultConfigurations (
            void  )  [inline]
```

Definition at line 2360 of file easylogging++.h.

8.40.3.2 flushAll()

```
void el::base::RegisteredLoggers::flushAll (
            void ) [inline]
```

Definition at line 2397 of file easylogging++.h.

8.40.3.3 get()

```
Logger * el::base::RegisteredLoggers::get (
            const std::string & id,
            bool forceCreation = true )
```

Definition at line 1887 of file easylogging++.cc.

References ELPP_ASSERT, el::Callback< T >::enabled(), el::base::utils::Registry< T_Ptr, T_Key >::get(), el::Callback< T >::handle(), el::Logger::isValidId(), el::base::threading::ThreadSafe::lock(), m_defaultConfigurations, m_defaultLogBuilder, el::Logger::m_logBuilder, m_loggerRegistrationCallbacks, m_logStreamsReference, and el::base::utils::Registry< Logger, std::string >::registerNew().

8.40.3.4 has()

```
bool el::base::RegisteredLoggers::has (
            const std::string & id ) [inline]
```

Definition at line 2384 of file easylogging++.h.

8.40.3.5 installLoggerRegistrationCallback()

```
template<typename T >
bool el::base::RegisteredLoggers::installLoggerRegistrationCallback (
            const std::string & id ) [inline]
```

Definition at line 2367 of file easylogging++.h.

8.40.3.6 loggerRegistrationCallback()

```
template<typename T >
T * el::base::RegisteredLoggers::loggerRegistrationCallback (
            const std::string & id ) [inline]
```

Definition at line 2378 of file easylogging++.h.

8.40.3.7 logStreamsReference()

```
LogStreamsReferenceMapPtr el::base::RegisteredLoggers::logStreamsReference (
            void ) [inline]
```

Definition at line 2393 of file easylogging++.h.

8.40.3.8 remove()

```
bool el::base::RegisteredLoggers::remove (
            const std::string & id )
```

Definition at line 1911 of file easylogging++.cc.

References el::base::utils::Registry< T_Ptr, T_Key >::get(), el::base::consts::kDefaultLoggerId, and unregister().

8.40.3.9 setDefaultConfigurations()

```
void el::base::RegisteredLoggers::setDefaultConfigurations (
            const Configurations & configurations ) [inline]
```

Definition at line 2355 of file easylogging++.h.

8.40.3.10 setDefaultLogBuilder()

```
void el::base::RegisteredLoggers::setDefaultLogBuilder (
            LogBuilderPtr & logBuilderPtr ) [inline]
```

Definition at line 2402 of file easylogging++.h.

8.40.3.11 uninstallLoggerRegistrationCallback()

```
template<typename T >
void el::base::RegisteredLoggers::uninstallLoggerRegistrationCallback (
            const std::string & id ) [inline]
```

Definition at line 2373 of file easylogging++.h.

8.40.3.12 unregister()

```
void el::base::RegisteredLoggers::unregister (
            Logger *& logger ) [inline]
```

Definition at line 2388 of file easylogging++.h.

References el::Logger::id().

8.40.3.13 unsafeFlushAll()

```
void el::base::RegisteredLoggers::unsafeFlushAll (
            void ) [private]
```

Definition at line 1924 of file easylogging++.cc.

References ELPP_INTERNAL_INFO, and m_logStreamsReference.

8.40.4 Friends And Related Symbol Documentation

8.40.4.1 el::base::Storage

```
friend class el::base::Storage  [friend]
```

Definition at line 2412 of file easylogging++.h.

8.40.5 Field Documentation

8.40.5.1 m_defaultConfigurations

```
Configurations el::base::RegisteredLoggers::m_defaultConfigurations  [private]
```

Definition at line 2409 of file easylogging++.h.

8.40.5.2 m_defaultLogBuilder

```
LogBuilderPtr el::base::RegisteredLoggers::m_defaultLogBuilder  [private]
```

Definition at line 2408 of file easylogging++.h.

8.40.5.3 m_loggerRegistrationCallbacks

```
std::unordered_map<std::string, base::type::LoggerRegistrationCallbackPtr> el::base::Registered↩
Loggers::m_loggerRegistrationCallbacks  [private]
```

Definition at line 2411 of file easylogging++.h.

8.40.5.4 m_logStreamsReference

```
base::LogStreamsReferenceMapPtr el::base::RegisteredLoggers::m_logStreamsReference = nullptr
[private]
```

Definition at line 2410 of file easylogging++.h.

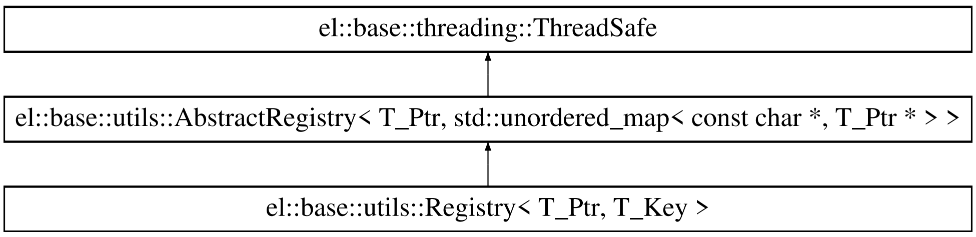
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.41 el::base::utils::Registry< T_Ptr, T_Key > Class Template Reference

A pointer registry mechanism to manage memory and provide search functionalities. (non-predicate version)

Inheritance diagram for el::base::utils::Registry< T_Ptr, T_Key >:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< T_Ptr, std::unordered_map< const char *, T_Ptr * > >
el::base::utils::Registry< T_Ptr, T_Key >

Public Types

- typedef Registry< T_Ptr, T_Key >::iterator iterator
- typedef Registry< T_Ptr, T_Key >::const_iterator const_iterator

Public Types inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Public Member Functions

- Registry (void)
- Registry (const Registry &sr)

  Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.
- Registry & operator= (const Registry &sr)

  Assignment operator that unregisters all the existing registries and deeply copies each of repo element.
- virtual ~Registry (void)

Public Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, Container > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, Container > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ~AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual Container & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const Container & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Protected Member Functions

- virtual void unregisterAll (void) ELPP_FINAL

    Unregisters all the pointers from current repository.
- virtual void registerNew (const T_Key &uniqKey, T_Ptr ∗ptr) ELPP_FINAL

    Registers new registry to repository.
- void unregister (const T_Key &uniqKey)

    Unregisters single entry mapped to specified unique key.
- T_Ptr ∗ get (const T_Key &uniqKey)

    Gets pointer from repository. If none found, nullptr is returned.

Protected Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, Container >

- virtual void deepCopy (const AbstractRegistry< T_Ptr, Container > &)=0
- void reinitDeepCopy (const AbstractRegistry< T_Ptr, Container > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

Private Member Functions

- virtual void deepCopy (const AbstractRegistry< T_Ptr, std::unordered_map< T_Key, T_Ptr ∗ > > &sr) ELPP_FINAL

8.41.1 Detailed Description

template<typename T_Ptr, typename T_Key = const char∗>
class el::base::utils::Registry< T_Ptr, T_Key >

A pointer registry mechanism to manage memory and provide search functionalities. (non-predicate version)

@detail NOTE: This is thread-unsafe implementation (although it contains lock function, it does not use these functions) of AbstractRegistry<T_Ptr, Container>. Any implementation of this class should be explicitly (by using lock functions)

Definition at line 1370 of file easylogging++.h.

8.41.2 Member Typedef Documentation

8.41.2.1 const_iterator

```
template<typename T_Ptr , typename T_Key = const char*>
typedef Registry<T_Ptr,T_Key>::const_iterator el::base::utils::Registry< T_Ptr, T_Key >↩
::const_iterator
```

Definition at line 1373 of file easylogging++.h.

8.41.2.2 iterator

```
template<typename T_Ptr , typename T_Key = const char*>
typedef Registry<T_Ptr,T_Key>::iterator el::base::utils::Registry< T_Ptr, T_Key >::iterator
```

Definition at line 1372 of file easylogging++.h.

8.41.3 Constructor & Destructor Documentation

8.41.3.1 Registry() [1/2]

```
template<typename T_Ptr , typename T_Key = const char*>
el::base::utils::Registry< T_Ptr, T_Key >::Registry (
            void ) [inline]
```

Definition at line 1375 of file easylogging++.h.

8.41.3.2 Registry() [2/2]

```
template<typename T_Ptr , typename T_Key = const char*>
el::base::utils::Registry< T_Ptr, T_Key >::Registry (
            const Registry< T_Ptr, T_Key > & sr ) [inline]
```

Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.

Definition at line 1378 of file easylogging++.h.

8.41.3.3 ∼Registry()

```
template<typename T_Ptr , typename T_Key = const char*>
virtual el::base::utils::Registry< T_Ptr, T_Key >::∼Registry (
            void ) [inline], [virtual]
```

Definition at line 1396 of file easylogging++.h.

8.41.4 Member Function Documentation

8.41.4.1 deepCopy()

```
template<typename T_Ptr , typename T_Key = const char*>
virtual void el::base::utils::Registry< T_Ptr, T_Key >::deepCopy (
             const AbstractRegistry< T_Ptr, std::unordered_map< T_Key, T_Ptr * > > & sr )
[inline], [private], [virtual]
```

Definition at line 1434 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::cbegin(), and el::base::utils::AbstractRegistry< T_Ptr, Container >

8.41.4.2 get()

```
template<typename T_Ptr , typename T_Key = const char*>
T_Ptr * el::base::utils::Registry< T_Ptr, T_Key >::get (
             const T_Key & uniqKey )  [inline], [protected]
```

Gets pointer from repository. If none found, nullptr is returned.

Definition at line 1426 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end().

8.41.4.3 operator=()

```
template<typename T_Ptr , typename T_Key = const char*>
Registry & el::base::utils::Registry< T_Ptr, T_Key >::operator= (
             const Registry< T_Ptr, T_Key > & sr )  [inline]
```

Assignment operator that unregisters all the existing registries and deeply copies each of repo element.

See also

unregisterAll()
deepCopy(const AbstractRegistry&)

Definition at line 1388 of file easylogging++.h.

8.41.4.4 registerNew()

```
template<typename T_Ptr , typename T_Key = const char*>
virtual void el::base::utils::Registry< T_Ptr, T_Key >::registerNew (
             const T_Key & uniqKey,
             T_Ptr * ptr )  [inline], [protected], [virtual]
```

Registers new registry to repository.

Definition at line 1411 of file easylogging++.h.

8.41.4.5 unregister()

```
template<typename T_Ptr , typename T_Key = const char*>
void el::base::utils::Registry< T_Ptr, T_Key >::unregister (
             const T_Key & uniqKey )  [inline], [protected]
```

Unregisters single entry mapped to specified unique key.

Definition at line 1417 of file easylogging++.h.

8.41.4.6 unregisterAll()

```
template<typename T_Ptr , typename T_Key = const char*>
virtual void el::base::utils::Registry< T_Ptr, T_Key >::unregisterAll (
             void  )  [inline], [protected], [virtual]
```

Unregisters all the pointers from current repository.

Implements el::base::utils::AbstractRegistry< T_Ptr, Container >.

Definition at line 1401 of file easylogging++.h.

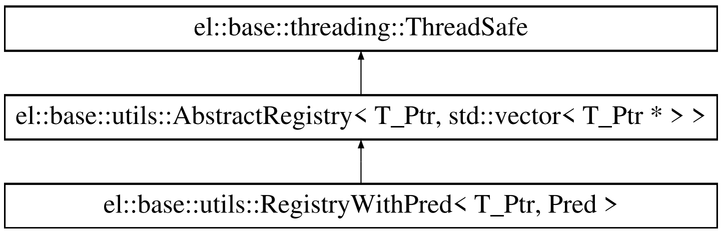
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.42 el::base::utils::RegistryWithPred< T_Ptr, Pred > Class Template Reference

A pointer registry mechanism to manage memory and provide search functionalities. (predicate version)

Inheritance diagram for el::base::utils::RegistryWithPred< T_Ptr, Pred >:


el::base::threading::ThreadSafe
el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr * > >
el::base::utils::RegistryWithPred< T_Ptr, Pred >

Public Types

- typedef RegistryWithPred< T_Ptr, Pred >::iterator iterator
- typedef RegistryWithPred< T_Ptr, Pred >::const_iterator const_iterator

Public Types inherited from el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > >

- typedef Container::iterator iterator
- typedef Container::const_iterator const_iterator

Public Member Functions

- RegistryWithPred (void)
- virtual ∼RegistryWithPred (void)
- RegistryWithPred (const RegistryWithPred &sr)

  Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.
- RegistryWithPred & operator= (const RegistryWithPred &sr)

  Assignment operator that unregisters all the existing registries and deeply copies each of repo element.

Public Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > >

- AbstractRegistry (void)

  Default constructor.
- AbstractRegistry (AbstractRegistry &&sr)

  Move constructor that is useful for base classes.
- bool operator== (const AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > > &other)
- bool operator!= (const AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > > &other)
- AbstractRegistry & operator= (AbstractRegistry &&sr)

  Assignment move operator.
- virtual ∼AbstractRegistry (void)
- virtual iterator begin (void) ELPP_FINAL
- virtual iterator end (void) ELPP_FINAL
- virtual const_iterator cbegin (void) const ELPP_FINAL
- virtual const_iterator cend (void) const ELPP_FINAL
- virtual bool empty (void) const ELPP_FINAL
- virtual std::size_t size (void) const ELPP_FINAL
- virtual std::vector< T_Ptr ∗ > & list (void) ELPP_FINAL

  Returns underlying container by reference.
- virtual const std::vector< T_Ptr ∗ > & list (void) const ELPP_FINAL

  Returns underlying container by constant reference.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Protected Member Functions

- virtual void unregisterAll (void) ELPP_FINAL

  Unregisters all the pointers from current repository.
- virtual void unregister (T_Ptr ∗&ptr) ELPP_FINAL
- virtual void registerNew (T_Ptr ∗ptr) ELPP_FINAL
- template<typename T , typename T2 >
  T_Ptr ∗ get (const T &arg1, const T2 arg2)

  Gets pointer from repository with specified arguments. Arguments are passed to predicate in order to validate pointer.

Protected Member Functions inherited from el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > >

- void reinitDeepCopy (const AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > > &sr)

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

Private Member Functions

- virtual void deepCopy (const AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > > &sr)

Friends

- base::type::ostream_t & operator<< (base::type::ostream_t &os, const RegistryWithPred &sr)

8.42.1 Detailed Description

template<typename T_Ptr, typename Pred>
class el::base::utils::RegistryWithPred< T_Ptr, Pred >

A pointer registry mechanism to manage memory and provide search functionalities. (predicate version)

@detail NOTE: This is thread-unsafe implementation of AbstractRegistry<T_Ptr, Container>. Any implementation of this class should be made thread-safe explicitly

Definition at line 1446 of file easylogging++.h.

8.42.2 Member Typedef Documentation

8.42.2.1 const_iterator

```
template<typename T_Ptr , typename Pred >
typedef RegistryWithPred<T_Ptr,Pred>::const_iterator el::base::utils::RegistryWithPred< T_Ptr,
Pred >::const_iterator
```

Definition at line 1449 of file easylogging++.h.

8.42.2.2 iterator

```
template<typename T_Ptr , typename Pred >
typedef RegistryWithPred<T_Ptr,Pred>::iterator el::base::utils::RegistryWithPred< T_Ptr, Pred
>::iterator
```

Definition at line 1448 of file easylogging++.h.

8.42.3 Constructor & Destructor Documentation

8.42.3.1 RegistryWithPred() [1/2]

```
template<typename T_Ptr , typename Pred >
el::base::utils::RegistryWithPred< T_Ptr, Pred >::RegistryWithPred (
            void ) [inline]
```

Definition at line 1451 of file easylogging++.h.

8.42.3.2 ∼RegistryWithPred()

```
template<typename T_Ptr , typename Pred >
virtual el::base::utils::RegistryWithPred< T_Ptr, Pred >::∼RegistryWithPred (
            void ) [inline], [virtual]
```

Definition at line 1454 of file easylogging++.h.

8.42.3.3 RegistryWithPred() [2/2]

```
template<typename T_Ptr , typename Pred >
el::base::utils::RegistryWithPred< T_Ptr, Pred >::RegistryWithPred (
            const RegistryWithPred< T_Ptr, Pred > & sr ) [inline]
```

Copy constructor that is useful for base classes. Try to avoid this constructor, use move constructor.

Definition at line 1459 of file easylogging++.h.

8.42.4 Member Function Documentation

8.42.4.1 deepCopy()

```
template<typename T_Ptr , typename Pred >
virtual void el::base::utils::RegistryWithPred< T_Ptr, Pred >::deepCopy (
            const AbstractRegistry< T_Ptr, std::vector< T_Ptr * > > & sr ) [inline], [private],
[virtual]
```

Implements el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr ∗ > >.

Definition at line 1525 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin(), el::base::utils::AbstractRegistry< T_Ptr, Container >::end
and el::base::utils::AbstractRegistry< T_Ptr, Container >::list().

8.42.4.2 get()

```
template<typename T_Ptr , typename Pred >
template<typename T , typename T2 >
T_Ptr * el::base::utils::RegistryWithPred< T_Ptr, Pred >::get (
             const T & arg1,
             const T2 arg2 )  [inline], [protected]
```

Gets pointer from repository with specified arguments. Arguments are passed to predicate in order to validate pointer.

Definition at line 1516 of file easylogging++.h.

8.42.4.3 operator=()

```
template<typename T_Ptr , typename Pred >
RegistryWithPred & el::base::utils::RegistryWithPred< T_Ptr, Pred >::operator= (
             const RegistryWithPred< T_Ptr, Pred > & sr )  [inline]
```

Assignment operator that unregisters all the existing registries and deeply copies each of repo element.

See also

unregisterAll()

deepCopy(const AbstractRegistry&)

Definition at line 1469 of file easylogging++.h.

8.42.4.4 registerNew()

```
template<typename T_Ptr , typename Pred >
virtual void el::base::utils::RegistryWithPred< T_Ptr, Pred >::registerNew (
             T_Ptr * ptr )  [inline], [protected], [virtual]
```

Definition at line 1509 of file easylogging++.h.

8.42.4.5 unregister()

```
template<typename T_Ptr , typename Pred >
virtual void el::base::utils::RegistryWithPred< T_Ptr, Pred >::unregister (
             T_Ptr *& ptr )  [inline], [protected], [virtual]
```

Definition at line 1494 of file easylogging++.h.

8.42.4.6 unregisterAll()

```
template<typename T_Ptr , typename Pred >
virtual void el::base::utils::RegistryWithPred< T_Ptr, Pred >::unregisterAll (
             void  )  [inline], [protected], [virtual]
```

Unregisters all the pointers from current repository.

Implements el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr * > >.

Definition at line 1485 of file easylogging++.h.

8.42.5 Friends And Related Symbol Documentation

8.42.5.1 operator<<

```
template<typename T_Ptr , typename Pred >
base::type::ostream_t & operator<< (
            base::type::ostream_t & os,
            const RegistryWithPred< T_Ptr, Pred > & sr )  [friend]
```

Definition at line 1477 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.43 el::Loggers::ScopedAddFlag Class Reference

Adds flag and removes it when scope goes out.

```
#include <easylogging++.h>
```

Public Member Functions

- ScopedAddFlag (LoggingFlag flag)
- ∼ScopedAddFlag (void)

Private Attributes

- LoggingFlag m_flag

8.43.1 Detailed Description

Adds flag and removes it when scope goes out.

Definition at line 3858 of file easylogging++.h.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 ScopedAddFlag()

```
el::Loggers::ScopedAddFlag::ScopedAddFlag (
            LoggingFlag flag )  [inline]
```

Definition at line 3860 of file easylogging++.h.

8.43.2.2 ∼ScopedAddFlag()

```
el::Loggers::ScopedAddFlag::∼ScopedAddFlag (
            void ) [inline]
```

Definition at line 3863 of file easylogging++.h.

8.43.3 Field Documentation

8.43.3.1 m_flag

```
LoggingFlag el::Loggers::ScopedAddFlag::m_flag [private]
```

Definition at line 3867 of file easylogging++.h.

The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.44 el::Loggers::ScopedRemoveFlag Class Reference

Removes flag and add it when scope goes out.

```
#include <easylogging++.h>
```

Public Member Functions

- ScopedRemoveFlag (LoggingFlag flag)
- ∼ScopedRemoveFlag (void)

Private Attributes

- LoggingFlag m_flag

8.44.1 Detailed Description

Removes flag and add it when scope goes out.

Definition at line 3870 of file easylogging++.h.

8.44.2 Constructor & Destructor Documentation

8.44.2.1 ScopedRemoveFlag()

```
el::Loggers::ScopedRemoveFlag::ScopedRemoveFlag (
            LoggingFlag flag ) [inline]
```

Definition at line 3872 of file easylogging++.h.

8.44.2.2 ∼ScopedRemoveFlag()

```
el::Loggers::ScopedRemoveFlag::∼ScopedRemoveFlag (
             void  )  [inline]
```

Definition at line 3875 of file easylogging++.h.

8.44.3 Field Documentation

8.44.3.1 m_flag

```
LoggingFlag el::Loggers::ScopedRemoveFlag::m_flag  [private]
```

Definition at line 3879 of file easylogging++.h.

The documentation for this class was generated from the following file:

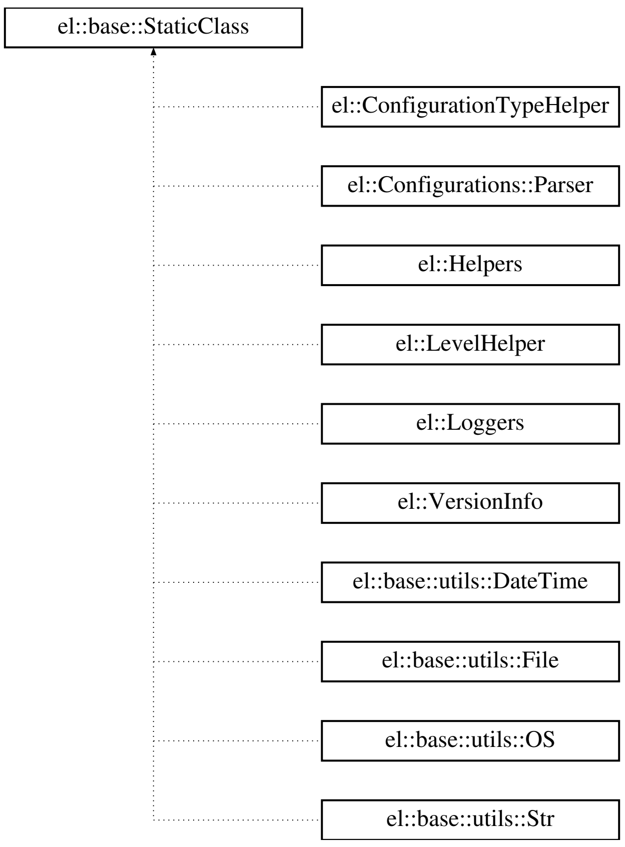
- lib/easylogging++.h

8.45 el::base::StaticClass Class Reference

Internal helper class that makes all default constructors private.

```
#include <easylogging++.h>
```

Inheritance diagram for el::base::StaticClass:



Private Member Functions

- StaticClass (void)
- StaticClass (const StaticClass &)
- StaticClass & operator= (const StaticClass &)

8.45.1 Detailed Description

Internal helper class that makes all default constructors private.

@detail This prevents initializing class making it static unless an explicit constructor is declared. When using this class simply inherit it privately

Definition at line 562 of file easylogging++.h.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 StaticClass() [1/2]

```
el::base::StaticClass::StaticClass (
            void  )  [private]
```

8.45.2.2 StaticClass() [2/2]

```
el::base::StaticClass::StaticClass (
            const StaticClass &  )  [private]
```

8.45.3 Member Function Documentation

8.45.3.1 operator=()

```
StaticClass & el::base::StaticClass::operator= (
            const StaticClass &  )  [private]
```

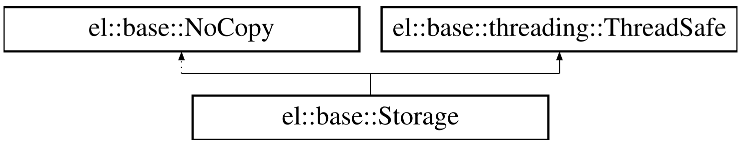
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.46 el::base::Storage Class Reference

Easylogging++ management storage.

Inheritance diagram for el::base::Storage:



Public Member Functions

- Storage (const LogBuilderPtr &defaultLogBuilder)
- virtual ∼Storage (void)
- bool validateEveryNCounter (const char ∗filename, base::type::LineNumber lineNumber, std::size_t occasion)
- bool validateAfterNCounter (const char ∗filename, base::type::LineNumber lineNumber, std::size_t n)
- bool validateNTimesCounter (const char ∗filename, base::type::LineNumber lineNumber, std::size_t n)
- base::RegisteredHitCounters ∗ hitCounters (void) const
- base::RegisteredLoggers ∗ registeredLoggers (void) const
- base::VRegistry ∗ vRegistry (void) const
- const base::utils::CommandLineArgs ∗ commandLineArgs (void) const
- void addFlag (LoggingFlag flag)
- void removeFlag (LoggingFlag flag)
- bool hasFlag (LoggingFlag flag) const
- base::type::EnumType flags (void) const
- void setFlags (base::type::EnumType flags)
- void setPreRollOutCallback (const PreRollOutCallback &callback)
- void unsetPreRollOutCallback (void)
- PreRollOutCallback & preRollOutCallback (void)
- bool hasCustomFormatSpecifier (const char ∗formatSpecifier)
- void installCustomFormatSpecifier (const CustomFormatSpecifier &customFormatSpecifier)
- bool uninstallCustomFormatSpecifier (const char ∗formatSpecifier)
- const std::vector< CustomFormatSpecifier > ∗ customFormatSpecifiers (void) const
- base::threading::Mutex & customFormatSpecifiersLock ()
- void setLoggingLevel (Level level)
- template<typename T >
  bool installLogDispatchCallback (const std::string &id)
- template<typename T >
  void uninstallLogDispatchCallback (const std::string &id)
- template<typename T >
  T ∗ logDispatchCallback (const std::string &id)
- void setThreadName (const std::string &name)

  Sets thread name for current thread. Requires std::thread.
- std::string getThreadName (const std::string &threadId)

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Member Functions

- void setApplicationArguments (int argc, char ∗∗argv)
- void setApplicationArguments (int argc, const char ∗∗argv)

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

Private Attributes

- base::RegisteredHitCounters $*$ m_registeredHitCounters
- base::RegisteredLoggers $*$ m_registeredLoggers
- base::type::EnumType m_flags
- base::VRegistry $*$ m_vRegistry
- base::utils::CommandLineArgs m_commandLineArgs
- PreRollOutCallback m_preRollOutCallback
- std::unordered_map< std::string, base::type::LogDispatchCallbackPtr > m_logDispatchCallbacks
- std::unordered_map< std::string, base::type::PerformanceTrackingCallbackPtr > m_performanceTrackingCallbacks
- std::unordered_map< std::string, std::string > m_threadNames
- std::vector< CustomFormatSpecifier > m_customFormatSpecifiers
- base::threading::Mutex m_customFormatSpecifiersLock
- base::threading::Mutex m_threadNamesLock
- Level m_loggingLevel

Friends

- class el::Helpers
- class el::base::DefaultLogDispatchCallback
- class el::LogBuilder
- class el::base::MessageBuilder
- class el::base::Writer
- class el::base::PerformanceTracker
- class el::base::LogDispatcher

Additional Inherited Members

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

8.46.1 Detailed Description

Easylogging++ management storage.

Definition at line 2551 of file easylogging++.h.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 Storage()

```
el::base::Storage::Storage (
            const LogBuilderPtr & defaultLogBuilder )  [explicit]
```

Definition at line 2061 of file easylogging++.cc.

References el::AllowVerboseIfModuleNotSpecified, el::Logger::configurations(), ELPP_INTERNAL_INFO, el::Format, el::base::consts::kDefaultLoggerId, el::Logger::reconfigure(), and el::Configurations::setGlobally().

8.46.2.2 ∼Storage()

```
el::base::Storage::∼Storage (
            void ) [virtual]
```

Definition at line 2109 of file easylogging++.cc.

References ELPP_INTERNAL_INFO, m_registeredHitCounters, m_registeredLoggers, m_vRegistry, and el::base::utils::safeDelete().

8.46.3 Member Function Documentation

8.46.3.1 addFlag()

```
void el::base::Storage::addFlag (
            LoggingFlag flag ) [inline]
```

Definition at line 2595 of file easylogging++.h.

8.46.3.2 commandLineArgs()

```
const base::utils::CommandLineArgs * el::base::Storage::commandLineArgs (
            void ) const [inline]
```

Definition at line 2591 of file easylogging++.h.

8.46.3.3 customFormatSpecifiers()

```
const std::vector< CustomFormatSpecifier > * el::base::Storage::customFormatSpecifiers (
            void ) const [inline]
```

Definition at line 2631 of file easylogging++.h.

8.46.3.4 customFormatSpecifiersLock()

```
base::threading::Mutex & el::base::Storage::customFormatSpecifiersLock ( ) [inline]
```

Definition at line 2635 of file easylogging++.h.

8.46.3.5 flags()

```
base::type::EnumType el::base::Storage::flags (
            void ) const [inline]
```

Definition at line 2607 of file easylogging++.h.

8.46.3.6 getThreadName()

```
std::string el::base::Storage::getThreadName (
             const std::string & threadId )  [inline]
```

Definition at line 2683 of file easylogging++.h.

8.46.3.7 hasCustomFormatSpecifier()

```
bool el::base::Storage::hasCustomFormatSpecifier (
             const char * formatSpecifier )
```

Definition at line 2128 of file easylogging++.cc.

References customFormatSpecifiersLock(), and m_customFormatSpecifiers.

8.46.3.8 hasFlag()

```
bool el::base::Storage::hasFlag (
             LoggingFlag flag ) const  [inline]
```

Definition at line 2603 of file easylogging++.h.

8.46.3.9 hitCounters()

```
base::RegisteredHitCounters * el::base::Storage::hitCounters (
             void  ) const  [inline]
```

Definition at line 2573 of file easylogging++.h.

8.46.3.10 installCustomFormatSpecifier()

```
void el::base::Storage::installCustomFormatSpecifier (
             const CustomFormatSpecifier & customFormatSpecifier )
```

Definition at line 2134 of file easylogging++.cc.

References customFormatSpecifiersLock(), el::CustomFormatSpecifier::formatSpecifier(), hasCustomFormatSpecifier(), and m_customFormatSpecifiers.

8.46.3.11 installLogDispatchCallback()

```
template<typename T >
bool el::base::Storage::installLogDispatchCallback (
             const std::string & id )  [inline]
```

Definition at line 2644 of file easylogging++.h.

8.46.3.12 logDispatchCallback()

```
template<typename T >
T * el::base::Storage::logDispatchCallback (
            const std::string & id )  [inline]
```

Definition at line 2653 of file easylogging++.h.

8.46.3.13 preRollOutCallback()

```
PreRollOutCallback & el::base::Storage::preRollOutCallback (
            void  )  [inline]
```

Definition at line 2623 of file easylogging++.h.

8.46.3.14 registeredLoggers()

```
base::RegisteredLoggers * el::base::Storage::registeredLoggers (
            void  ) const  [inline]
```

Definition at line 2577 of file easylogging++.h.

8.46.3.15 removeFlag()

```
void el::base::Storage::removeFlag (
            LoggingFlag flag )  [inline]
```

Definition at line 2599 of file easylogging++.h.

8.46.3.16 setApplicationArguments() [1/2]

```
void el::base::Storage::setApplicationArguments (
            int argc,
            char ** argv )  [private]
```

Definition at line 2153 of file easylogging++.cc.

References commandLineArgs(), ELPP_DEFAULT_LOGGING_FLAGS, el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), el::Filename, el::base::utils::CommandLineArgs::getParamValue(), el::base::utils::CommandLineArgs::hasParamWithValue(), el::base::consts::kDefaultLogFileParam, m_commandLineArgs, m_flags, m_vRegistry, registeredLoggers(), el::base::utils::CommandLineArgs::setArgs(), el::base::RegisteredLoggers::setDefaultConfigurations(), el::base::VRegistry::setFromA
and el::Configurations::setGlobally().

8.46.3.17 setApplicationArguments() [2/2]

```
void el::base::Storage::setApplicationArguments (
            int argc,
            const char ** argv )  [inline], [private]
```

Definition at line 2720 of file easylogging++.h.

8.46.3.18 setFlags()

```
void el::base::Storage::setFlags (
            base::type::EnumType flags )  [inline]
```

Definition at line 2611 of file easylogging++.h.

8.46.3.19 setLoggingLevel()

```
void el::base::Storage::setLoggingLevel (
            Level level )  [inline]
```

Definition at line 2639 of file easylogging++.h.

8.46.3.20 setPreRollOutCallback()

```
void el::base::Storage::setPreRollOutCallback (
            const PreRollOutCallback & callback )  [inline]
```

Definition at line 2615 of file easylogging++.h.

8.46.3.21 setThreadName()

```
void el::base::Storage::setThreadName (
            const std::string & name )  [inline]
```

Sets thread name for current thread. Requires std::thread.

Definition at line 2677 of file easylogging++.h.

8.46.3.22 uninstallCustomFormatSpecifier()

```
bool el::base::Storage::uninstallCustomFormatSpecifier (
            const char * formatSpecifier )
```

Definition at line 2142 of file easylogging++.cc.

References customFormatSpecifiersLock(), and m_customFormatSpecifiers.

8.46.3.23 uninstallLogDispatchCallback()

```
template<typename T >
void el::base::Storage::uninstallLogDispatchCallback (
            const std::string & id )  [inline]
```

Definition at line 2649 of file easylogging++.h.

8.46.3.24 unsetPreRollOutCallback()

```
void el::base::Storage::unsetPreRollOutCallback (
             void  )  [inline]
```

Definition at line 2619 of file easylogging++.h.

8.46.3.25 validateAfterNCounter()

```
bool el::base::Storage::validateAfterNCounter (
             const char * filename,
             base::type::LineNumber lineNumber,
             std::size_t n )  [inline]
```

Definition at line 2565 of file easylogging++.h.

8.46.3.26 validateEveryNCounter()

```
bool el::base::Storage::validateEveryNCounter (
             const char * filename,
             base::type::LineNumber lineNumber,
             std::size_t occasion )  [inline]
```

Definition at line 2561 of file easylogging++.h.

8.46.3.27 validateNTimesCounter()

```
bool el::base::Storage::validateNTimesCounter (
             const char * filename,
             base::type::LineNumber lineNumber,
             std::size_t n )  [inline]
```

Definition at line 2569 of file easylogging++.h.

8.46.3.28 vRegistry()

```
base::VRegistry * el::base::Storage::vRegistry (
             void  ) const  [inline]
```

Definition at line 2581 of file easylogging++.h.

8.46.4 Friends And Related Symbol Documentation

8.46.4.1 el::base::DefaultLogDispatchCallback

```
friend class el::base::DefaultLogDispatchCallback  [friend]
```

Definition at line 2711 of file easylogging++.h.

8.46.4.2 el::base::LogDispatcher

```
friend class el::base::LogDispatcher  [friend]
```

Definition at line 2716 of file easylogging++.h.

8.46.4.3 el::base::MessageBuilder

```
friend class el::base::MessageBuilder  [friend]
```

Definition at line 2713 of file easylogging++.h.

8.46.4.4 el::base::PerformanceTracker

```
friend class el::base::PerformanceTracker  [friend]
```

Definition at line 2715 of file easylogging++.h.

8.46.4.5 el::base::Writer

```
friend class el::base::Writer  [friend]
```

Definition at line 2714 of file easylogging++.h.

8.46.4.6 el::Helpers

```
friend class el::Helpers  [friend]
```

Definition at line 2710 of file easylogging++.h.

8.46.4.7 el::LogBuilder

```
friend class el::LogBuilder  [friend]
```

Definition at line 2712 of file easylogging++.h.

8.46.5 Field Documentation

8.46.5.1 m_commandLineArgs

```
base::utils::CommandLineArgs el::base::Storage::m_commandLineArgs  [private]
```

Definition at line 2700 of file easylogging++.h.

8.46.5.2 m_customFormatSpecifiers

```
std::vector<CustomFormatSpecifier> el::base::Storage::m_customFormatSpecifiers  [private]
```

Definition at line 2705 of file easylogging++.h.

8.46.5.3 m_customFormatSpecifiersLock

```
base::threading::Mutex el::base::Storage::m_customFormatSpecifiersLock  [private]
```

Definition at line 2706 of file easylogging++.h.

8.46.5.4 m_flags

```
base::type::EnumType el::base::Storage::m_flags  [private]
```

Definition at line 2694 of file easylogging++.h.

8.46.5.5 m_logDispatchCallbacks

```
std::unordered_map<std::string, base::type::LogDispatchCallbackPtr> el::base::Storage::m_log←
DispatchCallbacks  [private]
```

Definition at line 2702 of file easylogging++.h.

8.46.5.6 m_loggingLevel

```
Level el::base::Storage::m_loggingLevel  [private]
```

Definition at line 2708 of file easylogging++.h.

8.46.5.7 m_performanceTrackingCallbacks

```
std::unordered_map<std::string, base::type::PerformanceTrackingCallbackPtr> el::base::Storage←
::m_performanceTrackingCallbacks  [private]
```

Definition at line 2703 of file easylogging++.h.

8.46.5.8 m_preRollOutCallback

```
PreRollOutCallback el::base::Storage::m_preRollOutCallback  [private]
```

Definition at line 2701 of file easylogging++.h.

8.46.5.9 m_registeredHitCounters

```
base::RegisteredHitCounters* el::base::Storage::m_registeredHitCounters  [private]
```

Definition at line 2692 of file easylogging++.h.

8.46.5.10 m_registeredLoggers

```
base::RegisteredLoggers* el::base::Storage::m_registeredLoggers  [private]
```

Definition at line 2693 of file easylogging++.h.

8.46.5.11 m_threadNames

```
std::unordered_map<std::string, std::string> el::base::Storage::m_threadNames  [private]
```

Definition at line 2704 of file easylogging++.h.

8.46.5.12 m_threadNamesLock

```
base::threading::Mutex el::base::Storage::m_threadNamesLock  [private]
```

Definition at line 2707 of file easylogging++.h.

8.46.5.13 m_vRegistry

```
base::VRegistry* el::base::Storage::m_vRegistry  [private]
```

Definition at line 2695 of file easylogging++.h.

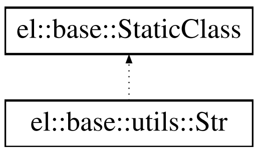
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.47 el::base::utils::Str Class Reference

String utilities helper class used internally. You should not use it.

Inheritance diagram for el::base::utils::Str:



Static Public Member Functions

- static bool isDigit (char c)

  Checks if character is digit. Dont use libc implementation of it to prevent locale issues.
- static bool wildCardMatch (const char ∗str, const char ∗pattern)

  Matches wildcards, '∗' and '?' only supported.
- static std::string & ltrim (std::string &str)
- static std::string & rtrim (std::string &str)
- static std::string & trim (std::string &str)
- static bool startsWith (const std::string &str, const std::string &start)

  Determines whether or not str starts with specified string.
- static bool endsWith (const std::string &str, const std::string &end)

  Determines whether or not str ends with specified string.
- static std::string & replaceAll (std::string &str, char replaceWhat, char replaceWith)

  Replaces all instances of replaceWhat with 'replaceWith'. Original variable is changed for performance.
- static std::string & replaceAll (std::string &str, const std::string &replaceWhat, const std::string &replaceWith)

  Replaces all instances of 'replaceWhat' with 'replaceWith'. (String version) Replaces in place.
- static void replaceFirstWithEscape (base::type::string_t &str, const base::type::string_t &replaceWhat, const base::type::string_t &replaceWith)
- static std::string & toUpper (std::string &str)

  Converts string to uppercase.
- static bool cStringEq (const char ∗s1, const char ∗s2)

  Compares cstring equality - uses strcmp.
- static bool cStringCaseEq (const char ∗s1, const char ∗s2)

  Compares cstring equality (case-insensitive) - uses toupper(char) Dont use strcasecmp because of CRT (VC++)
- static bool contains (const char ∗str, char c)

  Returns true if c exist in str.
- static char ∗ convertAndAddToBuff (std::size_t n, int len, char ∗buf, const char ∗bufLim, bool zeroPadded=true)
- static char ∗ addToBuff (const char ∗str, char ∗buf, const char ∗bufLim)
- static char ∗ clearBuff (char buff[ ], std::size_t lim)
- static char ∗ wcharPtrToCharPtr (const wchar_t ∗line)

  Converts wchar∗ to char∗ NOTE: Need to free return value after use!

8.47.1 Detailed Description

String utilities helper class used internally. You should not use it.

Definition at line 1066 of file easylogging++.h.

8.47.2 Member Function Documentation

8.47.2.1 addToBuff()

```
char * el::base::utils::Str::addToBuff (
            const char * str,
            char * buf,
            const char * bufLim ) [static]
```

Definition at line 1000 of file easylogging++.cc.

8.47.2.2 clearBuff()

```
char * el::base::utils::Str::clearBuff (
             char buff[],
             std::size_t lim )  [static]
```

Definition at line 1006 of file easylogging++.cc.

References ELPP_UNUSED, and STRCPY.

8.47.2.3 contains()

```
bool el::base::utils::Str::contains (
             const char * str,
             char c )  [static]
```

Returns true if c exist in str.

Definition at line 977 of file easylogging++.cc.

8.47.2.4 convertAndAddToBuff()

```
char * el::base::utils::Str::convertAndAddToBuff (
             std::size_t n,
             int len,
             char * buf,
             const char * bufLim,
             bool zeroPadded = true )  [static]
```

Definition at line 985 of file easylogging++.cc.

References addToBuff().

8.47.2.5 cStringCaseEq()

```
bool el::base::utils::Str::cStringCaseEq (
             const char * s1,
             const char * s2 )  [static]
```

Compares cstring equality (case-insensitive) - uses toupper(char) Dont use strcasecmp because of CRT (VC++)

Definition at line 958 of file easylogging++.cc.

8.47.2.6 cStringEq()

```
bool el::base::utils::Str::cStringEq (
             const char * s1,
             const char * s2 )  [static]
```

Compares cstring equality - uses strcmp.

Definition at line 952 of file easylogging++.cc.

8.47.2.7 endsWith()

```
bool el::base::utils::Str::endsWith (
             const std::string & str,
             const std::string & end )  [static]
```

Determines whether or not str ends with specified string.

Parameters

str	String to check
end	String to check against

Returns

Returns true if ends with specified string, false otherwise

Definition at line 904 of file easylogging++.cc.

8.47.2.8 isDigit()

```
static bool el::base::utils::Str::isDigit (
            char c )  [inline], [static]
```

Checks if character is digit. Dont use libc implementation of it to prevent locale issues.

Definition at line 1069 of file easylogging++.h.

8.47.2.9 ltrim()

```
std::string & el::base::utils::Str::ltrim (
            std::string & str )  [static]
```

Definition at line 882 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin(), and el::base::utils::AbstractRegistry< T_Ptr, Container >::

8.47.2.10 replaceAll() `[1/2]`

```
std::string & el::base::utils::Str::replaceAll (
            std::string & str,
            char replaceWhat,
            char replaceWith )  [static]
```

Replaces all instances of replaceWhat with 'replaceWith'. Original variable is changed for performance.

Parameters

`in,out`	*str*	String to replace from
	replaceWhat	Character to replace
	replaceWith	Character to replace with

Returns

Modified version of str

Definition at line 908 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin(), and el::base::utils::AbstractRegistry< T_Ptr, Container >::

8.47.2.11 replaceAll() [2/2]

```
std::string & el::base::utils::Str::replaceAll (
             std::string & str,
             const std::string & replaceWhat,
             const std::string & replaceWith )  [static]
```

Replaces all instances of 'replaceWhat' with 'replaceWith'. (String version) Replaces in place.

Parameters

str	String to replace from
replaceWhat	Character to replace
replaceWith	Character to replace with

Returns

Modified (original) str

Definition at line 913 of file easylogging++.cc.

8.47.2.12 replaceFirstWithEscape()

```
void el::base::utils::Str::replaceFirstWithEscape (
             base::type::string_t & str,
             const base::type::string_t & replaceWhat,
             const base::type::string_t & replaceWith )  [static]
```

Definition at line 924 of file easylogging++.cc.

References el::base::consts::kFormatSpecifierChar.

8.47.2.13 rtrim()

```
std::string & el::base::utils::Str::rtrim (
             std::string & str )  [static]
```

Definition at line 889 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end().

8.47.2.14 startsWith()

```
bool el::base::utils::Str::startsWith (
             const std::string & str,
             const std::string & start )  [static]
```

Determines whether or not str starts with specified string.

Parameters

str	String to check
start	String to check against

Returns

Returns true if starts with specified string, false otherwise

Definition at line 900 of file easylogging++.cc.

8.47.2.15 toUpper()

```
std::string & el::base::utils::Str::toUpper (
            std::string & str )  [static]
```

Converts string to uppercase.

Parameters

str	String to convert

Returns

Uppercase string

Definition at line 944 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin(), and el::base::utils::AbstractRegistry< T_Ptr, Container >::

8.47.2.16 trim()

```
std::string & el::base::utils::Str::trim (
            std::string & str )  [static]
```

Definition at line 896 of file easylogging++.cc.

References ltrim(), and rtrim().

8.47.2.17 wcharPtrToCharPtr()

```
char * el::base::utils::Str::wcharPtrToCharPtr (
            const wchar_t * line )  [static]
```

Converts wchar∗ to char∗ NOTE: Need to free return value after use!

Definition at line 1014 of file easylogging++.cc.

8.47.2.18 wildCardMatch()

```
bool el::base::utils::Str::wildCardMatch (
             const char * str,
             const char * pattern )  [static]
```

Matches wildcards, '*' and '?' only supported.

Definition at line 858 of file easylogging++.cc.

References wildCardMatch().

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.48 el::StringToLevelItem Struct Reference

Data Fields

- const char * levelString
- Level level

8.48.1 Detailed Description

Definition at line 145 of file easylogging++.cc.

8.48.2 Field Documentation

8.48.2.1 level

```
Level el::StringToLevelItem::level
```

Definition at line 147 of file easylogging++.cc.

8.48.2.2 levelString

```
const char* el::StringToLevelItem::levelString
```

Definition at line 146 of file easylogging++.cc.

The documentation for this struct was generated from the following file:

- lib/easylogging++.cc

8.49 el::base::SubsecondPrecision Class Reference

A subsecond precision class containing actual width and offset of the subsecond part.

```
#include <easylogging++.h>
```

Public Member Functions

- SubsecondPrecision (void)
- SubsecondPrecision (int width)
- bool operator== (const SubsecondPrecision &ssPrec)

Data Fields

- int m_width
- unsigned int m_offset

Private Member Functions

- void init (int width)

8.49.1 Detailed Description

A subsecond precision class containing actual width and offset of the subsecond part.

Definition at line 834 of file easylogging++.h.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 SubsecondPrecision() [1/2]

```
el::base::SubsecondPrecision::SubsecondPrecision (
            void ) [inline]
```

Definition at line 836 of file easylogging++.h.

References init(), and el::base::consts::kDefaultSubsecondPrecision.

8.49.2.2 SubsecondPrecision() [2/2]

```
el::base::SubsecondPrecision::SubsecondPrecision (
            int width ) [inline], [explicit]
```

Definition at line 839 of file easylogging++.h.

References init().

8.49.3 Member Function Documentation

8.49.3.1 init()

```
void el::base::SubsecondPrecision::init (
             int width )  [private]
```

Definition at line 1404 of file easylogging++.cc.

References el::base::consts::kDefaultSubsecondPrecision, m_offset, and m_width.

8.49.3.2 operator==()

```
bool el::base::SubsecondPrecision::operator== (
             const SubsecondPrecision & ssPrec )  [inline]
```

Definition at line 842 of file easylogging++.h.

References m_offset, and m_width.

8.49.4 Field Documentation

8.49.4.1 m_offset

```
unsigned int el::base::SubsecondPrecision::m_offset
```

Definition at line 846 of file easylogging++.h.

8.49.4.2 m_width

```
int el::base::SubsecondPrecision::m_width
```

Definition at line 845 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.50 el::SysLogInitializer Class Reference

Initializes syslog with process ID, options and facility. calls closelog() on d'tor.

```
#include <easylogging++.h>
```

Public Member Functions

- SysLogInitializer (const char ∗processIdent, int options=0, int facility=0)
- virtual ∼SysLogInitializer (void)

8.50.1 Detailed Description

Initializes syslog with process ID, options and facility. calls closelog() on d'tor.

Definition at line 3633 of file easylogging++.h.

8.50.2 Constructor & Destructor Documentation

8.50.2.1 SysLogInitializer()

```
el::SysLogInitializer::SysLogInitializer (
            const char * processIdent,
            int options = 0,
            int facility = 0 )  [inline]
```

Definition at line 3635 of file easylogging++.h.

References ELPP_UNUSED.

8.50.2.2 ∼SysLogInitializer()

```
virtual el::SysLogInitializer::∼SysLogInitializer (
            void  )  [inline], [virtual]
```

Definition at line 3645 of file easylogging++.h.

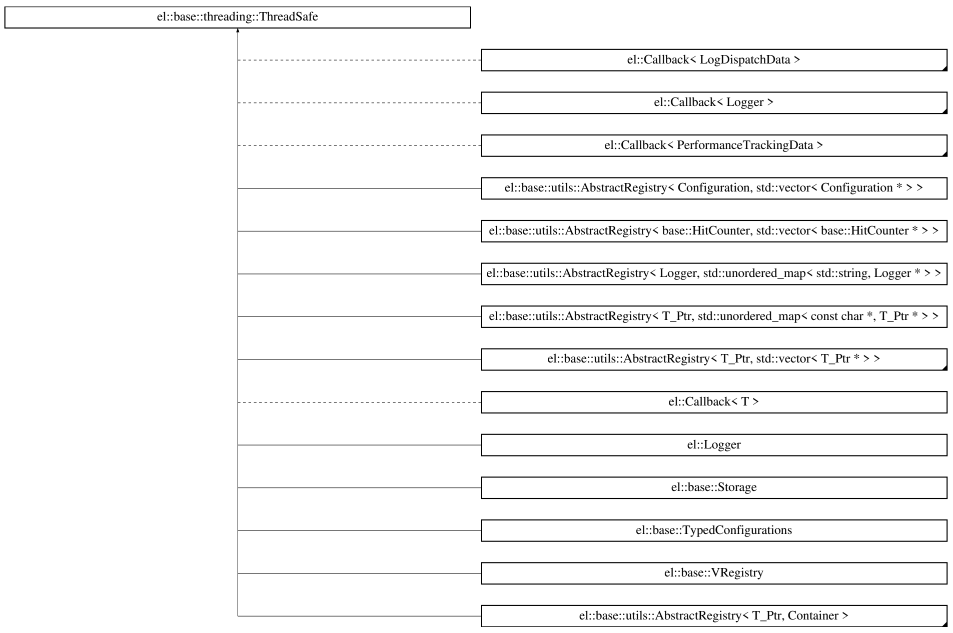
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.51 el::base::threading::ThreadSafe Class Reference

Base of thread safe class, this class is inheritable-only.

Inheritance diagram for el::base::threading::ThreadSafe:


el::base::threading::ThreadSafe
el::Callback< LogDispatchData >
el::Callback< Logger >
el::Callback< PerformanceTrackingData >
el::base::utils::AbstractRegistry< Configuration, std::vector< Configuration * > >
el::base::utils::AbstractRegistry< base::HitCounter, std::vector< base::HitCounter * > >
el::base::utils::AbstractRegistry< Logger, std::unordered_map< std::string, Logger * > >
el::base::utils::AbstractRegistry< T_Ptr, std::unordered_map< const char *, T_Ptr * > >
el::base::utils::AbstractRegistry< T_Ptr, std::vector< T_Ptr * > >
el::Callback< T >
el::Logger
el::base::Storage
el::base::TypedConfigurations
el::base::VRegistry
el::base::utils::AbstractRegistry< T_Ptr, Container >

Public Member Functions

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Protected Member Functions

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

Private Attributes

- base::threading::Mutex m_mutex

8.51.1 Detailed Description

Base of thread safe class, this class is inheritable-only.

Definition at line 1002 of file easylogging++.h.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 ThreadSafe()

```
el::base::threading::ThreadSafe::ThreadSafe (
            void  )  [inline], [protected]
```

Definition at line 1008 of file easylogging++.h.

8.51.2.2 ∼ThreadSafe()

```
virtual el::base::threading::ThreadSafe::∼ThreadSafe (
            void  )  [inline], [protected], [virtual]
```

Definition at line 1009 of file easylogging++.h.

8.51.3 Member Function Documentation

8.51.3.1 acquireLock()

```
virtual void el::base::threading::ThreadSafe::acquireLock (
            void  )  [inline], [virtual]
```

Definition at line 1004 of file easylogging++.h.

8.51.3.2 lock()

```
virtual base::threading::Mutex & el::base::threading::ThreadSafe::lock (
            void  )  [inline], [virtual]
```

Definition at line 1006 of file easylogging++.h.

8.51.3.3 releaseLock()

```
virtual void el::base::threading::ThreadSafe::releaseLock (
            void  )  [inline], [virtual]
```

Definition at line 1005 of file easylogging++.h.

8.51.4 Field Documentation

8.51.4.1 m_mutex

```
base::threading::Mutex el::base::threading::ThreadSafe::m_mutex  [private]
```

Definition at line 1011 of file easylogging++.h.

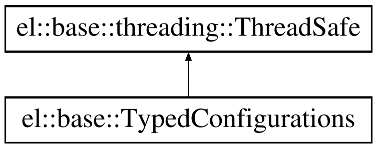
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.52 el::base::TypedConfigurations Class Reference

Configurations with data types.

Inheritance diagram for el::base::TypedConfigurations:


el::base::threading::ThreadSafe
el::base::TypedConfigurations

Public Member Functions

- TypedConfigurations (Configurations ∗configurations, LogStreamsReferenceMapPtr logStreamsReference)
  Constructor to initialize (construct) the object off el::Configurations.
- TypedConfigurations (const TypedConfigurations &other)
- virtual ∼TypedConfigurations (void)
- const Configurations ∗ configurations (void) const
- bool enabled (Level level)
- bool toFile (Level level)
- const std::string & filename (Level level)
- bool toStandardOutput (Level level)
- const base::LogFormat & logFormat (Level level)
- const base::SubsecondPrecision & subsecondPrecision (Level level=Level::Global)
- const base::MillisecondsWidth & millisecondsWidth (Level level=Level::Global)
- bool performanceTracking (Level level=Level::Global)
- base::type::fstream_t ∗ fileStream (Level level)
- std::size_t maxLogFileSize (Level level)
- std::size_t logFlushThreshold (Level level)

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Member Functions

- template<typename Conf_T >
  Conf_T getConfigByVal (Level level, const std::unordered_map< Level, Conf_T > ∗confMap, const char ∗confName)
- template<typename Conf_T >
  Conf_T & getConfigByRef (Level level, std::unordered_map< Level, Conf_T > ∗confMap, const char ∗conf←Name)
- template<typename Conf_T >
  Conf_T unsafeGetConfigByVal (Level level, const std::unordered_map< Level, Conf_T > ∗confMap, const char ∗confName)
- template<typename Conf_T >
  Conf_T & unsafeGetConfigByRef (Level level, std::unordered_map< Level, Conf_T > ∗confMap, const char ∗confName)

- template<typename Conf_T >
  void setValue (Level level, const Conf_T &value, std::unordered_map< Level, Conf_T > *confMap, bool includeGlobalLevel=true)
- void build (Configurations *configurations)
- unsigned long getULong (std::string confVal)
- std::string resolveFilename (const std::string &filename)
- void insertFile (Level level, const std::string &fullFilename)
- bool unsafeValidateFileRolling (Level level, const PreRollOutCallback &preRollOutCallback)
- bool validateFileRolling (Level level, const PreRollOutCallback &preRollOutCallback)

Private Attributes

- Configurations * m_configurations
- std::unordered_map< Level, bool > m_enabledMap
- std::unordered_map< Level, bool > m_toFileMap
- std::unordered_map< Level, std::string > m_filenameMap
- std::unordered_map< Level, bool > m_toStandardOutputMap
- std::unordered_map< Level, base::LogFormat > m_logFormatMap
- std::unordered_map< Level, base::SubsecondPrecision > m_subsecondPrecisionMap
- std::unordered_map< Level, bool > m_performanceTrackingMap
- std::unordered_map< Level, base::FileStreamPtr > m_fileStreamMap
- std::unordered_map< Level, std::size_t > m_maxLogFileSizeMap
- std::unordered_map< Level, std::size_t > m_logFlushThresholdMap
- LogStreamsReferenceMapPtr m_logStreamsReference = nullptr

Friends

- class el::Helpers
- class el::base::MessageBuilder
- class el::base::Writer
- class el::base::DefaultLogDispatchCallback
- class el::base::LogDispatcher

Additional Inherited Members

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ~ThreadSafe (void)

8.52.1 Detailed Description

Configurations with data types.

@detail el::Configurations have string based values. This is whats used internally in order to read correct configurations. This is to perform faster while writing logs using correct configurations.

This is thread safe and final class containing non-virtual destructor (means nothing should inherit this class)

Definition at line 1904 of file easylogging++.h.

8.52.2 Constructor & Destructor Documentation

8.52.2.1 TypedConfigurations() [1/2]

```
el::base::TypedConfigurations::TypedConfigurations (
            Configurations * configurations,
            LogStreamsReferenceMapPtr logStreamsReference )
```

Constructor to initialize (construct) the object off el::Configurations.

Parameters

configurations	Configurations pointer/reference to base this typed configurations off.
logStreamsReference	Use ELPP->registeredLoggers()->logStreamsReference()

Definition at line 1613 of file easylogging++.cc.

References build(), configurations(), m_configurations, and m_logStreamsReference.

8.52.2.2 TypedConfigurations() [2/2]

```
el::base::TypedConfigurations::TypedConfigurations (
            const TypedConfigurations & other )
```

Definition at line 1620 of file easylogging++.cc.

References build(), m_configurations, and m_logStreamsReference.

8.52.2.3 ∼TypedConfigurations()

```
virtual el::base::TypedConfigurations::∼TypedConfigurations (
            void ) [inline], [virtual]
```

Definition at line 1913 of file easylogging++.h.

8.52.3 Member Function Documentation

8.52.3.1 build()

```
void el::base::TypedConfigurations::build (
            Configurations * configurations ) [private]
```

Definition at line 1670 of file easylogging++.cc.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::begin(), configurations(), el::Configuration::configurationType(), el::base::defaultPreRollOutCallback(), el::Enabled, el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), el::Filename, el::Format, getULong(), el::Global, insertFile(), el::Configuration::level(), el::base::threading::ThreadSafe::lock(), el::LogFlushThreshold, m_enabledMap, m_logFlushThresholdMap, m_logFormatMap, m_maxLogFileSizeMap, m_performanceTrackingMap, m_subsecondPrecisionMap, m_toFileMap, m_toStandardOutputMap, el::MaxLogFileSize, el::PerformanceTracking, setValue(), el::SubsecondPrecision, el::ToFile, el::ToStandardOutput, el::base::utils::Str::trim(), unsafeValidateFileRolling(), and el::Configuration::value().

8.52.3.2 configurations()

```
const Configurations * el::base::TypedConfigurations::configurations (
            void ) const [inline]
```

Definition at line 1916 of file easylogging++.h.

8.52.3.3 enabled()

```
bool el::base::TypedConfigurations::enabled (
            Level level )
```

Definition at line 1626 of file easylogging++.cc.

References m_enabledMap.

8.52.3.4 filename()

```
const std::string & el::base::TypedConfigurations::filename (
            Level level )
```

Definition at line 1634 of file easylogging++.cc.

References m_filenameMap.

8.52.3.5 fileStream()

```
base::type::fstream_t * el::base::TypedConfigurations::fileStream (
            Level level )
```

Definition at line 1658 of file easylogging++.cc.

References m_fileStreamMap.

8.52.3.6 getConfigByRef()

```
template<typename Conf_T >
Conf_T & el::base::TypedConfigurations::getConfigByRef (
            Level level,
            std::unordered_map< Level, Conf_T > * confMap,
            const char * confName ) [inline], [private]
```

Definition at line 1959 of file easylogging++.h.

8.52.3.7 getConfigByVal()

```
template<typename Conf_T >
Conf_T el::base::TypedConfigurations::getConfigByVal (
            Level level,
            const std::unordered_map< Level, Conf_T > * confMap,
            const char * confName ) [inline], [private]
```

Definition at line 1953 of file easylogging++.h.

8.52.3.8 getULong()

```
unsigned long el::base::TypedConfigurations::getULong (
            std::string confVal )  [private]
```

Definition at line 1724 of file easylogging++.cc.

References ELPP_ASSERT, and el::base::utils::Str::trim().

8.52.3.9 insertFile()

```
void el::base::TypedConfigurations::insertFile (
            Level level,
            const std::string & fullFilename )  [private]
```

Definition at line 1778 of file easylogging++.cc.

References el::LevelHelper::convertToString(), el::base::utils::File::createPath(), ELPP_INTERNAL_ERROR, el::base::utils::File::extractPathFromFilename(), el::Global, el::base::consts::kFilePathSeparator, m_filenameMap, m_fileStreamMap, m_logStreamsReference, m_toFileMap, el::base::utils::File::newFileStream(), resolveFilename(), and setValue().

8.52.3.10 logFlushThreshold()

```
std::size_t el::base::TypedConfigurations::logFlushThreshold (
            Level level )
```

Definition at line 1666 of file easylogging++.cc.

References m_logFlushThresholdMap.

8.52.3.11 logFormat()

```
const base::LogFormat & el::base::TypedConfigurations::logFormat (
            Level level )
```

Definition at line 1642 of file easylogging++.cc.

References m_logFormatMap.

8.52.3.12 maxLogFileSize()

```
std::size_t el::base::TypedConfigurations::maxLogFileSize (
            Level level )
```

Definition at line 1662 of file easylogging++.cc.

References m_maxLogFileSizeMap.

8.52.3.13 millisecondsWidth()

```
const base::MillisecondsWidth & el::base::TypedConfigurations::millisecondsWidth (
            Level level = Level::Global )
```

Definition at line 1650 of file easylogging++.cc.

References m_subsecondPrecisionMap.

8.52.3.14 performanceTracking()

```
bool el::base::TypedConfigurations::performanceTracking (
            Level level = Level::Global )
```

Definition at line 1654 of file easylogging++.cc.

References m_performanceTrackingMap.

8.52.3.15 resolveFilename()

```
std::string el::base::TypedConfigurations::resolveFilename (
            const std::string & filename )  [private]
```

Definition at line 1739 of file easylogging++.cc.

References filename(), el::base::utils::DateTime::getDateTime(), el::base::consts::kDateTimeFormatSpecifierForFilename, el::base::consts::kDefaultDateTimeFormatInFilename, el::base::consts::kFormatSpecifierChar, and el::base::utils::Str::replaceAll().

8.52.3.16 setValue()

```
template<typename Conf_T >
void el::base::TypedConfigurations::setValue (
            Level level,
            const Conf_T & value,
            std::unordered_map< Level, Conf_T > * confMap,
            bool includeGlobalLevel = true )  [inline], [private]
```

Definition at line 1998 of file easylogging++.h.

8.52.3.17 subsecondPrecision()

```
const base::SubsecondPrecision & el::base::TypedConfigurations::subsecondPrecision (
            Level level = Level::Global )
```

Definition at line 1646 of file easylogging++.cc.

References m_subsecondPrecisionMap.

8.52.3.18 toFile()

```
bool el::base::TypedConfigurations::toFile (
            Level level )
```

Definition at line 1630 of file easylogging++.cc.

References m_toFileMap.

8.52.3.19 toStandardOutput()

```
bool el::base::TypedConfigurations::toStandardOutput (
            Level level )
```

Definition at line 1638 of file easylogging++.cc.

References m_toStandardOutputMap.

8.52.3.20 unsafeGetConfigByRef()

```
template<typename Conf_T >
Conf_T & el::base::TypedConfigurations::unsafeGetConfigByRef (
            Level level,
            std::unordered_map< Level, Conf_T > * confMap,
            const char * confName )  [inline], [private]
```

Definition at line 1982 of file easylogging++.h.

References ELPP_INTERNAL_ERROR, and ELPP_UNUSED.

8.52.3.21 unsafeGetConfigByVal()

```
template<typename Conf_T >
Conf_T el::base::TypedConfigurations::unsafeGetConfigByVal (
            Level level,
            const std::unordered_map< Level, Conf_T > * confMap,
            const char * confName )  [inline], [private]
```

Definition at line 1965 of file easylogging++.h.

References ELPP_INTERNAL_ERROR, and ELPP_UNUSED.

8.52.3.22 unsafeValidateFileRolling()

```
bool el::base::TypedConfigurations::unsafeValidateFileRolling (
            Level level,
            const PreRollOutCallback & preRollOutCallback )  [private]
```

Definition at line 1815 of file easylogging++.cc.

References el::LevelHelper::convertToString(), ELPP_INTERNAL_INFO, el::base::utils::File::getSizeOfFile(), m_filenameMap, m_fileStreamMap, m_maxLogFileSizeMap, maxLogFileSize(), unsafeGetConfigByRef(), and unsafeGetConfigByVal().

8.52.3.23 validateFileRolling()

```
bool el::base::TypedConfigurations::validateFileRolling (
            Level level,
            const PreRollOutCallback & preRollOutCallback ) [inline], [private]
```

Definition at line 2027 of file easylogging++.h.

8.52.4 Friends And Related Symbol Documentation

8.52.4.1 el::base::DefaultLogDispatchCallback

```
friend class el::base::DefaultLogDispatchCallback [friend]
```

Definition at line 1949 of file easylogging++.h.

8.52.4.2 el::base::LogDispatcher

```
friend class el::base::LogDispatcher [friend]
```

Definition at line 1950 of file easylogging++.h.

8.52.4.3 el::base::MessageBuilder

```
friend class el::base::MessageBuilder [friend]
```

Definition at line 1947 of file easylogging++.h.

8.52.4.4 el::base::Writer

```
friend class el::base::Writer [friend]
```

Definition at line 1948 of file easylogging++.h.

8.52.4.5 el::Helpers

```
friend class el::Helpers [friend]
```

Definition at line 1946 of file easylogging++.h.

8.52.5 Field Documentation

8.52.5.1 m_configurations

```
Configurations* el::base::TypedConfigurations::m_configurations [private]
```

Definition at line 1933 of file easylogging++.h.

8.52.5.2 m_enabledMap

`std::unordered_map<Level, bool> el::base::TypedConfigurations::m_enabledMap [private]`

Definition at line 1934 of file easylogging++.h.

8.52.5.3 m_filenameMap

`std::unordered_map<Level, std::string> el::base::TypedConfigurations::m_filenameMap [private]`

Definition at line 1936 of file easylogging++.h.

8.52.5.4 m_fileStreamMap

`std::unordered_map<Level, base::FileStreamPtr> el::base::TypedConfigurations::m_fileStreamMap [private]`

Definition at line 1941 of file easylogging++.h.

8.52.5.5 m_logFlushThresholdMap

`std::unordered_map<Level, std::size_t> el::base::TypedConfigurations::m_logFlushThresholdMap [private]`

Definition at line 1943 of file easylogging++.h.

8.52.5.6 m_logFormatMap

`std::unordered_map<Level, base::LogFormat> el::base::TypedConfigurations::m_logFormatMap [private]`

Definition at line 1938 of file easylogging++.h.

8.52.5.7 m_logStreamsReference

`LogStreamsReferenceMapPtr el::base::TypedConfigurations::m_logStreamsReference = nullptr [private]`

Definition at line 1944 of file easylogging++.h.

8.52.5.8 m_maxLogFileSizeMap

`std::unordered_map<Level, std::size_t> el::base::TypedConfigurations::m_maxLogFileSizeMap [private]`

Definition at line 1942 of file easylogging++.h.

8.52.5.9 m_performanceTrackingMap

```
std::unordered_map<Level, bool> el::base::TypedConfigurations::m_performanceTrackingMap  [private]
```

Definition at line 1940 of file easylogging++.h.

8.52.5.10 m_subsecondPrecisionMap

```
std::unordered_map<Level, base::SubsecondPrecision> el::base::TypedConfigurations::m_subsecond↩
PrecisionMap  [private]
```

Definition at line 1939 of file easylogging++.h.

8.52.5.11 m_toFileMap

```
std::unordered_map<Level, bool> el::base::TypedConfigurations::m_toFileMap  [private]
```

Definition at line 1935 of file easylogging++.h.

8.52.5.12 m_toStandardOutputMap

```
std::unordered_map<Level, bool> el::base::TypedConfigurations::m_toStandardOutputMap  [private]
```

Definition at line 1937 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.53 el::base::utils::Utils Class Reference

Static Public Member Functions

- template<typename T , typename TPtr >
  static bool installCallback (const std::string &id, std::unordered_map< std::string, TPtr > *mapT)
- template<typename T , typename TPtr >
  static void uninstallCallback (const std::string &id, std::unordered_map< std::string, TPtr > *mapT)
- template<typename T , typename TPtr >
  static T * callback (const std::string &id, std::unordered_map< std::string, TPtr > *mapT)

8.53.1 Detailed Description

Definition at line 1531 of file easylogging++.h.

8.53.2 Member Function Documentation

8.53.2.1 callback()

```
template<typename T , typename TPtr >
static T * el::base::utils::Utils::callback (
            const std::string & id,
            std::unordered_map< std::string, TPtr > * mapT )  [inline], [static]
```

Definition at line 1550 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end(), and el::base::utils::RegistryWithPred< T_Ptr, Pred >::get().

8.53.2.2 installCallback()

```
template<typename T , typename TPtr >
static bool el::base::utils::Utils::installCallback (
            const std::string & id,
            std::unordered_map< std::string, TPtr > * mapT )  [inline], [static]
```

Definition at line 1534 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end().

8.53.2.3 uninstallCallback()

```
template<typename T , typename TPtr >
static void el::base::utils::Utils::uninstallCallback (
            const std::string & id,
            std::unordered_map< std::string, TPtr > * mapT )  [inline], [static]
```

Definition at line 1543 of file easylogging++.h.

References el::base::utils::AbstractRegistry< T_Ptr, Container >::end().

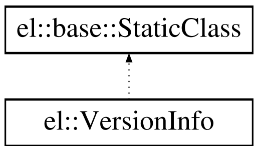
The documentation for this class was generated from the following file:

- lib/easylogging++.h

8.54 el::VersionInfo Class Reference

```
#include <easylogging++.h>
```

Inheritance diagram for el::VersionInfo:



Static Public Member Functions

- static const std::string version (void)

  Current version number.
- static const std::string releaseDate (void)

  Release date of current version.

8.54.1 Detailed Description

Definition at line 3894 of file easylogging++.h.

8.54.2 Member Function Documentation

8.54.2.1 releaseDate()

```
const std::string el::VersionInfo::releaseDate (
             void  )  [static]
```

Release date of current version.

Definition at line 3112 of file easylogging++.cc.

8.54.2.2 version()

```
const std::string el::VersionInfo::version (
             void  )  [static]
```

Current version number.

Definition at line 3108 of file easylogging++.cc.

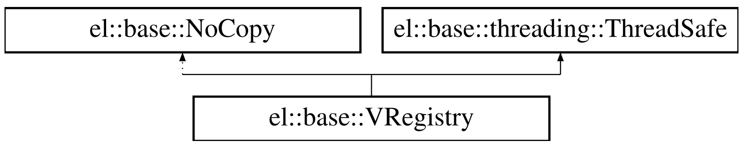
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.55 el::base::VRegistry Class Reference

Represents registries for verbose logging.

Inheritance diagram for el::base::VRegistry:



Public Member Functions

- VRegistry (base::type::VerboseLevel level, base::type::EnumType ∗pFlags)
- void setLevel (base::type::VerboseLevel level)

    Sets verbose level. Accepted range is 0-9.
- base::type::VerboseLevel level (void) const
- void clearModules (void)
- void setModules (const char ∗modules)
- bool allowed (base::type::VerboseLevel vlevel, const char ∗file)
- const std::unordered_map< std::string, base::type::VerboseLevel > & modules (void) const
- void setFromArgs (const base::utils::CommandLineArgs ∗commandLineArgs)
- bool vModulesEnabled (void)

    Whether or not vModules enabled.

Public Member Functions inherited from el::base::threading::ThreadSafe

- virtual void acquireLock (void) ELPP_FINAL
- virtual void releaseLock (void) ELPP_FINAL
- virtual base::threading::Mutex & lock (void) ELPP_FINAL

Private Attributes

- base::type::VerboseLevel m_level
- base::type::EnumType ∗ m_pFlags
- std::unordered_map< std::string, base::type::VerboseLevel > m_modules

Additional Inherited Members

Protected Member Functions inherited from el::base::threading::ThreadSafe

- ThreadSafe (void)
- virtual ∼ThreadSafe (void)

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.55.1 Detailed Description

Represents registries for verbose logging.

Definition at line 2417 of file easylogging++.h.

8.55.2 Constructor & Destructor Documentation

8.55.2.1 VRegistry()

```
el::base::VRegistry::VRegistry (
            base::type::VerboseLevel level,
            base::type::EnumType * pFlags )  [explicit]
```

Definition at line 1935 of file easylogging++.cc.

8.55.3 Member Function Documentation

8.55.3.1 allowed()

```
bool el::base::VRegistry::allowed (
            base::type::VerboseLevel vlevel,
            const char * file )
```

Definition at line 2019 of file easylogging++.cc.

References el::AllowVerboseIfModuleNotSpecified, el::base::utils::File::buildBaseFilename(), el::base::utils::hasFlag(), el::base::consts::kSourceFilenameMaxLength, el::base::threading::ThreadSafe::lock(), m_level, m_modules, m_pFlags, and el::base::utils::Str::wildCardMatch().

8.55.3.2 clearModules()

```
void el::base::VRegistry::clearModules (
            void )  [inline]
```

Definition at line 2428 of file easylogging++.h.

8.55.3.3 level()

```
base::type::VerboseLevel el::base::VRegistry::level (
            void ) const  [inline]
```

Definition at line 2424 of file easylogging++.h.

8.55.3.4 modules()

```
const std::unordered_map< std::string, base::type::VerboseLevel > & el::base::VRegistry↩
::modules (
            void ) const  [inline]
```

Definition at line 2437 of file easylogging++.h.

8.55.3.5 setFromArgs()

```
void el::base::VRegistry::setFromArgs (
            const base::utils::CommandLineArgs * commandLineArgs )
```

Definition at line 2039 of file easylogging++.cc.

References el::base::utils::CommandLineArgs::getParamValue(), el::base::utils::CommandLineArgs::hasParam(), el::base::utils::CommandLineArgs::hasParamWithValue(), el::base::consts::kMaxVerboseLevel, setLevel(), setModules(), and vModulesEnabled().

8.55.3.6 setLevel()

```
void el::base::VRegistry::setLevel (
            base::type::VerboseLevel level )
```

Sets verbose level. Accepted range is 0-9.

Definition at line 1939 of file easylogging++.cc.

References el::base::consts::kMaxVerboseLevel, level(), el::base::threading::ThreadSafe::lock(), and m_level.

8.55.3.7 setModules()

```
void el::base::VRegistry::setModules (
            const char * modules )
```

Definition at line 1947 of file easylogging++.cc.

References el::DisableVModulesExtensions, el::base::utils::Str::endsWith(), el::base::utils::hasFlag(), level(), el::base::threading::ThreadSafe::lock(), m_modules, m_pFlags, and modules().

8.55.3.8 vModulesEnabled()

```
bool el::base::VRegistry::vModulesEnabled (
            void ) [inline]
```

Whether or not vModules enabled.

Definition at line 2444 of file easylogging++.h.

8.55.4 Field Documentation

8.55.4.1 m_level

```
base::type::VerboseLevel el::base::VRegistry::m_level [private]
```

Definition at line 2449 of file easylogging++.h.

8.55.4.2 m_modules

```
std::unordered_map<std::string, base::type::VerboseLevel> el::base::VRegistry::m_modules
[private]
```

Definition at line 2451 of file easylogging++.h.

8.55.4.3 m_pFlags

```
base::type::EnumType* el::base::VRegistry::m_pFlags  [private]
```

Definition at line 2450 of file easylogging++.h.

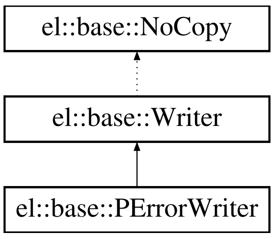
The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

8.56 el::base::Writer Class Reference

Main entry point of each logging.

Inheritance diagram for el::base::Writer:



Public Member Functions

- Writer (Level level, const char ∗file, base::type::LineNumber line, const char ∗func, base::DispatchAction dispatchAction=base::DispatchAction::NormalLog, base::type::VerboseLevel verboseLevel=0)
- Writer (LogMessage ∗msg, base::DispatchAction dispatchAction=base::DispatchAction::NormalLog)
- virtual ∼Writer (void)
- template<typename T >
  Writer & operator<< (const T &log)
- Writer & operator<< (std::ostream &(∗log)(std::ostream &))
- operator bool ()
- Writer & construct (Logger ∗logger, bool needLock=true)
- Writer & construct (int count, const char ∗loggerIds,...)

Protected Member Functions

- void initializeLogger (const std::string &loggerId, bool lookup=true, bool needLock=true)
- void processDispatch ()
- void triggerDispatch (void)

Protected Attributes

- LogMessage ∗ m_msg
- Level m_level
- const char ∗ m_file
- const base::type::LineNumber m_line
- const char ∗ m_func
- base::type::VerboseLevel m_verboseLevel
- Logger ∗ m_logger
- bool m_proceed
- base::MessageBuilder m_messageBuilder
- base::DispatchAction m_dispatchAction
- std::vector< std::string > m_loggerIds

Friends

- class el::Helpers

Additional Inherited Members

Private Member Functions inherited from el::base::NoCopy

- NoCopy (void)

8.56.1 Detailed Description

Main entry point of each logging.

Definition at line 3190 of file easylogging++.h.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 Writer() [1/2]

```
el::base::Writer::Writer (
            Level level,
            const char * file,
            base::type::LineNumber line,
            const char * func,
            base::DispatchAction dispatchAction = base::DispatchAction::NormalLog,
            base::type::VerboseLevel verboseLevel = 0 )  [inline]
```

Definition at line 3192 of file easylogging++.h.

8.56.2.2 Writer() [2/2]

```
el::base::Writer::Writer (
            LogMessage * msg,
            base::DispatchAction dispatchAction = base::DispatchAction::NormalLog )  [inline]
```

Definition at line 3199 of file easylogging++.h.

References el::Unknown.

8.56.2.3 ∼Writer()

```
virtual el::base::Writer::∼Writer (
            void )  [inline], [virtual]
```

Definition at line 3204 of file easylogging++.h.

8.56.3 Member Function Documentation

8.56.3.1 construct() [1/2]

```
Writer & el::base::Writer::construct (
            int count,
            const char * loggerIds,
             ... )
```

Definition at line 2538 of file easylogging++.cc.

References ELPP, el::base::MessageBuilder::initialize(), initializeLogger(), m_logger, m_loggerIds, m_messageBuilder, and el::MultiLoggerSupport.

8.56.3.2 construct() [2/2]

```
Writer & el::base::Writer::construct (
            Logger * logger,
            bool needLock = true )
```

Definition at line 2531 of file easylogging++.cc.

References el::Logger::id(), el::base::MessageBuilder::initialize(), initializeLogger(), m_logger, and m_messageBuilder.

8.56.3.3 initializeLogger()

```
void el::base::Writer::initializeLogger (
            const std::string & loggerId,
            bool lookup = true,
            bool needLock = true )  [protected]
```

Definition at line 2557 of file easylogging++.cc.

References el::base::threading::ThreadSafe::acquireLock(), el::LevelHelper::castToInt(), construct(), el::CreateLoggerAutomatically, el::Debug, ELPP, el::Logger::enabled(), el::HierarchicalLogging, el::base::consts::kDefaultLoggerId, m_file, m_func, m_level, m_line, m_logger, m_proceed, and el::Verbose.

8.56.3.4 operator bool()

```
el::base::Writer::operator bool ( )  [inline]
```

Definition at line 3227 of file easylogging++.h.

8.56.3.5 operator<<() [1/2]

```
template<typename T >
Writer & el::base::Writer::operator<< (
             const T & log )  [inline]
```

Definition at line 3209 of file easylogging++.h.

8.56.3.6 operator<<() [2/2]

```
Writer & el::base::Writer::operator<< (
             std::ostream &(*)(std::ostream &) log )  [inline]
```

Definition at line 3218 of file easylogging++.h.

8.56.3.7 processDispatch()

```
void el::base::Writer::processDispatch ( )  [protected]
```

Definition at line 2585 of file easylogging++.cc.

References ELPP, ELPP_LITERAL, initializeLogger(), m_logger, m_loggerIds, m_proceed, el::MultiLoggerSupport, el::base::threading::ThreadSafe::releaseLock(), el::Logger::stream(), and triggerDispatch().

8.56.3.8 triggerDispatch()

```
void el::base::Writer::triggerDispatch (
             void )  [protected]
```

Definition at line 2626 of file easylogging++.cc.

References el::base::utils::abort(), construct(), el::DisableApplicationAbortOnFatalLog, el::base::LogDispatcher::dispatch(), ELPP, ELPP_LITERAL, el::Fatal, el::base::consts::kDefaultLoggerId, m_dispatchAction, m_file, m_func, m_level, m_line, m_logger, m_msg, m_proceed, m_verboseLevel, el::base::threading::ThreadSafe::releaseLock(), el::Logger::stream(), and el::Warning.

8.56.4 Friends And Related Symbol Documentation

8.56.4.1 el::Helpers

```
friend class el::Helpers  [friend]
```

Definition at line 3245 of file easylogging++.h.

8.56.5 Field Documentation

8.56.5.1 m_dispatchAction

```
base::DispatchAction el::base::Writer::m_dispatchAction  [protected]
```

Definition at line 3243 of file easylogging++.h.

8.56.5.2 m_file

```
const char* el::base::Writer::m_file [protected]
```

Definition at line 3236 of file easylogging++.h.

8.56.5.3 m_func

```
const char* el::base::Writer::m_func [protected]
```

Definition at line 3238 of file easylogging++.h.

8.56.5.4 m_level

```
Level el::base::Writer::m_level [protected]
```

Definition at line 3235 of file easylogging++.h.

8.56.5.5 m_line

```
const base::type::LineNumber el::base::Writer::m_line [protected]
```

Definition at line 3237 of file easylogging++.h.

8.56.5.6 m_logger

```
Logger* el::base::Writer::m_logger [protected]
```

Definition at line 3240 of file easylogging++.h.

8.56.5.7 m_loggerIds

```
std::vector<std::string> el::base::Writer::m_loggerIds [protected]
```

Definition at line 3244 of file easylogging++.h.

8.56.5.8 m_messageBuilder

```
base::MessageBuilder el::base::Writer::m_messageBuilder [protected]
```

Definition at line 3242 of file easylogging++.h.

8.56.5.9 m_msg

```
LogMessage* el::base::Writer::m_msg [protected]
```

Definition at line 3234 of file easylogging++.h.

8.56.5.10 m_proceed

```
bool el::base::Writer::m_proceed  [protected]
```

Definition at line 3241 of file easylogging++.h.

8.56.5.11 m_verboseLevel

```
base::type::VerboseLevel el::base::Writer::m_verboseLevel  [protected]
```

Definition at line 3239 of file easylogging++.h.

The documentation for this class was generated from the following files:

- lib/easylogging++.h
- lib/easylogging++.cc

Chapter 9

File Documentation

9.1 lib/easylogging++.cc File Reference

```
#include "easylogging++.h"
```

Data Structures

- struct el::StringToLevelItem
- struct el::ConfigurationStringToTypeItem

Namespaces

- namespace el

  Easylogging++ entry namespace.
- namespace el::base

  Namespace containing base/internal functionality used by Easylogging++.
- namespace el::base::consts

  Namespace containing constants used internally.
- namespace el::base::utils

  Namespace containing utility functions/static classes used internally.
- namespace el::base::threading
- namespace el::base::debug

  Contains some internal debugging tools like crash handler and stack tracer.

Macros

- #define ELPP_DEFAULT_LOGGING_FLAGS 0x0

Functions

- static void el::base::utils::abort (int status, const std::string &reason)

  Aborts application due with user-defined status.
- base::type::ostream_t & el::base::utils::operator<< (base::type::ostream_t &os, const CommandLineArgs &c)

Variables

- static const base::type::char_t ∗ el::base::consts::kInfoLevelLogValue = ELPP_LITERAL("INFO")
- static const base::type::char_t ∗ el::base::consts::kDebugLevelLogValue = ELPP_LITERAL("DEBUG")
- static const base::type::char_t ∗ el::base::consts::kWarningLevelLogValue = ELPP_LITERAL("WARNING")
- static const base::type::char_t ∗ el::base::consts::kErrorLevelLogValue = ELPP_LITERAL("ERROR")
- static const base::type::char_t ∗ el::base::consts::kFatalLevelLogValue = ELPP_LITERAL("FATAL")
- static const base::type::char_t ∗ el::base::consts::kVerboseLevelLogValue
- static const base::type::char_t ∗ el::base::consts::kTraceLevelLogValue = ELPP_LITERAL("TRACE")
- static const base::type::char_t ∗ el::base::consts::kInfoLevelShortLogValue = ELPP_LITERAL("I")
- static const base::type::char_t ∗ el::base::consts::kDebugLevelShortLogValue = ELPP_LITERAL("D")
- static const base::type::char_t ∗ el::base::consts::kWarningLevelShortLogValue = ELPP_LITERAL("W")
- static const base::type::char_t ∗ el::base::consts::kErrorLevelShortLogValue = ELPP_LITERAL("E")
- static const base::type::char_t ∗ el::base::consts::kFatalLevelShortLogValue = ELPP_LITERAL("F")
- static const base::type::char_t ∗ el::base::consts::kVerboseLevelShortLogValue = ELPP_LITERAL("V")
- static const base::type::char_t ∗ el::base::consts::kTraceLevelShortLogValue = ELPP_LITERAL("T")
- static const base::type::char_t ∗ el::base::consts::kAppNameFormatSpecifier = ELPP_LITERAL("%app")
- static const base::type::char_t ∗ el::base::consts::kLoggerIdFormatSpecifier = ELPP_LITERAL("%logger")
- static const base::type::char_t ∗ el::base::consts::kThreadIdFormatSpecifier = ELPP_LITERAL("%thread")
- static const base::type::char_t ∗ el::base::consts::kSeverityLevelFormatSpecifier = ELPP_LITERAL("%level")
- static const base::type::char_t ∗ el::base::consts::kSeverityLevelShortFormatSpecifier = ELPP_LITERAL("%levshort")
- static const base::type::char_t ∗ el::base::consts::kDateTimeFormatSpecifier = ELPP_LITERAL("%datetime")
- static const base::type::char_t ∗ el::base::consts::kLogFileFormatSpecifier = ELPP_LITERAL("%file")
- static const base::type::char_t ∗ el::base::consts::kLogFileBaseFormatSpecifier = ELPP_LITERAL("%fbase")
- static const base::type::char_t ∗ el::base::consts::kLogLineFormatSpecifier = ELPP_LITERAL("%line")
- static const base::type::char_t ∗ el::base::consts::kLogLocationFormatSpecifier = ELPP_LITERAL("%loc")
- static const base::type::char_t ∗ el::base::consts::kLogFunctionFormatSpecifier = ELPP_LITERAL("%func")
- static const base::type::char_t ∗ el::base::consts::kCurrentUserFormatSpecifier = ELPP_LITERAL("%user")
- static const base::type::char_t ∗ el::base::consts::kCurrentHostFormatSpecifier = ELPP_LITERAL("%host")
- static const base::type::char_t ∗ el::base::consts::kMessageFormatSpecifier = ELPP_LITERAL("%msg")
- static const base::type::char_t ∗ el::base::consts::kVerboseLevelFormatSpecifier = ELPP_LITERAL("%vlevel")
- static const char ∗ el::base::consts::kDateTimeFormatSpecifierForFilename = "%datetime"
- static const char ∗ el::base::consts::kDays [7] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" }
- static const char ∗ el::base::consts::kDaysAbbrev [7] = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" }
- static const char ∗ el::base::consts::kMonths [12]
- static const char ∗ el::base::consts::kMonthsAbbrev [12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" }
- static const char ∗ el::base::consts::kDefaultDateTimeFormat = "%Y-%M-%d %H:%m:%s,%g"
- static const char ∗ el::base::consts::kDefaultDateTimeFormatInFilename = "%Y-%M-%d_%H-%m"
- static const int el::base::consts::kYearBase = 1900
- static const char ∗ el::base::consts::kAm = "AM"
- static const char ∗ el::base::consts::kPm = "PM"
- static const char ∗ el::base::consts::kNullPointer = "nullptr"
- static const base::type::VerboseLevel el::base::consts::kMaxVerboseLevel = 9
- static const char ∗ el::base::consts::kUnknownUser = "unknown-user"
- static const char ∗ el::base::consts::kUnknownHost = "unknown-host"
- static const char ∗ el::base::consts::kDefaultLogFile = "myeasylog.log"
- static const char ∗ el::base::consts::kDefaultLogFileParam = "--default-log-file"
- static const char ∗ el::base::consts::kValidLoggerIdSymbols
- static const char ∗ el::base::consts::kConfigurationComment = "##"
- static const char ∗ el::base::consts::kConfigurationLevel = "∗"
- static const char ∗ el::base::consts::kConfigurationLoggerId = "--"
- static struct StringToLevelItem el::stringToLevelMap [ ]
- static struct ConfigurationStringToTypeItem el::configStringToTypeMap [ ]

9.1.1 Macro Definition Documentation

9.1.1.1 ELPP_DEFAULT_LOGGING_FLAGS

```
#define ELPP_DEFAULT_LOGGING_FLAGS 0x0
```

Definition at line 2055 of file easylogging++.cc.

9.2 easylogging++.cc

Go to the documentation of this file.

```
//
//  Bismillah ar-Rahmaan ar-Raheem
//
//  Easylogging++ v9.97.1
//  Cross-platform logging library for C++ applications
//
//  Copyright (c) 2012-present @abumq (Majid Q.)
//
//  This library is released under the MIT Licence.
//  https://github.com/amrayn/easyloggingpp/blob/master/LICENSE
//

#include "easylogging++.h"

#if defined(AUTO_INITIALIZE_EASYLOGGINGPP)
INITIALIZE_EASYLOGGINGPP
#endif

namespace el {

// el::base
namespace base {
// el::base::consts
namespace consts {

// Level log values - These are values that are replaced in place of %level format specifier
// Extra spaces after format specifiers are only for readability purposes in log files
static const base::type::char_t* kInfoLevelLogValue     =   ELPP_LITERAL("INFO");
static const base::type::char_t* kDebugLevelLogValue    =   ELPP_LITERAL("DEBUG");
static const base::type::char_t* kWarningLevelLogValue  =   ELPP_LITERAL("WARNING");
static const base::type::char_t* kErrorLevelLogValue    =   ELPP_LITERAL("ERROR");
static const base::type::char_t* kFatalLevelLogValue    =   ELPP_LITERAL("FATAL");
static const base::type::char_t* kVerboseLevelLogValue  =
  ELPP_LITERAL("VERBOSE"); // will become VERBOSE-x where x = verbose level
static const base::type::char_t* kTraceLevelLogValue    =   ELPP_LITERAL("TRACE");
static const base::type::char_t* kInfoLevelShortLogValue     =   ELPP_LITERAL("I");
static const base::type::char_t* kDebugLevelShortLogValue    =   ELPP_LITERAL("D");
static const base::type::char_t* kWarningLevelShortLogValue  =   ELPP_LITERAL("W");
static const base::type::char_t* kErrorLevelShortLogValue    =   ELPP_LITERAL("E");
static const base::type::char_t* kFatalLevelShortLogValue    =   ELPP_LITERAL("F");
static const base::type::char_t* kVerboseLevelShortLogValue  =   ELPP_LITERAL("V");
static const base::type::char_t* kTraceLevelShortLogValue    =   ELPP_LITERAL("T");
// Format specifiers - These are used to define log format
static const base::type::char_t* kAppNameFormatSpecifier          =      ELPP_LITERAL("%app");
static const base::type::char_t* kLoggerIdFormatSpecifier         =      ELPP_LITERAL("%logger");
static const base::type::char_t* kThreadIdFormatSpecifier         =      ELPP_LITERAL("%thread");
static const base::type::char_t* kSeverityLevelFormatSpecifier    =      ELPP_LITERAL("%level");
static const base::type::char_t* kSeverityLevelShortFormatSpecifier    =
ELPP_LITERAL("%levshort");
static const base::type::char_t* kDateTimeFormatSpecifier         =      ELPP_LITERAL("%datetime");
static const base::type::char_t* kLogFileFormatSpecifier          =      ELPP_LITERAL("%file");
static const base::type::char_t* kLogFileBaseFormatSpecifier      =      ELPP_LITERAL("%fbase");
static const base::type::char_t* kLogLineFormatSpecifier          =      ELPP_LITERAL("%line");
static const base::type::char_t* kLogLocationFormatSpecifier      =      ELPP_LITERAL("%loc");
static const base::type::char_t* kLogFunctionFormatSpecifier      =      ELPP_LITERAL("%func");
static const base::type::char_t* kCurrentUserFormatSpecifier      =      ELPP_LITERAL("%user");
static const base::type::char_t* kCurrentHostFormatSpecifier      =      ELPP_LITERAL("%host");
static const base::type::char_t* kMessageFormatSpecifier          =      ELPP_LITERAL("%msg");
static const base::type::char_t* kVerboseLevelFormatSpecifier     =      ELPP_LITERAL("%vlevel");
static const char* kDateTimeFormatSpecifierForFilename            =      "%datetime";
// Date/time
static const char* kDays[7]                         =      { "Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday" };
static const char* kDaysAbbrev[7]                   =      { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat" };
static const char* kMonths[12]                      =      { "January", "February", "March", "April",
"May", "June", "July", "August",
                                                             "September", "October", "November",
"December"
                                                           };
static const char* kMonthsAbbrev[12]                =      { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
static const char* kDefaultDateTimeFormat           =      "%Y-%M-%d %H:%m:%s,%g";
static const char* kDefaultDateTimeFormatInFilename =      "%Y-%M-%d_%H-%m";
static const int kYearBase                          =      1900;
static const char* kAm                              =      "AM";
static const char* kPm                              =      "PM";
// Miscellaneous constants

static const char* kNullPointer                            =      "nullptr";
#if ELPP_VARIADIC_TEMPLATES_SUPPORTED
#endif  // ELPP_VARIADIC_TEMPLATES_SUPPORTED
static const base::type::VerboseLevel kMaxVerboseLevel     =      9;
static const char* kUnknownUser                            =      "unknown-user";
static const char* kUnknownHost                            =      "unknown-host";


//---------------- DEFAULT LOG FILE -----------------------

#if defined(ELPP_NO_DEFAULT_LOG_FILE)
#  if ELPP_OS_UNIX
static const char* kDefaultLogFile                         =      "/dev/null";
#  elif ELPP_OS_WINDOWS
static const char* kDefaultLogFile                         =      "nul";
#  endif  // ELPP_OS_UNIX
#elif defined(ELPP_DEFAULT_LOG_FILE)
static const char* kDefaultLogFile                         =      ELPP_DEFAULT_LOG_FILE;
#else
static const char* kDefaultLogFile                         =      "myeasylog.log";
#endif // defined(ELPP_NO_DEFAULT_LOG_FILE)


#if !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
static const char* kDefaultLogFileParam                    =      "--default-log-file";
#endif  // !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
#if defined(ELPP_LOGGING_FLAGS_FROM_ARG)
static const char* kLoggingFlagsParam                      =      "--logging-flags";
#endif  // defined(ELPP_LOGGING_FLAGS_FROM_ARG)
static const char* kValidLoggerIdSymbols                   =
  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._";
static const char* kConfigurationComment                   =      "##";
static const char* kConfigurationLevel                     =      "*";
static const char* kConfigurationLoggerId                  =      "--";
}
// el::base::utils
namespace utils {

static void abort(int status, const std::string& reason) {
  // Both status and reason params are there for debugging with tools like gdb etc
  ELPP_UNUSED(status);
  ELPP_UNUSED(reason);
#if defined(ELPP_COMPILER_MSVC) && defined(_M_IX86) && defined(_DEBUG)
  // Ignore msvc critical error dialog - break instead (on debug mode)
  _asm int 3
#else
  ::abort();
#endif  // defined(ELPP_COMPILER_MSVC) && defined(_M_IX86) && defined(_DEBUG)
}

} // namespace utils
} // namespace base

// el

// LevelHelper

const char* LevelHelper::convertToString(Level level) {
  // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
  if (level == Level::Global) return "GLOBAL";
  if (level == Level::Debug) return "DEBUG";
  if (level == Level::Info) return "INFO";
  if (level == Level::Warning) return "WARNING";
  if (level == Level::Error) return "ERROR";
  if (level == Level::Fatal) return "FATAL";
  if (level == Level::Verbose) return "VERBOSE";
  if (level == Level::Trace) return "TRACE";
  return "UNKNOWN";
}

struct StringToLevelItem {
  const char* levelString;
  Level level;
```

```
};

static struct StringToLevelItem stringToLevelMap[] = {
  { "global", Level::Global },
  { "debug", Level::Debug },
  { "info", Level::Info },
  { "warning", Level::Warning },
  { "error", Level::Error },
  { "fatal", Level::Fatal },
  { "verbose", Level::Verbose },
  { "trace", Level::Trace }
};

Level LevelHelper::convertFromString(const char* levelStr) {
  for (auto& item : stringToLevelMap) {
    if (base::utils::Str::cStringCaseEq(levelStr, item.levelString)) {
      return item.level;
    }
  }
  return Level::Unknown;
}

void LevelHelper::forEachLevel(base::type::EnumType* startIndex, const std::function<bool(void)>& fn)
{
  base::type::EnumType lIndexMax = LevelHelper::kMaxValid;
  do {
    if (fn()) {
      break;
    }
    *startIndex = static_cast<base::type::EnumType>(*startIndex « 1);
  } while (*startIndex <= lIndexMax);
}

// ConfigurationTypeHelper

const char* ConfigurationTypeHelper::convertToString(ConfigurationType configurationType) {
  // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
  if (configurationType == ConfigurationType::Enabled) return "ENABLED";
  if (configurationType == ConfigurationType::Filename) return "FILENAME";
  if (configurationType == ConfigurationType::Format) return "FORMAT";
  if (configurationType == ConfigurationType::ToFile) return "TO_FILE";
  if (configurationType == ConfigurationType::ToStandardOutput) return "TO_STANDARD_OUTPUT";
  if (configurationType == ConfigurationType::SubsecondPrecision) return "SUBSECOND_PRECISION";
  if (configurationType == ConfigurationType::PerformanceTracking) return "PERFORMANCE_TRACKING";
  if (configurationType == ConfigurationType::MaxLogFileSize) return "MAX_LOG_FILE_SIZE";
  if (configurationType == ConfigurationType::LogFlushThreshold) return "LOG_FLUSH_THRESHOLD";
  return "UNKNOWN";
}

struct ConfigurationStringToTypeItem {
  const char* configString;
  ConfigurationType configType;
};

static struct ConfigurationStringToTypeItem configStringToTypeMap[] = {
  { "enabled", ConfigurationType::Enabled },
  { "to_file", ConfigurationType::ToFile },
  { "to_standard_output", ConfigurationType::ToStandardOutput },
  { "format", ConfigurationType::Format },
  { "filename", ConfigurationType::Filename },
  { "subsecond_precision", ConfigurationType::SubsecondPrecision },
  { "milliseconds_width", ConfigurationType::MillisecondsWidth },
  { "performance_tracking", ConfigurationType::PerformanceTracking },
  { "max_log_file_size", ConfigurationType::MaxLogFileSize },
  { "log_flush_threshold", ConfigurationType::LogFlushThreshold },
};

ConfigurationType ConfigurationTypeHelper::convertFromString(const char* configStr) {
  for (auto& item : configStringToTypeMap) {
    if (base::utils::Str::cStringCaseEq(configStr, item.configString)) {
      return item.configType;
    }
  }
  return ConfigurationType::Unknown;
}

void ConfigurationTypeHelper::forEachConfigType(base::type::EnumType* startIndex, const
std::function<bool(void)>& fn) {
  base::type::EnumType cIndexMax = ConfigurationTypeHelper::kMaxValid;
  do {
    if (fn()) {
      break;
    }
    *startIndex = static_cast<base::type::EnumType>(*startIndex « 1);
  } while (*startIndex <= cIndexMax);
}

```

```
// Configuration

Configuration::Configuration(const Configuration& c) :
  m_level(c.m_level),
  m_configurationType(c.m_configurationType),
  m_value(c.m_value) {
}

Configuration& Configuration::operator=(const Configuration& c) {
  if (&c != this) {
    m_level = c.m_level;
    m_configurationType = c.m_configurationType;
    m_value = c.m_value;
  }
  return *this;
}

Configuration::Configuration(Level level, ConfigurationType configurationType, const std::string& value) :
  m_level(level),
  m_configurationType(configurationType),
  m_value(value) {
}

void Configuration::log(el::base::type::ostream_t& os) const {
  os « LevelHelper::convertToString(m_level)
     « ELPP_LITERAL(" ") « ConfigurationTypeHelper::convertToString(m_configurationType)
     « ELPP_LITERAL(" = ") « m_value.c_str();
}

Configuration::Predicate::Predicate(Level level, ConfigurationType configurationType) :
  m_level(level),
  m_configurationType(configurationType) {
}

bool Configuration::Predicate::operator()(const Configuration* conf) const {
  return ((conf != nullptr) && (conf->level() == m_level) && (conf->configurationType() == m_configurationType));
}

// Configurations

Configurations::Configurations(void) :
  m_configurationFile(std::string()),
  m_isFromFile(false) {
}

Configurations::Configurations(const std::string& configurationFile, bool useDefaultsForRemaining,
                               Configurations* base) :
  m_configurationFile(configurationFile),
  m_isFromFile(false) {
  parseFromFile(configurationFile, base);
  if (useDefaultsForRemaining) {
    setRemainingToDefault();
  }
}

bool Configurations::parseFromFile(const std::string& configurationFile, Configurations* base) {
  // We initial assertion with true because if we have assertion disabled, we want to pass this
  // check and if assertion is enabled we will have values re-assigned any way.
  bool assertionPassed = true;
  ELPP_ASSERT((assertionPassed = base::utils::File::pathExists(configurationFile.c_str(), true)) == true,
              "Configuration file [" « configurationFile « "] does not exist!");
  if (!assertionPassed) {
    return false;
  }
  bool success = Parser::parseFromFile(configurationFile, this, base);
  m_isFromFile = success;
  return success;
}

bool Configurations::parseFromText(const std::string& configurationsString, Configurations* base) {
  bool success = Parser::parseFromText(configurationsString, this, base);
  if (success) {
    m_isFromFile = false;
  }
  return success;
}

void Configurations::setFromBase(Configurations* base) {
  if (base == nullptr || base == this) {
    return;
  }
  base::threading::ScopedLock scopedLock(base->lock());
  for (Configuration*& conf : base->list()) {
    set(conf);
  }
}

bool Configurations::hasConfiguration(ConfigurationType configurationType) {
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  bool result = false;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
    if (hasConfiguration(LevelHelper::castFromInt(lIndex), configurationType)) {
      result = true;
    }
    return result;
  });
  return result;
}

bool Configurations::hasConfiguration(Level level, ConfigurationType configurationType) {
  base::threading::ScopedLock scopedLock(lock());
#if ELPP_COMPILER_INTEL
  // We cant specify template types here, Intel C++ throws compilation error
  // "error: type name is not allowed"
  return RegistryWithPred::get(level, configurationType) != nullptr;
#else
  return RegistryWithPred<Configuration, Configuration::Predicate>::get(level, configurationType) !=
nullptr;
#endif  // ELPP_COMPILER_INTEL
}

void Configurations::set(Level level, ConfigurationType configurationType, const std::string& value) {
  base::threading::ScopedLock scopedLock(lock());
  unsafeSet(level, configurationType, value);  // This is not unsafe anymore as we have locked mutex
  if (level == Level::Global) {
    unsafeSetGlobally(configurationType, value, false);  // Again this is not unsafe either
  }
}

void Configurations::set(Configuration* conf) {
  if (conf == nullptr) {
    return;
  }
  set(conf->level(), conf->configurationType(), conf->value());
}

void Configurations::setToDefault(void) {
  setGlobally(ConfigurationType::Enabled, std::string("true"), true);
  setGlobally(ConfigurationType::Filename, std::string(base::consts::kDefaultLogFile), true);
#if defined(ELPP_NO_LOG_TO_FILE)
  setGlobally(ConfigurationType::ToFile, std::string("false"), true);
#else
  setGlobally(ConfigurationType::ToFile, std::string("true"), true);
#endif // defined(ELPP_NO_LOG_TO_FILE)
  setGlobally(ConfigurationType::ToStandardOutput, std::string("true"), true);
  setGlobally(ConfigurationType::SubsecondPrecision, std::string("3"), true);
  setGlobally(ConfigurationType::PerformanceTracking, std::string("true"), true);
  setGlobally(ConfigurationType::MaxLogFileSize, std::string("0"), true);
  setGlobally(ConfigurationType::LogFlushThreshold, std::string("0"), true);

  setGlobally(ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"), true);
  set(Level::Debug, ConfigurationType::Format,
      std::string("%datetime %level [%logger] [%user@%host] [%func] [%loc] %msg"));
  // INFO and WARNING are set to default by Level::Global
  set(Level::Error, ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"));
  set(Level::Fatal, ConfigurationType::Format, std::string("%datetime %level [%logger] %msg"));
  set(Level::Verbose, ConfigurationType::Format, std::string("%datetime %level-%vlevel [%logger]
%msg"));
  set(Level::Trace, ConfigurationType::Format, std::string("%datetime %level [%logger] [%func] [%loc]
%msg"));
}

void Configurations::setRemainingToDefault(void) {
  base::threading::ScopedLock scopedLock(lock());
#if defined(ELPP_NO_LOG_TO_FILE)
  unsafeSetIfNotExist(Level::Global, ConfigurationType::Enabled, std::string("false"));
#else
  unsafeSetIfNotExist(Level::Global, ConfigurationType::Enabled, std::string("true"));
#endif // defined(ELPP_NO_LOG_TO_FILE)
  unsafeSetIfNotExist(Level::Global, ConfigurationType::Filename,
std::string(base::consts::kDefaultLogFile));
  unsafeSetIfNotExist(Level::Global, ConfigurationType::ToStandardOutput, std::string("true"));
  unsafeSetIfNotExist(Level::Global, ConfigurationType::SubsecondPrecision, std::string("3"));
  unsafeSetIfNotExist(Level::Global, ConfigurationType::PerformanceTracking, std::string("true"));
  unsafeSetIfNotExist(Level::Global, ConfigurationType::MaxLogFileSize, std::string("0"));
  unsafeSetIfNotExist(Level::Global, ConfigurationType::Format, std::string("%datetime %level
[%logger] %msg"));
  unsafeSetIfNotExist(Level::Debug, ConfigurationType::Format,
                      std::string("%datetime %level [%logger] [%user@%host] [%func] [%loc] %msg"));
  // INFO and WARNING are set to default by Level::Global
  unsafeSetIfNotExist(Level::Error, ConfigurationType::Format, std::string("%datetime %level [%logger]
%msg"));
  unsafeSetIfNotExist(Level::Fatal, ConfigurationType::Format, std::string("%datetime %level [%logger]
%msg"));
  unsafeSetIfNotExist(Level::Verbose, ConfigurationType::Format, std::string("%datetime %level-%vlevel
[%logger] %msg"));
  unsafeSetIfNotExist(Level::Trace, ConfigurationType::Format,
                      std::string("%datetime %level [%logger] [%func] [%loc] %msg"));
}

bool Configurations::Parser::parseFromFile(const std::string& configurationFile, Configurations*
sender,
    Configurations* base) {
  sender->setFromBase(base);
  std::ifstream fileStream_(configurationFile.c_str(), std::ifstream::in);
  ELPP_ASSERT(fileStream_.is_open(), "Unable to open configuration file [" « configurationFile « "]
for parsing.");
  bool parsedSuccessfully = false;
  std::string line = std::string();
  Level currLevel = Level::Unknown;
  std::string currConfigStr = std::string();
  std::string currLevelStr = std::string();
  while (fileStream_.good()) {
    std::getline(fileStream_, line);
    parsedSuccessfully = parseLine(&line, &currConfigStr, &currLevelStr, &currLevel, sender);
    ELPP_ASSERT(parsedSuccessfully, "Unable to parse configuration line: " « line);
  }
  return parsedSuccessfully;
}

bool Configurations::Parser::parseFromText(const std::string& configurationsString, Configurations*
sender,
    Configurations* base) {
  sender->setFromBase(base);
  bool parsedSuccessfully = false;
  std::stringstream ss(configurationsString);
  std::string line = std::string();
  Level currLevel = Level::Unknown;
  std::string currConfigStr = std::string();
  std::string currLevelStr = std::string();
  while (std::getline(ss, line)) {
    parsedSuccessfully = parseLine(&line, &currConfigStr, &currLevelStr, &currLevel, sender);
    ELPP_ASSERT(parsedSuccessfully, "Unable to parse configuration line: " « line);
  }
  return parsedSuccessfully;
}

void Configurations::Parser::ignoreComments(std::string* line) {
  std::size_t foundAt = 0;
  std::size_t quotesStart = line->find("\"");
  std::size_t quotesEnd = std::string::npos;
  if (quotesStart != std::string::npos) {
    quotesEnd = line->find("\"", quotesStart + 1);
    while (quotesEnd != std::string::npos && line->at(quotesEnd - 1) == '\\') {
      // Do not erase slash yet - we will erase it in parseLine(..) while loop
      quotesEnd = line->find("\"", quotesEnd + 2);
    }
  }
  if ((foundAt = line->find(base::consts::kConfigurationComment)) != std::string::npos) {
    if (foundAt < quotesEnd) {
      foundAt = line->find(base::consts::kConfigurationComment, quotesEnd + 1);
    }
    *line = line->substr(0, foundAt);
  }
}

bool Configurations::Parser::isLevel(const std::string& line) {
  return base::utils::Str::startsWith(line, std::string(base::consts::kConfigurationLevel));
}

bool Configurations::Parser::isComment(const std::string& line) {
  return base::utils::Str::startsWith(line, std::string(base::consts::kConfigurationComment));
}

bool Configurations::Parser::isConfig(const std::string& line) {
  std::size_t assignment = line.find('=');
  return line != "" &&
         ((line[0] >= 'A' && line[0] <= 'Z') || (line[0] >= 'a' && line[0] <= 'z')) &&
         (assignment != std::string::npos) &&
         (line.size() > assignment);
}

bool Configurations::Parser::parseLine(std::string* line, std::string* currConfigStr, std::string*
currLevelStr,
                                       Level* currLevel,
                                       Configurations* conf) {
  ConfigurationType currConfig = ConfigurationType::Unknown;
  std::string currValue = std::string();
  *line = base::utils::Str::trim(*line);
  if (isComment(*line)) return true;
  ignoreComments(line);
  *line = base::utils::Str::trim(*line);
  if (line->empty()) {
    // Comment ignored
    return true;
  }
  if (isLevel(*line)) {
    if (line->size() <= 2) {
      return true;
    }
    *currLevelStr = line->substr(1, line->size() - 2);
    *currLevelStr = base::utils::Str::toUpper(*currLevelStr);
    *currLevelStr = base::utils::Str::trim(*currLevelStr);
    *currLevel = LevelHelper::convertFromString(currLevelStr->c_str());
    return true;
  }
  if (isConfig(*line)) {
    std::size_t assignment = line->find('=');
    *currConfigStr = line->substr(0, assignment);
    *currConfigStr = base::utils::Str::toUpper(*currConfigStr);
    *currConfigStr = base::utils::Str::trim(*currConfigStr);
    currConfig = ConfigurationTypeHelper::convertFromString(currConfigStr->c_str());
    currValue = line->substr(assignment + 1);
    currValue = base::utils::Str::trim(currValue);
    std::size_t quotesStart = currValue.find("\"", 0);
    std::size_t quotesEnd = std::string::npos;
    if (quotesStart != std::string::npos) {
      quotesEnd = currValue.find("\"", quotesStart + 1);
      while (quotesEnd != std::string::npos && currValue.at(quotesEnd - 1) == '\\') {
        currValue = currValue.erase(quotesEnd - 1, 1);
        quotesEnd = currValue.find("\"", quotesEnd + 2);
      }
    }
    if (quotesStart != std::string::npos && quotesEnd != std::string::npos) {
      // Quote provided - check and strip if valid
      ELPP_ASSERT((quotesStart < quotesEnd), "Configuration error - No ending quote found in ["
                  « currConfigStr « "]");
      ELPP_ASSERT((quotesStart + 1 != quotesEnd), "Empty configuration value for [" « currConfigStr «
"]");
      if ((quotesStart != quotesEnd) && (quotesStart + 1 != quotesEnd)) {
        // Explicit check in case if assertion is disabled
        currValue = currValue.substr(quotesStart + 1, quotesEnd - 1);
      }
    }
  }
  ELPP_ASSERT(*currLevel != Level::Unknown, "Unrecognized severity level [" « *currLevelStr « "]");
  ELPP_ASSERT(currConfig != ConfigurationType::Unknown, "Unrecognized configuration [" «
*currConfigStr « "]");
  if (*currLevel == Level::Unknown || currConfig == ConfigurationType::Unknown) {
    return false;  // unrecognizable level or config
  }
  conf->set(*currLevel, currConfig, currValue);
  return true;
}

void Configurations::unsafeSetIfNotExist(Level level, ConfigurationType configurationType, const
std::string& value) {
  Configuration* conf = RegistryWithPred<Configuration, Configuration::Predicate>::get(level,
configurationType);
  if (conf == nullptr) {
    unsafeSet(level, configurationType, value);
  }
}

void Configurations::unsafeSet(Level level, ConfigurationType configurationType, const std::string&
value) {
  Configuration* conf = RegistryWithPred<Configuration, Configuration::Predicate>::get(level,
configurationType);
  if (conf == nullptr) {
    registerNew(new Configuration(level, configurationType, value));
  } else {
    conf->setValue(value);
  }
  if (level == Level::Global) {
    unsafeSetGlobally(configurationType, value, false);
  }
}

void Configurations::setGlobally(ConfigurationType configurationType, const std::string& value,
                                 bool includeGlobalLevel) {
  if (includeGlobalLevel) {
    set(Level::Global, configurationType, value);
  }
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
    set(LevelHelper::castFromInt(lIndex), configurationType, value);
    return false;  // Do not break lambda function yet as we need to set all levels regardless
  });
}

void Configurations::unsafeSetGlobally(ConfigurationType configurationType, const std::string& value,
                                       bool includeGlobalLevel) {
  if (includeGlobalLevel) {
    unsafeSet(Level::Global, configurationType, value);
  }
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool  {
    unsafeSet(LevelHelper::castFromInt(lIndex), configurationType, value);
    return false;  // Do not break lambda function yet as we need to set all levels regardless
  });
}

// LogBuilder

void LogBuilder::convertToColoredOutput(base::type::string_t* logLine, Level level) {
  if (!m_termSupportsColor) return;
  const base::type::char_t* resetColor = ELPP_LITERAL("\x1b[0m");
  if (level == Level::Error || level == Level::Fatal)
    *logLine = ELPP_LITERAL("\x1b[31m") + *logLine + resetColor;
  else if (level == Level::Warning)
    *logLine = ELPP_LITERAL("\x1b[33m") + *logLine + resetColor;
  else if (level == Level::Debug)
    *logLine = ELPP_LITERAL("\x1b[32m") + *logLine + resetColor;
  else if (level == Level::Info)
    *logLine = ELPP_LITERAL("\x1b[36m") + *logLine + resetColor;
  else if (level == Level::Trace)
    *logLine = ELPP_LITERAL("\x1b[35m") + *logLine + resetColor;
}

// Logger

Logger::Logger(const std::string& id, base::LogStreamsReferenceMapPtr logStreamsReference) :
  m_id(id),
  m_typedConfigurations(nullptr),
  m_parentApplicationName(std::string()),
  m_isConfigured(false),
  m_logStreamsReference(logStreamsReference) {
  initUnflushedCount();
}

Logger::Logger(const std::string& id, const Configurations& configurations,
               base::LogStreamsReferenceMapPtr logStreamsReference) :
  m_id(id),
  m_typedConfigurations(nullptr),
  m_parentApplicationName(std::string()),
  m_isConfigured(false),
  m_logStreamsReference(logStreamsReference) {
  initUnflushedCount();
  configure(configurations);
}

Logger::Logger(const Logger& logger) {
  base::utils::safeDelete(m_typedConfigurations);
  m_id = logger.m_id;
  m_typedConfigurations = logger.m_typedConfigurations;
  m_parentApplicationName = logger.m_parentApplicationName;
  m_isConfigured = logger.m_isConfigured;
  m_configurations = logger.m_configurations;
  m_unflushedCount = logger.m_unflushedCount;
  m_logStreamsReference = logger.m_logStreamsReference;
}

Logger& Logger::operator=(const Logger& logger) {
  if (&logger != this) {
    base::utils::safeDelete(m_typedConfigurations);
    m_id = logger.m_id;
    m_typedConfigurations = logger.m_typedConfigurations;
    m_parentApplicationName = logger.m_parentApplicationName;
    m_isConfigured = logger.m_isConfigured;
    m_configurations = logger.m_configurations;
    m_unflushedCount = logger.m_unflushedCount;
    m_logStreamsReference = logger.m_logStreamsReference;
  }
  return *this;
}

void Logger::configure(const Configurations& configurations) {
  m_isConfigured = false;  // we set it to false in case if we fail
  initUnflushedCount();
  if (m_typedConfigurations != nullptr) {
    Configurations* c = const_cast<Configurations*>(m_typedConfigurations->configurations());
    if (c->hasConfiguration(Level::Global, ConfigurationType::Filename)) {
      flush();
    }
  }
  base::threading::ScopedLock scopedLock(lock());
  if (m_configurations != configurations) {
    m_configurations.setFromBase(const_cast<Configurations*>(&configurations));
  }
  base::utils::safeDelete(m_typedConfigurations);
  m_typedConfigurations = new base::TypedConfigurations(&m_configurations, m_logStreamsReference);
  resolveLoggerFormatSpec();
  m_isConfigured = true;
}

void Logger::reconfigure(void) {
  ELPP_INTERNAL_INFO(1, "Reconfiguring logger [" « m_id « "]");
  configure(m_configurations);
}

bool Logger::isValidId(const std::string& id) {
  for (std::string::const_iterator it = id.begin(); it != id.end(); ++it) {
    if (!base::utils::Str::contains(base::consts::kValidLoggerIdSymbols, *it)) {
      return false;
    }
  }
  return true;
}

void Logger::flush(void) {
  ELPP_INTERNAL_INFO(3, "Flushing logger [" « m_id « "] all levels");
  base::threading::ScopedLock scopedLock(lock());
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
    flush(LevelHelper::castFromInt(lIndex), nullptr);
    return false;
  });
}

void Logger::flush(Level level, base::type::fstream_t* fs) {
  if (fs == nullptr && m_typedConfigurations->toFile(level)) {
    fs = m_typedConfigurations->fileStream(level);
  }
  if (fs != nullptr) {
    fs->flush();
    std::unordered_map<Level, unsigned int>::iterator iter = m_unflushedCount.find(level);
    if (iter != m_unflushedCount.end()) {
      iter->second = 0;
    }
    Helpers::validateFileRolling(this, level);
  }
}

void Logger::initUnflushedCount(void) {
  m_unflushedCount.clear();
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
    m_unflushedCount.insert(std::make_pair(LevelHelper::castFromInt(lIndex), 0));
    return false;
  });
}

void Logger::resolveLoggerFormatSpec(void) const {
  base::type::EnumType lIndex = LevelHelper::kMinValid;
  LevelHelper::forEachLevel(&lIndex, [&](void) -> bool {
    base::LogFormat* logFormat =
    const_cast<base::LogFormat*>(&m_typedConfigurations->logFormat(LevelHelper::castFromInt(lIndex)));
    base::utils::Str::replaceFirstWithEscape(logFormat->m_format,
base::consts::kLoggerIdFormatSpecifier, m_id);
    return false;
  });
}

// el::base
namespace base {

// el::base::utils
namespace utils {

// File

base::type::fstream_t* File::newFileStream(const std::string& filename) {
  base::type::fstream_t *fs = new base::type::fstream_t(filename.c_str(),
      base::type::fstream_t::out
#if !defined(ELPP_FRESH_LOG_FILE)
        | base::type::fstream_t::app
#endif
                                                        );
#if defined(ELPP_UNICODE)
```

```
  std::locale elppUnicodeLocale("");
#  if ELPP_OS_WINDOWS
  std::locale elppUnicodeLocaleWindows(elppUnicodeLocale, new std::codecvt_utf8_utf16<wchar_t>);
  elppUnicodeLocale = elppUnicodeLocaleWindows;
#  endif // ELPP_OS_WINDOWS
  fs->imbue(elppUnicodeLocale);
#endif  // defined(ELPP_UNICODE)
  if (fs->is_open()) {
    fs->flush();
  } else {
    base::utils::safeDelete(fs);
    ELPP_INTERNAL_ERROR("Bad file [" « filename « "]", true);
  }
  return fs;
}

std::size_t File::getSizeOfFile(base::type::fstream_t* fs) {
  if (fs == nullptr) {
    return 0;
  }
  // Since the file stream is appended to or truncated, the current
  // offset is the file size.
  std::size_t size = static_cast<std::size_t>(fs->tellg());
  return size;
}

bool File::pathExists(const char* path, bool considerFile) {
  if (path == nullptr) {
    return false;
  }
#if ELPP_OS_UNIX
  ELPP_UNUSED(considerFile);
  struct stat st;
  return (stat(path, &st) == 0);
#elif ELPP_OS_WINDOWS
  DWORD fileType = GetFileAttributesA(path);
  if (fileType == INVALID_FILE_ATTRIBUTES) {
    return false;
  }
  return considerFile ? true : ((fileType & FILE_ATTRIBUTE_DIRECTORY) == 0 ? false : true);
#endif  // ELPP_OS_UNIX
}

bool File::createPath(const std::string& path) {
  if (path.empty()) {
    return false;
  }
  if (base::utils::File::pathExists(path.c_str())) {
    return true;
  }
  int status = -1;

  char* currPath = const_cast<char*>(path.c_str());
  std::string builtPath = std::string();
#if ELPP_OS_UNIX
  if (path[0] == '/') {
    builtPath = "/";
  }
  currPath = STRTOK(currPath, base::consts::kFilePathSeparator, 0);
#elif ELPP_OS_WINDOWS
  // Use secure functions API
  char* nextTok_ = nullptr;
  currPath = STRTOK(currPath, base::consts::kFilePathSeparator, &nextTok_);
  ELPP_UNUSED(nextTok_);
#endif  // ELPP_OS_UNIX
  while (currPath != nullptr) {
    builtPath.append(currPath);
    builtPath.append(base::consts::kFilePathSeparator);
#if ELPP_OS_UNIX
    status = mkdir(builtPath.c_str(), ELPP_LOG_PERMS);
    currPath = STRTOK(nullptr, base::consts::kFilePathSeparator, 0);
#elif ELPP_OS_WINDOWS
    status = _mkdir(builtPath.c_str());
    currPath = STRTOK(nullptr, base::consts::kFilePathSeparator, &nextTok_);
#endif  // ELPP_OS_UNIX
  }
  if (status == -1) {
    ELPP_INTERNAL_ERROR("Error while creating path [" « path « "]", true);
    return false;
  }
  return true;
}

std::string File::extractPathFromFilename(const std::string& fullPath, const char* separator) {
  if ((fullPath == "") || (fullPath.find(separator) == std::string::npos)) {
    return fullPath;
  }
  std::size_t lastSlashAt = fullPath.find_last_of(separator);
  if (lastSlashAt == 0) {
    return std::string(separator);
  }
  return fullPath.substr(0, lastSlashAt + 1);
}

void File::buildStrippedFilename(const char* filename, char buff[], std::size_t limit) {
  std::size_t sizeOfFilename = strlen(filename);
  if (sizeOfFilename >= limit) {
    filename += (sizeOfFilename - limit);
    if (filename[0] != '.' && filename[1] != '.') {  // prepend if not already
      filename += 3;  // 3 = '..'
      STRCAT(buff, "..", limit);
    }
  }
  STRCAT(buff, filename, limit);
}

void File::buildBaseFilename(const std::string& fullPath, char buff[], std::size_t limit, const char*
separator) {
  const char *filename = fullPath.c_str();
  std::size_t lastSlashAt = fullPath.find_last_of(separator);
  filename += lastSlashAt ? lastSlashAt+1 : 0;
  std::size_t sizeOfFilename = strlen(filename);
  if (sizeOfFilename >= limit) {
    filename += (sizeOfFilename - limit);
    if (filename[0] != '.' && filename[1] != '.') {  // prepend if not already
      filename += 3;  // 3 = '..'
      STRCAT(buff, "..", limit);
    }
  }
  STRCAT(buff, filename, limit);
}

// Str

bool Str::wildCardMatch(const char* str, const char* pattern) {
  while (*pattern) {
    switch (*pattern) {
    case '?':
      if (!*str)
        return false;
      ++str;
      ++pattern;
      break;
    case '*':
      if (wildCardMatch(str, pattern + 1))
        return true;
      if (*str && wildCardMatch(str + 1, pattern))
        return true;
      return false;
    default:
      if (*str++ != *pattern++)
        return false;
      break;
    }
  }
  return !*str && !*pattern;
}

std::string& Str::ltrim(std::string& str) {
  str.erase(str.begin(), std::find_if(str.begin(), str.end(), [](char c) {
    return !std::isspace(c);
  } ));
  return str;
}

std::string& Str::rtrim(std::string& str) {
  str.erase(std::find_if(str.rbegin(), str.rend(), [](char c) {
    return !std::isspace(c);
  }).base(), str.end());
  return str;
}

std::string& Str::trim(std::string& str) {
  return ltrim(rtrim(str));
}

bool Str::startsWith(const std::string& str, const std::string& start) {
  return (str.length() >= start.length()) && (str.compare(0, start.length(), start) == 0);
}

bool Str::endsWith(const std::string& str, const std::string& end) {
  return (str.length() >= end.length()) && (str.compare(str.length() - end.length(), end.length(),
end) == 0);
}
```

```

std::string& Str::replaceAll(std::string& str, char replaceWhat, char replaceWith) {
  std::replace(str.begin(), str.end(), replaceWhat, replaceWith);
  return str;
}

std::string& Str::replaceAll(std::string& str, const std::string& replaceWhat,
                             const std::string& replaceWith) {
  if (replaceWhat == replaceWith)
    return str;
  std::size_t foundAt = std::string::npos;
  while ((foundAt = str.find(replaceWhat, foundAt + 1)) != std::string::npos) {
    str.replace(foundAt, replaceWhat.length(), replaceWith);
  }
  return str;
}

void Str::replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t& replaceWhat,
                                 const base::type::string_t& replaceWith) {
  std::size_t foundAt = base::type::string_t::npos;
  while ((foundAt = str.find(replaceWhat, foundAt + 1)) != base::type::string_t::npos) {
    if (foundAt > 0 && str[foundAt - 1] == base::consts::kFormatSpecifierChar) {
      str.erase(foundAt - 1, 1);
      ++foundAt;
    } else {
      str.replace(foundAt, replaceWhat.length(), replaceWith);
      return;
    }
  }
}
#if defined(ELPP_UNICODE)
void Str::replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t& replaceWhat,
                                 const std::string& replaceWith) {
  replaceFirstWithEscape(str, replaceWhat, base::type::string_t(replaceWith.begin(),
replaceWith.end()));
}
#endif  // defined(ELPP_UNICODE)

std::string& Str::toUpper(std::string& str) {
  std::transform(str.begin(), str.end(), str.begin(),
  [](char c) {
    return static_cast<char>(::toupper(c));
  });
  return str;
}

bool Str::cStringEq(const char* s1, const char* s2) {
  if (s1 == nullptr && s2 == nullptr) return true;
  if (s1 == nullptr || s2 == nullptr) return false;
  return strcmp(s1, s2) == 0;
}

bool Str::cStringCaseEq(const char* s1, const char* s2) {
  if (s1 == nullptr && s2 == nullptr) return true;
  if (s1 == nullptr || s2 == nullptr) return false;

  // With thanks to cygwin for this code
  int d = 0;

  while (true) {
    const int c1 = toupper(*s1++);
    const int c2 = toupper(*s2++);

    if (((d = c1 - c2) != 0) || (c2 == '\0')) {
      break;
    }
  }

  return d == 0;
}

bool Str::contains(const char* str, char c) {
  for (; *str; ++str) {
    if (*str == c)
      return true;
  }
  return false;
}

char* Str::convertAndAddToBuff(std::size_t n, int len, char* buf, const char* bufLim, bool zeroPadded)
{
  char localBuff[10] = "";
  char* p = localBuff + sizeof(localBuff) - 2;
  if (n > 0) {
    for (; n > 0 && p > localBuff && len > 0; n /= 10, --len)
      *--p = static_cast<char>(n % 10 + '0');
  } else {
    *--p = '0';
    --len;
  }
  if (zeroPadded)
    while (p > localBuff && len-- > 0) *--p = static_cast<char>('0');
  return addToBuff(p, buf, bufLim);
}

char* Str::addToBuff(const char* str, char* buf, const char* bufLim) {
  while ((buf < bufLim) && ((*buf = *str++) != '\0'))
    ++buf;
  return buf;
}

char* Str::clearBuff(char buff[], std::size_t lim) {
  STRCPY(buff, "", lim);
  ELPP_UNUSED(lim);  // For *nix we dont have anything using lim in above STRCPY macro
  return buff;
}

char* Str::wcharPtrToCharPtr(const wchar_t* line) {
  std::size_t len_ = wcslen(line) + 1;
  char* buff_ = static_cast<char*>(malloc(len_ + 1));
#      if ELPP_OS_UNIX || (ELPP_OS_WINDOWS && !ELPP_CRT_DBG_WARNINGS)
  std::wcstombs(buff_, line, len_);
#      elif ELPP_OS_WINDOWS
  std::size_t convCount_ = 0;
  mbstate_t mbState_;
  ::memset(static_cast<void*>(&mbState_), 0, sizeof(mbState_));
  wcsrtombs_s(&convCount_, buff_, len_, &line, len_, &mbState_);
#      endif  // ELPP_OS_UNIX || (ELPP_OS_WINDOWS && !ELPP_CRT_DBG_WARNINGS)
  return buff_;
}

// OS

#if ELPP_OS_WINDOWS
const char* OS::getWindowsEnvironmentVariable(const char* varname) {
  const DWORD bufferLen = 50;
  static char buffer[bufferLen];
  if (GetEnvironmentVariableA(varname, buffer, bufferLen)) {
    return buffer;
  }
  return nullptr;
}
#endif  // ELPP_OS_WINDOWS
#if ELPP_OS_ANDROID
std::string OS::getProperty(const char* prop) {
  char propVal[PROP_VALUE_MAX + 1];
  int ret = __system_property_get(prop, propVal);
  return ret == 0 ? std::string() : std::string(propVal);
}

std::string OS::getDeviceName(void) {
  std::stringstream ss;
  std::string manufacturer = getProperty("ro.product.manufacturer");
  std::string model = getProperty("ro.product.model");
  if (manufacturer.empty() || model.empty()) {
    return std::string();
  }
  ss « manufacturer « "-" « model;
  return ss.str();
}
#endif  // ELPP_OS_ANDROID

const std::string OS::getBashOutput(const char* command) {
#if (ELPP_OS_UNIX && !ELPP_OS_ANDROID && !ELPP_CYGWIN)
  if (command == nullptr) {
    return std::string();
  }
  FILE* proc = nullptr;
  if ((proc = popen(command, "r")) == nullptr) {
    ELPP_INTERNAL_ERROR("\nUnable to run command [" « command « "]", true);
    return std::string();
  }
  char hBuff[4096];
  if (fgets(hBuff, sizeof(hBuff), proc) != nullptr) {
    pclose(proc);
    const std::size_t buffLen = strlen(hBuff);
    if (buffLen > 0 && hBuff[buffLen - 1] == '\n') {
      hBuff[buffLen - 1] = '\0';
    }
    return std::string(hBuff);
  } else {
    pclose(proc);
  }
  return std::string();
```

```
#else
  ELPP_UNUSED(command);
  return std::string();
#endif  // (ELPP_OS_UNIX && !ELPP_OS_ANDROID && !ELPP_CYGWIN)
}

std::string OS::getEnvironmentVariable(const char* variableName, const char* defaultVal,
                                       const char* alternativeBashCommand) {
#if ELPP_OS_UNIX
  const char* val = getenv(variableName);
#elif ELPP_OS_WINDOWS
  const char* val = getWindowsEnvironmentVariable(variableName);
#endif  // ELPP_OS_UNIX
  if ((val == nullptr) || ((strcmp(val, "") == 0))) {
#if ELPP_OS_UNIX && defined(ELPP_FORCE_ENV_VAR_FROM_BASH)
    // Try harder on unix-based systems
    std::string valBash = base::utils::OS::getBashOutput(alternativeBashCommand);
    if (valBash.empty()) {
      return std::string(defaultVal);
    } else {
      return valBash;
    }
#elif ELPP_OS_WINDOWS || ELPP_OS_UNIX
    ELPP_UNUSED(alternativeBashCommand);
    return std::string(defaultVal);
#endif  // ELPP_OS_UNIX && defined(ELPP_FORCE_ENV_VAR_FROM_BASH)
  }
  return std::string(val);
}

std::string OS::currentUser(void) {
#if ELPP_OS_UNIX && !ELPP_OS_ANDROID
  return getEnvironmentVariable("USER", base::consts::kUnknownUser, "whoami");
#elif ELPP_OS_WINDOWS
  return getEnvironmentVariable("USERNAME", base::consts::kUnknownUser);
#elif ELPP_OS_ANDROID
  ELPP_UNUSED(base::consts::kUnknownUser);
  return std::string("android");
#else
  return std::string();
#endif  // ELPP_OS_UNIX && !ELPP_OS_ANDROID
}

std::string OS::currentHost(void) {
#if ELPP_OS_UNIX && !ELPP_OS_ANDROID
  return getEnvironmentVariable("HOSTNAME", base::consts::kUnknownHost, "hostname");
#elif ELPP_OS_WINDOWS
  return getEnvironmentVariable("COMPUTERNAME", base::consts::kUnknownHost);
#elif ELPP_OS_ANDROID
  ELPP_UNUSED(base::consts::kUnknownHost);
  return getDeviceName();
#else
  return std::string();
#endif  // ELPP_OS_UNIX && !ELPP_OS_ANDROID
}

bool OS::termSupportsColor(void) {
  std::string term = getEnvironmentVariable("TERM", "");
  return term == "xterm" || term == "xterm-color" || term == "xterm-256color"
         || term == "screen" || term == "linux" || term == "cygwin"
         || term == "screen-256color";
}

// DateTime

void DateTime::gettimeofday(struct timeval* tv) {
#if ELPP_OS_WINDOWS
  if (tv != nullptr) {
#  if ELPP_COMPILER_MSVC || defined(_MSC_EXTENSIONS)
    const unsigned __int64 delta_ = 11644473600000000Ui64;
#  else
    const unsigned __int64 delta_ = 11644473600000000ULL;
#  endif  // ELPP_COMPILER_MSVC || defined(_MSC_EXTENSIONS)
    const double secOffSet = 0.000001;
    const unsigned long usecOffSet = 1000000;
    FILETIME fileTime;
    GetSystemTimeAsFileTime(&fileTime);
    unsigned __int64 present = 0;
    present |= fileTime.dwHighDateTime;
    present = present « 32;
    present |= fileTime.dwLowDateTime;
    present /= 10;  // mic-sec
    // Subtract the difference
    present -= delta_;
    tv->tv_sec = static_cast<long>(present * secOffSet);
    tv->tv_usec = static_cast<long>(present % usecOffSet);
  }
```

```
#else
  ::gettimeofday(tv, nullptr);
#endif  // ELPP_OS_WINDOWS
}

std::string DateTime::getDateTime(const char* format, const base::SubsecondPrecision* ssPrec) {
  struct timeval currTime;
  gettimeofday(&currTime);
  return timevalToString(currTime, format, ssPrec);
}

std::string DateTime::timevalToString(struct timeval tval, const char* format,
                                      const el::base::SubsecondPrecision* ssPrec) {
  struct ::tm timeInfo;
  buildTimeInfo(&tval, &timeInfo);
  const int kBuffSize = 30;
  char buff_[kBuffSize] = "";
  parseFormat(buff_, kBuffSize, format, &timeInfo, static_cast<std::size_t>(tval.tv_usec /
ssPrec->m_offset),
              ssPrec);
  return std::string(buff_);
}

base::type::string_t DateTime::formatTime(unsigned long long time, base::TimestampUnit timestampUnit)
{
  base::type::EnumType start = static_cast<base::type::EnumType>(timestampUnit);
  const base::type::char_t* unit = base::consts::kTimeFormats[start].unit;
  for (base::type::EnumType i = start; i < base::consts::kTimeFormatsCount - 1; ++i) {
    if (time <= base::consts::kTimeFormats[i].value) {
      break;
    }
    if (base::consts::kTimeFormats[i].value == 1000.0f && time / 1000.0f < 1.9f) {
      break;
    }
    time /= static_cast<decltype(time)>(base::consts::kTimeFormats[i].value);
    unit = base::consts::kTimeFormats[i + 1].unit;
  }
  base::type::stringstream_t ss;
  ss « time « " " « unit;
  return ss.str();
}

unsigned long long DateTime::getTimeDifference(const struct timeval& endTime, const struct timeval&
startTime,
    base::TimestampUnit timestampUnit) {
  if (timestampUnit == base::TimestampUnit::Microsecond) {
    return static_cast<unsigned long long>(static_cast<unsigned long long>(1000000 * endTime.tv_sec +
endTime.tv_usec) -
                                           static_cast<unsigned long long>(1000000 * startTime.tv_sec
+ startTime.tv_usec));
  }
  // milliseconds
  auto conv = [](const struct timeval& tim) {
    return static_cast<unsigned long long>((tim.tv_sec * 1000) + (tim.tv_usec / 1000));
  };
  return static_cast<unsigned long long>(conv(endTime) - conv(startTime));
}

struct ::tm* DateTime::buildTimeInfo(struct timeval* currTime, struct ::tm* timeInfo) {
#if ELPP_OS_UNIX
  time_t rawTime = currTime->tv_sec;
  ::elpptime_r(&rawTime, timeInfo);
  return timeInfo;
#else
#  if ELPP_COMPILER_MSVC
  ELPP_UNUSED(currTime);
  time_t t;
#    if defined(_USE_32BIT_TIME_T)
  _time32(&t);
#    else
  _time64(&t);
#    endif
  elpptime_s(timeInfo, &t);
  return timeInfo;
#  else
  // For any other compilers that don't have CRT warnings issue e.g, MinGW or TDM GCC- we use
different method
  time_t rawTime = currTime->tv_sec;
  struct tm* tmInf = elpptime(&rawTime);
  *timeInfo = *tmInf;
  return timeInfo;
#  endif  // ELPP_COMPILER_MSVC
#endif  // ELPP_OS_UNIX
}

char* DateTime::parseFormat(char* buf, std::size_t bufSz, const char* format, const struct tm* tInfo,
                            std::size_t msec, const base::SubsecondPrecision* ssPrec) {
```

```
  const char* bufLim = buf + bufSz;
  for (; *format; ++format) {
    if (*format == base::consts::kFormatSpecifierChar) {
      switch (*++format) {
      case base::consts::kFormatSpecifierChar:  // Escape
        break;
      case '\0':  // End
        --format;
        break;
      case 'd':  // Day
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_mday, 2, buf, bufLim);
        continue;
      case 'a':  // Day of week (short)
        buf = base::utils::Str::addToBuff(base::consts::kDaysAbbrev[tInfo->tm_wday], buf, bufLim);
        continue;
      case 'A':  // Day of week (long)
        buf = base::utils::Str::addToBuff(base::consts::kDays[tInfo->tm_wday], buf, bufLim);
        continue;
      case 'M':  // month
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_mon + 1, 2, buf, bufLim);
        continue;
      case 'b':  // month (short)
        buf = base::utils::Str::addToBuff(base::consts::kMonthsAbbrev[tInfo->tm_mon], buf, bufLim);
        continue;
      case 'B':  // month (long)
        buf = base::utils::Str::addToBuff(base::consts::kMonths[tInfo->tm_mon], buf, bufLim);
        continue;
      case 'y':  // year (two digits)
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_year + base::consts::kYearBase, 2, buf,
bufLim);
        continue;
      case 'Y':  // year (four digits)
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_year + base::consts::kYearBase, 4, buf,
bufLim);
        continue;
      case 'h':  // hour (12-hour)
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_hour % 12, 2, buf, bufLim);
        continue;
      case 'H':  // hour (24-hour)
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_hour, 2, buf, bufLim);
        continue;
      case 'm':  // minute
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_min, 2, buf, bufLim);
        continue;
      case 's':  // second
        buf = base::utils::Str::convertAndAddToBuff(tInfo->tm_sec, 2, buf, bufLim);
        continue;
      case 'z':  // subsecond part
      case 'g':
        buf = base::utils::Str::convertAndAddToBuff(msec, ssPrec->m_width, buf, bufLim);
        continue;
      case 'F':  // AM/PM
        buf = base::utils::Str::addToBuff((tInfo->tm_hour >= 12) ? base::consts::kPm :
base::consts::kAm, buf, bufLim);
        continue;
      default:
        continue;
      }
    }
    if (buf == bufLim) break;
    *buf++ = *format;
  }
  return buf;
}

// CommandLineArgs

void CommandLineArgs::setArgs(int argc, char** argv) {
  m_params.clear();
  m_paramsWithValue.clear();
  if (argc == 0 || argv == nullptr) {
    return;
  }
  m_argc = argc;
  m_argv = argv;
  for (int i = 1; i < m_argc; ++i) {
    const char* v = (strstr(m_argv[i], "="));
    if (v != nullptr && strlen(v) > 0) {
      std::string key = std::string(m_argv[i]);
      key = key.substr(0, key.find_first_of('='));
      if (hasParamWithValue(key.c_str())) {
        ELPP_INTERNAL_INFO(1, "Skipping [" « key « "] arg since it already has value ["
                           « getParamValue(key.c_str()) « "]");
      } else {
        m_paramsWithValue.insert(std::make_pair(key, std::string(v + 1)));
      }
    }
    if (v == nullptr) {
      if (hasParam(m_argv[i])) {
        ELPP_INTERNAL_INFO(1, "Skipping [" « m_argv[i] « "] arg since it already exists");
      } else {
        m_params.push_back(std::string(m_argv[i]));
      }
    }
  }
}

bool CommandLineArgs::hasParamWithValue(const char* paramKey) const {
  return m_paramsWithValue.find(std::string(paramKey)) != m_paramsWithValue.end();
}

const char* CommandLineArgs::getParamValue(const char* paramKey) const {
  std::unordered_map<std::string, std::string>::const_iterator iter =
m_paramsWithValue.find(std::string(paramKey));
  return iter != m_paramsWithValue.end() ? iter->second.c_str() : "";
}

bool CommandLineArgs::hasParam(const char* paramKey) const {
  return std::find(m_params.begin(), m_params.end(), std::string(paramKey)) != m_params.end();
}

bool CommandLineArgs::empty(void) const {
  return m_params.empty() && m_paramsWithValue.empty();
}

std::size_t CommandLineArgs::size(void) const {
  return m_params.size() + m_paramsWithValue.size();
}

base::type::ostream_t& operator«(base::type::ostream_t& os, const CommandLineArgs& c) {
  for (int i = 1; i < c.m_argc; ++i) {
    os « ELPP_LITERAL("[") « c.m_argv[i] « ELPP_LITERAL("]");
    if (i < c.m_argc - 1) {
      os « ELPP_LITERAL(" ");
    }
  }
  return os;
}

} // namespace utils

// el::base::threading
namespace threading {

#if ELPP_THREADING_ENABLED
#  if ELPP_USE_STD_THREADING
#      if ELPP_ASYNC_LOGGING
static void msleep(int ms) {
  // Only when async logging enabled - this is because async is strict on compiler
#          if defined(ELPP_NO_SLEEP_FOR)
  usleep(ms * 1000);
#          else
  std::this_thread::sleep_for(std::chrono::milliseconds(ms));
#          endif  // defined(ELPP_NO_SLEEP_FOR)
}
#      endif  // ELPP_ASYNC_LOGGING
#  endif  // !ELPP_USE_STD_THREADING
#endif  // ELPP_THREADING_ENABLED

} // namespace threading

// el::base

// SubsecondPrecision

void SubsecondPrecision::init(int width) {
  if (width < 1 || width > 6) {
    width = base::consts::kDefaultSubsecondPrecision;
  }
  m_width = width;
  switch (m_width) {
  case 3:
    m_offset = 1000;
    break;
  case 4:
    m_offset = 100;
    break;
  case 5:
    m_offset = 10;
    break;
  case 6:
    m_offset = 1;
    break;
  default:
    m_offset = 1000;
    break;
  }
}

// LogFormat

LogFormat::LogFormat(void) :
  m_level(Level::Unknown),
  m_userFormat(base::type::string_t()),
  m_format(base::type::string_t()),
  m_dateTimeFormat(std::string()),
  m_flags(0x0),
  m_currentUser(base::utils::OS::currentUser()),
  m_currentHost(base::utils::OS::currentHost()) {
}

LogFormat::LogFormat(Level level, const base::type::string_t& format)
  : m_level(level), m_userFormat(format), m_currentUser(base::utils::OS::currentUser()),
    m_currentHost(base::utils::OS::currentHost()) {
  parseFromFormat(m_userFormat);
}

LogFormat::LogFormat(const LogFormat& logFormat):
  m_level(logFormat.m_level),
  m_userFormat(logFormat.m_userFormat),
  m_format(logFormat.m_format),
  m_dateTimeFormat(logFormat.m_dateTimeFormat),
  m_flags(logFormat.m_flags),
  m_currentUser(logFormat.m_currentUser),
  m_currentHost(logFormat.m_currentHost) {
}

LogFormat::LogFormat(LogFormat&& logFormat) {
  m_level = std::move(logFormat.m_level);
  m_userFormat = std::move(logFormat.m_userFormat);
  m_format = std::move(logFormat.m_format);
  m_dateTimeFormat = std::move(logFormat.m_dateTimeFormat);
  m_flags = std::move(logFormat.m_flags);
  m_currentUser = std::move(logFormat.m_currentUser);
  m_currentHost = std::move(logFormat.m_currentHost);
}

LogFormat& LogFormat::operator=(const LogFormat& logFormat) {
  if (&logFormat != this) {
    m_level = logFormat.m_level;
    m_userFormat = logFormat.m_userFormat;
    m_dateTimeFormat = logFormat.m_dateTimeFormat;
    m_flags = logFormat.m_flags;
    m_currentUser = logFormat.m_currentUser;
    m_currentHost = logFormat.m_currentHost;
  }
  return *this;
}

bool LogFormat::operator==(const LogFormat& other) {
  return m_level == other.m_level && m_userFormat == other.m_userFormat && m_format == other.m_format
&&
         m_dateTimeFormat == other.m_dateTimeFormat && m_flags == other.m_flags;
}

void LogFormat::parseFromFormat(const base::type::string_t& userFormat) {
  // We make copy because we will be changing the format
  // i.e, removing user provided date format from original format
  // and then storing it.
  base::type::string_t formatCopy = userFormat;
  m_flags = 0x0;
  auto conditionalAddFlag = [&](const base::type::char_t* specifier, base::FormatFlags flag) {
    std::size_t foundAt = base::type::string_t::npos;
    while ((foundAt = formatCopy.find(specifier, foundAt + 1)) != base::type::string_t::npos) {
      if (foundAt > 0 && formatCopy[foundAt - 1] == base::consts::kFormatSpecifierChar) {
        if (hasFlag(flag)) {
          // If we already have flag we remove the escape chars so that '%%' is turned to '%'
          // even after specifier resolution - this is because we only replaceFirst specifier
          formatCopy.erase(foundAt - 1, 1);
          ++foundAt;
        }
      } else {
        if (!hasFlag(flag)) addFlag(flag);
      }
    }
  };
  conditionalAddFlag(base::consts::kAppNameFormatSpecifier, base::FormatFlags::AppName);
  conditionalAddFlag(base::consts::kSeverityLevelFormatSpecifier, base::FormatFlags::Level);
  conditionalAddFlag(base::consts::kSeverityLevelShortFormatSpecifier, base::FormatFlags::LevelShort);
  conditionalAddFlag(base::consts::kLoggerIdFormatSpecifier, base::FormatFlags::LoggerId);
  conditionalAddFlag(base::consts::kThreadIdFormatSpecifier, base::FormatFlags::ThreadId);
  conditionalAddFlag(base::consts::kLogFileFormatSpecifier, base::FormatFlags::File);
  conditionalAddFlag(base::consts::kLogFileBaseFormatSpecifier, base::FormatFlags::FileBase);
  conditionalAddFlag(base::consts::kLogLineFormatSpecifier, base::FormatFlags::Line);
  conditionalAddFlag(base::consts::kLogLocationFormatSpecifier, base::FormatFlags::Location);
  conditionalAddFlag(base::consts::kLogFunctionFormatSpecifier, base::FormatFlags::Function);
  conditionalAddFlag(base::consts::kCurrentUserFormatSpecifier, base::FormatFlags::User);
  conditionalAddFlag(base::consts::kCurrentHostFormatSpecifier, base::FormatFlags::Host);
  conditionalAddFlag(base::consts::kMessageFormatSpecifier, base::FormatFlags::LogMessage);
  conditionalAddFlag(base::consts::kVerboseLevelFormatSpecifier, base::FormatFlags::VerboseLevel);
  // For date/time we need to extract user's date format first
  std::size_t dateIndex = std::string::npos;
  if ((dateIndex = formatCopy.find(base::consts::kDateTimeFormatSpecifier)) != std::string::npos) {
    while (dateIndex != std::string::npos && dateIndex > 0 && formatCopy[dateIndex - 1] ==
base::consts::kFormatSpecifierChar) {
      dateIndex = formatCopy.find(base::consts::kDateTimeFormatSpecifier, dateIndex + 1);
    }
    if (dateIndex != std::string::npos) {
      addFlag(base::FormatFlags::DateTime);
      updateDateFormat(dateIndex, formatCopy);
    }
  }
  m_format = formatCopy;
  updateFormatSpec();
}

void LogFormat::updateDateFormat(std::size_t index, base::type::string_t& currFormat) {
  if (hasFlag(base::FormatFlags::DateTime)) {
    index += ELPP_STRLEN(base::consts::kDateTimeFormatSpecifier);
  }
  const base::type::char_t* ptr = currFormat.c_str() + index;
  if ((currFormat.size() > index) && (ptr[0] == '{')) {
    // User has provided format for date/time
    ++ptr;
    int count = 1;  // Start by 1 in order to remove starting brace
    std::stringstream ss;
    for (; *ptr; ++ptr, ++count) {
      if (*ptr == '}') {
        ++count;  // In order to remove ending brace
        break;
      }
      ss « static_cast<char>(*ptr);
    }
    currFormat.erase(index, count);
    m_dateTimeFormat = ss.str();
  } else {
    // No format provided, use default
    if (hasFlag(base::FormatFlags::DateTime)) {
      m_dateTimeFormat = std::string(base::consts::kDefaultDateTimeFormat);
    }
  }
}

void LogFormat::updateFormatSpec(void) {
  // Do not use switch over strongly typed enums because Intel C++ compilers dont support them yet.
  if (m_level == Level::Debug) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kDebugLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kDebugLevelShortLogValue);
  } else if (m_level == Level::Info) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kInfoLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kInfoLevelShortLogValue);
  } else if (m_level == Level::Warning) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kWarningLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kWarningLevelShortLogValue);
  } else if (m_level == Level::Error) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kErrorLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kErrorLevelShortLogValue);
  } else if (m_level == Level::Fatal) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kFatalLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kFatalLevelShortLogValue);
  } else if (m_level == Level::Verbose) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kVerboseLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kVerboseLevelShortLogValue);
  } else if (m_level == Level::Trace) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kSeverityLevelFormatSpecifier,
        base::consts::kTraceLevelLogValue);
    base::utils::Str::replaceFirstWithEscape(m_format,
base::consts::kSeverityLevelShortFormatSpecifier,
        base::consts::kTraceLevelShortLogValue);
  }
  if (hasFlag(base::FormatFlags::User)) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kCurrentUserFormatSpecifier,
        m_currentUser);
  }
  if (hasFlag(base::FormatFlags::Host)) {
    base::utils::Str::replaceFirstWithEscape(m_format, base::consts::kCurrentHostFormatSpecifier,
        m_currentHost);
  }
  // Ignore Level::Global and Level::Unknown
}

// TypedConfigurations

TypedConfigurations::TypedConfigurations(Configurations* configurations,
    LogStreamsReferenceMapPtr logStreamsReference) {
  m_configurations = configurations;
  m_logStreamsReference = logStreamsReference;
  build(m_configurations);
}

TypedConfigurations::TypedConfigurations(const TypedConfigurations& other) {
  this->m_configurations = other.m_configurations;
  this->m_logStreamsReference = other.m_logStreamsReference;
  build(m_configurations);
}

bool TypedConfigurations::enabled(Level level) {
  return getConfigByVal<bool>(level, &m_enabledMap, "enabled");
}

bool TypedConfigurations::toFile(Level level) {
  return getConfigByVal<bool>(level, &m_toFileMap, "toFile");
}

const std::string& TypedConfigurations::filename(Level level) {
  return getConfigByRef<std::string>(level, &m_filenameMap, "filename");
}

bool TypedConfigurations::toStandardOutput(Level level) {
  return getConfigByVal<bool>(level, &m_toStandardOutputMap, "toStandardOutput");
}

const base::LogFormat& TypedConfigurations::logFormat(Level level) {
  return getConfigByRef<base::LogFormat>(level, &m_logFormatMap, "logFormat");
}

const base::SubsecondPrecision& TypedConfigurations::subsecondPrecision(Level level) {
  return getConfigByRef<base::SubsecondPrecision>(level, &m_subsecondPrecisionMap,
"subsecondPrecision");
}

const base::MillisecondsWidth& TypedConfigurations::millisecondsWidth(Level level) {
  return getConfigByRef<base::MillisecondsWidth>(level, &m_subsecondPrecisionMap,
"millisecondsWidth");
}

bool TypedConfigurations::performanceTracking(Level level) {
  return getConfigByVal<bool>(level, &m_performanceTrackingMap, "performanceTracking");
}

base::type::fstream_t* TypedConfigurations::fileStream(Level level) {
  return getConfigByRef<base::FileStreamPtr>(level, &m_fileStreamMap, "fileStream").get();
}

std::size_t TypedConfigurations::maxLogFileSize(Level level) {
  return getConfigByVal<std::size_t>(level, &m_maxLogFileSizeMap, "maxLogFileSize");
}

std::size_t TypedConfigurations::logFlushThreshold(Level level) {
  return getConfigByVal<std::size_t>(level, &m_logFlushThresholdMap, "logFlushThreshold");
}

void TypedConfigurations::build(Configurations* configurations) {
  base::threading::ScopedLock scopedLock(lock());
  auto getBool = [] (std::string boolStr) -> bool {  // Pass by value for trimming
    base::utils::Str::trim(boolStr);
    return (boolStr == "TRUE" || boolStr == "true" || boolStr == "1");
  };
  std::vector<Configuration*> withFileSizeLimit;
  for (Configurations::const_iterator it = configurations->begin(); it != configurations->end(); ++it)
{
    Configuration* conf = *it;
    // We cannot use switch on strong enums because Intel C++ dont support them yet
    if (conf->configurationType() == ConfigurationType::Enabled) {
      setValue(conf->level(), getBool(conf->value()), &m_enabledMap);
    } else if (conf->configurationType() == ConfigurationType::ToFile) {
      setValue(conf->level(), getBool(conf->value()), &m_toFileMap);
    } else if (conf->configurationType() == ConfigurationType::ToStandardOutput) {
      setValue(conf->level(), getBool(conf->value()), &m_toStandardOutputMap);
    } else if (conf->configurationType() == ConfigurationType::Filename) {
      // We do not yet configure filename but we will configure in another
      // loop. This is because if file cannot be created, we will force ToFile
      // to be false. Because configuring logger is not necessarily performance
      // sensitive operation, we can live with another loop; (by the way this loop
      // is not very heavy either)
    } else if (conf->configurationType() == ConfigurationType::Format) {
      setValue(conf->level(), base::LogFormat(conf->level(),
                                              base::type::string_t(conf->value().begin(),
conf->value().end())), &m_logFormatMap);
    } else if (conf->configurationType() == ConfigurationType::SubsecondPrecision) {
      setValue(Level::Global,
               base::SubsecondPrecision(static_cast<int>(getULong(conf->value()))),
&m_subsecondPrecisionMap);
    } else if (conf->configurationType() == ConfigurationType::PerformanceTracking) {
      setValue(Level::Global, getBool(conf->value()), &m_performanceTrackingMap);
    } else if (conf->configurationType() == ConfigurationType::MaxLogFileSize) {
      auto v = getULong(conf->value());
      setValue(conf->level(), static_cast<std::size_t>(v), &m_maxLogFileSizeMap);
      if (v != 0) {
        withFileSizeLimit.push_back(conf);
      }
    } else if (conf->configurationType() == ConfigurationType::LogFlushThreshold) {
      setValue(conf->level(), static_cast<std::size_t>(getULong(conf->value())),
&m_logFlushThresholdMap);
    }
  }
  // As mentioned earlier, we will now set filename configuration in separate loop to deal with
non-existent files
  for (Configurations::const_iterator it = configurations->begin(); it != configurations->end(); ++it)
{
    Configuration* conf = *it;
    if (conf->configurationType() == ConfigurationType::Filename) {
      insertFile(conf->level(), conf->value());
    }
  }
  for (std::vector<Configuration*>::iterator conf = withFileSizeLimit.begin();
       conf != withFileSizeLimit.end(); ++conf) {
    // This is not unsafe as mutex is locked in currect scope
    unsafeValidateFileRolling((*conf)->level(), base::defaultPreRollOutCallback);
  }
}

unsigned long TypedConfigurations::getULong(std::string confVal) {
  bool valid = true;
  base::utils::Str::trim(confVal);
  valid = !confVal.empty() && std::find_if(confVal.begin(), confVal.end(),
  [](char c) {
    return !base::utils::Str::isDigit(c);
  }) == confVal.end();
  if (!valid) {
    valid = false;
    ELPP_ASSERT(valid, "Configuration value not a valid integer [" « confVal « "]");
    return 0;
  }
  return atol(confVal.c_str());
}

std::string TypedConfigurations::resolveFilename(const std::string& filename) {
  std::string resultingFilename = filename;
  std::size_t dateIndex = std::string::npos;
  std::string dateTimeFormatSpecifierStr =
std::string(base::consts::kDateTimeFormatSpecifierForFilename);
  if ((dateIndex = resultingFilename.find(dateTimeFormatSpecifierStr.c_str())) != std::string::npos) {
    while (dateIndex > 0 && resultingFilename[dateIndex - 1] == base::consts::kFormatSpecifierChar) {
      dateIndex = resultingFilename.find(dateTimeFormatSpecifierStr.c_str(), dateIndex + 1);
    }
    if (dateIndex != std::string::npos) {
      const char* ptr = resultingFilename.c_str() + dateIndex;
      // Goto end of specifier
      ptr += dateTimeFormatSpecifierStr.size();
      std::string fmt;
      if ((resultingFilename.size() > dateIndex) && (ptr[0] == '{')) {
        // User has provided format for date/time
        ++ptr;
        int count = 1;  // Start by 1 in order to remove starting brace
        std::stringstream ss;
        for (; *ptr; ++ptr, ++count) {
          if (*ptr == '}') {
            ++count;  // In order to remove ending brace
            break;
          }
          ss « *ptr;
        }
        resultingFilename.erase(dateIndex + dateTimeFormatSpecifierStr.size(), count);
        fmt = ss.str();
      } else {
        fmt = std::string(base::consts::kDefaultDateTimeFormatInFilename);
      }
      base::SubsecondPrecision ssPrec(3);
      std::string now = base::utils::DateTime::getDateTime(fmt.c_str(), &ssPrec);
      base::utils::Str::replaceAll(now, '/', '-'); // Replace path element since we are dealing with
filename
      base::utils::Str::replaceAll(resultingFilename, dateTimeFormatSpecifierStr, now);
    }
  }
  return resultingFilename;
}

void TypedConfigurations::insertFile(Level level, const std::string& fullFilename) {
  std::string resolvedFilename = resolveFilename(fullFilename);
  if (resolvedFilename.empty()) {
    std::cerr « "Could not load empty file for logging, please re-check your configurations for level
["
              « LevelHelper::convertToString(level) « "]";
  }
  std::string filePath = base::utils::File::extractPathFromFilename(resolvedFilename,
base::consts::kFilePathSeparator);
  if (filePath.size() < resolvedFilename.size()) {
    base::utils::File::createPath(filePath);
  }
  auto create = [&](Level level) {
    base::LogStreamsReferenceMap::iterator filestreamIter =
m_logStreamsReference->find(resolvedFilename);
    base::type::fstream_t* fs = nullptr;
    if (filestreamIter == m_logStreamsReference->end()) {
      // We need a completely new stream, nothing to share with
      fs = base::utils::File::newFileStream(resolvedFilename);
      m_filenameMap.insert(std::make_pair(level, resolvedFilename));
      m_fileStreamMap.insert(std::make_pair(level, base::FileStreamPtr(fs)));
      m_logStreamsReference->insert(std::make_pair(resolvedFilename,
base::FileStreamPtr(m_fileStreamMap.at(level))));
    } else {
      // Woops! we have an existing one, share it!
      m_filenameMap.insert(std::make_pair(level, filestreamIter->first));
      m_fileStreamMap.insert(std::make_pair(level, base::FileStreamPtr(filestreamIter->second)));
      fs = filestreamIter->second.get();
    }
    if (fs == nullptr) {
      // We display bad file error from newFileStream()
      ELPP_INTERNAL_ERROR("Setting [TO_FILE] of ["
                          « LevelHelper::convertToString(level) « "] to FALSE", false);
      setValue(level, false, &m_toFileMap);
    }
  };
  // If we dont have file conf for any level, create it for Level::Global first
  // otherwise create for specified level
  create(m_filenameMap.empty() && m_fileStreamMap.empty() ? Level::Global : level);
}

bool TypedConfigurations::unsafeValidateFileRolling(Level level, const PreRollOutCallback&
preRollOutCallback) {
  base::type::fstream_t* fs = unsafeGetConfigByRef(level, &m_fileStreamMap, "fileStream").get();
  if (fs == nullptr) {
    return true;
  }
  std::size_t maxLogFileSize = unsafeGetConfigByVal(level, &m_maxLogFileSizeMap, "maxLogFileSize");
  std::size_t currFileSize = base::utils::File::getSizeOfFile(fs);
  if (maxLogFileSize != 0 && currFileSize >= maxLogFileSize) {
    std::string fname = unsafeGetConfigByRef(level, &m_filenameMap, "filename");
    ELPP_INTERNAL_INFO(1, "Truncating log file [" « fname « "] as a result of configurations for level
["
                       « LevelHelper::convertToString(level) « "]");
    fs->close();
    preRollOutCallback(fname.c_str(), currFileSize);
    fs->open(fname, std::fstream::out | std::fstream::trunc);
    return true;
  }
  return false;
}

// RegisteredHitCounters
```

```

bool RegisteredHitCounters::validateEveryN(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
  base::threading::ScopedLock scopedLock(lock());
  base::HitCounter* counter = get(filename, lineNumber);
  if (counter == nullptr) {
    registerNew(counter = new base::HitCounter(filename, lineNumber));
  }
  counter->validateHitCounts(n);
  bool result = (n >= 1 && counter->hitCounts() != 0 && counter->hitCounts() % n == 0);
  return result;
}

bool RegisteredHitCounters::validateAfterN(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
  base::threading::ScopedLock scopedLock(lock());
  base::HitCounter* counter = get(filename, lineNumber);
  if (counter == nullptr) {
    registerNew(counter = new base::HitCounter(filename, lineNumber));
  }
  // Do not use validateHitCounts here since we do not want to reset counter here
  // Note the >= instead of > because we are incrementing
  // after this check
  if (counter->hitCounts() >= n)
    return true;
  counter->increment();
  return false;
}

bool RegisteredHitCounters::validateNTimes(const char* filename, base::type::LineNumber lineNumber,
std::size_t n) {
  base::threading::ScopedLock scopedLock(lock());
  base::HitCounter* counter = get(filename, lineNumber);
  if (counter == nullptr) {
    registerNew(counter = new base::HitCounter(filename, lineNumber));
  }
  counter->increment();
  // Do not use validateHitCounts here since we do not want to reset counter here
  if (counter->hitCounts() <= n)
    return true;
  return false;
}

// RegisteredLoggers

RegisteredLoggers::RegisteredLoggers(const LogBuilderPtr& defaultLogBuilder) :
  m_defaultLogBuilder(defaultLogBuilder) {
  m_defaultConfigurations.setToDefault();
  m_logStreamsReference = std::make_shared<base::LogStreamsReferenceMap>();
}

Logger* RegisteredLoggers::get(const std::string& id, bool forceCreation) {
  base::threading::ScopedLock scopedLock(lock());
  Logger* logger_ = base::utils::Registry<Logger, std::string>::get(id);
  if (logger_ == nullptr && forceCreation) {
    bool validId = Logger::isValidId(id);
    if (!validId) {
      ELPP_ASSERT(validId, "Invalid logger ID [" « id « "]. Not registering this logger.");
      return nullptr;
    }
    logger_ = new Logger(id, m_defaultConfigurations, m_logStreamsReference);
    logger_->m_logBuilder = m_defaultLogBuilder;
    registerNew(id, logger_);
    LoggerRegistrationCallback* callback = nullptr;
    for (const std::pair<std::string, base::type::LoggerRegistrationCallbackPtr>& h
         : m_loggerRegistrationCallbacks) {
      callback = h.second.get();
      if (callback != nullptr && callback->enabled()) {
        callback->handle(logger_);
      }
    }
  }
  return logger_;
}

bool RegisteredLoggers::remove(const std::string& id) {
  if (id == base::consts::kDefaultLoggerId) {
    return false;
  }
  // get has internal lock
  Logger* logger = base::utils::Registry<Logger, std::string>::get(id);
  if (logger != nullptr) {
    // unregister has internal lock
    unregister(logger);
  }
  return true;
}
```

```

void RegisteredLoggers::unsafeFlushAll(void) {
  ELPP_INTERNAL_INFO(1, "Flushing all log files");
  for (base::LogStreamsReferenceMap::iterator it = m_logStreamsReference->begin();
       it != m_logStreamsReference->end(); ++it) {
    if (it->second.get() == nullptr) continue;
    it->second->flush();
  }
}

// VRegistry

VRegistry::VRegistry(base::type::VerboseLevel level, base::type::EnumType* pFlags) : m_level(level),
m_pFlags(pFlags) {
}

void VRegistry::setLevel(base::type::VerboseLevel level) {
  base::threading::ScopedLock scopedLock(lock());
  if (level > 9)
    m_level = base::consts::kMaxVerboseLevel;
  else
    m_level = level;
}

void VRegistry::setModules(const char* modules) {
  base::threading::ScopedLock scopedLock(lock());
  auto addSuffix = [](std::stringstream& ss, const char* sfx, const char* prev) {
    if (prev != nullptr && base::utils::Str::endsWith(ss.str(), std::string(prev))) {
      std::string chr(ss.str().substr(0, ss.str().size() - strlen(prev)));
      ss.str(std::string(""));
      ss « chr;
    }
    if (base::utils::Str::endsWith(ss.str(), std::string(sfx))) {
      std::string chr(ss.str().substr(0, ss.str().size() - strlen(sfx)));
      ss.str(std::string(""));
      ss « chr;
    }
    ss « sfx;
  };
  auto insert = [&](std::stringstream& ss, base::type::VerboseLevel level) {
    if (!base::utils::hasFlag(LoggingFlag::DisableVModulesExtensions, *m_pFlags)) {
      addSuffix(ss, ".h", nullptr);
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".c", ".h");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".cpp", ".c");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".cc", ".cpp");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".cxx", ".cc");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".-inl.h", ".cxx");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".hxx", ".-inl.h");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".hpp", ".hxx");
      m_modules.insert(std::make_pair(ss.str(), level));
      addSuffix(ss, ".hh", ".hpp");
    }
    m_modules.insert(std::make_pair(ss.str(), level));
  };
  bool isMod = true;
  bool isLevel = false;
  std::stringstream ss;
  int level = -1;
  for (; *modules; ++modules) {
    switch (*modules) {
    case '=':
      isLevel = true;
      isMod = false;
      break;
    case ',':
      isLevel = false;
      isMod = true;
      if (!ss.str().empty() && level != -1) {
        insert(ss, static_cast<base::type::VerboseLevel>(level));
        ss.str(std::string(""));
        level = -1;
      }
      break;
    default:
      if (isMod) {
        ss « *modules;
      } else if (isLevel) {
        if (isdigit(*modules)) {
          level = static_cast<base::type::VerboseLevel>(*modules) - 48;
        }
      }
      break;
    }
  }
  if (!ss.str().empty() && level != -1) {
    insert(ss, static_cast<base::type::VerboseLevel>(level));
  }
}

bool VRegistry::allowed(base::type::VerboseLevel vlevel, const char* file) {
  base::threading::ScopedLock scopedLock(lock());
  if (m_modules.empty() || file == nullptr) {
    return vlevel <= m_level;
  } else {
    char baseFilename[base::consts::kSourceFilenameMaxLength] = "";
    base::utils::File::buildBaseFilename(file, baseFilename);
    std::unordered_map<std::string, base::type::VerboseLevel>::iterator it = m_modules.begin();
    for (; it != m_modules.end(); ++it) {
      if (base::utils::Str::wildCardMatch(baseFilename, it->first.c_str())) {
        return vlevel <= it->second;
      }
    }
    if (base::utils::hasFlag(LoggingFlag::AllowVerboseIfModuleNotSpecified, *m_pFlags)) {
      return true;
    }
    return false;
  }
}

void VRegistry::setFromArgs(const base::utils::CommandLineArgs* commandLineArgs) {
  if (commandLineArgs->hasParam("-v") || commandLineArgs->hasParam("--verbose") ||
      commandLineArgs->hasParam("-V") || commandLineArgs->hasParam("--VERBOSE")) {
    setLevel(base::consts::kMaxVerboseLevel);
  } else if (commandLineArgs->hasParamWithValue("--v")) {
    setLevel(static_cast<base::type::VerboseLevel>(atoi(commandLineArgs->getParamValue("--v"))));
  } else if (commandLineArgs->hasParamWithValue("--V")) {
    setLevel(static_cast<base::type::VerboseLevel>(atoi(commandLineArgs->getParamValue("--V"))));
  } else if ((commandLineArgs->hasParamWithValue("-vmodule")) && vModulesEnabled()) {
    setModules(commandLineArgs->getParamValue("-vmodule"));
  } else if (commandLineArgs->hasParamWithValue("-VMODULE") && vModulesEnabled()) {
    setModules(commandLineArgs->getParamValue("-VMODULE"));
  }
}

#if !defined(ELPP_DEFAULT_LOGGING_FLAGS)
#   define ELPP_DEFAULT_LOGGING_FLAGS 0x0
#endif // !defined(ELPP_DEFAULT_LOGGING_FLAGS)
// Storage
#if ELPP_ASYNC_LOGGING
Storage::Storage(const LogBuilderPtr& defaultLogBuilder, base::IWorker* asyncDispatchWorker) :
#else
Storage::Storage(const LogBuilderPtr& defaultLogBuilder) :
#endif  // ELPP_ASYNC_LOGGING
  m_registeredHitCounters(new base::RegisteredHitCounters()),
  m_registeredLoggers(new base::RegisteredLoggers(defaultLogBuilder)),
  m_flags(ELPP_DEFAULT_LOGGING_FLAGS),
  m_vRegistry(new base::VRegistry(0, &m_flags)),

#if ELPP_ASYNC_LOGGING
  m_asyncLogQueue(new base::AsyncLogQueue()),
  m_asyncDispatchWorker(asyncDispatchWorker),
#endif  // ELPP_ASYNC_LOGGING

  m_preRollOutCallback(base::defaultPreRollOutCallback) {
  // Register default logger
  m_registeredLoggers->get(std::string(base::consts::kDefaultLoggerId));
  // We register default logger anyway (worse case it's not going to register) just in case
  m_registeredLoggers->get("default");

#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  // Register performance logger and reconfigure format
  Logger* performanceLogger =
m_registeredLoggers->get(std::string(base::consts::kPerformanceLoggerId));
  m_registeredLoggers->get("performance");
  performanceLogger->configurations()->setGlobally(ConfigurationType::Format, std::string("%datetime
%level %msg"));
  performanceLogger->reconfigure();
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)

#if defined(ELPP_SYSLOG)
  // Register syslog logger and reconfigure format
  Logger* sysLogLogger = m_registeredLoggers->get(std::string(base::consts::kSysLogLoggerId));
  sysLogLogger->configurations()->setGlobally(ConfigurationType::Format, std::string("%level: %msg"));
  sysLogLogger->reconfigure();
#endif //  defined(ELPP_SYSLOG)
  addFlag(LoggingFlag::AllowVerboseIfModuleNotSpecified);
#if ELPP_ASYNC_LOGGING
```

```
  installLogDispatchCallback<base::AsyncLogDispatchCallback>(std::string("AsyncLogDispatchCallback"));
#else
installLogDispatchCallback<base::DefaultLogDispatchCallback>(std::string("DefaultLogDispatchCallback"));
#endif  // ELPP_ASYNC_LOGGING
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  installPerformanceTrackingCallback<base::DefaultPerformanceTrackingCallback>
  (std::string("DefaultPerformanceTrackingCallback"));
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  ELPP_INTERNAL_INFO(1, "Easylogging++ has been initialized");
#if ELPP_ASYNC_LOGGING
  m_asyncDispatchWorker->start();
#endif  // ELPP_ASYNC_LOGGING
}

Storage::~Storage(void) {
  ELPP_INTERNAL_INFO(4, "Destroying storage");
#if ELPP_ASYNC_LOGGING
  ELPP_INTERNAL_INFO(5, "Replacing log dispatch callback to synchronous");
uninstallLogDispatchCallback<base::AsyncLogDispatchCallback>(std::string("AsyncLogDispatchCallback"));
installLogDispatchCallback<base::DefaultLogDispatchCallback>(std::string("DefaultLogDispatchCallback"));
  ELPP_INTERNAL_INFO(5, "Destroying asyncDispatchWorker");
  base::utils::safeDelete(m_asyncDispatchWorker);
  ELPP_INTERNAL_INFO(5, "Destroying asyncLogQueue");
  base::utils::safeDelete(m_asyncLogQueue);
#endif  // ELPP_ASYNC_LOGGING
  ELPP_INTERNAL_INFO(5, "Destroying registeredHitCounters");
  base::utils::safeDelete(m_registeredHitCounters);
  ELPP_INTERNAL_INFO(5, "Destroying registeredLoggers");
  base::utils::safeDelete(m_registeredLoggers);
  ELPP_INTERNAL_INFO(5, "Destroying vRegistry");
  base::utils::safeDelete(m_vRegistry);
}

bool Storage::hasCustomFormatSpecifier(const char* formatSpecifier) {
  base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
  return std::find(m_customFormatSpecifiers.begin(), m_customFormatSpecifiers.end(),
                   formatSpecifier) != m_customFormatSpecifiers.end();
}

void Storage::installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier) {
  if (hasCustomFormatSpecifier(customFormatSpecifier.formatSpecifier())) {
    return;
  }
  base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
  m_customFormatSpecifiers.push_back(customFormatSpecifier);
}

bool Storage::uninstallCustomFormatSpecifier(const char* formatSpecifier) {
  base::threading::ScopedLock scopedLock(customFormatSpecifiersLock());
  std::vector<CustomFormatSpecifier>::iterator it = std::find(m_customFormatSpecifiers.begin(),
      m_customFormatSpecifiers.end(), formatSpecifier);
  if (it != m_customFormatSpecifiers.end() && strcmp(formatSpecifier, it->formatSpecifier()) == 0) {
    m_customFormatSpecifiers.erase(it);
    return true;
  }
  return false;
}

void Storage::setApplicationArguments(int argc, char** argv) {
  m_commandLineArgs.setArgs(argc, argv);
  m_vRegistry->setFromArgs(commandLineArgs());
  // default log file
#if !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
  if (m_commandLineArgs.hasParamWithValue(base::consts::kDefaultLogFileParam)) {
    Configurations c;
    c.setGlobally(ConfigurationType::Filename,
                  std::string(m_commandLineArgs.getParamValue(base::consts::kDefaultLogFileParam)));
    registeredLoggers()->setDefaultConfigurations(c);
    for (base::RegisteredLoggers::iterator it = registeredLoggers()->begin();
         it != registeredLoggers()->end(); ++it) {
      it->second->configure(c);
    }
  }
#endif  // !defined(ELPP_DISABLE_LOG_FILE_FROM_ARG)
#if defined(ELPP_LOGGING_FLAGS_FROM_ARG)
  if (m_commandLineArgs.hasParamWithValue(base::consts::kLoggingFlagsParam)) {
    int userInput = atoi(m_commandLineArgs.getParamValue(base::consts::kLoggingFlagsParam));
    if (ELPP_DEFAULT_LOGGING_FLAGS == 0x0) {
      m_flags = userInput;
    } else {
      base::utils::addFlag<base::type::EnumType>(userInput, &m_flags);
    }
  }
#endif  // defined(ELPP_LOGGING_FLAGS_FROM_ARG)
```

```
}

} // namespace base

// LogDispatchCallback
#if defined(ELPP_THREAD_SAFE)
void LogDispatchCallback::handle(const LogDispatchData* data) {
  base::threading::ScopedLock scopedLock(m_fileLocksMapLock);
  std::string filename = data->logMessage()->logger()->typedConfigurations()->filename(data->logMessage()->level());
  auto lock = m_fileLocks.find(filename);
  if (lock == m_fileLocks.end()) {
    m_fileLocks.emplace(std::make_pair(filename, std::unique_ptr<base::threading::Mutex>(new base::threading::Mutex)));
  }
}
#else
void LogDispatchCallback::handle(const LogDispatchData* /*data*/) {}
#endif

base::threading::Mutex& LogDispatchCallback::fileHandle(const LogDispatchData* data) {
  auto it = m_fileLocks.find(data->logMessage()->logger()->typedConfigurations()->filename(data->logMessage()->level()));
  return *(it->second.get());
}

namespace base {
// DefaultLogDispatchCallback

void DefaultLogDispatchCallback::handle(const LogDispatchData* data) {
#if defined(ELPP_THREAD_SAFE)
  LogDispatchCallback::handle(data);
  base::threading::ScopedLock scopedLock(fileHandle(data));
#endif
  m_data = data;
  dispatch(m_data->logMessage()->logger()->logBuilder()->build(m_data->logMessage(),
           m_data->dispatchAction() == base::DispatchAction::NormalLog));
}

void DefaultLogDispatchCallback::dispatch(base::type::string_t&& logLine) {
  if (m_data->dispatchAction() == base::DispatchAction::NormalLog) {
    if (m_data->logMessage()->logger()->m_typedConfigurations->toFile(m_data->logMessage()->level())) {
      base::type::fstream_t* fs = m_data->logMessage()->logger()->m_typedConfigurations->fileStream(
                                    m_data->logMessage()->level());
      if (fs != nullptr) {
        fs->write(logLine.c_str(), logLine.size());
        if (fs->fail()) {
          ELPP_INTERNAL_ERROR("Unable to write log to file ["
                              « m_data->logMessage()->logger()->m_typedConfigurations->filename(m_data->logMessage()->level()) « "].\n"
                              « "Few possible reasons (could be something else):\n" « "      * Permission denied\n"
                              « "      * Disk full\n" « "      * Disk is not writable", true);
        } else {
          if (ELPP->hasFlag(LoggingFlag::ImmediateFlush)
              || (m_data->logMessage()->logger()->isFlushNeeded(m_data->logMessage()->level()))) {
            m_data->logMessage()->logger()->flush(m_data->logMessage()->level(), fs);
          }
        }
      } else {
        ELPP_INTERNAL_ERROR("Log file for [" « LevelHelper::convertToString(m_data->logMessage()->level()) « "] "
                            « "has not been configured but [TO_FILE] is configured to TRUE. [Logger ID: "
                            « m_data->logMessage()->logger()->id() « "]", false);
      }
    }
    if (m_data->logMessage()->logger()->m_typedConfigurations->toStandardOutput(m_data->logMessage()->level())) {
      if (ELPP->hasFlag(LoggingFlag::ColoredTerminalOutput))
        m_data->logMessage()->logger()->logBuilder()->convertToColoredOutput(&logLine, m_data->logMessage()->level());
      ELPP_COUT « ELPP_COUT_LINE(logLine);
    }
  }
#if defined(ELPP_SYSLOG)
  else if (m_data->dispatchAction() == base::DispatchAction::SysLog) {
    // Determine syslog priority
    int sysLogPriority = 0;
    if (m_data->logMessage()->level() == Level::Fatal)
      sysLogPriority = LOG_EMERG;
    else if (m_data->logMessage()->level() == Level::Error)
      sysLogPriority = LOG_ERR;
    else if (m_data->logMessage()->level() == Level::Warning)
```

```
      sysLogPriority = LOG_WARNING;
    else if (m_data->logMessage()->level() == Level::Info)
      sysLogPriority = LOG_INFO;
    else if (m_data->logMessage()->level() == Level::Debug)
      sysLogPriority = LOG_DEBUG;
    else
      sysLogPriority = LOG_NOTICE;
#  if defined(ELPP_UNICODE)
    char* line = base::utils::Str::wcharPtrToCharPtr(logLine.c_str());
    syslog(sysLogPriority, "%s", line);
    free(line);
#  else
    syslog(sysLogPriority, "%s", logLine.c_str());
#  endif
  }
#endif  // defined(ELPP_SYSLOG)
}

#if ELPP_ASYNC_LOGGING

// AsyncLogDispatchCallback

void AsyncLogDispatchCallback::handle(const LogDispatchData* data) {
  base::type::string_t logLine = data->logMessage()->logger()->logBuilder()->build(data->logMessage(),
                                 data->dispatchAction() == base::DispatchAction::NormalLog);
  if (data->dispatchAction() == base::DispatchAction::NormalLog
      &&
data->logMessage()->logger()->typedConfigurations()->toStandardOutput(data->logMessage()->level())) {
    if (ELPP->hasFlag(LoggingFlag::ColoredTerminalOutput))
      data->logMessage()->logger()->logBuilder()->convertToColoredOutput(&logLine,
data->logMessage()->level());
    ELPP_COUT « ELPP_COUT_LINE(logLine);
  }
  // Save resources and only queue if we want to write to file otherwise just ignore handler
  if (data->logMessage()->logger()->typedConfigurations()->toFile(data->logMessage()->level())) {
    ELPP->asyncLogQueue()->push(AsyncLogItem(*(data->logMessage()), *data, logLine));
  }
}

// AsyncDispatchWorker
AsyncDispatchWorker::AsyncDispatchWorker() {
  setContinueRunning(false);
}

AsyncDispatchWorker::~AsyncDispatchWorker() {
  setContinueRunning(false);
  ELPP_INTERNAL_INFO(6, "Stopping dispatch worker - Cleaning log queue");
  clean();
  ELPP_INTERNAL_INFO(6, "Log queue cleaned");
}

bool AsyncDispatchWorker::clean(void) {
  std::mutex m;
  std::unique_lock<std::mutex> lk(m);
  cv.wait(lk, [] { return !ELPP->asyncLogQueue()->empty(); });
  emptyQueue();
  lk.unlock();
  cv.notify_one();
  return ELPP->asyncLogQueue()->empty();
}

void AsyncDispatchWorker::emptyQueue(void) {
  while (!ELPP->asyncLogQueue()->empty()) {
    AsyncLogItem data = ELPP->asyncLogQueue()->next();
    handle(&data);
    base::threading::msleep(100);
  }
}

void AsyncDispatchWorker::start(void) {
  base::threading::msleep(5000); // 5s (why?)
  setContinueRunning(true);
  std::thread t1(&AsyncDispatchWorker::run, this);
  t1.join();
}

void AsyncDispatchWorker::handle(AsyncLogItem* logItem) {
  LogDispatchData* data = logItem->data();
  LogMessage* logMessage = logItem->logMessage();
  Logger* logger = logMessage->logger();
  base::TypedConfigurations* conf = logger->typedConfigurations();
  base::type::string_t logLine = logItem->logLine();
  if (data->dispatchAction() == base::DispatchAction::NormalLog) {
    if (conf->toFile(logMessage->level())) {
      base::type::fstream_t* fs = conf->fileStream(logMessage->level());
      if (fs != nullptr) {
        fs->write(logLine.c_str(), logLine.size());
        if (fs->fail()) {
          ELPP_INTERNAL_ERROR("Unable to write log to file ["
                              « conf->filename(logMessage->level()) « "].\n"
                              « "Few possible reasons (could be something else):\n" « "      *
Permission denied\n"
                              « "      * Disk full\n" « "      * Disk is not writable", true);
        } else {
          if (ELPP->hasFlag(LoggingFlag::ImmediateFlush) ||
(logger->isFlushNeeded(logMessage->level()))) {
            logger->flush(logMessage->level(), fs);
          }
        }
      } else {
        ELPP_INTERNAL_ERROR("Log file for [" « LevelHelper::convertToString(logMessage->level()) « "]
"
                            « "has not been configured but [TO_FILE] is configured to TRUE. [Logger
ID: " « logger->id() « "]", false);
      }
    }
  }
#  if defined(ELPP_SYSLOG)
  else if (data->dispatchAction() == base::DispatchAction::SysLog) {
    // Determine syslog priority
    int sysLogPriority = 0;
    if (logMessage->level() == Level::Fatal)
      sysLogPriority = LOG_EMERG;
    else if (logMessage->level() == Level::Error)
      sysLogPriority = LOG_ERR;
    else if (logMessage->level() == Level::Warning)
      sysLogPriority = LOG_WARNING;
    else if (logMessage->level() == Level::Info)
      sysLogPriority = LOG_INFO;
    else if (logMessage->level() == Level::Debug)
      sysLogPriority = LOG_DEBUG;
    else
      sysLogPriority = LOG_NOTICE;
#      if defined(ELPP_UNICODE)
    char* line = base::utils::Str::wcharPtrToCharPtr(logLine.c_str());
    syslog(sysLogPriority, "%s", line);
    free(line);
#      else
    syslog(sysLogPriority, "%s", logLine.c_str());
#      endif
  }
#  endif  // defined(ELPP_SYSLOG)
}

void AsyncDispatchWorker::run(void) {
  while (continueRunning()) {
    emptyQueue();
    base::threading::msleep(10); // 10ms
  }
}
#endif  // ELPP_ASYNC_LOGGING

// DefaultLogBuilder

base::type::string_t DefaultLogBuilder::build(const LogMessage* logMessage, bool appendNewLine) const
{
  base::TypedConfigurations* tc = logMessage->logger()->typedConfigurations();
  const base::LogFormat* logFormat = &tc->logFormat(logMessage->level());
  base::type::string_t logLine = logFormat->format();
  char buff[base::consts::kSourceFilenameMaxLength + base::consts::kSourceLineMaxLength] = "";
  const char* bufLim = buff + sizeof(buff);
  if (logFormat->hasFlag(base::FormatFlags::AppName)) {
    // App name
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kAppNameFormatSpecifier,
        logMessage->logger()->parentApplicationName());
  }
  if (logFormat->hasFlag(base::FormatFlags::ThreadId)) {
    // Thread ID
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kThreadIdFormatSpecifier,
        ELPP->getThreadName(base::threading::getCurrentThreadId()));
  }
  if (logFormat->hasFlag(base::FormatFlags::DateTime)) {
    // DateTime
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kDateTimeFormatSpecifier,
        base::utils::DateTime::getDateTime(logFormat->dateTimeFormat().c_str(),
                                           &tc->subsecondPrecision(logMessage->level())));
  }
  if (logFormat->hasFlag(base::FormatFlags::Function)) {
    // Function
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFunctionFormatSpecifier,
logMessage->func());
  }
  if (logFormat->hasFlag(base::FormatFlags::File)) {
    // File
    base::utils::Str::clearBuff(buff, base::consts::kSourceFilenameMaxLength);
    base::utils::File::buildStrippedFilename(logMessage->file().c_str(), buff);
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFileFormatSpecifier,
std::string(buff));
  }
  if (logFormat->hasFlag(base::FormatFlags::FileBase)) {
    // FileBase
    base::utils::Str::clearBuff(buff, base::consts::kSourceFilenameMaxLength);
    base::utils::File::buildBaseFilename(logMessage->file(), buff);
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogFileBaseFormatSpecifier,
std::string(buff));
  }
  if (logFormat->hasFlag(base::FormatFlags::Line)) {
    // Line
    char* buf = base::utils::Str::clearBuff(buff, base::consts::kSourceLineMaxLength);
    buf = base::utils::Str::convertAndAddToBuff(logMessage->line(),
base::consts::kSourceLineMaxLength, buf, bufLim, false);
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogLineFormatSpecifier,
std::string(buff));
  }
  if (logFormat->hasFlag(base::FormatFlags::Location)) {
    // Location
    char* buf = base::utils::Str::clearBuff(buff,
                                            base::consts::kSourceFilenameMaxLength +
base::consts::kSourceLineMaxLength);
    base::utils::File::buildStrippedFilename(logMessage->file().c_str(), buff);
    buf = base::utils::Str::addToBuff(buff, buf, bufLim);
    buf = base::utils::Str::addToBuff(":", buf, bufLim);
    buf = base::utils::Str::convertAndAddToBuff(logMessage->line(),
base::consts::kSourceLineMaxLength, buf, bufLim,
          false);
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kLogLocationFormatSpecifier,
std::string(buff));
  }
  if (logMessage->level() == Level::Verbose && logFormat->hasFlag(base::FormatFlags::VerboseLevel)) {
    // Verbose level
    char* buf = base::utils::Str::clearBuff(buff, 1);
    buf = base::utils::Str::convertAndAddToBuff(logMessage->verboseLevel(), 1, buf, bufLim, false);
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kVerboseLevelFormatSpecifier,
std::string(buff));
  }
  if (logFormat->hasFlag(base::FormatFlags::LogMessage)) {
    // Log message
    base::utils::Str::replaceFirstWithEscape(logLine, base::consts::kMessageFormatSpecifier,
logMessage->message());
  }
#if !defined(ELPP_DISABLE_CUSTOM_FORMAT_SPECIFIERS)
  el::base::threading::ScopedLock lock_(ELPP->customFormatSpecifiersLock());
  ELPP_UNUSED(lock_);
  for (std::vector<CustomFormatSpecifier>::const_iterator it =
ELPP->customFormatSpecifiers()->begin();
       it != ELPP->customFormatSpecifiers()->end(); ++it) {
    std::string fs(it->formatSpecifier());
    base::type::string_t wcsFormatSpecifier(fs.begin(), fs.end());
    base::utils::Str::replaceFirstWithEscape(logLine, wcsFormatSpecifier, it->resolver()(logMessage));
  }
#endif  // !defined(ELPP_DISABLE_CUSTOM_FORMAT_SPECIFIERS)
  if (appendNewLine) logLine += ELPP_LITERAL("\n");
  return logLine;
}

// LogDispatcher

void LogDispatcher::dispatch(void) {
  if (m_proceed && m_dispatchAction == base::DispatchAction::None) {
    m_proceed = false;
  }
  if (!m_proceed) {
    return;
  }
#ifndef ELPP_NO_GLOBAL_LOCK
  // see https://github.com/muflihun/easyloggingpp/issues/580
  // global lock is turned on by default unless
  // ELPP_NO_GLOBAL_LOCK is defined
  base::threading::ScopedLock scopedLock(ELPP->lock());
#endif
  base::TypedConfigurations* tc = m_logMessage->logger()->m_typedConfigurations;
  if (ELPP->hasFlag(LoggingFlag::StrictLogFileSizeCheck)) {
    tc->validateFileRolling(m_logMessage->level(), ELPP->preRollOutCallback());
  }
  LogDispatchCallback* callback = nullptr;
  LogDispatchData data;
  for (const std::pair<std::string, base::type::LogDispatchCallbackPtr>& h
       : ELPP->m_logDispatchCallbacks) {
    callback = h.second.get();
    if (callback != nullptr && callback->enabled()) {
      data.setLogMessage(m_logMessage);
      data.setDispatchAction(m_dispatchAction);
      callback->handle(&data);
    }
  }
}

// MessageBuilder

void MessageBuilder::initialize(Logger* logger) {
  m_logger = logger;
  m_containerLogSeparator = ELPP->hasFlag(LoggingFlag::NewLineForContainer) ?
                            ELPP_LITERAL("\n    ") : ELPP_LITERAL(", ");
}

MessageBuilder& MessageBuilder::operator«(const wchar_t* msg) {
  if (msg == nullptr) {
    m_logger->stream() « base::consts::kNullPointer;
    return *this;
  }
#  if defined(ELPP_UNICODE)
  m_logger->stream() « msg;
#  else
  char* buff_ = base::utils::Str::wcharPtrToCharPtr(msg);
  m_logger->stream() « buff_;
  free(buff_);
#  endif
  if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {
    m_logger->stream() « " ";
  }
  return *this;
}

// Writer

Writer& Writer::construct(Logger* logger, bool needLock) {
  m_logger = logger;
  initializeLogger(logger->id(), false, needLock);
  m_messageBuilder.initialize(m_logger);
  return *this;
}

Writer& Writer::construct(int count, const char* loggerIds, ...) {
  if (ELPP->hasFlag(LoggingFlag::MultiLoggerSupport)) {
    va_list loggersList;
    va_start(loggersList, loggerIds);
    const char* id = loggerIds;
    m_loggerIds.reserve(count);
    for (int i = 0; i < count; ++i) {
      m_loggerIds.push_back(std::string(id));
      id = va_arg(loggersList, const char*);
    }
    va_end(loggersList);
    initializeLogger(m_loggerIds.at(0));
  } else {
    initializeLogger(std::string(loggerIds));
  }
  m_messageBuilder.initialize(m_logger);
  return *this;
}

void Writer::initializeLogger(const std::string& loggerId, bool lookup, bool needLock) {
  if (lookup) {
    m_logger = ELPP->registeredLoggers()->get(loggerId,
ELPP->hasFlag(LoggingFlag::CreateLoggerAutomatically));
  }
  if (m_logger == nullptr) {
    {
      if (!ELPP->registeredLoggers()->has(std::string(base::consts::kDefaultLoggerId))) {
        // Somehow default logger has been unregistered. Not good! Register again
        ELPP->registeredLoggers()->get(std::string(base::consts::kDefaultLoggerId));
      }
    }
    Writer(Level::Debug, m_file, m_line, m_func).construct(1, base::consts::kDefaultLoggerId)
        « "Logger [" « loggerId « "] is not registered yet!";
    m_proceed = false;
  } else {
    if (needLock) {
      m_logger->acquireLock();  // This should not be unlocked by checking m_proceed because
      // m_proceed can be changed by lines below
    }
    if (ELPP->hasFlag(LoggingFlag::HierarchicalLogging)) {
      m_proceed = m_level == Level::Verbose ? m_logger->enabled(m_level) :
                  LevelHelper::castToInt(m_level) >= LevelHelper::castToInt(ELPP->m_loggingLevel);
    } else {
      m_proceed = m_logger->enabled(m_level);
    }
  }
```

```
}

void Writer::processDispatch() {
#if ELPP_LOGGING_ENABLED
  if (ELPP->hasFlag(LoggingFlag::MultiLoggerSupport)) {
    bool firstDispatched = false;
    base::type::string_t logMessage;
    std::size_t i = 0;
    do {
      if (m_proceed) {
        if (firstDispatched) {
          m_logger->stream() « logMessage;
        } else {
          firstDispatched = true;
          if (m_loggerIds.size() > 1) {
            logMessage = m_logger->stream().str();
          }
        }
        triggerDispatch();
      } else if (m_logger != nullptr) {
        m_logger->stream().str(ELPP_LITERAL(""));
        m_logger->releaseLock();
      }
      if (i + 1 < m_loggerIds.size()) {
        initializeLogger(m_loggerIds.at(i + 1));
      }
    } while (++i < m_loggerIds.size());
  } else {
    if (m_proceed) {
      triggerDispatch();
    } else if (m_logger != nullptr) {
      m_logger->stream().str(ELPP_LITERAL(""));
      m_logger->releaseLock();
    }
  }
#else
  if (m_logger != nullptr) {
    m_logger->stream().str(ELPP_LITERAL(""));
    m_logger->releaseLock();
  }
#endif // ELPP_LOGGING_ENABLED
}

void Writer::triggerDispatch(void) {
  try {
      if (m_proceed) {
        if (m_msg == nullptr) {
          LogMessage msg(m_level, m_file, m_line, m_func, m_verboseLevel,
                         m_logger);
          base::LogDispatcher(m_proceed, &msg, m_dispatchAction).dispatch();
        } else {
          base::LogDispatcher(m_proceed, m_msg, m_dispatchAction).dispatch();
        }
      }
      if (m_logger != nullptr) {
        m_logger->stream().str(ELPP_LITERAL(""));
        m_logger->releaseLock();
      }
      if (m_proceed && m_level == Level::Fatal
          && !ELPP->hasFlag(LoggingFlag::DisableApplicationAbortOnFatalLog)) {
        base::Writer(Level::Warning, m_file, m_line, m_func).construct(1,
base::consts::kDefaultLoggerId)
            « "Aborting application. Reason: Fatal log at [" « m_file « ":" « m_line « "]";
        std::stringstream reasonStream;
        reasonStream « "Fatal log at [" « m_file « ":" « m_line « "]"
                     « " If you wish to disable 'abort on fatal log' please use "
                     « "el::Loggers::addFlag(el::LoggingFlag::DisableApplicationAbortOnFatalLog)";
        base::utils::abort(1, reasonStream.str());
      }
      m_proceed = false;
      }
    catch(std::exception & ex){
        // Extremely low memory situation; don't let exception be unhandled.
    }
}

// PErrorWriter

PErrorWriter::~PErrorWriter(void) {
  if (m_proceed) {
#if ELPP_COMPILER_MSVC
    char buff[256];
    strerror_s(buff, 256, errno);
    m_logger->stream() « ": " « buff « " [" « errno « "]";
#else
    m_logger->stream() « ": " « strerror(errno) « " [" « errno « "]";
#endif
```

```
  }
}

// PerformanceTracker

#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)

PerformanceTracker::PerformanceTracker(const std::string& blockName,
                                       base::TimestampUnit timestampUnit,
                                       const std::string& loggerId,
                                       bool scopedLog, Level level) :
  m_blockName(blockName), m_timestampUnit(timestampUnit), m_loggerId(loggerId),
m_scopedLog(scopedLog),
  m_level(level), m_hasChecked(false), m_lastCheckpointId(std::string()), m_enabled(false) {
#if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
  // We store it locally so that if user happen to change configuration by the end of scope
  // or before calling checkpoint, we still depend on state of configuration at time of construction
  el::Logger* loggerPtr = ELPP->registeredLoggers()->get(loggerId, false);
  m_enabled = loggerPtr != nullptr && loggerPtr->m_typedConfigurations->performanceTracking(m_level);
  if (m_enabled) {
    base::utils::DateTime::gettimeofday(&m_startTime);
  }
#endif  // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
}

PerformanceTracker::~PerformanceTracker(void) {
#if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
  if (m_enabled) {
    base::threading::ScopedLock scopedLock(lock());
    if (m_scopedLog) {
      base::utils::DateTime::gettimeofday(&m_endTime);
      base::type::string_t formattedTime = getFormattedTimeTaken();
      PerformanceTrackingData data(PerformanceTrackingData::DataType::Complete);
      data.init(this);
      data.m_formattedTimeTaken = formattedTime;
      PerformanceTrackingCallback* callback = nullptr;
      for (const std::pair<std::string, base::type::PerformanceTrackingCallbackPtr>& h
           : ELPP->m_performanceTrackingCallbacks) {
        callback = h.second.get();
        if (callback != nullptr && callback->enabled()) {
          callback->handle(&data);
        }
      }
    }
  }
#endif  // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING)
}

void PerformanceTracker::checkpoint(const std::string& id, const char* file, base::type::LineNumber
line,
                                    const char* func) {
#if !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
  if (m_enabled) {
    base::threading::ScopedLock scopedLock(lock());
    base::utils::DateTime::gettimeofday(&m_endTime);
    base::type::string_t formattedTime = m_hasChecked ? getFormattedTimeTaken(m_lastCheckpointTime) :
ELPP_LITERAL("");
    PerformanceTrackingData data(PerformanceTrackingData::DataType::Checkpoint);
    data.init(this);
    data.m_checkpointId = id;
    data.m_file = file;
    data.m_line = line;
    data.m_func = func;
    data.m_formattedTimeTaken = formattedTime;
    PerformanceTrackingCallback* callback = nullptr;
    for (const std::pair<std::string, base::type::PerformanceTrackingCallbackPtr>& h
         : ELPP->m_performanceTrackingCallbacks) {
      callback = h.second.get();
      if (callback != nullptr && callback->enabled()) {
        callback->handle(&data);
      }
    }
    base::utils::DateTime::gettimeofday(&m_lastCheckpointTime);
    m_hasChecked = true;
    m_lastCheckpointId = id;
  }
#endif  // !defined(ELPP_DISABLE_PERFORMANCE_TRACKING) && ELPP_LOGGING_ENABLED
  ELPP_UNUSED(id);
  ELPP_UNUSED(file);
  ELPP_UNUSED(line);
  ELPP_UNUSED(func);
}

const base::type::string_t PerformanceTracker::getFormattedTimeTaken(struct timeval startTime) const {
  if (ELPP->hasFlag(LoggingFlag::FixedTimeFormat)) {
    base::type::stringstream_t ss;
    ss « base::utils::DateTime::getTimeDifference(m_endTime,
        startTime, m_timestampUnit) « " " « base::consts::kTimeFormats[static_cast<base::type::EnumType>
            (m_timestampUnit)].unit;
    return ss.str();
  }
  return base::utils::DateTime::formatTime(base::utils::DateTime::getTimeDifference(m_endTime,
         startTime, m_timestampUnit), m_timestampUnit);
}

#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)

namespace debug {
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)

// StackTrace

StackTrace::StackTraceEntry::StackTraceEntry(std::size_t index, const std::string& loc, const std::string& demang,
    const std::string& hex,
    const std::string& addr) :
  m_index(index),
  m_location(loc),
  m_demangled(demang),
  m_hex(hex),
  m_addr(addr) {
}

std::ostream& operator«(std::ostream& ss, const StackTrace::StackTraceEntry& si) {
  ss « "[" « si.m_index « "] " « si.m_location « (si.m_hex.empty() ? "" : "+") « si.m_hex « " " « si.m_addr «
     (si.m_demangled.empty() ? "" : ":") « si.m_demangled;
  return ss;
}

std::ostream& operator«(std::ostream& os, const StackTrace& st) {
  std::vector<StackTrace::StackTraceEntry>::const_iterator it = st.m_stack.begin();
  while (it != st.m_stack.end()) {
    os « "    " « *it++ « "\n";
  }
  return os;
}

void StackTrace::generateNew(void) {
#ifdef HAVE_EXECINFO
  m_stack.clear();
  void* stack[kMaxStack];
  unsigned int size = backtrace(stack, kMaxStack);
  char** strings = backtrace_symbols(stack, size);
  if (size > kStackStart) {  // Skip StackTrace c'tor and generateNew
    for (std::size_t i = kStackStart; i < size; ++i) {
      std::string mangName;
      std::string location;
      std::string hex;
      std::string addr;

      // entry: 2   crash.cpp.bin                       0x0000000101552be5 _ZN2el4base5debug10StackTraceC1Ev + 21
      const std::string line(strings[i]);
      auto p = line.find("_");
      if (p != std::string::npos) {
        mangName = line.substr(p);
        mangName = mangName.substr(0, mangName.find(" +"));
      }
      p = line.find("0x");
      if (p != std::string::npos) {
        addr = line.substr(p);
        addr = addr.substr(0, addr.find("_"));
      }
      // Perform demangling if parsed properly
      if (!mangName.empty()) {
        int status = 0;
        char* demangName = abi::__cxa_demangle(mangName.data(), 0, 0, &status);
        // if demangling is successful, output the demangled function name
        if (status == 0) {
          // Success (see http://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.3/a01696.html)
          StackTraceEntry entry(i - 1, location, demangName, hex, addr);
          m_stack.push_back(entry);
        } else {
          // Not successful - we will use mangled name
          StackTraceEntry entry(i - 1, location, mangName, hex, addr);
          m_stack.push_back(entry);
        }
        free(demangName);
      } else {
        StackTraceEntry entry(i - 1, line);
        m_stack.push_back(entry);
      }
    }
  }
  free(strings);
#else
  ELPP_INTERNAL_INFO(1, "Stacktrace generation not supported for selected compiler");
#endif  // ELPP_STACKTRACE
}

// Static helper functions

static std::string crashReason(int sig) {
  std::stringstream ss;
  bool foundReason = false;
  for (int i = 0; i < base::consts::kCrashSignalsCount; ++i) {
    if (base::consts::kCrashSignals[i].numb == sig) {
      ss « "Application has crashed due to [" « base::consts::kCrashSignals[i].name « "] signal";
      if (ELPP->hasFlag(el::LoggingFlag::LogDetailedCrashReason)) {
        ss « std::endl «
           "    " « base::consts::kCrashSignals[i].brief « std::endl «
           "    " « base::consts::kCrashSignals[i].detail;
      }
      foundReason = true;
    }
  }
  if (!foundReason) {
    ss « "Application has crashed due to unknown signal [" « sig « "]";
  }
  return ss.str();
}
static void logCrashReason(int sig, bool stackTraceIfAvailable, Level level, const char* logger) {
  if (sig == SIGINT && ELPP->hasFlag(el::LoggingFlag::IgnoreSigInt)) {
    return;
  }
  std::stringstream ss;
  ss « "CRASH HANDLED; ";
  ss « crashReason(sig);
#if ELPP_STACKTRACE
  if (stackTraceIfAvailable) {
    ss « std::endl « "    ======= Backtrace: =========" « std::endl « base::debug::StackTrace();
  }
#else
  ELPP_UNUSED(stackTraceIfAvailable);
#endif  // ELPP_STACKTRACE
  ELPP_WRITE_LOG(el::base::Writer, level, base::DispatchAction::NormalLog, logger) « ss.str();
}

static inline void crashAbort(int sig) {
  base::utils::abort(sig, std::string());
}

static inline void defaultCrashHandler(int sig) {
  base::debug::logCrashReason(sig, true, Level::Fatal, base::consts::kDefaultLoggerId);
  base::debug::crashAbort(sig);
}

// CrashHandler

CrashHandler::CrashHandler(bool useDefault) {
  if (useDefault) {
    setHandler(defaultCrashHandler);
  }
}

void CrashHandler::setHandler(const Handler& cHandler) {
  m_handler = cHandler;
#if defined(ELPP_HANDLE_SIGABRT)
  int i = 0;  // SIGABRT is at base::consts::kCrashSignals[0]
#else
  int i = 1;
#endif  // defined(ELPP_HANDLE_SIGABRT)
  for (; i < base::consts::kCrashSignalsCount; ++i) {
    m_handler = signal(base::consts::kCrashSignals[i].numb, cHandler);
  }
}

#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
}  // namespace debug
} // namespace base

// el

// Helpers

#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)

void Helpers::crashAbort(int sig, const char* sourceFile, unsigned int long line) {
  std::stringstream ss;
  ss « base::debug::crashReason(sig).c_str();
  ss « " - [Called el::Helpers::crashAbort(" « sig « ")]";
  if (sourceFile != nullptr && strlen(sourceFile) > 0) {
    ss « " - Source: " « sourceFile;
    if (line > 0)
      ss « ":" « line;
    else
      ss « " (line number not specified)";
  }
  base::utils::abort(sig, ss.str());
}

void Helpers::logCrashReason(int sig, bool stackTraceIfAvailable, Level level, const char* logger) {
  el::base::debug::logCrashReason(sig, stackTraceIfAvailable, level, logger);
}

#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)

// Loggers

Logger* Loggers::getLogger(const std::string& identity, bool registerIfNotAvailable) {
  return ELPP->registeredLoggers()->get(identity, registerIfNotAvailable);
}

void Loggers::setDefaultLogBuilder(el::LogBuilderPtr& logBuilderPtr) {
  ELPP->registeredLoggers()->setDefaultLogBuilder(logBuilderPtr);
}

bool Loggers::unregisterLogger(const std::string& identity) {
  return ELPP->registeredLoggers()->remove(identity);
}

bool Loggers::hasLogger(const std::string& identity) {
  return ELPP->registeredLoggers()->has(identity);
}

Logger* Loggers::reconfigureLogger(Logger* logger, const Configurations& configurations) {
  if (!logger) return nullptr;
  logger->configure(configurations);
  return logger;
}

Logger* Loggers::reconfigureLogger(const std::string& identity, const Configurations& configurations)
{
  return Loggers::reconfigureLogger(Loggers::getLogger(identity), configurations);
}

Logger* Loggers::reconfigureLogger(const std::string& identity, ConfigurationType configurationType,
                                   const std::string& value) {
  Logger* logger = Loggers::getLogger(identity);
  if (logger == nullptr) {
    return nullptr;
  }
  logger->configurations()->set(Level::Global, configurationType, value);
  logger->reconfigure();
  return logger;
}

void Loggers::reconfigureAllLoggers(const Configurations& configurations) {
  for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->begin();
       it != ELPP->registeredLoggers()->end(); ++it) {
    Loggers::reconfigureLogger(it->second, configurations);
  }
}

void Loggers::reconfigureAllLoggers(Level level, ConfigurationType configurationType,
                                    const std::string& value) {
  for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->begin();
       it != ELPP->registeredLoggers()->end(); ++it) {
    Logger* logger = it->second;
    logger->configurations()->set(level, configurationType, value);
    logger->reconfigure();
  }
}

void Loggers::setDefaultConfigurations(const Configurations& configurations, bool
reconfigureExistingLoggers) {
  ELPP->registeredLoggers()->setDefaultConfigurations(configurations);
  if (reconfigureExistingLoggers) {
    Loggers::reconfigureAllLoggers(configurations);
  }
}

const Configurations* Loggers::defaultConfigurations(void) {
  return ELPP->registeredLoggers()->defaultConfigurations();
}
```

```

const base::LogStreamsReferenceMapPtr Loggers::logStreamsReference(void) {
  return ELPP->registeredLoggers()->logStreamsReference();
}

base::TypedConfigurations Loggers::defaultTypedConfigurations(void) {
  return base::TypedConfigurations(
           ELPP->registeredLoggers()->defaultConfigurations(),
           ELPP->registeredLoggers()->logStreamsReference());
}

std::vector<std::string>* Loggers::populateAllLoggerIds(std::vector<std::string>* targetList) {
  targetList->clear();
  for (base::RegisteredLoggers::iterator it = ELPP->registeredLoggers()->list().begin();
       it != ELPP->registeredLoggers()->list().end(); ++it) {
    targetList->push_back(it->first);
  }
  return targetList;
}

void Loggers::configureFromGlobal(const char* globalConfigurationFilePath) {
  std::ifstream gcfStream(globalConfigurationFilePath, std::ifstream::in);
  ELPP_ASSERT(gcfStream.is_open(), "Unable to open global configuration file [" «
globalConfigurationFilePath
              « "] for parsing.");
  std::string line = std::string();
  std::stringstream ss;
  Logger* logger = nullptr;
  auto configure = [&](void) {
    ELPP_INTERNAL_INFO(8, "Configuring logger: '" « logger->id() « "' with configurations \n" «
ss.str()
                       « "\n--------------");
    Configurations c;
    c.parseFromText(ss.str());
    logger->configure(c);
  };
  while (gcfStream.good()) {
    std::getline(gcfStream, line);
    ELPP_INTERNAL_INFO(1, "Parsing line: " « line);
    base::utils::Str::trim(line);
    if (Configurations::Parser::isComment(line)) continue;
    Configurations::Parser::ignoreComments(&line);
    base::utils::Str::trim(line);
    if (line.size() > 2 && base::utils::Str::startsWith(line,
std::string(base::consts::kConfigurationLoggerId))) {
      if (!ss.str().empty() && logger != nullptr) {
        configure();
      }
      ss.str(std::string(""));
      line = line.substr(2);
      base::utils::Str::trim(line);
      if (line.size() > 1) {
        ELPP_INTERNAL_INFO(1, "Getting logger: '" « line « "'");
        logger = getLogger(line);
      }
    } else {
      ss « line « "\n";
    }
  }
  if (!ss.str().empty() && logger != nullptr) {
    configure();
  }
}

bool Loggers::configureFromArg(const char* argKey) {
#if defined(ELPP_DISABLE_CONFIGURATION_FROM_PROGRAM_ARGS)
  ELPP_UNUSED(argKey);
#else
  if (!Helpers::commandLineArgs()->hasParamWithValue(argKey)) {
    return false;
  }
  configureFromGlobal(Helpers::commandLineArgs()->getParamValue(argKey));
#endif  // defined(ELPP_DISABLE_CONFIGURATION_FROM_PROGRAM_ARGS)
  return true;
}

void Loggers::flushAll(void) {
  ELPP->registeredLoggers()->flushAll();
}

void Loggers::setVerboseLevel(base::type::VerboseLevel level) {
  ELPP->vRegistry()->setLevel(level);
}

base::type::VerboseLevel Loggers::verboseLevel(void) {
  return ELPP->vRegistry()->level();
}
```

```
void Loggers::setVModules(const char* modules) {
  if (ELPP->vRegistry()->vModulesEnabled()) {
    ELPP->vRegistry()->setModules(modules);
  }
}

void Loggers::clearVModules(void) {
  ELPP->vRegistry()->clearModules();
}

// VersionInfo

const std::string VersionInfo::version(void) {
  return std::string("9.97.1");
}
const std::string VersionInfo::releaseDate(void) {
  return std::string("Thu Jul 20 2023 13:45:52 GMT+1000");
}

} // namespace el
```

9.3 lib/easylogging++.h File Reference

```
#include <ctime>
#include <cstring>
#include <cstdlib>
#include <cctype>
#include <cwchar>
#include <csignal>
#include <cerrno>
#include <cstdarg>
#include <string>
#include <vector>
#include <map>
#include <unordered_map>
#include <utility>
#include <functional>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <memory>
#include <type_traits>
```

Data Structures

- class el::base::NoCopy

  Internal helper class that prevent copy constructor for class.
- class el::base::StaticClass

  Internal helper class that makes all default constructors private.
- struct std::hash< el::Level >
- class el::LevelHelper

  Static class that contains helper functions for el::Level.
- class el::ConfigurationTypeHelper

  Static class that contains helper functions for el::ConfigurationType.
- class el::base::SubsecondPrecision

  A subsecond precision class containing actual width and offset of the subsecond part.

- class el::base::threading::internal::NoMutex

  Mutex wrapper used when multi-threading is disabled.
- class el::base::threading::internal::NoScopedLock< Mutex >

  Lock guard wrapper used when multi-threading is disabled.
- class el::base::threading::ThreadSafe

  Base of thread safe class, this class is inheritable-only.
- class el::base::utils::File
- class el::base::utils::Str

  String utilities helper class used internally. You should not use it.
- class el::base::utils::OS

  Operating System helper static class used internally. You should not use it.
- class el::base::utils::DateTime

  Contains utilities for cross-platform date/time. This class make use of el::base::utils::Str.
- class el::base::utils::CommandLineArgs

  Command line arguments for application if specified using el::Helpers::setArgs(..) or START_EASYLOGGINGPP(..)
- class el::base::utils::AbstractRegistry< T_Ptr, Container >

  Abstract registry (aka repository) that provides basic interface for pointer repository specified by T_Ptr type.
- class el::base::utils::Registry< T_Ptr, T_Key >

  A pointer registry mechanism to manage memory and provide search functionalities. (non-predicate version)
- class el::base::utils::RegistryWithPred< T_Ptr, Pred >

  A pointer registry mechanism to manage memory and provide search functionalities. (predicate version)
- class el::base::utils::Utils
- class el::Loggable

  Base of Easylogging++ friendly class.
- class el::base::LogFormat

  Represents log format containing flags and date format. This is used internally to start initial log.
- class el::CustomFormatSpecifier

  User-provided custom format specifier.
- class el::Configuration

  Represents single configuration that has representing level, configuration type and a string based value.
- class el::Configuration::Predicate

  Used to find configuration from configuration (pointers) repository. Avoid using it.
- class el::Configurations

  Thread-safe Configuration repository.
- class el::Configurations::Parser

  Parser used internally to parse configurations from file or text.
- class el::base::TypedConfigurations

  Configurations with data types.
- class el::base::HitCounter

  Class that keeps record of current line hit for occasional logging.
- class el::base::HitCounter::Predicate
- class el::base::RegisteredHitCounters

  Repository for hit counters used across the application.
- class el::Callback< T >
- class el::LogDispatchData
- class el::LogDispatchCallback
- class el::PerformanceTrackingCallback
- class el::LoggerRegistrationCallback
- class el::LogBuilder
- class el::Logger

  Represents a logger holding ID and configurations we need to write logs.

- class el::base::RegisteredLoggers

  Loggers repository.
- class el::base::VRegistry

  Represents registries for verbose logging.
- class el::LogMessage
- class el::base::Storage

  Easylogging++ management storage.
- class el::base::DefaultLogDispatchCallback
- class el::base::DefaultLogBuilder
- class el::base::LogDispatcher

  Dispatches log messages.
- class el::base::MessageBuilder
- class el::base::NullWriter

  Writes nothing - Used when certain log is disabled.
- class el::base::Writer

  Main entry point of each logging.
- class el::base::PErrorWriter
- class el::base::debug::CrashHandler
- class el::SysLogInitializer

  Initializes syslog with process ID, options and facility. calls closelog() on d'tor.
- class el::Helpers

  Static helpers for developers.
- class el::Loggers

  Static helpers to deal with loggers and their configurations.
- class el::Loggers::ScopedAddFlag

  Adds flag and removes it when scope goes out.
- class el::Loggers::ScopedRemoveFlag

  Removes flag and add it when scope goes out.
- class el::VersionInfo

Namespaces

- namespace el

  Easylogging++ entry namespace.
- namespace el::base

  Namespace containing base/internal functionality used by Easylogging++.
- namespace el::base::type

  Data types used by Easylogging++.
- namespace std
- namespace el::base::consts

  Namespace containing constants used internally.
- namespace el::base::utils

  Namespace containing utility functions/static classes used internally.
- namespace el::base::utils::bitwise

  Bitwise operations for C++11 strong enum class. This casts e into Flag_T and returns value after bitwise operation Use these function as.
- namespace el::base::threading
- namespace el::base::threading::internal
- namespace el::base::debug

  Contains some internal debugging tools like crash handler and stack tracer.

Macros

- #define ELPP_COMPILER_GCC 0
- #define ELPP_COMPILER_MSVC 0
- #define ELPP_CRT_DBG_WARNINGS ELPP_COMPILER_MSVC
- #define ELPP_COMPILER_CLANG 0
- #define ELPP_MINGW 0
- #define ELPP_CYGWIN 0
- #define ELPP_COMPILER_INTEL 0
- #define ELPP_OS_WINDOWS 0
- #define ELPP_OS_LINUX 0
- #define ELPP_OS_MAC 0
- #define ELPP_OS_FREEBSD 0
- #define ELPP_OS_SOLARIS 0
- #define ELPP_OS_AIX 0
- #define ELPP_OS_NETBSD 0
- #define ELPP_OS_EMSCRIPTEN 0
- #define ELPP_OS_QNX 0
- #define ELPP_OS_UNIX 0
- #define ELPP_OS_ANDROID 0
- #define ELPP_INTERNAL_DEBUGGING_OUT_INFO std::cout
- #define ELPP_INTERNAL_DEBUGGING_OUT_ERROR std::cerr
- #define ELPP_INTERNAL_DEBUGGING_ENDL std::endl
- #define ELPP_INTERNAL_DEBUGGING_MSG(msg) msg
- #define ELPP_ASSERT(expr, msg)
- #define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR ELPP_INTERNAL_DEBUGGING_OUT_ERROR << ": " << strerror(errno) << " [" << errno << "]"; (void)0
- #define ELPP_INTERNAL_ERROR(msg, pe)
- #define ELPP_INTERNAL_INFO(lvl, msg)
- #define ELPP_STACKTRACE 0
- #define ELPP_UNUSED(x) (void)x
- #define ELPP_EXPORT
- #define STRTOK(a, b, c) strtok(a, b)
- #define STRERROR(a, b, c) strerror(c)
- #define STRCAT(a, b, len) strcat(a, b)
- #define STRCPY(a, b, len) strcpy(a, b)
- #define ELPP_USE_STD_THREADING 0
- #define ELPP_FINAL
- #define ELPP_ASYNC_LOGGING 0
- #define ELPP_THREADING_ENABLED 0
- #define ELPP_FUNC ""
- #define ELPP_VARIADIC_TEMPLATES_SUPPORTED (ELPP_COMPILER_GCC || ELPP_COMPILER_CLANG || ELPP_COMPILER_INTEL || (ELPP_COMPILER_MSVC && _MSC_VER >= 1800))
- #define ELPP_LOGGING_ENABLED 1
- #define ELPP_DEBUG_LOG 1
- #define ELPP_INFO_LOG 1
- #define ELPP_WARNING_LOG 1
- #define ELPP_ERROR_LOG 1
- #define ELPP_FATAL_LOG 1
- #define ELPP_TRACE_LOG 1
- #define ELPP_VERBOSE_LOG 1
- #define elpptime_r localtime_r
- #define elpptime_s localtime_s
- #define elpptime localtime
- #define ELPP_LITERAL(txt) txt

- #define ELPP_STRLEN strlen
- #define ELPP_COUT std::cout
- #define ELPP_COUT_LINE(logLine) logLine << std::flush
- #define ELPP el::base::elStorage
- #define ELPP_SIMPLE_LOG(LOG_TYPE)
- #define ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(temp)
- #define ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(temp)
- #define ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(temp)
- #define ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(temp)
- #define ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(temp)
- #define MAKE_CONTAINERELPP_FRIENDLY(ContainerType, SizeMethod, ElementInstance)

  Macro used internally that can be used externally to make containers easylogging++ friendly.
- #define ELPP_WX_PTR_ENABLED(ContainerType)
- #define ELPP_WX_ENABLED(ContainerType)
- #define ELPP_WX_HASH_MAP_ENABLED(ContainerType)
- #define el_getVALength(...) el_resolveVALength(0, ## __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
- #define el_resolveVALength(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
- #define ELPP_WRITE_LOG(writer, level, dispatchAction, ...) writer(level, __FILE__, __LINE__↩
, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__), __VA_ARGS__)
- #define ELPP_WRITE_LOG_IF(writer, condition, level, dispatchAction, ...)
- #define ELPP_WRITE_LOG_EVERY_N(writer, occasion, level, dispatchAction, ...)
- #define ELPP_WRITE_LOG_AFTER_N(writer, n, level, dispatchAction, ...)
- #define ELPP_WRITE_LOG_N_TIMES(writer, n, level, dispatchAction, ...)
- #define MAKE_LOGGABLE(ClassType, ClassInstance, OutputStreamInstance) el::base::type::ostream_t& operator<<(el::base::type::ostream_t& OutputStreamInstance, const ClassType& ClassInstance)
- #define ELPP_INITIALIZE_SYSLOG(id, opt, fac) el::SysLogInitializer elSyslogInit(id, opt, fac)
- #define VLOG_IS_ON(verboseLevel) (ELPP->vRegistry()->allowed(verboseLevel, __FILE__))

  Determines whether verbose logging is on for specified level current file.
- #define ELPP_MIN_UNIT el::base::TimestampUnit::Millisecond
- #define TIMED_SCOPE_IF(obj, blockname, condition)

  Performance tracked scope. Performance gets written when goes out of scope using 'performance' logger.
- #define TIMED_SCOPE(obj, blockname) TIMED_SCOPE_IF(obj, blockname, true)
- #define TIMED_BLOCK(obj, blockName)
- #define TIMED_FUNC_IF(obj, condition) TIMED_SCOPE_IF(obj, ELPP_FUNC, condition)

  Performance tracked function. Performance gets written when goes out of scope using 'performance' logger.
- #define TIMED_FUNC(obj) TIMED_SCOPE(obj, ELPP_FUNC)
- #define PERFORMANCE_CHECKPOINT(obj) obj->checkpoint(std::string(), __FILE__, __LINE__↩
, ELPP_FUNC)
- #define PERFORMANCE_CHECKPOINT_WITH_ID(obj, id) obj->checkpoint(id, __FILE__, __LINE__↩
, ELPP_FUNC)
- #define ELPP_COUNTER (ELPP->hitCounters()->getCounter(__FILE__, __LINE__))

  Gets hit counter for file/line.
- #define ELPP_COUNTER_POS (ELPP_COUNTER == nullptr ? -1 : ELPP_COUNTER->hitCounts())

  Gets hit counter position for file/line, -1 if not registered yet.
- #define CINFO(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Info, dispatchAction, __VA↩
_ARGS__)
- #define CWARNING(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Warning, dispatch↩
Action, __VA_ARGS__)
- #define CDEBUG(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Debug, dispatchAction, ↩
__VA_ARGS__)
- #define CERROR(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Error, dispatchAction, _↩
_VA_ARGS__)
- #define CFATAL(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Fatal, dispatchAction, __↩
VA_ARGS__)

- #define CTRACE(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Trace, dispatchAction, _↵ _VA_ARGS__)
- #define CVERBOSE(writer, vlevel, dispatchAction, ...)
- #define CINFO_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Info, dispatchAction, __VA_ARGS__)
- #define CWARNING_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Warning, dispatchAction, __VA_ARGS__)
- #define CDEBUG_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Debug, dispatchAction, __VA_ARGS__)
- #define CERROR_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Error, dispatchAction, __VA_ARGS__)
- #define CFATAL_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Fatal, dispatchAction, __VA_ARGS__)
- #define CTRACE_IF(writer, condition_, dispatchAction, ...) ELPP_WRITE_LOG_IF(writer, (condition_↵), el::Level::Trace, dispatchAction, __VA_ARGS__)
- #define CVERBOSE_IF(writer, condition_, vlevel, dispatchAction, ...)
- #define CINFO_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Info, dispatchAction, __VA_ARGS__)
- #define CWARNING_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Warning, dispatchAction, __VA_ARGS__)
- #define CDEBUG_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Debug, dispatchAction, __VA_ARGS__)
- #define CERROR_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Error, dispatchAction, __VA_ARGS__)
- #define CFATAL_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Fatal, dispatchAction, __VA_ARGS__)
- #define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Trace, dispatchAction, __VA_ARGS__)
- #define CVERBOSE_EVERY_N(writer, occasion, vlevel, dispatchAction, ...) CVERBOSE_IF(writer, ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion), vlevel, dispatchAction, __VA_ARGS__)
- #define CINFO_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
- #define CWARNING_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
- #define CDEBUG_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
- #define CERROR_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
- #define CFATAL_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
- #define CTRACE_AFTER_N(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)
- #define CVERBOSE_AFTER_N(writer, n, vlevel, dispatchAction, ...) CVERBOSE_IF(writer, ELPP->validateAfterNCounter(__FILE__, __LINE__, n), vlevel, dispatchAction, __VA_ARGS__)
- #define CINFO_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
- #define CWARNING_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
- #define CDEBUG_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
- #define CERROR_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
- #define CFATAL_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
- #define CTRACE_N_TIMES(writer, n, dispatchAction, ...) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)

- #define CVERBOSE_N_TIMES(writer, n, vlevel, dispatchAction, ...) CVERBOSE_IF(writer, ELPP->validateNTimesCounter(__FILE__, __LINE__, n), vlevel, dispatchAction, __VA_ARGS__)
- #define CLOG(LEVEL, ...) C##LEVEL(el::base::Writer, el::base::DispatchAction::NormalLog, __VA_ARGS↩ __)
- #define CVLOG(vlevel, ...) CVERBOSE(el::base::Writer, vlevel, el::base::DispatchAction::NormalLog, __↩ VA_ARGS__)
- #define CLOG_IF(condition, LEVEL, ...) C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define CVLOG_IF(condition, vlevel, ...) CVERBOSE_IF(el::base::Writer, condition, vlevel, el::base::DispatchAction::NormalLog __VA_ARGS__)
- #define CLOG_EVERY_N(n, LEVEL, ...) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define CVLOG_EVERY_N(n, vlevel, ...) CVERBOSE_EVERY_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLo __VA_ARGS__)
- #define CLOG_AFTER_N(n, LEVEL, ...) C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define CVLOG_AFTER_N(n, vlevel, ...) CVERBOSE_AFTER_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLo __VA_ARGS__)
- #define CLOG_N_TIMES(n, LEVEL, ...) C##LEVEL##_N_TIMES(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define CVLOG_N_TIMES(n, vlevel, ...) CVERBOSE_N_TIMES(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog __VA_ARGS__)
- #define ELPP_CURR_FILE_LOGGER_ID el::base::consts::kDefaultLoggerId
- #define ELPP_TRACE CLOG(TRACE, ELPP_CURR_FILE_LOGGER_ID)
- #define LOG(LEVEL) CLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define VLOG(vlevel) CVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define LOG_IF(condition, LEVEL) CLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define VLOG_IF(condition, vlevel) CVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define LOG_EVERY_N(n, LEVEL) CLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define VLOG_EVERY_N(n, vlevel) CVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define LOG_AFTER_N(n, LEVEL) CLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define VLOG_AFTER_N(n, vlevel) CVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define LOG_N_TIMES(n, LEVEL) CLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define VLOG_N_TIMES(n, vlevel) CVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define CPLOG(LEVEL, ...) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA↩ _ARGS__)
- #define CPLOG_IF(condition, LEVEL, ...) C##LEVEL##_IF(el::base::PErrorWriter, condition, el::base::DispatchAction::NormalL __VA_ARGS__)
- #define DCPLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define DCPLOG_IF(condition, LEVEL, ...) C##LEVEL##_IF(el::base::PErrorWriter, (ELPP_DEBUG_LOG) && (condition), el::base::DispatchAction::NormalLog, __VA_ARGS__)
- #define PLOG(LEVEL) CPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define PLOG_IF(condition, LEVEL) CPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DPLOG(LEVEL) DCPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DPLOG_IF(condition, LEVEL) DCPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define CSYSLOG(LEVEL, ...) el::base::NullWriter()
- #define CSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
- #define CSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
- #define CSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
- #define CSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
- #define SYSLOG(LEVEL) el::base::NullWriter()
- #define SYSLOG_IF(condition, LEVEL) el::base::NullWriter()
- #define SYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
- #define SYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()

- #define SYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
- #define DCSYSLOG(LEVEL, ...) el::base::NullWriter()
- #define DCSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
- #define DCSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
- #define DCSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
- #define DCSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
- #define DSYSLOG(LEVEL) el::base::NullWriter()
- #define DSYSLOG_IF(condition, LEVEL) el::base::NullWriter()
- #define DSYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
- #define DSYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()
- #define DSYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
- #define DCLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG(LEVEL, __VA_ARGS__)
- #define DCLOG_VERBOSE(vlevel, ...) if (ELPP_DEBUG_LOG) CLOG_VERBOSE(vlevel, __VA_ARGS__)
- #define DCVLOG(vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG(vlevel, __VA_ARGS__)
- #define DCLOG_IF(condition, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_IF(condition, LEVEL, __VA_↩ ARGS__)
- #define DCVLOG_IF(condition, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_IF(condition, vlevel, __VA_↩ ARGS__)
- #define DCLOG_EVERY_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_EVERY_N(n, LEVEL, __VA_↩ ARGS__)
- #define DCVLOG_EVERY_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_EVERY_N(n, vlevel, __VA_↩ ARGS__)
- #define DCLOG_AFTER_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_AFTER_N(n, LEVEL, __VA_↩ ARGS__)
- #define DCVLOG_AFTER_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_AFTER_N(n, vlevel, __VA_↩ ARGS__)
- #define DCLOG_N_TIMES(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_N_TIMES(n, LEVEL, __VA_↩ ARGS__)
- #define DCVLOG_N_TIMES(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_N_TIMES(n, vlevel, __VA_↩ ARGS__)
- #define DLOG(LEVEL) DCLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DVLOG(vlevel) DCVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define DLOG_IF(condition, LEVEL) DCLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DVLOG_IF(condition, vlevel) DCVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define DLOG_EVERY_N(n, LEVEL) DCLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DVLOG_EVERY_N(n, vlevel) DCVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define DLOG_AFTER_N(n, LEVEL) DCLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DVLOG_AFTER_N(n, vlevel) DCVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define DLOG_N_TIMES(n, LEVEL) DCLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
- #define DVLOG_N_TIMES(n, vlevel) DCVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
- #define CCHECK(condition, ...) CLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed: [" << #condition << "] "
- #define CPCHECK(condition, ...) CPLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed: [" << #condition << "] "
- #define CHECK(condition) CCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
- #define PCHECK(condition) CPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
- #define CCHECK_EQ(a, b, ...) CCHECK(a == b, __VA_ARGS__)
- #define CCHECK_NE(a, b, ...) CCHECK(a != b, __VA_ARGS__)
- #define CCHECK_LT(a, b, ...) CCHECK(a < b, __VA_ARGS__)
- #define CCHECK_GT(a, b, ...) CCHECK(a > b, __VA_ARGS__)
- #define CCHECK_LE(a, b, ...) CCHECK(a <= b, __VA_ARGS__)
- #define CCHECK_GE(a, b, ...) CCHECK(a >= b, __VA_ARGS__)
- #define CCHECK_BOUNDS(val, min, max, ...) CCHECK(val >= min && val <= max, __VA_ARGS__)
- #define CHECK_EQ(a, b) CCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_NE(a, b) CCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)

- #define CHECK_LT(a, b) CCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_GT(a, b) CCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_LE(a, b) CCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_GE(a, b) CCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_BOUNDS(val, min, max) CCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
- #define CCHECK_NOTNULL(ptr, ...) CCHECK((ptr) != nullptr, __VA_ARGS__)
- #define CCHECK_STREQ(str1, str2, ...)
- #define CCHECK_STRNE(str1, str2, ...)
- #define CCHECK_STRCASEEQ(str1, str2, ...)
- #define CCHECK_STRCASENE(str1, str2, ...)
- #define CHECK_NOTNULL(ptr) CCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_STREQ(str1, str2) CCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_STRNE(str1, str2) CCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_STRCASEEQ(str1, str2) CCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define CHECK_STRCASENE(str1, str2) CCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define DCCHECK(condition, ...) if (ELPP_DEBUG_LOG) CCHECK(condition, __VA_ARGS__)
- #define DCCHECK_EQ(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_EQ(a, b, __VA_ARGS__)
- #define DCCHECK_NE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_NE(a, b, __VA_ARGS__)
- #define DCCHECK_LT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LT(a, b, __VA_ARGS__)
- #define DCCHECK_GT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GT(a, b, __VA_ARGS__)
- #define DCCHECK_LE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LE(a, b, __VA_ARGS__)
- #define DCCHECK_GE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GE(a, b, __VA_ARGS__)
- #define DCCHECK_BOUNDS(val, min, max, ...) if (ELPP_DEBUG_LOG) CCHECK_BOUNDS(val, min, max, __VA_ARGS__)
- #define DCCHECK_NOTNULL(ptr, ...) if (ELPP_DEBUG_LOG) CCHECK_NOTNULL((ptr), __VA_ARGS_↩ _)
- #define DCCHECK_STREQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STREQ(str1, str2, __VA_↩ ARGS__)
- #define DCCHECK_STRNE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRNE(str1, str2, __VA_↩ ARGS__)
- #define DCCHECK_STRCASEEQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASEEQ(str1, str2, __VA_ARGS__)
- #define DCCHECK_STRCASENE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASENE(str1, str2, __VA_ARGS__)
- #define DCPCHECK(condition, ...) if (ELPP_DEBUG_LOG) CPCHECK(condition, __VA_ARGS__)
- #define DCHECK(condition) DCCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_EQ(a, b) DCCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_NE(a, b) DCCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_LT(a, b) DCCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_GT(a, b) DCCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_LE(a, b) DCCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_GE(a, b) DCCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_BOUNDS(val, min, max) DCCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_NOTNULL(ptr) DCCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_STREQ(str1, str2) DCCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_STRNE(str1, str2) DCCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_STRCASEEQ(str1, str2) DCCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define DCHECK_STRCASENE(str1, str2) DCCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
- #define DPCHECK(condition) DCPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
- #define ELPP_USE_DEF_CRASH_HANDLER true
- #define ELPP_CRASH_HANDLER_INIT
- #define ELPP_INIT_EASYLOGGINGPP(val)
- #define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new el::base::DefaultLogBuilder())))
- #define INITIALIZE_NULL_EASYLOGGINGPP
- #define SHARE_EASYLOGGINGPP(initializedStorage)
- #define START_EASYLOGGINGPP(argc, argv) el::Helpers::setArgs(argc, argv)

Typedefs

- typedef char el::base::type::char_t
- typedef std::string el::base::type::string_t
- typedef std::stringstream el::base::type::stringstream_t
- typedef std::fstream el::base::type::fstream_t
- typedef std::ostream el::base::type::ostream_t
- typedef unsigned int el::base::type::EnumType
- typedef unsigned short el::base::type::VerboseLevel
- typedef unsigned long int el::base::type::LineNumber
- typedef std::shared_ptr< base::Storage > el::base::type::StoragePointer
- typedef std::shared_ptr< LogDispatchCallback > el::base::type::LogDispatchCallbackPtr
- typedef std::shared_ptr< PerformanceTrackingCallback > el::base::type::PerformanceTrackingCallbackPtr
- typedef std::shared_ptr< LoggerRegistrationCallback > el::base::type::LoggerRegistrationCallbackPtr
- typedef std::unique_ptr< el::base::PerformanceTracker > el::base::type::PerformanceTrackerPtr
- typedef std::function< void(const char ∗, std::size_t)> el::PreRollOutCallback
- typedef SubsecondPrecision el::base::MillisecondsWidth

  Type alias of SubsecondPrecision.
- typedef base::threading::internal::NoMutex el::base::threading::Mutex
- typedef base::threading::internal::NoScopedLock< base::threading::Mutex > el::base::threading::ScopedLock
- typedef std::function< std::string(const LogMessage ∗)> el::FormatSpecifierValueResolver

  Resolving function for format specifier.
- typedef std::shared_ptr< base::type::fstream_t > el::base::FileStreamPtr
- typedef std::unordered_map< std::string, FileStreamPtr > el::base::LogStreamsReferenceMap
- typedef std::shared_ptr< base::LogStreamsReferenceMap > el::base::LogStreamsReferenceMapPtr
- typedef std::shared_ptr< LogBuilder > el::LogBuilderPtr

Enumerations

- enum class el::Level : base::type::EnumType {
  el::Global = 1 , el::Trace = 2 , el::Debug = 4 , el::Fatal = 8 ,
  el::Error = 16 , el::Warning = 32 , el::Verbose = 64 , el::Info = 128 ,
  el::Unknown = 1010 }

  Represents enumeration for severity level used to determine level of logging.
- enum class el::ConfigurationType : base::type::EnumType {
  el::Enabled = 1 , el::ToFile = 2 , el::ToStandardOutput = 4 , el::Format = 8 ,
  el::Filename = 16 , el::SubsecondPrecision = 32 , el::MillisecondsWidth = SubsecondPrecision ,
  el::PerformanceTracking = 64 ,
  el::MaxLogFileSize = 128 , el::LogFlushThreshold = 256 , el::Unknown = 1010 }

  Represents enumeration of ConfigurationType used to configure or access certain aspect of logging.
- enum class el::LoggingFlag : base::type::EnumType {
  el::NewLineForContainer = 1 , el::AllowVerboseIfModuleNotSpecified = 2 , el::LogDetailedCrashReason = 4
  , el::DisableApplicationAbortOnFatalLog = 8 ,
  el::ImmediateFlush = 16 , el::StrictLogFileSizeCheck = 32 , el::ColoredTerminalOutput = 64 , el::MultiLoggerSupport
  = 128 ,
  el::DisablePerformanceTrackingCheckpointComparison = 256 , el::DisableVModules = 512 , el::DisableVModulesExtensions
  = 1024 , el::HierarchicalLogging = 2048 ,
  el::CreateLoggerAutomatically = 4096 , el::AutoSpacing = 8192 , el::FixedTimeFormat = 16384 ,
  el::IgnoreSigInt = 32768 }

  Flags used while writing logs. This flags are set by user.
- enum class el::base::TimestampUnit : base::type::EnumType {
  el::base::Microsecond = 0 , el::base::Millisecond = 1 , el::base::Second = 2 , el::base::Minute = 3 ,
  el::base::Hour = 4 , el::base::Day = 5 }

  Enum to represent timestamp unit.

- enum class el::base::FormatFlags : base::type::EnumType {
  el::base::DateTime = 1 << 1 , el::base::LoggerId = 1 << 2 , el::base::File = 1 << 3 , el::base::Line = 1 << 4 ,
  el::base::Location = 1 << 5 , el::base::Function = 1 << 6 , el::base::User = 1 << 7 , el::base::Host = 1 << 8 ,
  el::base::LogMessage = 1 << 9 , el::base::VerboseLevel = 1 << 10 , el::base::AppName = 1 << 11 , el::base::ThreadId = 1 << 12 ,
  el::base::Level = 1 << 13 , el::base::FileBase = 1 << 14 , el::base::LevelShort = 1 << 15 }

  Format flags used to determine specifiers that are active for performance improvements.
- enum class el::base::DispatchAction : base::type::EnumType { el::base::None = 1 , el::base::NormalLog = 2 , el::base::SysLog = 4 }

  Action to be taken for dispatching.

Functions

- static void el::base::defaultPreRollOutCallback (const char ∗, std::size_t)
- template<typename T >
  static std::enable_if< std::is_pointer< T ∗ >::value, void >::type el::base::utils::safeDelete (T ∗&pointer)

  Deletes memory safely and points to null.
- template<typename Enum >
  static base::type::EnumType el::base::utils::bitwise::And (Enum e, base::type::EnumType flag)
- template<typename Enum >
  static base::type::EnumType el::base::utils::bitwise::Not (Enum e, base::type::EnumType flag)
- template<typename Enum >
  static base::type::EnumType el::base::utils::bitwise::Or (Enum e, base::type::EnumType flag)
- template<typename Enum >
  static void el::base::utils::addFlag (Enum e, base::type::EnumType ∗flag)
- template<typename Enum >
  static void el::base::utils::removeFlag (Enum e, base::type::EnumType ∗flag)
- template<typename Enum >
  static bool el::base::utils::hasFlag (Enum e, base::type::EnumType flag)
- static std::string el::base::threading::getCurrentThreadId (void)

Variables

- static const char el::base::consts::kFormatSpecifierCharValue = 'v'
- static const char el::base::consts::kFormatSpecifierChar = '%'
- static const unsigned int el::base::consts::kMaxLogPerCounter = 100000
- static const unsigned int el::base::consts::kMaxLogPerContainer = 100
- static const unsigned int el::base::consts::kDefaultSubsecondPrecision = 3
- static const char ∗ el::base::consts::kDefaultLoggerId = "default"
- static const char ∗ el::base::consts::kFilePathSeparator = "/"
- static const std::size_t el::base::consts::kSourceFilenameMaxLength = 100
- static const std::size_t el::base::consts::kSourceLineMaxLength = 10
- static const Level el::base::consts::kPerformanceTrackerDefaultLevel = Level::Info
- struct {
    double el::base::consts::value
    const base::type::char_t ∗ el::base::consts::unit
  } el::base::consts::kTimeFormats [ ]
- static const int el::base::consts::kTimeFormatsCount = sizeof(kTimeFormats) / sizeof(kTimeFormats[0])

- struct {
    int el::base::consts::numb
    const char ∗ el::base::consts::name
    const char ∗ el::base::consts::brief
    const char ∗ el::base::consts::detail
  } el::base::consts::kCrashSignals [ ]
- static const int el::base::consts::kCrashSignalsCount = sizeof(kCrashSignals) / sizeof(kCrashSignals[0])
- ELPP_EXPORT base::type::StoragePointer el::base::elStorage
- base::debug::CrashHandler el::elCrashHandler

9.3.1 Macro Definition Documentation

9.3.1.1 CCHECK

```
#define CCHECK(
              condition,
              ...  ) CLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed:  [" << #condition
<< "] "
```

Definition at line 4447 of file easylogging++.h.

9.3.1.2 CCHECK_BOUNDS

```
#define CCHECK_BOUNDS(
              val,
              min,
              max,
              ...  ) CCHECK(val >= min && val <= max, __VA_ARGS__)
```

Definition at line 4457 of file easylogging++.h.

9.3.1.3 CCHECK_EQ

```
#define CCHECK_EQ(
              a,
              b,
              ...  ) CCHECK(a == b, __VA_ARGS__)
```

Definition at line 4451 of file easylogging++.h.

9.3.1.4 CCHECK_GE

```
#define CCHECK_GE(
              a,
              b,
              ...  ) CCHECK(a >= b, __VA_ARGS__)
```

Definition at line 4456 of file easylogging++.h.

9.3.1.5 CCHECK_GT

```
#define CCHECK_GT(
              a,
              b,
              ...  ) CCHECK(a > b, __VA_ARGS__)
```

Definition at line 4454 of file easylogging++.h.

9.3.1.6 CCHECK_LE

```
#define CCHECK_LE(
              a,
              b,
              ...  ) CCHECK(a <= b, __VA_ARGS__)
```

Definition at line 4455 of file easylogging++.h.

9.3.1.7 CCHECK_LT

```
#define CCHECK_LT(
              a,
              b,
              ...  ) CCHECK(a < b, __VA_ARGS__)
```

Definition at line 4453 of file easylogging++.h.

9.3.1.8 CCHECK_NE

```
#define CCHECK_NE(
              a,
              b,
              ...  ) CCHECK(a != b, __VA_ARGS__)
```

Definition at line 4452 of file easylogging++.h.

9.3.1.9 CCHECK_NOTNULL

```
#define CCHECK_NOTNULL(
              ptr,
              ...  ) CCHECK((ptr) != nullptr, __VA_ARGS__)
```

Definition at line 4465 of file easylogging++.h.

9.3.1.10 CCHECK_STRCASEEQ

```
#define CCHECK_STRCASEEQ(
                str1,
                str2,
                ... )
```

Value:
```
CLOG_IF(!el::base::utils::Str::cStringCaseEq(str1, str2), FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " == " « #str2 « "] "
```

Definition at line 4470 of file easylogging++.h.

9.3.1.11 CCHECK_STRCASENE

```
#define CCHECK_STRCASENE(
                str1,
                str2,
                ... )
```

Value:
```
CLOG_IF(el::base::utils::Str::cStringCaseEq(str1, str2), FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " != " « #str2 « "] "
```

Definition at line 4472 of file easylogging++.h.

9.3.1.12 CCHECK_STREQ

```
#define CCHECK_STREQ(
                str1,
                str2,
                ... )
```

Value:
```
CLOG_IF(!el::base::utils::Str::cStringEq(str1, str2), FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " == " « #str2 « "] "
```

Definition at line 4466 of file easylogging++.h.

9.3.1.13 CCHECK_STRNE

```
#define CCHECK_STRNE(
                str1,
                str2,
                ... )
```

Value:
```
CLOG_IF(el::base::utils::Str::cStringEq(str1, str2), FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " != " « #str2 « "] "
```

Definition at line 4468 of file easylogging++.h.

9.3.1.14 CDEBUG

```
#define CDEBUG(
              writer,
              dispatchAction,
              ... ) ELPP_WRITE_LOG(writer, el::Level::Debug, dispatchAction, __VA_ARGS__)
```

Definition at line 4003 of file easylogging++.h.

9.3.1.15 CDEBUG_AFTER_N

```
#define CDEBUG_AFTER_N(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Debug, dispatchAction, __VA↩
_ARGS__)
```

Definition at line 4128 of file easylogging++.h.

9.3.1.16 CDEBUG_EVERY_N

```
#define CDEBUG_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Debug, dispatch↩
Action, __VA_ARGS__)
```

Definition at line 4085 of file easylogging++.h.

9.3.1.17 CDEBUG_IF

```
#define CDEBUG_IF(
              writer,
              condition_,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Debug, dispatchAction,
__VA_ARGS__)
```

Definition at line 4042 of file easylogging++.h.

9.3.1.18 CDEBUG_N_TIMES

```
#define CDEBUG_N_TIMES(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Debug, dispatchAction, __VA↩
_ARGS__)
```

Definition at line 4171 of file easylogging++.h.

9.3.1.19 CERROR

```
#define CERROR(
              writer,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG(writer, el::Level::Error, dispatchAction, __VA_ARGS__)
```

Definition at line 4008 of file easylogging++.h.

9.3.1.20 CERROR_AFTER_N

```
#define CERROR_AFTER_N(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Error, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4134 of file easylogging++.h.

9.3.1.21 CERROR_EVERY_N

```
#define CERROR_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Error, dispatch←
Action, __VA_ARGS__)
```

Definition at line 4091 of file easylogging++.h.

9.3.1.22 CERROR_IF

```
#define CERROR_IF(
              writer,
              condition_,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Error, dispatchAction,
__VA_ARGS__)
```

Definition at line 4048 of file easylogging++.h.

9.3.1.23 CERROR_N_TIMES

```
#define CERROR_N_TIMES(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Error, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4177 of file easylogging++.h.

9.3.1.24 CFATAL

```
#define CFATAL(
              writer,
              dispatchAction,
              ... ) ELPP_WRITE_LOG(writer, el::Level::Fatal, dispatchAction, __VA_ARGS__)
```

Definition at line 4013 of file easylogging++.h.

9.3.1.25 CFATAL_AFTER_N

```
#define CFATAL_AFTER_N(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Fatal, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4140 of file easylogging++.h.

9.3.1.26 CFATAL_EVERY_N

```
#define CFATAL_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Fatal, dispatch←
Action, __VA_ARGS__)
```

Definition at line 4097 of file easylogging++.h.

9.3.1.27 CFATAL_IF

```
#define CFATAL_IF(
              writer,
              condition_,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Fatal, dispatchAction,
__VA_ARGS__)
```

Definition at line 4054 of file easylogging++.h.

9.3.1.28 CFATAL_N_TIMES

```
#define CFATAL_N_TIMES(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Fatal, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4183 of file easylogging++.h.

9.3.1.29 CHECK

```
#define CHECK(
                condition ) CCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4449 of file easylogging++.h.

9.3.1.30 CHECK_BOUNDS

```
#define CHECK_BOUNDS(
                val,
                min,
                max ) CCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4464 of file easylogging++.h.

9.3.1.31 CHECK_EQ

```
#define CHECK_EQ(
                a,
                b ) CCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4458 of file easylogging++.h.

9.3.1.32 CHECK_GE

```
#define CHECK_GE(
                a,
                b ) CCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4463 of file easylogging++.h.

9.3.1.33 CHECK_GT

```
#define CHECK_GT(
                a,
                b ) CCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4461 of file easylogging++.h.

9.3.1.34 CHECK_LE

```
#define CHECK_LE(
                a,
                b ) CCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4462 of file easylogging++.h.

9.3.1.35 CHECK_LT

```
#define CHECK_LT(
            a,
            b ) CCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4460 of file easylogging++.h.

9.3.1.36 CHECK_NE

```
#define CHECK_NE(
            a,
            b ) CCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4459 of file easylogging++.h.

9.3.1.37 CHECK_NOTNULL

```
#define CHECK_NOTNULL(
            ptr ) CCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4474 of file easylogging++.h.

9.3.1.38 CHECK_STRCASEEQ

```
#define CHECK_STRCASEEQ(
            str1,
            str2 ) CCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4477 of file easylogging++.h.

9.3.1.39 CHECK_STRCASENE

```
#define CHECK_STRCASENE(
            str1,
            str2 ) CCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4478 of file easylogging++.h.

9.3.1.40 CHECK_STREQ

```
#define CHECK_STREQ(
            str1,
            str2 ) CCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4475 of file easylogging++.h.

9.3.1.41 CHECK_STRNE

```
#define CHECK_STRNE(
              str1,
              str2 ) CCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4476 of file easylogging++.h.

9.3.1.42 CINFO

```
#define CINFO(
              writer,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG(writer, el::Level::Info, dispatchAction, __VA_ARGS__)
```

Definition at line 3993 of file easylogging++.h.

9.3.1.43 CINFO_AFTER_N

```
#define CINFO_AFTER_N(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Info, dispatchAction, __VA_←
ARGS__)
```

Definition at line 4116 of file easylogging++.h.

9.3.1.44 CINFO_EVERY_N

```
#define CINFO_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Info, dispatchAction,
__VA_ARGS__)
```

Definition at line 4073 of file easylogging++.h.

9.3.1.45 CINFO_IF

```
#define CINFO_IF(
              writer,
              condition_,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Info, dispatchAction,
__VA_ARGS__)
```

Definition at line 4030 of file easylogging++.h.

9.3.1.46 CINFO_N_TIMES

```
#define CINFO_N_TIMES(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Info, dispatchAction, __VA_←
ARGS__)
```

Definition at line 4159 of file easylogging++.h.

9.3.1.47 CLOG

```
#define CLOG(
              LEVEL,
              ...  ) C##LEVEL(el::base::Writer, el::base::DispatchAction::NormalLog, __VA_←
ARGS__)
```

Definition at line 4217 of file easylogging++.h.

9.3.1.48 CLOG_AFTER_N

```
#define CLOG_AFTER_N(
              n,
              LEVEL,
              ...  ) C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4230 of file easylogging++.h.

9.3.1.49 CLOG_EVERY_N

```
#define CLOG_EVERY_N(
              n,
              LEVEL,
              ...  ) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4226 of file easylogging++.h.

9.3.1.50 CLOG_IF

```
#define CLOG_IF(
              condition,
              LEVEL,
              ...  ) C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4221 of file easylogging++.h.

9.3.1.51 CLOG_N_TIMES

```
#define CLOG_N_TIMES(
              n,
              LEVEL,
              ...  ) C##LEVEL##_N_TIMES(el::base::Writer, n, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4234 of file easylogging++.h.

9.3.1.52 CPCHECK

```
#define CPCHECK(
              condition,
              ...  ) CPLOG_IF(!(condition), FATAL, __VA_ARGS__) << "Check failed:  [" <<
#condition << "] "
```

Definition at line 4448 of file easylogging++.h.

9.3.1.53 CPLOG

```
#define CPLOG(
              LEVEL,
              ...  ) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __↩
VA_ARGS__)
```

Definition at line 4282 of file easylogging++.h.

9.3.1.54 CPLOG_IF

```
#define CPLOG_IF(
              condition,
              LEVEL,
              ...  ) C##LEVEL##_IF(el::base::PErrorWriter, condition, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4284 of file easylogging++.h.

9.3.1.55 CSYSLOG

```
#define CSYSLOG(
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4343 of file easylogging++.h.

9.3.1.56 CSYSLOG_AFTER_N

```
#define CSYSLOG_AFTER_N(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4346 of file easylogging++.h.

9.3.1.57 CSYSLOG_EVERY_N

```
#define CSYSLOG_EVERY_N(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4345 of file easylogging++.h.

9.3.1.58 CSYSLOG_IF

```
#define CSYSLOG_IF(
              condition,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4344 of file easylogging++.h.

9.3.1.59 CSYSLOG_N_TIMES

```
#define CSYSLOG_N_TIMES(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4347 of file easylogging++.h.

9.3.1.60 CTRACE

```
#define CTRACE(
              writer,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG(writer, el::Level::Trace, dispatchAction, __VA_ARGS__)
```

Definition at line 4018 of file easylogging++.h.

9.3.1.61 CTRACE_AFTER_N

```
#define CTRACE_AFTER_N(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Trace, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4146 of file easylogging++.h.

9.3.1.62 CTRACE_EVERY_N

```
#define CTRACE_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Trace, dispatch←
Action, __VA_ARGS__)
```

Definition at line 4103 of file easylogging++.h.

9.3.1.63 CTRACE_IF

```
#define CTRACE_IF(
              writer,
              condition_,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Trace, dispatchAction,
__VA_ARGS__)
```

Definition at line 4060 of file easylogging++.h.

9.3.1.64 CTRACE_N_TIMES

```
#define CTRACE_N_TIMES(
              writer,
              n,
              dispatchAction,
              ... ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Trace, dispatchAction, __VA←
_ARGS__)
```

Definition at line 4189 of file easylogging++.h.

9.3.1.65 CVERBOSE

```
#define CVERBOSE(
              writer,
              vlevel,
              dispatchAction,
              ... )
```

Value:

```
if (VLOG_IS_ON(vlevel)) writer(\
el::Level::Verbose, __FILE__, __LINE__, ELPP_FUNC, dispatchAction,
     vlevel).construct(el_getVALength(__VA_ARGS__), __VA_ARGS__)
```

Definition at line 4023 of file easylogging++.h.

9.3.1.66 CVERBOSE_AFTER_N

```
#define CVERBOSE_AFTER_N(
              writer,
              n,
              vlevel,
              dispatchAction,
              ... ) CVERBOSE_IF(writer, ELPP->validateAfterNCounter(__FILE__, __LINE__, n),
vlevel, dispatchAction, __VA_ARGS__)
```

Definition at line 4152 of file easylogging++.h.

9.3.1.67 CVERBOSE_EVERY_N

```
#define CVERBOSE_EVERY_N(
              writer,
              occasion,
              vlevel,
              dispatchAction,
              ... ) CVERBOSE_IF(writer, ELPP->validateEveryNCounter(__FILE__, __LINE__←
, occasion), vlevel, dispatchAction, __VA_ARGS__)
```

Definition at line 4109 of file easylogging++.h.

9.3.1.68 CVERBOSE_IF

```
#define CVERBOSE_IF(
              writer,
              condition_,
              vlevel,
              dispatchAction,
              ... )
```

Value:

```
if (VLOG_IS_ON(vlevel) && (condition_)) writer( \
el::Level::Verbose, __FILE__, __LINE__, ELPP_FUNC, dispatchAction,
      vlevel).construct(el_getVALength(__VA_ARGS__), __VA_ARGS__)
```

Definition at line 4066 of file easylogging++.h.

9.3.1.69 CVERBOSE_N_TIMES

```
#define CVERBOSE_N_TIMES(
              writer,
              n,
              vlevel,
              dispatchAction,
              ... ) CVERBOSE_IF(writer, ELPP->validateNTimesCounter(__FILE__, __LINE__, n),
vlevel, dispatchAction, __VA_ARGS__)
```

Definition at line 4195 of file easylogging++.h.

9.3.1.70 CVLOG

```
#define CVLOG(
              vlevel,
              ... ) CVERBOSE(el::base::Writer, vlevel, el::base::DispatchAction::NormalLog, ↩
__VA_ARGS__)
```

Definition at line 4219 of file easylogging++.h.

9.3.1.71 CVLOG_AFTER_N

```
#define CVLOG_AFTER_N(
              n,
              vlevel,
              ... ) CVERBOSE_AFTER_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4232 of file easylogging++.h.

9.3.1.72 CVLOG_EVERY_N

```
#define CVLOG_EVERY_N(
              n,
              vlevel,
              ... ) CVERBOSE_EVERY_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4228 of file easylogging++.h.

9.3.1.73 CVLOG_IF

```
#define CVLOG_IF(
              condition,
              vlevel,
              ... ) CVERBOSE_IF(el::base::Writer, condition, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4223 of file easylogging++.h.

9.3.1.74 CVLOG_N_TIMES

```
#define CVLOG_N_TIMES(
              n,
              vlevel,
              ... ) CVERBOSE_N_TIMES(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4236 of file easylogging++.h.

9.3.1.75 CWARNING

```
#define CWARNING(
              writer,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG(writer, el::Level::Warning, dispatchAction, __VA_ARGS__)
```

Definition at line 3998 of file easylogging++.h.

9.3.1.76 CWARNING_AFTER_N

```
#define CWARNING_AFTER_N(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Warning, dispatchAction, __↩
VA_ARGS__)
```

Definition at line 4122 of file easylogging++.h.

9.3.1.77 CWARNING_EVERY_N

```
#define CWARNING_EVERY_N(
              writer,
              occasion,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Warning, dispatch↩
Action, __VA_ARGS__)
```

Definition at line 4079 of file easylogging++.h.

9.3.1.78 CWARNING_IF

```
#define CWARNING_IF(
              writer,
              condition_,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Warning, dispatch↩
Action, __VA_ARGS__)
```

Definition at line 4036 of file easylogging++.h.

9.3.1.79 CWARNING_N_TIMES

```
#define CWARNING_N_TIMES(
              writer,
              n,
              dispatchAction,
              ...  ) ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Warning, dispatchAction, __↩
VA_ARGS__)
```

Definition at line 4165 of file easylogging++.h.

9.3.1.80 DCCHECK

```
#define DCCHECK(
              condition,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK(condition, __VA_ARGS__)
```

Definition at line 4503 of file easylogging++.h.

9.3.1.81 DCCHECK_BOUNDS

```
#define DCCHECK_BOUNDS(
              val,
              min,
              max,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_BOUNDS(val, min, max, __VA_ARGS__)
```

Definition at line 4510 of file easylogging++.h.

9.3.1.82 DCCHECK_EQ

```
#define DCCHECK_EQ(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_EQ(a, b, __VA_ARGS__)
```

Definition at line 4504 of file easylogging++.h.

9.3.1.83 DCCHECK_GE

```
#define DCCHECK_GE(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_GE(a, b, __VA_ARGS__)
```

Definition at line 4509 of file easylogging++.h.

9.3.1.84 DCCHECK_GT

```
#define DCCHECK_GT(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_GT(a, b, __VA_ARGS__)
```

Definition at line 4507 of file easylogging++.h.

9.3.1.85 DCCHECK_LE

```
#define DCCHECK_LE(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_LE(a, b, __VA_ARGS__)
```

Definition at line 4508 of file easylogging++.h.

9.3.1.86 DCCHECK_LT

```
#define DCCHECK_LT(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_LT(a, b, __VA_ARGS__)
```

Definition at line 4506 of file easylogging++.h.

9.3.1.87 DCCHECK_NE

```
#define DCCHECK_NE(
              a,
              b,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_NE(a, b, __VA_ARGS__)
```

Definition at line 4505 of file easylogging++.h.

9.3.1.88 DCCHECK_NOTNULL

```
#define DCCHECK_NOTNULL(
              ptr,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_NOTNULL((ptr), __VA_ARGS__)
```

Definition at line 4511 of file easylogging++.h.

9.3.1.89 DCCHECK_STRCASEEQ

```
#define DCCHECK_STRCASEEQ(
              str1,
              str2,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_STRCASEEQ(str1, str2, __VA_ARGS__)
```

Definition at line 4514 of file easylogging++.h.

9.3.1.90 DCCHECK_STRCASENE

```
#define DCCHECK_STRCASENE(
              str1,
              str2,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_STRCASENE(str1, str2, __VA_ARGS__)
```

Definition at line 4515 of file easylogging++.h.

9.3.1.91 DCCHECK_STREQ

```
#define DCCHECK_STREQ(
              str1,
              str2,
              ...  ) if (ELPP_DEBUG_LOG) CCHECK_STREQ(str1, str2, __VA_ARGS__)
```

Definition at line 4512 of file easylogging++.h.

9.3.1.92 DCCHECK_STRNE

```
#define DCCHECK_STRNE(
              str1,
              str2,
              ... ) if (ELPP_DEBUG_LOG) CCHECK_STRNE(str1, str2, __VA_ARGS__)
```

Definition at line 4513 of file easylogging++.h.

9.3.1.93 DCHECK

```
#define DCHECK(
              condition ) DCCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4517 of file easylogging++.h.

9.3.1.94 DCHECK_BOUNDS

```
#define DCHECK_BOUNDS(
              val,
              min,
              max ) DCCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4524 of file easylogging++.h.

9.3.1.95 DCHECK_EQ

```
#define DCHECK_EQ(
              a,
              b ) DCCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4518 of file easylogging++.h.

9.3.1.96 DCHECK_GE

```
#define DCHECK_GE(
              a,
              b ) DCCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4523 of file easylogging++.h.

9.3.1.97 DCHECK_GT

```
#define DCHECK_GT(
              a,
              b ) DCCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4521 of file easylogging++.h.

9.3.1.98 DCHECK_LE

```
#define DCHECK_LE(
              a,
              b ) DCCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4522 of file easylogging++.h.

9.3.1.99 DCHECK_LT

```
#define DCHECK_LT(
              a,
              b ) DCCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4520 of file easylogging++.h.

9.3.1.100 DCHECK_NE

```
#define DCHECK_NE(
              a,
              b ) DCCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4519 of file easylogging++.h.

9.3.1.101 DCHECK_NOTNULL

```
#define DCHECK_NOTNULL(
              ptr ) DCCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4525 of file easylogging++.h.

9.3.1.102 DCHECK_STRCASEEQ

```
#define DCHECK_STRCASEEQ(
              str1,
              str2 ) DCCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4528 of file easylogging++.h.

9.3.1.103 DCHECK_STRCASENE

```
#define DCHECK_STRCASENE(
              str1,
              str2 ) DCCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4529 of file easylogging++.h.

9.3.1.104 DCHECK_STREQ

```
#define DCHECK_STREQ(
              str1,
              str2 ) DCCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4526 of file easylogging++.h.

9.3.1.105 DCHECK_STRNE

```
#define DCHECK_STRNE(
              str1,
              str2 ) DCCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4527 of file easylogging++.h.

9.3.1.106 DCLOG

```
#define DCLOG(
              LEVEL,
              ...  ) if (ELPP_DEBUG_LOG) CLOG(LEVEL, __VA_ARGS__)
```

Definition at line 4379 of file easylogging++.h.

9.3.1.107 DCLOG_AFTER_N

```
#define DCLOG_AFTER_N(
              n,
              LEVEL,
              ...  ) if (ELPP_DEBUG_LOG) CLOG_AFTER_N(n, LEVEL, __VA_ARGS__)
```

Definition at line 4388 of file easylogging++.h.

9.3.1.108 DCLOG_EVERY_N

```
#define DCLOG_EVERY_N(
              n,
              LEVEL,
              ...  ) if (ELPP_DEBUG_LOG) CLOG_EVERY_N(n, LEVEL, __VA_ARGS__)
```

Definition at line 4386 of file easylogging++.h.

9.3.1.109 DCLOG_IF

```
#define DCLOG_IF(
              condition,
              LEVEL,
              ...  ) if (ELPP_DEBUG_LOG) CLOG_IF(condition, LEVEL, __VA_ARGS__)
```

Definition at line 4383 of file easylogging++.h.

9.3.1.110 DCLOG_N_TIMES

```
#define DCLOG_N_TIMES(
              n,
              LEVEL,
              ... ) if (ELPP_DEBUG_LOG) CLOG_N_TIMES(n, LEVEL, __VA_ARGS__)
```

Definition at line 4390 of file easylogging++.h.

9.3.1.111 DCLOG_VERBOSE

```
#define DCLOG_VERBOSE(
              vlevel,
              ... ) if (ELPP_DEBUG_LOG) CLOG_VERBOSE(vlevel, __VA_ARGS__)
```

Definition at line 4380 of file easylogging++.h.

9.3.1.112 DCPCHECK

```
#define DCPCHECK(
              condition,
              ... ) if (ELPP_DEBUG_LOG) CPCHECK(condition, __VA_ARGS__)
```

Definition at line 4516 of file easylogging++.h.

9.3.1.113 DCPLOG

```
#define DCPLOG(
              LEVEL,
              ... ) if (ELPP_DEBUG_LOG) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
```

Definition at line 4286 of file easylogging++.h.

9.3.1.114 DCPLOG_IF

```
#define DCPLOG_IF(
              condition,
              LEVEL,
              ... ) C##LEVEL##_IF(el::base::PErrorWriter, (ELPP_DEBUG_LOG) && (condition),
el::base::DispatchAction::NormalLog, __VA_ARGS__)
```

Definition at line 4288 of file easylogging++.h.

9.3.1.115 DCSYSLOG

```
#define DCSYSLOG(
              LEVEL,
              ... ) el::base::NullWriter()
```

Definition at line 4353 of file easylogging++.h.

9.3.1.116 DCSYSLOG_AFTER_N

```
#define DCSYSLOG_AFTER_N(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4356 of file easylogging++.h.

9.3.1.117 DCSYSLOG_EVERY_N

```
#define DCSYSLOG_EVERY_N(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4355 of file easylogging++.h.

9.3.1.118 DCSYSLOG_IF

```
#define DCSYSLOG_IF(
              condition,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4354 of file easylogging++.h.

9.3.1.119 DCSYSLOG_N_TIMES

```
#define DCSYSLOG_N_TIMES(
              n,
              LEVEL,
              ...  ) el::base::NullWriter()
```

Definition at line 4357 of file easylogging++.h.

9.3.1.120 DCVLOG

```
#define DCVLOG(
              vlevel,
              ...  ) if (ELPP_DEBUG_LOG) CVLOG(vlevel, __VA_ARGS__)
```

Definition at line 4381 of file easylogging++.h.

9.3.1.121 DCVLOG_AFTER_N

```
#define DCVLOG_AFTER_N(
              n,
              vlevel,
              ...  ) if (ELPP_DEBUG_LOG) CVLOG_AFTER_N(n, vlevel, __VA_ARGS__)
```

Definition at line 4389 of file easylogging++.h.

9.3.1.122 DCVLOG_EVERY_N

```
#define DCVLOG_EVERY_N(
              n,
              vlevel,
              ...  ) if (ELPP_DEBUG_LOG) CVLOG_EVERY_N(n, vlevel, __VA_ARGS__)
```

Definition at line 4387 of file easylogging++.h.

9.3.1.123 DCVLOG_IF

```
#define DCVLOG_IF(
              condition,
              vlevel,
              ...  ) if (ELPP_DEBUG_LOG) CVLOG_IF(condition, vlevel, __VA_ARGS__)
```

Definition at line 4384 of file easylogging++.h.

9.3.1.124 DCVLOG_N_TIMES

```
#define DCVLOG_N_TIMES(
              n,
              vlevel,
              ...  ) if (ELPP_DEBUG_LOG) CVLOG_N_TIMES(n, vlevel, __VA_ARGS__)
```

Definition at line 4391 of file easylogging++.h.

9.3.1.125 DLOG

```
#define DLOG(
              LEVEL ) DCLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4408 of file easylogging++.h.

9.3.1.126 DLOG_AFTER_N

```
#define DLOG_AFTER_N(
              n,
              LEVEL ) DCLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4416 of file easylogging++.h.

9.3.1.127 DLOG_EVERY_N

```
#define DLOG_EVERY_N(
              n,
              LEVEL ) DCLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4414 of file easylogging++.h.

9.3.1.128 DLOG_IF

```
#define DLOG_IF(
              condition,
              LEVEL ) DCLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4411 of file easylogging++.h.

9.3.1.129 DLOG_N_TIMES

```
#define DLOG_N_TIMES(
              n,
              LEVEL ) DCLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4418 of file easylogging++.h.

9.3.1.130 DPCHECK

```
#define DPCHECK(
              condition ) DCPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4530 of file easylogging++.h.

9.3.1.131 DPLOG

```
#define DPLOG(
              LEVEL ) DCPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4292 of file easylogging++.h.

9.3.1.132 DPLOG_IF

```
#define DPLOG_IF(
              condition,
              LEVEL ) DCPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4293 of file easylogging++.h.

9.3.1.133 DSYSLOG

```
#define DSYSLOG(
              LEVEL ) el::base::NullWriter()
```

Definition at line 4358 of file easylogging++.h.

9.3.1.134 DSYSLOG_AFTER_N

```
#define DSYSLOG_AFTER_N(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4361 of file easylogging++.h.

9.3.1.135 DSYSLOG_EVERY_N

```
#define DSYSLOG_EVERY_N(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4360 of file easylogging++.h.

9.3.1.136 DSYSLOG_IF

```
#define DSYSLOG_IF(
              condition,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4359 of file easylogging++.h.

9.3.1.137 DSYSLOG_N_TIMES

```
#define DSYSLOG_N_TIMES(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4362 of file easylogging++.h.

9.3.1.138 DVLOG

```
#define DVLOG(
              vlevel ) DCVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4409 of file easylogging++.h.

9.3.1.139 DVLOG_AFTER_N

```
#define DVLOG_AFTER_N(
              n,
              vlevel ) DCVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4417 of file easylogging++.h.

9.3.1.140 DVLOG_EVERY_N

```
#define DVLOG_EVERY_N(
              n,
              vlevel ) DCVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4415 of file easylogging++.h.

9.3.1.141 DVLOG_IF

```
#define DVLOG_IF(
              condition,
              vlevel ) DCVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4412 of file easylogging++.h.

9.3.1.142 DVLOG_N_TIMES

```
#define DVLOG_N_TIMES(
              n,
              vlevel ) DCVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4419 of file easylogging++.h.

9.3.1.143 el_getVALength

```
#define el_getVALength(
              ...  ) el_resolveVALength(0, ## __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
```

Definition at line 3391 of file easylogging++.h.

9.3.1.144 el_resolveVALength

```
#define el_resolveVALength(
              _0,
              _1,
              _2,
              _3,
              _4,
              _5,
              _6,
              _7,
              _8,
              _9,
              _10,
              N,
              ...  ) N
```

Definition at line 3394 of file easylogging++.h.

9.3.1.145 ELPP

```
#define ELPP el::base::elStorage
```

Definition at line 2725 of file easylogging++.h.

9.3.1.146 ELPP_ASSERT

```
#define ELPP_ASSERT(
                expr,
                msg )
```

Value:

```
if (!(expr)) { \
std::stringstream internalInfoStream; internalInfoStream « msg; \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR\
« "ASSERTION FAILURE FROM EASYLOGGING++ (LINE: " \
« __LINE__ « ") [" #expr « "] WITH MESSAGE \"" « ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) «
      "\"" \
« ELPP_INTERNAL_DEBUGGING_ENDL; }
```

Definition at line 163 of file easylogging++.h.

9.3.1.147 ELPP_ASYNC_LOGGING

```
#define ELPP_ASYNC_LOGGING 0
```

Definition at line 274 of file easylogging++.h.

9.3.1.148 ELPP_COMPILER_CLANG

```
#define ELPP_COMPILER_CLANG 0
```

Definition at line 50 of file easylogging++.h.

9.3.1.149 ELPP_COMPILER_GCC

```
#define ELPP_COMPILER_GCC 0
```

Definition at line 22 of file easylogging++.h.

9.3.1.150 ELPP_COMPILER_INTEL

```
#define ELPP_COMPILER_INTEL 0
```

Definition at line 73 of file easylogging++.h.

9.3.1.151 ELPP_COMPILER_MSVC

```
#define ELPP_COMPILER_MSVC 0
```

Definition at line 36 of file easylogging++.h.

9.3.1.152 ELPP_COUNTER

```
#define ELPP_COUNTER (ELPP->hitCounters()->getCounter(__FILE__, __LINE__))
```

Gets hit counter for file/line.

Definition at line 3944 of file easylogging++.h.

9.3.1.153 ELPP_COUNTER_POS

```
#define ELPP_COUNTER_POS (ELPP_COUNTER == nullptr ?  -1 :  ELPP_COUNTER->hitCounts())
```

Gets hit counter position for file/line, -1 if not registered yet.

Definition at line 3946 of file easylogging++.h.

9.3.1.154 ELPP_COUT

```
#define ELPP_COUT std::cout
```

Definition at line 526 of file easylogging++.h.

9.3.1.155 ELPP_COUT_LINE

```
#define ELPP_COUT_LINE(
              logLine ) logLine << std::flush
```

Definition at line 537 of file easylogging++.h.

9.3.1.156 ELPP_CRASH_HANDLER_INIT

```
#define ELPP_CRASH_HANDLER_INIT
```

Definition at line 4537 of file easylogging++.h.

9.3.1.157 ELPP_CRT_DBG_WARNINGS

```
#define ELPP_CRT_DBG_WARNINGS ELPP_COMPILER_MSVC
```

Definition at line 38 of file easylogging++.h.

9.3.1.158 ELPP_CURR_FILE_LOGGER_ID

```
#define ELPP_CURR_FILE_LOGGER_ID el::base::consts::kDefaultLoggerId
```

Definition at line 4256 of file easylogging++.h.

9.3.1.159 ELPP_CYGWIN

```
#define ELPP_CYGWIN 0
```

Definition at line 68 of file easylogging++.h.

9.3.1.160 ELPP_DEBUG_LOG

```
#define ELPP_DEBUG_LOG 1
```

Definition at line 309 of file easylogging++.h.

9.3.1.161 ELPP_ERROR_LOG

```
#define ELPP_ERROR_LOG 1
```

Definition at line 324 of file easylogging++.h.

9.3.1.162 ELPP_EXPORT

```
#define ELPP_EXPORT
```

Definition at line 235 of file easylogging++.h.

9.3.1.163 ELPP_FATAL_LOG

```
#define ELPP_FATAL_LOG 1
```

Definition at line 329 of file easylogging++.h.

9.3.1.164 ELPP_FINAL

```
#define ELPP_FINAL
```

Definition at line 267 of file easylogging++.h.

9.3.1.165 ELPP_FUNC

```
#define ELPP_FUNC ""
```

Definition at line 295 of file easylogging++.h.

9.3.1.166 ELPP_INFO_LOG

```
#define ELPP_INFO_LOG 1
```

Definition at line 314 of file easylogging++.h.

9.3.1.167 ELPP_INIT_EASYLOGGINGPP

```
#define ELPP_INIT_EASYLOGGINGPP(
              val )
```

Value:

```
namespace el { \
namespace base { \
el::base::type::StoragePointer elStorage(val); \
} \
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER); \
}
```

Definition at line 4538 of file easylogging++.h.

9.3.1.168 ELPP_INITIALIZE_SYSLOG

```
#define ELPP_INITIALIZE_SYSLOG(
              id,
              opt,
              fac ) el::SysLogInitializer elSyslogInit(id, opt, fac)
```

Definition at line 3651 of file easylogging++.h.

9.3.1.169 ELPP_INTERNAL_DEBUGGING_ENDL

```
#define ELPP_INTERNAL_DEBUGGING_ENDL std::endl
```

Definition at line 148 of file easylogging++.h.

9.3.1.170 ELPP_INTERNAL_DEBUGGING_MSG

```
#define ELPP_INTERNAL_DEBUGGING_MSG(
              msg ) msg
```

Definition at line 151 of file easylogging++.h.

9.3.1.171 ELPP_INTERNAL_DEBUGGING_OUT_ERROR

```
#define ELPP_INTERNAL_DEBUGGING_OUT_ERROR std::cerr
```

Definition at line 145 of file easylogging++.h.

9.3.1.172 ELPP_INTERNAL_DEBUGGING_OUT_INFO

```
#define ELPP_INTERNAL_DEBUGGING_OUT_INFO std::cout
```

Definition at line 142 of file easylogging++.h.

9.3.1.173 ELPP_INTERNAL_DEBUGGING_WRITE_PERROR

```
#define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR ELPP_INTERNAL_DEBUGGING_OUT_ERROR << ":  " <<
strerror(errno) << " [" << errno << "]"; (void)0
```

Definition at line 178 of file easylogging++.h.

9.3.1.174 ELPP_INTERNAL_ERROR

```
#define ELPP_INTERNAL_ERROR(
              msg,
              pe )
```

Definition at line 192 of file easylogging++.h.

9.3.1.175 ELPP_INTERNAL_INFO

```
#define ELPP_INTERNAL_INFO(
              lvl,
              msg )
```

Definition at line 206 of file easylogging++.h.

9.3.1.176 ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG

```
#define ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(
              temp )
```

Value:
```
template <typename T1, typename T2, typename T3, typename T4, typename T5>                    \
inline MessageBuilder& operator«(const temp<T1, T2, T3, T4, T5>& template_inst) {            \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
```

Definition at line 2922 of file easylogging++.h.

9.3.1.177 ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG

```
#define ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(
              temp )
```

Value:
```
template <typename T1, typename T2, typename T3, typename T4>                                 \
inline MessageBuilder& operator«(const temp<T1, T2, T3, T4>& template_inst) {                \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
```

Definition at line 2917 of file easylogging++.h.

9.3.1.178 ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG

```
#define ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(
              temp )
```

Value:
```
template <typename T>                                                                    \
inline MessageBuilder& operator«(const temp<T>& template_inst) {                        \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());     \
}
```

Definition at line 2902 of file easylogging++.h.

9.3.1.179 ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG

```
#define ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(
              temp )
```

Value:
```
template <typename T1, typename T2, typename T3>                                         \
inline MessageBuilder& operator«(const temp<T1, T2, T3>& template_inst) {               \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());     \
}
```

Definition at line 2912 of file easylogging++.h.

9.3.1.180 ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG

```
#define ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(
              temp )
```

Value:
```
template <typename T1, typename T2>                                                      \
inline MessageBuilder& operator«(const temp<T1, T2>& template_inst) {                   \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());     \
}
```

Definition at line 2907 of file easylogging++.h.

9.3.1.181 ELPP_LITERAL

```
#define ELPP_LITERAL(
              txt ) txt
```

Definition at line 521 of file easylogging++.h.

9.3.1.182 ELPP_LOGGING_ENABLED

```
#define ELPP_LOGGING_ENABLED 1
```

Definition at line 306 of file easylogging++.h.

9.3.1.183 ELPP_MIN_UNIT

```
#define ELPP_MIN_UNIT el::base::TimestampUnit::Millisecond
```

Definition at line 3915 of file easylogging++.h.

9.3.1.184 ELPP_MINGW

```
#define ELPP_MINGW 0
```

Definition at line 63 of file easylogging++.h.

9.3.1.185 ELPP_OS_AIX

```
#define ELPP_OS_AIX 0
```

Definition at line 106 of file easylogging++.h.

9.3.1.186 ELPP_OS_ANDROID

```
#define ELPP_OS_ANDROID 0
```

Definition at line 132 of file easylogging++.h.

9.3.1.187 ELPP_OS_EMSCRIPTEN

```
#define ELPP_OS_EMSCRIPTEN 0
```

Definition at line 116 of file easylogging++.h.

9.3.1.188 ELPP_OS_FREEBSD

```
#define ELPP_OS_FREEBSD 0
```

Definition at line 96 of file easylogging++.h.

9.3.1.189 ELPP_OS_LINUX

```
#define ELPP_OS_LINUX 0
```

Definition at line 86 of file easylogging++.h.

9.3.1.190 ELPP_OS_MAC

```
#define ELPP_OS_MAC 0
```

Definition at line 91 of file easylogging++.h.

9.3.1.191 ELPP_OS_NETBSD

```
#define ELPP_OS_NETBSD 0
```

Definition at line 111 of file easylogging++.h.

9.3.1.192 ELPP_OS_QNX

```
#define ELPP_OS_QNX 0
```

Definition at line 121 of file easylogging++.h.

9.3.1.193 ELPP_OS_SOLARIS

```
#define ELPP_OS_SOLARIS 0
```

Definition at line 101 of file easylogging++.h.

9.3.1.194 ELPP_OS_UNIX

```
#define ELPP_OS_UNIX 0
```

Definition at line 127 of file easylogging++.h.

9.3.1.195 ELPP_OS_WINDOWS

```
#define ELPP_OS_WINDOWS 0
```

Definition at line 80 of file easylogging++.h.

9.3.1.196 ELPP_SIMPLE_LOG

```
#define ELPP_SIMPLE_LOG(
              LOG_TYPE )
```

Value:
```
MessageBuilder& operator«(LOG_TYPE msg) {\
m_logger->stream() « msg;\
if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {\
m_logger->stream() « " ";\
}\
return *this;\
}
```

Definition at line 2867 of file easylogging++.h.

9.3.1.197 ELPP_STACKTRACE

```
#define ELPP_STACKTRACE 0
```

Definition at line 220 of file easylogging++.h.

9.3.1.198 ELPP_STRLEN

```
#define ELPP_STRLEN strlen
```

Definition at line 522 of file easylogging++.h.

9.3.1.199 ELPP_THREADING_ENABLED

```
#define ELPP_THREADING_ENABLED 0
```

Definition at line 279 of file easylogging++.h.

9.3.1.200 ELPP_TRACE

```
#define ELPP_TRACE CLOG(TRACE, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4259 of file easylogging++.h.

9.3.1.201 ELPP_TRACE_LOG

```
#define ELPP_TRACE_LOG 1
```

Definition at line 334 of file easylogging++.h.

9.3.1.202 ELPP_UNUSED

```
#define ELPP_UNUSED(
              x ) (void)x
```

Definition at line 223 of file easylogging++.h.

9.3.1.203 ELPP_USE_DEF_CRASH_HANDLER

```
#define ELPP_USE_DEF_CRASH_HANDLER true
```

Definition at line 4535 of file easylogging++.h.

9.3.1.204 ELPP_USE_STD_THREADING

```
#define ELPP_USE_STD_THREADING 0
```

Definition at line 262 of file easylogging++.h.

9.3.1.205 ELPP_VARIADIC_TEMPLATES_SUPPORTED

```
#define ELPP_VARIADIC_TEMPLATES_SUPPORTED (ELPP_COMPILER_GCC || ELPP_COMPILER_CLANG || ELPP_COMPILER_INTEL
|| (ELPP_COMPILER_MSVC && _MSC_VER >= 1800))
```

Definition at line 300 of file easylogging++.h.

9.3.1.206 ELPP_VERBOSE_LOG

```
#define ELPP_VERBOSE_LOG 1
```

Definition at line 339 of file easylogging++.h.

9.3.1.207 ELPP_WARNING_LOG

```
#define ELPP_WARNING_LOG 1
```

Definition at line 319 of file easylogging++.h.

9.3.1.208 ELPP_WRITE_LOG

```
#define ELPP_WRITE_LOG(
              writer,
              level,
              dispatchAction,
              ...  ) writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(←
__VA_ARGS__), __VA_ARGS__)
```

Definition at line 3395 of file easylogging++.h.

9.3.1.209 ELPP_WRITE_LOG_AFTER_N

```
#define ELPP_WRITE_LOG_AFTER_N(
              writer,
              n,
              level,
              dispatchAction,
              ...  )
```

Value:
```
ELPP->validateAfterNCounter(__FILE__, __LINE__, n) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
      __VA_ARGS__)
```

Definition at line 3402 of file easylogging++.h.

9.3.1.210 ELPP_WRITE_LOG_EVERY_N

```
#define ELPP_WRITE_LOG_EVERY_N(
                writer,
                occasion,
                level,
                dispatchAction,
                ... )
```

Value:

```
ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
      __VA_ARGS__)
```

Definition at line 3399 of file easylogging++.h.

9.3.1.211 ELPP_WRITE_LOG_IF

```
#define ELPP_WRITE_LOG_IF(
                writer,
                condition,
                level,
                dispatchAction,
                ... )
```

Value:

```
if (condition) \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
      __VA_ARGS__)
```

Definition at line 3397 of file easylogging++.h.

9.3.1.212 ELPP_WRITE_LOG_N_TIMES

```
#define ELPP_WRITE_LOG_N_TIMES(
                writer,
                n,
                level,
                dispatchAction,
                ... )
```

Value:

```
ELPP->validateNTimesCounter(__FILE__, __LINE__, n) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
      __VA_ARGS__)
```

Definition at line 3405 of file easylogging++.h.

9.3.1.213 ELPP_WX_ENABLED

```
#define ELPP_WX_ENABLED(
                ContainerType )
```

Definition at line 3137 of file easylogging++.h.

9.3.1.214 ELPP_WX_HASH_MAP_ENABLED

```
#define ELPP_WX_HASH_MAP_ENABLED(
            ContainerType )
```

Definition at line 3138 of file easylogging++.h.

9.3.1.215 ELPP_WX_PTR_ENABLED

```
#define ELPP_WX_PTR_ENABLED(
            ContainerType )
```

Definition at line 3136 of file easylogging++.h.

9.3.1.216 elpptime

```
#define elpptime localtime
```

Definition at line 467 of file easylogging++.h.

9.3.1.217 elpptime_r

```
#define elpptime_r localtime_r
```

Definition at line 465 of file easylogging++.h.

9.3.1.218 elpptime_s

```
#define elpptime_s localtime_s
```

Definition at line 466 of file easylogging++.h.

9.3.1.219 INITIALIZE_EASYLOGGINGPP

```
#define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new
el::base::DefaultLogBuilder())))
```

Definition at line 4550 of file easylogging++.h.

9.3.1.220 INITIALIZE_NULL_EASYLOGGINGPP

```
#define INITIALIZE_NULL_EASYLOGGINGPP
```

Value:

```
namespace el {\
namespace base {\
el::base::type::StoragePointer elStorage;\
}\
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
}
```

Definition at line 4552 of file easylogging++.h.

9.3.1.221 LOG

```
#define LOG(
              LEVEL ) CLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4261 of file easylogging++.h.

9.3.1.222 LOG_AFTER_N

```
#define LOG_AFTER_N(
              n,
              LEVEL ) CLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4269 of file easylogging++.h.

9.3.1.223 LOG_EVERY_N

```
#define LOG_EVERY_N(
              n,
              LEVEL ) CLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4267 of file easylogging++.h.

9.3.1.224 LOG_IF

```
#define LOG_IF(
              condition,
              LEVEL ) CLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4264 of file easylogging++.h.

9.3.1.225 LOG_N_TIMES

```
#define LOG_N_TIMES(
              n,
              LEVEL ) CLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4271 of file easylogging++.h.

9.3.1.226 MAKE_CONTAINERELPP_FRIENDLY

```
#define MAKE_CONTAINERELPP_FRIENDLY(
                ContainerType,
                SizeMethod,
                ElementInstance )
```

Value:

```
el::base::type::ostream_t& operator«(el::base::type::ostream_t& ss, const ContainerType& container) {\
const el::base::type::char_t* sep = ELPP->hasFlag(el::LoggingFlag::NewLineForContainer) ? \
ELPP_LITERAL("\n    ") : ELPP_LITERAL(", ");\
ContainerType::const_iterator elem = container.begin();\
ContainerType::const_iterator endElem = container.end();\
std::size_t size_ = container.SizeMethod; \
ss « ELPP_LITERAL("[");\
for (std::size_t i = 0; elem != endElem && i < el::base::consts::kMaxLogPerContainer; ++i, ++elem) { \
ss « ElementInstance;\
ss « ((i < size_ - 1) ? sep : ELPP_LITERAL(""));\
}\
if (elem != endElem) {\
ss « ELPP_LITERAL("...");\
}\
ss « ELPP_LITERAL("]");\
return ss;\
}
```

Macro used internally that can be used externally to make containers easylogging++ friendly.

@detail This macro expands to write an ostream& operator<< for container. This container is expected to have begin() and end() methods that return respective iterators

Parameters

ContainerType	Type of container e.g, MyList from WX_DECLARE_LIST(int, MyList); in wxwidgets
SizeMethod	Method used to get size of container.
ElementInstance	Instance of element to be fed out. Instance name is "elem". See WXELPP_ENABLED macro for an example usage

Definition at line 3111 of file easylogging++.h.

9.3.1.227 MAKE_LOGGABLE

```
#define MAKE_LOGGABLE(
                ClassType,
                ClassInstance,
                OutputStreamInstance ) el::base::type::ostream_t& operator<<(el::base::type::ostream_t&
OutputStreamInstance, const ClassType& ClassInstance)
```

Definition at line 3630 of file easylogging++.h.

9.3.1.228 PCHECK

```
#define PCHECK(
                condition ) CPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4450 of file easylogging++.h.

9.3.1.229 PERFORMANCE_CHECKPOINT

```
#define PERFORMANCE_CHECKPOINT(
              obj ) obj->checkpoint(std::string(), __FILE__, __LINE__, ELPP_FUNC)
```

Definition at line 3939 of file easylogging++.h.

9.3.1.230 PERFORMANCE_CHECKPOINT_WITH_ID

```
#define PERFORMANCE_CHECKPOINT_WITH_ID(
              obj,
              id ) obj->checkpoint(id, __FILE__, __LINE__, ELPP_FUNC)
```

Definition at line 3940 of file easylogging++.h.

9.3.1.231 PLOG

```
#define PLOG(
              LEVEL ) CPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4290 of file easylogging++.h.

9.3.1.232 PLOG_IF

```
#define PLOG_IF(
              condition,
              LEVEL ) CPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4291 of file easylogging++.h.

9.3.1.233 SHARE_EASYLOGGINGPP

```
#define SHARE_EASYLOGGINGPP(
              initializedStorage )
```

Value:

```
namespace el {\
namespace base {\
el::base::type::StoragePointer elStorage(initializedStorage);\
}\
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
}
```

Definition at line 4559 of file easylogging++.h.

9.3.1.234 START_EASYLOGGINGPP

```
#define START_EASYLOGGINGPP(
              argc,
              argv ) el::Helpers::setArgs(argc, argv)
```

Definition at line 4570 of file easylogging++.h.

9.3.1.235 STRCAT

```
#define STRCAT(
              a,
              b,
              len ) strcat(a, b)
```

Definition at line 250 of file easylogging++.h.

9.3.1.236 STRCPY

```
#define STRCPY(
              a,
              b,
              len ) strcpy(a, b)
```

Definition at line 251 of file easylogging++.h.

9.3.1.237 STRERROR

```
#define STRERROR(
              a,
              b,
              c ) strerror(c)
```

Definition at line 249 of file easylogging++.h.

9.3.1.238 STRTOK

```
#define STRTOK(
              a,
              b,
              c ) strtok(a, b)
```

Definition at line 248 of file easylogging++.h.

9.3.1.239 SYSLOG

```
#define SYSLOG(
              LEVEL ) el::base::NullWriter()
```

Definition at line 4348 of file easylogging++.h.

9.3.1.240 SYSLOG_AFTER_N

```
#define SYSLOG_AFTER_N(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4351 of file easylogging++.h.

9.3.1.241 SYSLOG_EVERY_N

```
#define SYSLOG_EVERY_N(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4350 of file easylogging++.h.

9.3.1.242 SYSLOG_IF

```
#define SYSLOG_IF(
              condition,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4349 of file easylogging++.h.

9.3.1.243 SYSLOG_N_TIMES

```
#define SYSLOG_N_TIMES(
              n,
              LEVEL ) el::base::NullWriter()
```

Definition at line 4352 of file easylogging++.h.

9.3.1.244 TIMED_BLOCK

```
#define TIMED_BLOCK(
              obj,
              blockName )
```

Value:

```
for (struct { int i; el::base::type::PerformanceTrackerPtr timer; } obj = { 0, \
el::base::type::PerformanceTrackerPtr(new el::base::PerformanceTracker(blockName, ELPP_MIN_UNIT)) }; obj.i
    < 1; ++obj.i)
```

Definition at line 3927 of file easylogging++.h.

9.3.1.245 TIMED_FUNC

```
#define TIMED_FUNC(
              obj ) TIMED_SCOPE(obj, ELPP_FUNC)
```

Definition at line 3936 of file easylogging++.h.

9.3.1.246 TIMED_FUNC_IF

```
#define TIMED_FUNC_IF(
              obj,
              condition ) TIMED_SCOPE_IF(obj, ELPP_FUNC, condition)
```

Performance tracked function. Performance gets written when goes out of scope using 'performance' logger.

@detail Please note in order to check the performance at a certain time you can use obj->checkpoint();

See also

el::base::PerformanceTracker

el::base::PerformanceTracker::checkpoint

Definition at line 3935 of file easylogging++.h.

9.3.1.247 TIMED_SCOPE

```
#define TIMED_SCOPE(
              obj,
              blockname ) TIMED_SCOPE_IF(obj, blockname, true)
```

Definition at line 3926 of file easylogging++.h.

9.3.1.248 TIMED_SCOPE_IF

```
#define TIMED_SCOPE_IF(
              obj,
              blockname,
              condition )
```

Value:

```
el::base::type::PerformanceTrackerPtr obj( condition ? \
new el::base::PerformanceTracker(blockname, ELPP_MIN_UNIT) : nullptr )
```

Performance tracked scope. Performance gets written when goes out of scope using 'performance' logger.

@detail Please note in order to check the performance at a certain time you can use obj->checkpoint();

See also

el::base::PerformanceTracker

el::base::PerformanceTracker::checkpoint

Definition at line 3924 of file easylogging++.h.

9.3.1.249 VLOG

```
#define VLOG(
              vlevel ) CVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4262 of file easylogging++.h.

9.3.1.250 VLOG_AFTER_N

```
#define VLOG_AFTER_N(
              n,
              vlevel ) CVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4270 of file easylogging++.h.

9.3.1.251 VLOG_EVERY_N

```
#define VLOG_EVERY_N(
              n,
              vlevel ) CVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4268 of file easylogging++.h.

9.3.1.252 VLOG_IF

```
#define VLOG_IF(
              condition,
              vlevel ) CVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4265 of file easylogging++.h.

9.3.1.253 VLOG_IS_ON

```
#define VLOG_IS_ON(
              verboseLevel ) (ELPP->vRegistry()->allowed(verboseLevel, __FILE__))
```

Determines whether verbose logging is on for specified level current file.

Definition at line 3905 of file easylogging++.h.

9.3.1.254 VLOG_N_TIMES

```
#define VLOG_N_TIMES(
              n,
              vlevel ) CVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
```

Definition at line 4272 of file easylogging++.h.

9.4 easylogging++.h

Go to the documentation of this file.

```
//
//  Bismillah ar-Rahmaan ar-Raheem
//
//  Easylogging++ v9.97.1
//  Single-header only, cross-platform logging library for C++ applications
//
//  Copyright (c) 2012-present @abumq (Majid Q.)
//
//  This library is released under the MIT Licence.
//  https://github.com/amrayn/easyloggingpp/blob/master/LICENSE
//

#ifndef EASYLOGGINGPP_H
#define EASYLOGGINGPP_H
// Compilers and C++0x/C++11 Evaluation
#if __cplusplus >= 201103L
#  define ELPP_CXX11 1
#endif  // __cplusplus >= 201103L
#if (defined(__GNUC__))
#  define ELPP_COMPILER_GCC 1
#else
#  define ELPP_COMPILER_GCC 0
#endif
#if ELPP_COMPILER_GCC
#    define ELPP_GCC_VERSION (__GNUC__ * 10000 \
+ __GNUC_MINOR__ * 100 \
+ __GNUC_PATCHLEVEL__)
#  if defined(__GXX_EXPERIMENTAL_CXX0X__)
#    define ELPP_CXX0X 1
#  endif
#endif
// Visual C++
#if defined(_MSC_VER)
#  define ELPP_COMPILER_MSVC 1
#else
#  define ELPP_COMPILER_MSVC 0
#endif
#define ELPP_CRT_DBG_WARNINGS ELPP_COMPILER_MSVC
#if ELPP_COMPILER_MSVC
#  if (_MSC_VER == 1600)
#    define ELPP_CXX0X 1
#  elif(_MSC_VER >= 1700)
#    define ELPP_CXX11 1
#  endif
#endif
// Clang++
#if (defined(__clang__) && (__clang__ == 1))
#  define ELPP_COMPILER_CLANG 1
#else
#  define ELPP_COMPILER_CLANG 0
#endif
#if ELPP_COMPILER_CLANG
#  if __has_include(<thread>)
#    include <cstddef> // Make __GLIBCXX__ defined when using libstdc++
#    if !defined(__GLIBCXX__) || __GLIBCXX__ >= 20150426
#      define ELPP_CLANG_SUPPORTS_THREAD
#    endif // !defined(__GLIBCXX__) || __GLIBCXX__ >= 20150426
#  endif // __has_include(<thread>)
#endif
#if (defined(__MINGW32__) || defined(__MINGW64__))
#  define ELPP_MINGW 1
#else
#  define ELPP_MINGW 0
#endif
#if (defined(__CYGWIN__) && (__CYGWIN__ == 1))
#  define ELPP_CYGWIN 1
#else
#  define ELPP_CYGWIN 0
#endif
#if (defined(__INTEL_COMPILER))
#  define ELPP_COMPILER_INTEL 1
#else
#  define ELPP_COMPILER_INTEL 0
#endif
// Operating System Evaluation
// Windows
#if (defined(_WIN32) || defined(_WIN64))
#  define ELPP_OS_WINDOWS 1
#else
#  define ELPP_OS_WINDOWS 0
#endif
// Linux
```

```
#if (defined(__linux) || defined(__linux__))
#  define ELPP_OS_LINUX 1
#else
#  define ELPP_OS_LINUX 0
#endif
#if (defined(__APPLE__))
#  define ELPP_OS_MAC 1
#else
#  define ELPP_OS_MAC 0
#endif
#if (defined(__FreeBSD__) || defined(__FreeBSD_kernel__))
#  define ELPP_OS_FREEBSD 1
#else
#  define ELPP_OS_FREEBSD 0
#endif
#if (defined(__sun))
#  define ELPP_OS_SOLARIS 1
#else
#  define ELPP_OS_SOLARIS 0
#endif
#if (defined(_AIX))
#  define ELPP_OS_AIX 1
#else
#  define ELPP_OS_AIX 0
#endif
#if (defined(__NetBSD__))
#  define ELPP_OS_NETBSD 1
#else
#  define ELPP_OS_NETBSD 0
#endif
#if defined(__EMSCRIPTEN__)
#  define ELPP_OS_EMSCRIPTEN 1
#else
#  define ELPP_OS_EMSCRIPTEN 0
#endif
#if (defined(__QNX__) || defined(__QNXNTO__))
#  define ELPP_OS_QNX 1
#else
#  define ELPP_OS_QNX 0
#endif
// Unix
#if ((ELPP_OS_LINUX || ELPP_OS_MAC || ELPP_OS_FREEBSD || ELPP_OS_NETBSD || ELPP_OS_SOLARIS ||
ELPP_OS_AIX || ELPP_OS_EMSCRIPTEN || ELPP_OS_QNX) && (!ELPP_OS_WINDOWS))
#  define ELPP_OS_UNIX 1
#else
#  define ELPP_OS_UNIX 0
#endif
#if (defined(__ANDROID__))
#  define ELPP_OS_ANDROID 1
#else
#  define ELPP_OS_ANDROID 0
#endif
// Evaluating Cygwin as *nix OS
#if !ELPP_OS_UNIX && !ELPP_OS_WINDOWS && ELPP_CYGWIN
#  undef ELPP_OS_UNIX
#  undef ELPP_OS_LINUX
#  define ELPP_OS_UNIX 1
#  define ELPP_OS_LINUX 1
#endif //  !ELPP_OS_UNIX && !ELPP_OS_WINDOWS && ELPP_CYGWIN
#if !defined(ELPP_INTERNAL_DEBUGGING_OUT_INFO)
#  define ELPP_INTERNAL_DEBUGGING_OUT_INFO std::cout
#endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
#if !defined(ELPP_INTERNAL_DEBUGGING_OUT_ERROR)
#  define ELPP_INTERNAL_DEBUGGING_OUT_ERROR std::cerr
#endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
#if !defined(ELPP_INTERNAL_DEBUGGING_ENDL)
#  define ELPP_INTERNAL_DEBUGGING_ENDL std::endl
#endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
#if !defined(ELPP_INTERNAL_DEBUGGING_MSG)
#  define ELPP_INTERNAL_DEBUGGING_MSG(msg) msg
#endif // !defined(ELPP_INTERNAL_DEBUGGING_OUT)
// Internal Assertions and errors
#if !defined(ELPP_DISABLE_ASSERT)
#  if (defined(ELPP_DEBUG_ASSERT_FAILURE))
#    define ELPP_ASSERT(expr, msg) if (!(expr)) { \
std::stringstream internalInfoStream; internalInfoStream « msg; \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR \
« "EASYLOGGING++ ASSERTION FAILED (LINE: " « __LINE__ « ") [" #expr « "] WITH MESSAGE \"" \
« ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) « "\"" « ELPP_INTERNAL_DEBUGGING_ENDL;
base::utils::abort(1, \
"ELPP Assertion failure, please define ELPP_DEBUG_ASSERT_FAILURE"); }
#  else
#    define ELPP_ASSERT(expr, msg) if (!(expr)) { \
std::stringstream internalInfoStream; internalInfoStream « msg; \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR\
« "ASSERTION FAILURE FROM EASYLOGGING++ (LINE: " \
« __LINE__ « ") [" #expr « "] WITH MESSAGE \"" « ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str())
```

```
  « "\"" \
« ELPP_INTERNAL_DEBUGGING_ENDL; }
#  endif  // (defined(ELPP_DEBUG_ASSERT_FAILURE))
#else
#  define ELPP_ASSERT(x, y)
#endif  //(!defined(ELPP_DISABLE_ASSERT)
#if ELPP_COMPILER_MSVC
#  define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR \
{ char buff[256]; strerror_s(buff, 256, errno); \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR « ": " « buff « " [" « errno « "]";} (void)0
#else
#  define ELPP_INTERNAL_DEBUGGING_WRITE_PERROR \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR « ": " « strerror(errno) « " [" « errno « "]"; (void)0
#endif  // ELPP_COMPILER_MSVC
#if defined(ELPP_DEBUG_ERRORS)
#  if !defined(ELPP_INTERNAL_ERROR)
#    define ELPP_INTERNAL_ERROR(msg, pe) { \
std::stringstream internalInfoStream; internalInfoStream « "<ERROR> " « msg; \
ELPP_INTERNAL_DEBUGGING_OUT_ERROR \
« "ERROR FROM EASYLOGGING++ (LINE: " « __LINE__ « ") " \
« ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) « ELPP_INTERNAL_DEBUGGING_ENDL; \
if (pe) { ELPP_INTERNAL_DEBUGGING_OUT_ERROR « "    "; ELPP_INTERNAL_DEBUGGING_WRITE_PERROR; }} (void)0
#  endif
#else
#  undef ELPP_INTERNAL_INFO
#  define ELPP_INTERNAL_ERROR(msg, pe)
#endif  // defined(ELPP_DEBUG_ERRORS)
#if (defined(ELPP_DEBUG_INFO))
#  if !(defined(ELPP_INTERNAL_INFO_LEVEL))
#    define ELPP_INTERNAL_INFO_LEVEL 9
#  endif  // !(defined(ELPP_INTERNAL_INFO_LEVEL))
#  if !defined(ELPP_INTERNAL_INFO)
#    define ELPP_INTERNAL_INFO(lvl, msg) { if (lvl <= ELPP_INTERNAL_INFO_LEVEL) { \
std::stringstream internalInfoStream; internalInfoStream « "<INFO> " « msg; \
ELPP_INTERNAL_DEBUGGING_OUT_INFO « ELPP_INTERNAL_DEBUGGING_MSG(internalInfoStream.str()) \
« ELPP_INTERNAL_DEBUGGING_ENDL; }}
#  endif
#else
#  undef ELPP_INTERNAL_INFO
#  define ELPP_INTERNAL_INFO(lvl, msg)
#endif  // (defined(ELPP_DEBUG_INFO))
#if (defined(ELPP_FEATURE_ALL)) || (defined(ELPP_FEATURE_CRASH_LOG))
#  if (ELPP_COMPILER_GCC && !ELPP_MINGW && !ELPP_CYGWIN && !ELPP_OS_ANDROID && !ELPP_OS_EMSCRIPTEN &&
!ELPP_OS_QNX)
#    define ELPP_STACKTRACE 1
#  else
#      if ELPP_COMPILER_MSVC
#         pragma message("Stack trace not available for this compiler")
#      else
#         warning "Stack trace not available for this compiler";
#      endif  // ELPP_COMPILER_MSVC
#    define ELPP_STACKTRACE 0
#  endif  // ELPP_COMPILER_GCC
#else
#    define ELPP_STACKTRACE 0
#endif  // (defined(ELPP_FEATURE_ALL)) || (defined(ELPP_FEATURE_CRASH_LOG))
// Miscellaneous macros
#define ELPP_UNUSED(x) (void)x
#if ELPP_OS_UNIX
// Log file permissions for unix-based systems
#  define ELPP_LOG_PERMS S_IRUSR | S_IWUSR | S_IXUSR | S_IWGRP | S_IRGRP | S_IXGRP | S_IWOTH | S_IXOTH
#endif  // ELPP_OS_UNIX
#if defined(ELPP_AS_DLL) && ELPP_COMPILER_MSVC
#  if defined(ELPP_EXPORT_SYMBOLS)
#    define ELPP_EXPORT __declspec(dllexport)
#  else
#    define ELPP_EXPORT __declspec(dllimport)
#  endif  // defined(ELPP_EXPORT_SYMBOLS)
#else
#  define ELPP_EXPORT
#endif  // defined(ELPP_AS_DLL) && ELPP_COMPILER_MSVC
// Some special functions that are VC++ specific
#undef STRTOK
#undef STRERROR
#undef STRCAT
#undef STRCPY
#if ELPP_CRT_DBG_WARNINGS
#  define STRTOK(a, b, c) strtok_s(a, b, c)
#  define STRERROR(a, b, c) strerror_s(a, b, c)
#  define STRCAT(a, b, len) strcat_s(a, len, b)
#  define STRCPY(a, b, len) strcpy_s(a, len, b)
#else
#  define STRTOK(a, b, c) strtok(a, b)
#  define STRERROR(a, b, c) strerror(c)
#  define STRCAT(a, b, len) strcat(a, b)
#  define STRCPY(a, b, len) strcpy(a, b)
#endif
```

```
// Compiler specific support evaluations
#if (ELPP_MINGW && !defined(ELPP_FORCE_USE_STD_THREAD))
#  define ELPP_USE_STD_THREADING 0
#else
#  if ((ELPP_COMPILER_CLANG && defined(ELPP_CLANG_SUPPORTS_THREAD)) || \
       (!ELPP_COMPILER_CLANG && defined(ELPP_CXX11)) || \
       defined(ELPP_FORCE_USE_STD_THREAD))
#    define ELPP_USE_STD_THREADING 1
#  else
#    define ELPP_USE_STD_THREADING 0
#  endif
#endif
#undef ELPP_FINAL
#if ELPP_COMPILER_INTEL || (ELPP_GCC_VERSION < 40702)
#  define ELPP_FINAL
#else
#  define ELPP_FINAL final
#endif  // ELPP_COMPILER_INTEL || (ELPP_GCC_VERSION < 40702)
#if defined(ELPP_EXPERIMENTAL_ASYNC)
#  define ELPP_ASYNC_LOGGING 1
#else
#  define ELPP_ASYNC_LOGGING 0
#endif // defined(ELPP_EXPERIMENTAL_ASYNC)
#if defined(ELPP_THREAD_SAFE) || ELPP_ASYNC_LOGGING
#  define ELPP_THREADING_ENABLED 1
#else
#  define ELPP_THREADING_ENABLED 0
#endif  // defined(ELPP_THREAD_SAFE) || ELPP_ASYNC_LOGGING
// Function macro ELPP_FUNC
#undef ELPP_FUNC
#if ELPP_COMPILER_MSVC  // Visual C++
#  define ELPP_FUNC __FUNCSIG__
#elif ELPP_COMPILER_GCC  // GCC
#  define ELPP_FUNC __PRETTY_FUNCTION__
#elif ELPP_COMPILER_INTEL  // Intel C++
#  define ELPP_FUNC __PRETTY_FUNCTION__
#elif ELPP_COMPILER_CLANG  // Clang++
#  define ELPP_FUNC __PRETTY_FUNCTION__
#else
#  if defined(__func__)
#    define ELPP_FUNC __func__
#  else
#    define ELPP_FUNC ""
#  endif  // defined(__func__)
#endif  // defined(_MSC_VER)
#undef ELPP_VARIADIC_TEMPLATES_SUPPORTED
// Keep following line commented until features are fixed
#define ELPP_VARIADIC_TEMPLATES_SUPPORTED \
(ELPP_COMPILER_GCC || ELPP_COMPILER_CLANG || ELPP_COMPILER_INTEL || (ELPP_COMPILER_MSVC && _MSC_VER >= 1800))
// Logging Enable/Disable macros
#if defined(ELPP_DISABLE_LOGS)
#define ELPP_LOGGING_ENABLED 0
#else
#define ELPP_LOGGING_ENABLED 1
#endif
#if (!defined(ELPP_DISABLE_DEBUG_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_DEBUG_LOG 1
#else
#  define ELPP_DEBUG_LOG 0
#endif  // (!defined(ELPP_DISABLE_DEBUG_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_INFO_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_INFO_LOG 1
#else
#  define ELPP_INFO_LOG 0
#endif  // (!defined(ELPP_DISABLE_INFO_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_WARNING_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_WARNING_LOG 1
#else
#  define ELPP_WARNING_LOG 0
#endif  // (!defined(ELPP_DISABLE_WARNING_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_ERROR_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_ERROR_LOG 1
#else
#  define ELPP_ERROR_LOG 0
#endif  // (!defined(ELPP_DISABLE_ERROR_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_FATAL_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_FATAL_LOG 1
#else
#  define ELPP_FATAL_LOG 0
#endif  // (!defined(ELPP_DISABLE_FATAL_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_TRACE_LOGS) && (ELPP_LOGGING_ENABLED))
#  define ELPP_TRACE_LOG 1
#else
#  define ELPP_TRACE_LOG 0
#endif  // (!defined(ELPP_DISABLE_TRACE_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!defined(ELPP_DISABLE_VERBOSE_LOGS) && (ELPP_LOGGING_ENABLED))
```

```
#  define ELPP_VERBOSE_LOG 1
#else
#  define ELPP_VERBOSE_LOG 0
#endif  // (!defined(ELPP_DISABLE_VERBOSE_LOGS) && (ELPP_LOGGING_ENABLED))
#if (!(ELPP_CXX0X || ELPP_CXX11))
#   error "C++0x (or higher) support not detected! (Is `-std=c++11' missing?)"
#endif  // (!(ELPP_CXX0X || ELPP_CXX11))
// Headers
#if defined(ELPP_SYSLOG)
#   include <syslog.h>
#endif  // defined(ELPP_SYSLOG)
#include <ctime>
#include <cstring>
#include <cstdlib>
#include <cctype>
#include <cwchar>
#include <csignal>
#include <cerrno>
#include <cstdarg>
#if defined(ELPP_UNICODE)
#   include <locale>
#  if ELPP_OS_WINDOWS
#      include <codecvt>
#  endif // ELPP_OS_WINDOWS
#endif  // defined(ELPP_UNICODE)
#ifdef HAVE_EXECINFO
#   include <cxxabi.h>
#   include <execinfo.h>
#endif // ENABLE_EXECINFO
#if ELPP_OS_ANDROID
#   include <sys/system_properties.h>
#endif  // ELPP_OS_ANDROID
#if ELPP_OS_UNIX
#   include <sys/stat.h>
#   include <sys/time.h>
#elif ELPP_OS_WINDOWS
#   include <direct.h>
#   include <windows.h>
#  if defined(WIN32_LEAN_AND_MEAN)
#      if defined(ELPP_WINSOCK2)
#         include <winsock2.h>
#      else
#         include <winsock.h>
#      endif // defined(ELPP_WINSOCK2)
#  endif // defined(WIN32_LEAN_AND_MEAN)
#endif  // ELPP_OS_UNIX
#include <string>
#include <vector>
#include <map>
#include <unordered_map>
#include <utility>
#include <functional>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <memory>
#include <type_traits>
#if ELPP_THREADING_ENABLED
#  if ELPP_USE_STD_THREADING
#      include <mutex>
#      include <thread>
#  else
#      if ELPP_OS_UNIX
#         include <pthread.h>
#      endif  // ELPP_OS_UNIX
#  endif  // ELPP_USE_STD_THREADING
#endif  // ELPP_THREADING_ENABLED
#if ELPP_ASYNC_LOGGING
#  if defined(ELPP_NO_SLEEP_FOR)
#      include <unistd.h>
#  endif  // defined(ELPP_NO_SLEEP_FOR)
#   include <thread>
#   include <queue>
#   include <condition_variable>
#endif  // ELPP_ASYNC_LOGGING
#if defined(ELPP_STL_LOGGING)
// For logging STL based templates
#   include <list>
#   include <queue>
#   include <deque>
#   include <set>
#   include <bitset>
#   include <stack>
#  if defined(ELPP_LOG_STD_ARRAY)
#      include <array>
#  endif  // defined(ELPP_LOG_STD_ARRAY)
#  if defined(ELPP_LOG_UNORDERED_SET)
#      include <unordered_set>
#  endif  // defined(ELPP_UNORDERED_SET)
#endif  // defined(ELPP_STL_LOGGING)
#if defined(ELPP_QT_LOGGING)
// For logging Qt based classes & templates
#   include <QString>
#   include <QByteArray>
#   include <QVector>
#   include <QList>
#   include <QPair>
#   include <QMap>
#   include <QQueue>
#   include <QSet>
#   include <QLinkedList>
#   include <QHash>
#   include <QMultiHash>
#   include <QStack>
#endif  // defined(ELPP_QT_LOGGING)
#if defined(ELPP_BOOST_LOGGING)
// For logging boost based classes & templates
#   include <boost/container/vector.hpp>
#   include <boost/container/stable_vector.hpp>
#   include <boost/container/list.hpp>
#   include <boost/container/deque.hpp>
#   include <boost/container/map.hpp>
#   include <boost/container/flat_map.hpp>
#   include <boost/container/set.hpp>
#   include <boost/container/flat_set.hpp>
#endif  // defined(ELPP_BOOST_LOGGING)
#if defined(ELPP_WXWIDGETS_LOGGING)
// For logging wxWidgets based classes & templates
#   include <wx/vector.h>
#endif  // defined(ELPP_WXWIDGETS_LOGGING)
#if defined(ELPP_UTC_DATETIME)
#   define elpptime_r gmtime_r
#   define elpptime_s gmtime_s
#   define elpptime   gmtime
#else
#   define elpptime_r localtime_r
#   define elpptime_s localtime_s
#   define elpptime   localtime
#endif  // defined(ELPP_UTC_DATETIME)
// Forward declarations
namespace el {
class Logger;
class LogMessage;
class PerformanceTrackingData;
class Loggers;
class Helpers;
template <typename T> class Callback;
class LogDispatchCallback;
class PerformanceTrackingCallback;
class LoggerRegistrationCallback;
class LogDispatchData;
namespace base {
class Storage;
class RegisteredLoggers;
class PerformanceTracker;
class MessageBuilder;
class Writer;
class PErrorWriter;
class LogDispatcher;
class DefaultLogBuilder;
class DefaultLogDispatchCallback;
#if ELPP_ASYNC_LOGGING
class AsyncLogDispatchCallback;
class AsyncDispatchWorker;
#endif // ELPP_ASYNC_LOGGING
class DefaultPerformanceTrackingCallback;
}  // namespace base
}  // namespace el
namespace el {
namespace base {
namespace type {
#undef ELPP_LITERAL
#undef ELPP_STRLEN
#undef ELPP_COUT
#if defined(ELPP_UNICODE)
#  define ELPP_LITERAL(txt) L##txt
#  define ELPP_STRLEN wcslen
#  if defined ELPP_CUSTOM_COUT
#    define ELPP_COUT ELPP_CUSTOM_COUT
#  else
#    define ELPP_COUT std::wcout
#  endif  // defined ELPP_CUSTOM_COUT
typedef wchar_t char_t;
typedef std::wstring string_t;
typedef std::wstringstream stringstream_t;
typedef std::wfstream fstream_t;
typedef std::wostream ostream_t;
#else
#  define ELPP_LITERAL(txt) txt
#  define ELPP_STRLEN strlen
#  if defined ELPP_CUSTOM_COUT
#    define ELPP_COUT ELPP_CUSTOM_COUT
#  else
#    define ELPP_COUT std::cout
#  endif  // defined ELPP_CUSTOM_COUT
typedef char char_t;
typedef std::string string_t;
typedef std::stringstream stringstream_t;
typedef std::fstream fstream_t;
typedef std::ostream ostream_t;
#endif  // defined(ELPP_UNICODE)
#if defined(ELPP_CUSTOM_COUT_LINE)
#  define ELPP_COUT_LINE(logLine) ELPP_CUSTOM_COUT_LINE(logLine)
#else
#  define ELPP_COUT_LINE(logLine) logLine « std::flush
#endif // defined(ELPP_CUSTOM_COUT_LINE)
typedef unsigned int EnumType;
typedef unsigned short VerboseLevel;
typedef unsigned long int LineNumber;
typedef std::shared_ptr<base::Storage> StoragePointer;
typedef std::shared_ptr<LogDispatchCallback> LogDispatchCallbackPtr;
typedef std::shared_ptr<PerformanceTrackingCallback> PerformanceTrackingCallbackPtr;
typedef std::shared_ptr<LoggerRegistrationCallback> LoggerRegistrationCallbackPtr;
typedef std::unique_ptr<el::base::PerformanceTracker> PerformanceTrackerPtr;
}  // namespace type
class NoCopy {
 protected:
  NoCopy(void) {}
 private:
  NoCopy(const NoCopy&);
  NoCopy& operator=(const NoCopy&);
};
class StaticClass {
 private:
  StaticClass(void);
  StaticClass(const StaticClass&);
  StaticClass& operator=(const StaticClass&);
};
}  // namespace base
enum class Level : base::type::EnumType {
  Global = 1,
  Trace = 2,
  Debug = 4,
  Fatal = 8,
  Error = 16,
  Warning = 32,
  Verbose = 64,
  Info = 128,
  Unknown = 1010
};
} // namespace el
namespace std {
template<> struct hash<el::Level> {
 public:
  std::size_t operator()(const el::Level& l) const {
    return hash<el::base::type::EnumType> {}(static_cast<el::base::type::EnumType>(l));
  }
};
}
namespace el {
class LevelHelper : base::StaticClass {
 public:
  static const base::type::EnumType kMinValid = static_cast<base::type::EnumType>(Level::Trace);
  static const base::type::EnumType kMaxValid = static_cast<base::type::EnumType>(Level::Info);
  static base::type::EnumType castToInt(Level level) {
    return static_cast<base::type::EnumType>(level);
  }
  static Level castFromInt(base::type::EnumType l) {
    return static_cast<Level>(l);
  }
  static const char* convertToString(Level level);
  static Level convertFromString(const char* levelStr);
  static void forEachLevel(base::type::EnumType* startIndex, const std::function<bool(void)>& fn);
};
enum class ConfigurationType : base::type::EnumType {
  Enabled = 1,
  ToFile = 2,
  ToStandardOutput = 4,
  Format = 8,
  Filename = 16,
  SubsecondPrecision = 32,
  MillisecondsWidth = SubsecondPrecision,
  PerformanceTracking = 64,
  MaxLogFileSize = 128,
  LogFlushThreshold = 256,
  Unknown = 1010
};
class ConfigurationTypeHelper : base::StaticClass {
 public:
  static const base::type::EnumType kMinValid = static_cast<base::type::EnumType>(ConfigurationType::Enabled);
  static const base::type::EnumType kMaxValid = static_cast<base::type::EnumType>(ConfigurationType::MaxLogFileSize);
  static base::type::EnumType castToInt(ConfigurationType configurationType) {
    return static_cast<base::type::EnumType>(configurationType);
  }
  static ConfigurationType castFromInt(base::type::EnumType c) {
    return static_cast<ConfigurationType>(c);
  }
  static const char* convertToString(ConfigurationType configurationType);
  static ConfigurationType convertFromString(const char* configStr);
  static inline void forEachConfigType(base::type::EnumType* startIndex, const std::function<bool(void)>& fn);
};
enum class LoggingFlag : base::type::EnumType {
  NewLineForContainer = 1,
  AllowVerboseIfModuleNotSpecified = 2,
  LogDetailedCrashReason = 4,
  DisableApplicationAbortOnFatalLog = 8,
  ImmediateFlush = 16,
  StrictLogFileSizeCheck = 32,
  ColoredTerminalOutput = 64,
  MultiLoggerSupport = 128,
  DisablePerformanceTrackingCheckpointComparison = 256,
  DisableVModules = 512,
  DisableVModulesExtensions = 1024,
  HierarchicalLogging = 2048,
  CreateLoggerAutomatically = 4096,
  AutoSpacing = 8192,
  FixedTimeFormat = 16384,
  // @brief Ignore SIGINT or crash
  IgnoreSigInt = 32768,
};
namespace base {
namespace consts {
static const char  kFormatSpecifierCharValue               =      'v';
static const char  kFormatSpecifierChar                    =      '%';
static const unsigned int kMaxLogPerCounter                =      100000;
static const unsigned int kMaxLogPerContainer              =      100;
static const unsigned int kDefaultSubsecondPrecision       =      3;

#ifdef ELPP_DEFAULT_LOGGER
static const char* kDefaultLoggerId                        =      ELPP_DEFAULT_LOGGER;
#else
static const char* kDefaultLoggerId                        =      "default";
#endif

#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
#ifdef ELPP_DEFAULT_PERFORMANCE_LOGGER
static const char* kPerformanceLoggerId                    =      ELPP_DEFAULT_PERFORMANCE_LOGGER;
#else
static const char* kPerformanceLoggerId                    =      "performance";
#endif // ELPP_DEFAULT_PERFORMANCE_LOGGER
#endif

#if defined(ELPP_SYSLOG)
static const char* kSysLogLoggerId                         =      "syslog";
#endif  // defined(ELPP_SYSLOG)

#if ELPP_OS_WINDOWS
static const char* kFilePathSeparator                      =      "\\";
#else
static const char* kFilePathSeparator                      =      "/";
#endif  // ELPP_OS_WINDOWS

static const std::size_t kSourceFilenameMaxLength          =      100;
static const std::size_t kSourceLineMaxLength              =      10;
static const Level kPerformanceTrackerDefaultLevel         =      Level::Info;
const struct {
  double value;
  const base::type::char_t* unit;
} kTimeFormats[] = {
  { 1000.0f, ELPP_LITERAL("us") },
  { 1000.0f, ELPP_LITERAL("ms") },
  { 60.0f, ELPP_LITERAL("seconds") },
  { 60.0f, ELPP_LITERAL("minutes") },
  { 24.0f, ELPP_LITERAL("hours") },
  { 7.0f, ELPP_LITERAL("days") }
};
static const int kTimeFormatsCount                           =      sizeof(kTimeFormats) /
sizeof(kTimeFormats[0]);
const struct {
  int numb;
  const char* name;
  const char* brief;
  const char* detail;
} kCrashSignals[] = {
  // NOTE: Do not re-order, if you do please check CrashHandler(bool) constructor and
CrashHandler::setHandler(..)
  {
    SIGABRT, "SIGABRT", "Abnormal termination",
    "Program was abnormally terminated."
  },
  {
    SIGFPE, "SIGFPE", "Erroneous arithmetic operation",
    "Arithmetic operation issue such as division by zero or operation resulting in overflow."
  },
  {
    SIGILL, "SIGILL", "Illegal instruction",
    "Generally due to a corruption in the code or to an attempt to execute data."
  },
  {
    SIGSEGV, "SIGSEGV", "Invalid access to memory",
    "Program is trying to read an invalid (unallocated, deleted or corrupted) or inaccessible memory."
  },
  {
    SIGINT, "SIGINT", "Interactive attention signal",
    "Interruption generated (generally) by user or operating system."
  },
};
static const int kCrashSignalsCount                          =      sizeof(kCrashSignals) /
sizeof(kCrashSignals[0]);
}  // namespace consts
}  // namespace base
typedef std::function<void(const char*, std::size_t)> PreRollOutCallback;
namespace base {
static inline void defaultPreRollOutCallback(const char*, std::size_t) {}
enum class TimestampUnit : base::type::EnumType {
  Microsecond = 0, Millisecond = 1, Second = 2, Minute = 3, Hour = 4, Day = 5
};
enum class FormatFlags : base::type::EnumType {
  DateTime = 1 « 1,
  LoggerId = 1 « 2,
  File = 1 « 3,
  Line = 1 « 4,
  Location = 1 « 5,
  Function = 1 « 6,
  User = 1 « 7,
  Host = 1 « 8,
  LogMessage = 1 « 9,
  VerboseLevel = 1 « 10,
  AppName = 1 « 11,
  ThreadId = 1 « 12,
  Level = 1 « 13,
  FileBase = 1 « 14,
  LevelShort = 1 « 15
};
class SubsecondPrecision {
 public:
  SubsecondPrecision(void) {
    init(base::consts::kDefaultSubsecondPrecision);
  }
  explicit SubsecondPrecision(int width) {
    init(width);
  }
  bool operator==(const SubsecondPrecision& ssPrec) {
    return m_width == ssPrec.m_width && m_offset == ssPrec.m_offset;
  }
  int m_width;
  unsigned int m_offset;
 private:
  void init(int width);
};
typedef SubsecondPrecision MillisecondsWidth;
namespace utils {
template <typename T>
static
typename std::enable_if<std::is_pointer<T*>::value, void>::type
safeDelete(T*& pointer) {
  if (pointer == nullptr)
    return;
  delete pointer;
  pointer = nullptr;
}
```

```
namespace bitwise {
template <typename Enum>
static inline base::type::EnumType And(Enum e, base::type::EnumType flag) {
  return static_cast<base::type::EnumType>(flag) & static_cast<base::type::EnumType>(e);
}
template <typename Enum>
static inline base::type::EnumType Not(Enum e, base::type::EnumType flag) {
  return static_cast<base::type::EnumType>(flag) & ~(static_cast<base::type::EnumType>(e));
}
template <typename Enum>
static inline base::type::EnumType Or(Enum e, base::type::EnumType flag) {
  return static_cast<base::type::EnumType>(flag) | static_cast<base::type::EnumType>(e);
}
}  // namespace bitwise
template <typename Enum>
static inline void addFlag(Enum e, base::type::EnumType* flag) {
  *flag = base::utils::bitwise::Or<Enum>(e, *flag);
}
template <typename Enum>
static inline void removeFlag(Enum e, base::type::EnumType* flag) {
  *flag = base::utils::bitwise::Not<Enum>(e, *flag);
}
template <typename Enum>
static inline bool hasFlag(Enum e, base::type::EnumType flag) {
  return base::utils::bitwise::And<Enum>(e, flag) > 0x0;
}
}  // namespace utils
namespace threading {
#if ELPP_THREADING_ENABLED
#  if !ELPP_USE_STD_THREADING
namespace internal {
class Mutex : base::NoCopy {
 public:
  Mutex(void) {
#  if ELPP_OS_UNIX
    pthread_mutexattr_t attr;
    pthread_mutexattr_init(&attr);
    pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_RECURSIVE);
    pthread_mutex_init(&m_underlyingMutex, &attr);
    pthread_mutexattr_destroy(&attr);
#  elif ELPP_OS_WINDOWS
    InitializeCriticalSection(&m_underlyingMutex);
#  endif  // ELPP_OS_UNIX
  }

  virtual ~Mutex(void) {
#  if ELPP_OS_UNIX
    pthread_mutex_destroy(&m_underlyingMutex);
#  elif ELPP_OS_WINDOWS
    DeleteCriticalSection(&m_underlyingMutex);
#  endif  // ELPP_OS_UNIX
  }

  inline void lock(void) {
#  if ELPP_OS_UNIX
    pthread_mutex_lock(&m_underlyingMutex);
#  elif ELPP_OS_WINDOWS
    EnterCriticalSection(&m_underlyingMutex);
#  endif  // ELPP_OS_UNIX
  }

  inline bool try_lock(void) {
#  if ELPP_OS_UNIX
    return (pthread_mutex_trylock(&m_underlyingMutex) == 0);
#  elif ELPP_OS_WINDOWS
    return TryEnterCriticalSection(&m_underlyingMutex);
#  endif  // ELPP_OS_UNIX
  }

  inline void unlock(void) {
#  if ELPP_OS_UNIX
    pthread_mutex_unlock(&m_underlyingMutex);
#  elif ELPP_OS_WINDOWS
    LeaveCriticalSection(&m_underlyingMutex);
#  endif  // ELPP_OS_UNIX
  }

 private:
#  if ELPP_OS_UNIX
  pthread_mutex_t m_underlyingMutex;
#  elif ELPP_OS_WINDOWS
  CRITICAL_SECTION m_underlyingMutex;
#  endif  // ELPP_OS_UNIX
};
template <typename M>
class ScopedLock : base::NoCopy {
 public:
  explicit ScopedLock(M& mutex) {
    m_mutex = &mutex;
    m_mutex->lock();
  }

  virtual ~ScopedLock(void) {
    m_mutex->unlock();
  }
 private:
  M* m_mutex;
  ScopedLock(void);
};
} // namespace internal
typedef base::threading::internal::Mutex Mutex;
typedef base::threading::internal::ScopedLock<base::threading::Mutex> ScopedLock;
#  else
typedef std::recursive_mutex Mutex;
typedef std::lock_guard<base::threading::Mutex> ScopedLock;
#  endif  // !ELPP_USE_STD_THREADING
#else
namespace internal {
class NoMutex : base::NoCopy {
 public:
  NoMutex(void) {}
  inline void lock(void) {}
  inline bool try_lock(void) {
    return true;
  }
  inline void unlock(void) {}
};
template <typename Mutex>
class NoScopedLock : base::NoCopy {
 public:
  explicit NoScopedLock(Mutex&) {
  }
  virtual ~NoScopedLock(void) {
  }
 private:
  NoScopedLock(void);
};
}  // namespace internal
typedef base::threading::internal::NoMutex Mutex;
typedef base::threading::internal::NoScopedLock<base::threading::Mutex> ScopedLock;
#endif  // ELPP_THREADING_ENABLED
class ThreadSafe {
 public:
  virtual inline void acquireLock(void) ELPP_FINAL { m_mutex.lock(); }
  virtual inline void releaseLock(void) ELPP_FINAL { m_mutex.unlock(); }
  virtual inline base::threading::Mutex& lock(void) ELPP_FINAL { return m_mutex; }
 protected:
  ThreadSafe(void) {}
  virtual ~ThreadSafe(void) {}
 private:
  base::threading::Mutex m_mutex;
};

#if ELPP_THREADING_ENABLED
#  if !ELPP_USE_STD_THREADING
static std::string getCurrentThreadId(void) {
  std::stringstream ss;
#      if (ELPP_OS_WINDOWS)
  ss « GetCurrentThreadId();
#      endif  // (ELPP_OS_WINDOWS)
  return ss.str();
}
#  else
static std::string getCurrentThreadId(void) {
  std::stringstream ss;
  ss « std::this_thread::get_id();
  return ss.str();
}
#  endif  // !ELPP_USE_STD_THREADING
#else
static inline std::string getCurrentThreadId(void) {
  return std::string();
}
#endif  // ELPP_THREADING_ENABLED
}  // namespace threading
namespace utils {
class File : base::StaticClass {
 public:
  static base::type::fstream_t* newFileStream(const std::string& filename);

  static std::size_t getSizeOfFile(base::type::fstream_t* fs);

  static bool pathExists(const char* path, bool considerFile = false);

  static bool createPath(const std::string& path);
  static std::string extractPathFromFilename(const std::string& fullPath,
      const char* separator = base::consts::kFilePathSeparator);
  static void buildStrippedFilename(const char* filename, char buff[],
                                    std::size_t limit = base::consts::kSourceFilenameMaxLength);
  static void buildBaseFilename(const std::string& fullPath, char buff[],
                                std::size_t limit = base::consts::kSourceFilenameMaxLength,
                                const char* separator = base::consts::kFilePathSeparator);
};
class Str : base::StaticClass {
 public:
  static inline bool isDigit(char c) {
    return c >= '0' && c <= '9';
  }

  static bool wildCardMatch(const char* str, const char* pattern);

  static std::string& ltrim(std::string& str);
  static std::string& rtrim(std::string& str);
  static std::string& trim(std::string& str);

  static bool startsWith(const std::string& str, const std::string& start);

  static bool endsWith(const std::string& str, const std::string& end);

  static std::string& replaceAll(std::string& str, char replaceWhat, char replaceWith);

  static std::string& replaceAll(std::string& str, const std::string& replaceWhat,
                                 const std::string& replaceWith);

  static void replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t&
replaceWhat,
                                     const base::type::string_t& replaceWith);
#if defined(ELPP_UNICODE)
  static void replaceFirstWithEscape(base::type::string_t& str, const base::type::string_t&
replaceWhat,
                                     const std::string& replaceWith);
#endif  // defined(ELPP_UNICODE)
  static std::string& toUpper(std::string& str);

  static bool cStringEq(const char* s1, const char* s2);

  static bool cStringCaseEq(const char* s1, const char* s2);

  static bool contains(const char* str, char c);

  static char* convertAndAddToBuff(std::size_t n, int len, char* buf, const char* bufLim, bool
zeroPadded = true);
  static char* addToBuff(const char* str, char* buf, const char* bufLim);
  static char* clearBuff(char buff[], std::size_t lim);

  static char* wcharPtrToCharPtr(const wchar_t* line);
};
class OS : base::StaticClass {
 public:
#if ELPP_OS_WINDOWS
  static const char* getWindowsEnvironmentVariable(const char* varname);
#endif  // ELPP_OS_WINDOWS
#if ELPP_OS_ANDROID
  static std::string getProperty(const char* prop);

  static std::string getDeviceName(void);
#endif  // ELPP_OS_ANDROID

  static const std::string getBashOutput(const char* command);

  static std::string getEnvironmentVariable(const char* variableName, const char* defaultVal,
      const char* alternativeBashCommand = nullptr);
  static std::string currentUser(void);

  static std::string currentHost(void);
  static bool termSupportsColor(void);
};
class DateTime : base::StaticClass {
 public:
  static void gettimeofday(struct timeval* tv);

  static std::string getDateTime(const char* format, const base::SubsecondPrecision* ssPrec);

  static std::string timevalToString(struct timeval tval, const char* format,
                                     const el::base::SubsecondPrecision* ssPrec);

  static base::type::string_t formatTime(unsigned long long time, base::TimestampUnit timestampUnit);

  static unsigned long long getTimeDifference(const struct timeval& endTime, const struct timeval&
startTime,
      base::TimestampUnit timestampUnit);


  static struct ::tm* buildTimeInfo(struct timeval* currTime, struct ::tm* timeInfo);
 private:
  static char* parseFormat(char* buf, std::size_t bufSz, const char* format, const struct tm* tInfo,
                           std::size_t msec, const base::SubsecondPrecision* ssPrec);
};
class CommandLineArgs {
 public:
  CommandLineArgs(void) {
    setArgs(0, static_cast<char**>(nullptr));
  }
  CommandLineArgs(int argc, const char** argv) {
    setArgs(argc, argv);
  }
  CommandLineArgs(int argc, char** argv) {
    setArgs(argc, argv);
  }
  virtual ~CommandLineArgs(void) {}
  inline void setArgs(int argc, const char** argv) {
    setArgs(argc, const_cast<char**>(argv));
  }
  void setArgs(int argc, char** argv);
  bool hasParamWithValue(const char* paramKey) const;
  const char* getParamValue(const char* paramKey) const;
  bool hasParam(const char* paramKey) const;
  bool empty(void) const;
  std::size_t size(void) const;
  friend base::type::ostream_t& operator«(base::type::ostream_t& os, const CommandLineArgs& c);

 private:
  int m_argc;
  char** m_argv;
  std::unordered_map<std::string, std::string> m_paramsWithValue;
  std::vector<std::string> m_params;
};
template <typename T_Ptr, typename Container>
class AbstractRegistry : public base::threading::ThreadSafe {
 public:
  typedef typename Container::iterator iterator;
  typedef typename Container::const_iterator const_iterator;

  AbstractRegistry(void) {}

  AbstractRegistry(AbstractRegistry&& sr) {
    if (this == &sr) {
      return;
    }
    unregisterAll();
    m_list = std::move(sr.m_list);
  }

  bool operator==(const AbstractRegistry<T_Ptr, Container>& other) {
    if (size() != other.size()) {
      return false;
    }
    for (std::size_t i = 0; i < m_list.size(); ++i) {
      if (m_list.at(i) != other.m_list.at(i)) {
        return false;
      }
    }
    return true;
  }

  bool operator!=(const AbstractRegistry<T_Ptr, Container>& other) {
    if (size() != other.size()) {
      return true;
    }
    for (std::size_t i = 0; i < m_list.size(); ++i) {
      if (m_list.at(i) != other.m_list.at(i)) {
        return true;
      }
    }
    return false;
  }

  AbstractRegistry& operator=(AbstractRegistry&& sr) {
    if (this == &sr) {
      return *this;
    }
    unregisterAll();
    m_list = std::move(sr.m_list);
    return *this;
  }

  virtual ~AbstractRegistry(void) {
  }

  virtual inline iterator begin(void) ELPP_FINAL {
    return m_list.begin();
  }

  virtual inline iterator end(void) ELPP_FINAL {
    return m_list.end();
  }


  virtual inline const_iterator cbegin(void) const ELPP_FINAL {
    return m_list.cbegin();
  }

  virtual inline const_iterator cend(void) const ELPP_FINAL {
    return m_list.cend();
  }

  virtual inline bool empty(void) const ELPP_FINAL {
    return m_list.empty();
  }

  virtual inline std::size_t size(void) const ELPP_FINAL {
    return m_list.size();
  }

  virtual inline Container& list(void) ELPP_FINAL {
    return m_list;
  }

  virtual inline const Container& list(void) const ELPP_FINAL {
    return m_list;
  }

  virtual void unregisterAll(void) = 0;

 protected:
  virtual void deepCopy(const AbstractRegistry<T_Ptr, Container>&) = 0;
  void reinitDeepCopy(const AbstractRegistry<T_Ptr, Container>& sr) {
    unregisterAll();
    deepCopy(sr);
  }

 private:
  Container m_list;
};

template <typename T_Ptr, typename T_Key = const char*>
class Registry : public AbstractRegistry<T_Ptr, std::unordered_map<T_Key, T_Ptr*» {
 public:
  typedef typename Registry<T_Ptr, T_Key>::iterator iterator;
  typedef typename Registry<T_Ptr, T_Key>::const_iterator const_iterator;

  Registry(void) {}

  Registry(const Registry& sr) : AbstractRegistry<T_Ptr, std::vector<T_Ptr*»() {
    if (this == &sr) {
      return;
    }
    this->reinitDeepCopy(sr);
  }

  Registry& operator=(const Registry& sr) {
    if (this == &sr) {
      return *this;
    }
    this->reinitDeepCopy(sr);
    return *this;
  }

  virtual ~Registry(void) {
    unregisterAll();
  }

 protected:
  virtual void unregisterAll(void) ELPP_FINAL {
    if (!this->empty()) {
      for (auto&& curr : this->list()) {
        base::utils::safeDelete(curr.second);
      }
      this->list().clear();
    }
  }

  virtual void registerNew(const T_Key& uniqKey, T_Ptr* ptr) ELPP_FINAL {
    unregister(uniqKey);
    this->list().insert(std::make_pair(uniqKey, ptr));
  }

  void unregister(const T_Key& uniqKey) {
    T_Ptr* existing = get(uniqKey);
    if (existing != nullptr) {
      this->list().erase(uniqKey);
      base::utils::safeDelete(existing);
    }
  }

  T_Ptr* get(const T_Key& uniqKey) {
    iterator it = this->list().find(uniqKey);
    return it == this->list().end()
           ? nullptr
           : it->second;
  }

 private:
  virtual void deepCopy(const AbstractRegistry<T_Ptr, std::unordered_map<T_Key, T_Ptr*»& sr)
ELPP_FINAL {
    for (const_iterator it = sr.cbegin(); it != sr.cend(); ++it) {
      registerNew(it->first, new T_Ptr(*it->second));
    }
  }
};

template <typename T_Ptr, typename Pred>
class RegistryWithPred : public AbstractRegistry<T_Ptr, std::vector<T_Ptr*» {
 public:
  typedef typename RegistryWithPred<T_Ptr, Pred>::iterator iterator;
  typedef typename RegistryWithPred<T_Ptr, Pred>::const_iterator const_iterator;

  RegistryWithPred(void) {
  }

  virtual ~RegistryWithPred(void) {
    unregisterAll();
  }

  RegistryWithPred(const RegistryWithPred& sr) : AbstractRegistry<T_Ptr, std::vector<T_Ptr*»() {
    if (this == &sr) {
      return;
    }
    this->reinitDeepCopy(sr);
  }

  RegistryWithPred& operator=(const RegistryWithPred& sr) {
    if (this == &sr) {
      return *this;
    }
    this->reinitDeepCopy(sr);
    return *this;
  }

  friend base::type::ostream_t& operator«(base::type::ostream_t& os, const RegistryWithPred& sr) {
    for (const_iterator it = sr.list().begin(); it != sr.list().end(); ++it) {
      os « ELPP_LITERAL("    ") « **it « ELPP_LITERAL("\n");
    }
    return os;
  }

 protected:
  virtual void unregisterAll(void) ELPP_FINAL {
    if (!this->empty()) {
      for (auto&& curr : this->list()) {
        base::utils::safeDelete(curr);
      }
      this->list().clear();
    }
  }

  virtual void unregister(T_Ptr*& ptr) ELPP_FINAL {
    if (ptr) {
      iterator iter = this->begin();
      for (; iter != this->end(); ++iter) {
        if (ptr == *iter) {
          break;
        }
      }
      if (iter != this->end() && *iter != nullptr) {
        this->list().erase(iter);
        base::utils::safeDelete(*iter);
      }
    }
  }

  virtual inline void registerNew(T_Ptr* ptr) ELPP_FINAL {
    this->list().push_back(ptr);
  }

  template <typename T, typename T2>
  T_Ptr* get(const T& arg1, const T2 arg2) {
    iterator iter = std::find_if(this->list().begin(), this->list().end(), Pred(arg1, arg2));
    if (iter != this->list().end() && *iter != nullptr) {
      return *iter;
    }
    return nullptr;
  }

 private:
  virtual void deepCopy(const AbstractRegistry<T_Ptr, std::vector<T_Ptr*»& sr) {
    for (const_iterator it = sr.list().begin(); it != sr.list().end(); ++it) {
      registerNew(new T_Ptr(**it));
    }
  }
};
class Utils {
 public:
  template <typename T, typename TPtr>
  static bool installCallback(const std::string& id, std::unordered_map<std::string, TPtr>* mapT) {
    if (mapT->find(id) == mapT->end()) {
      mapT->insert(std::make_pair(id, TPtr(new T())));
      return true;
    }
    return false;
  }

  template <typename T, typename TPtr>
  static void uninstallCallback(const std::string& id, std::unordered_map<std::string, TPtr>* mapT) {
    if (mapT->find(id) != mapT->end()) {
      mapT->erase(id);
    }
  }

  template <typename T, typename TPtr>
  static T* callback(const std::string& id, std::unordered_map<std::string, TPtr>* mapT) {
    typename std::unordered_map<std::string, TPtr>::iterator iter = mapT->find(id);
    if (iter != mapT->end()) {
      return static_cast<T*>(iter->second.get());
    }
    return nullptr;
  }
};
}  // namespace utils
} // namespace base
class Loggable {
 public:
  virtual ~Loggable(void) {}
  virtual void log(el::base::type::ostream_t&) const = 0;
 private:
  friend inline el::base::type::ostream_t& operator«(el::base::type::ostream_t& os, const Loggable&
loggable) {
    loggable.log(os);
    return os;
  }
};
namespace base {
class LogFormat : public Loggable {
 public:
  LogFormat(void);
  LogFormat(Level level, const base::type::string_t& format);
  LogFormat(const LogFormat& logFormat);
  LogFormat(LogFormat&& logFormat);
  LogFormat& operator=(const LogFormat& logFormat);
  virtual ~LogFormat(void) {}
  bool operator==(const LogFormat& other);

  void parseFromFormat(const base::type::string_t& userFormat);

  inline Level level(void) const {
    return m_level;
  }

  inline const base::type::string_t& userFormat(void) const {
    return m_userFormat;
  }

  inline const base::type::string_t& format(void) const {
    return m_format;
  }

  inline const std::string& dateTimeFormat(void) const {
    return m_dateTimeFormat;
  }

  inline base::type::EnumType flags(void) const {
    return m_flags;
  }

  inline bool hasFlag(base::FormatFlags flag) const {
    return base::utils::hasFlag(flag, m_flags);
  }

  virtual void log(el::base::type::ostream_t& os) const {
    os « m_format;
  }

 protected:
  virtual void updateDateFormat(std::size_t index, base::type::string_t& currFormat) ELPP_FINAL;

  virtual void updateFormatSpec(void) ELPP_FINAL;

  inline void addFlag(base::FormatFlags flag) {
    base::utils::addFlag(flag, &m_flags);
  }

 private:
  Level m_level;
  base::type::string_t m_userFormat;
  base::type::string_t m_format;
  std::string m_dateTimeFormat;
  base::type::EnumType m_flags;
  std::string m_currentUser;
  std::string m_currentHost;
  friend class el::Logger;  // To resolve loggerId format specifier easily
};
}  // namespace base
typedef std::function<std::string(const LogMessage*)> FormatSpecifierValueResolver;
class CustomFormatSpecifier {
 public:
  CustomFormatSpecifier(const char* formatSpecifier, const FormatSpecifierValueResolver& resolver) :
    m_formatSpecifier(formatSpecifier), m_resolver(resolver) {}
  inline const char* formatSpecifier(void) const {
    return m_formatSpecifier;
  }
  inline const FormatSpecifierValueResolver& resolver(void) const {
    return m_resolver;
  }
  inline bool operator==(const char* formatSpecifier) {
    return strcmp(m_formatSpecifier, formatSpecifier) == 0;
  }

 private:
  const char* m_formatSpecifier;
  FormatSpecifierValueResolver m_resolver;
};
class Configuration : public Loggable {
 public:
  Configuration(const Configuration& c);
  Configuration& operator=(const Configuration& c);

  virtual ~Configuration(void) {
  }

  Configuration(Level level, ConfigurationType configurationType, const std::string& value);

  inline Level level(void) const {
    return m_level;
  }

  inline ConfigurationType configurationType(void) const {
    return m_configurationType;
  }

  inline const std::string& value(void) const {
    return m_value;
  }

  inline void setValue(const std::string& value) {
    m_value = value;
  }

  virtual void log(el::base::type::ostream_t& os) const;

  class Predicate {
   public:
    Predicate(Level level, ConfigurationType configurationType);

    bool operator()(const Configuration* conf) const;

   private:
    Level m_level;
    ConfigurationType m_configurationType;
  };

 private:
  Level m_level;
  ConfigurationType m_configurationType;
  std::string m_value;
};

class Configurations : public base::utils::RegistryWithPred<Configuration, Configuration::Predicate> {
 public:
  Configurations(void);

  Configurations(const std::string& configurationFile, bool useDefaultsForRemaining = true,
                 Configurations* base = nullptr);

  virtual ~Configurations(void) {
  }

  bool parseFromFile(const std::string& configurationFile, Configurations* base = nullptr);

  bool parseFromText(const std::string& configurationsString, Configurations* base = nullptr);

  void setFromBase(Configurations* base);

  bool hasConfiguration(ConfigurationType configurationType);

  bool hasConfiguration(Level level, ConfigurationType configurationType);

  void set(Level level, ConfigurationType configurationType, const std::string& value);

  void set(Configuration* conf);

  inline Configuration* get(Level level, ConfigurationType configurationType) {
    base::threading::ScopedLock scopedLock(lock());
    return RegistryWithPred<Configuration, Configuration::Predicate>::get(level, configurationType);
  }

  inline void setGlobally(ConfigurationType configurationType, const std::string& value) {
    setGlobally(configurationType, value, false);
  }

  inline void clear(void) {
    base::threading::ScopedLock scopedLock(lock());
    unregisterAll();
  }

  inline const std::string& configurationFile(void) const {
    return m_configurationFile;
  }

  void setToDefault(void);

  void setRemainingToDefault(void);

  class Parser : base::StaticClass {
   public:
    static bool parseFromFile(const std::string& configurationFile, Configurations* sender,
                              Configurations* base = nullptr);

    static bool parseFromText(const std::string& configurationsString, Configurations* sender,
                              Configurations* base = nullptr);

   private:
    friend class el::Loggers;
    static void ignoreComments(std::string* line);
    static bool isLevel(const std::string& line);
    static bool isComment(const std::string& line);
    static inline bool isConfig(const std::string& line);
    static bool parseLine(std::string* line, std::string* currConfigStr, std::string* currLevelStr,
Level* currLevel,
                          Configurations* conf);
  };

 private:
  std::string m_configurationFile;
  bool m_isFromFile;
  friend class el::Loggers;

  void unsafeSetIfNotExist(Level level, ConfigurationType configurationType, const std::string&
value);

  void unsafeSet(Level level, ConfigurationType configurationType, const std::string& value);

  void setGlobally(ConfigurationType configurationType, const std::string& value, bool
includeGlobalLevel);
```

```

  void unsafeSetGlobally(ConfigurationType configurationType, const std::string& value, bool includeGlobalLevel);
};

namespace base {
typedef std::shared_ptr<base::type::fstream_t> FileStreamPtr;
typedef std::unordered_map<std::string, FileStreamPtr> LogStreamsReferenceMap;
typedef std::shared_ptr<base::LogStreamsReferenceMap> LogStreamsReferenceMapPtr;
class TypedConfigurations : public base::threading::ThreadSafe {
 public:
  TypedConfigurations(Configurations* configurations, LogStreamsReferenceMapPtr logStreamsReference);

  TypedConfigurations(const TypedConfigurations& other);

  virtual ~TypedConfigurations(void) {
  }

  const Configurations* configurations(void) const {
    return m_configurations;
  }

  bool enabled(Level level);
  bool toFile(Level level);
  const std::string& filename(Level level);
  bool toStandardOutput(Level level);
  const base::LogFormat& logFormat(Level level);
  const base::SubsecondPrecision& subsecondPrecision(Level level = Level::Global);
  const base::MillisecondsWidth& millisecondsWidth(Level level = Level::Global);
  bool performanceTracking(Level level = Level::Global);
  base::type::fstream_t* fileStream(Level level);
  std::size_t maxLogFileSize(Level level);
  std::size_t logFlushThreshold(Level level);

 private:
  Configurations* m_configurations;
  std::unordered_map<Level, bool> m_enabledMap;
  std::unordered_map<Level, bool> m_toFileMap;
  std::unordered_map<Level, std::string> m_filenameMap;
  std::unordered_map<Level, bool> m_toStandardOutputMap;
  std::unordered_map<Level, base::LogFormat> m_logFormatMap;
  std::unordered_map<Level, base::SubsecondPrecision> m_subsecondPrecisionMap;
  std::unordered_map<Level, bool> m_performanceTrackingMap;
  std::unordered_map<Level, base::FileStreamPtr> m_fileStreamMap;
  std::unordered_map<Level, std::size_t> m_maxLogFileSizeMap;
  std::unordered_map<Level, std::size_t> m_logFlushThresholdMap;
  LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;

  friend class el::Helpers;
  friend class el::base::MessageBuilder;
  friend class el::base::Writer;
  friend class el::base::DefaultLogDispatchCallback;
  friend class el::base::LogDispatcher;

  template <typename Conf_T>
  inline Conf_T getConfigByVal(Level level, const std::unordered_map<Level, Conf_T>* confMap, const char* confName) {
    base::threading::ScopedLock scopedLock(lock());
    return unsafeGetConfigByVal(level, confMap, confName);  // This is not unsafe anymore - mutex locked in scope
  }

  template <typename Conf_T>
  inline Conf_T& getConfigByRef(Level level, std::unordered_map<Level, Conf_T>* confMap, const char* confName) {
    base::threading::ScopedLock scopedLock(lock());
    return unsafeGetConfigByRef(level, confMap, confName);  // This is not unsafe anymore - mutex locked in scope
  }

  template <typename Conf_T>
  Conf_T unsafeGetConfigByVal(Level level, const std::unordered_map<Level, Conf_T>* confMap, const char* confName) {
    ELPP_UNUSED(confName);
    typename std::unordered_map<Level, Conf_T>::const_iterator it = confMap->find(level);
    if (it == confMap->end()) {
      try {
        return confMap->at(Level::Global);
      } catch (...) {
        ELPP_INTERNAL_ERROR("Unable to get configuration [" « confName « "] for level ["
                            « LevelHelper::convertToString(level) « "]"
                            « std::endl « "Please ensure you have properly configured logger.", false);
        return Conf_T();
      }
    }
    return it->second;
  }

  template <typename Conf_T>
  Conf_T& unsafeGetConfigByRef(Level level, std::unordered_map<Level, Conf_T>* confMap, const char*
confName) {
    ELPP_UNUSED(confName);
    typename std::unordered_map<Level, Conf_T>::iterator it = confMap->find(level);
    if (it == confMap->end()) {
      try {
        return confMap->at(Level::Global);
      } catch (...) {
        ELPP_INTERNAL_ERROR("Unable to get configuration [" « confName « "] for level ["
                              « LevelHelper::convertToString(level) « "]"
                              « std::endl « "Please ensure you have properly configured logger.",
false);
      }
    }
    return it->second;
  }

  template <typename Conf_T>
  void setValue(Level level, const Conf_T& value, std::unordered_map<Level, Conf_T>* confMap,
                bool includeGlobalLevel = true) {
    // If map is empty and we are allowed to add into generic level (Level::Global), do it!
    if (confMap->empty() && includeGlobalLevel) {
      confMap->insert(std::make_pair(Level::Global, value));
      return;
    }
    // If same value exist in generic level already, dont add it to explicit level
    typename std::unordered_map<Level, Conf_T>::iterator it = confMap->find(Level::Global);
    if (it != confMap->end() && it->second == value) {
      return;
    }
    // Now make sure we dont double up values if we really need to add it to explicit level
    it = confMap->find(level);
    if (it == confMap->end()) {
      // Value not found for level, add new
      confMap->insert(std::make_pair(level, value));
    } else {
      // Value found, just update value
      confMap->at(level) = value;
    }
  }

  void build(Configurations* configurations);
  unsigned long getULong(std::string confVal);
  std::string resolveFilename(const std::string& filename);
  void insertFile(Level level, const std::string& fullFilename);
  bool unsafeValidateFileRolling(Level level, const PreRollOutCallback& preRollOutCallback);

  inline bool validateFileRolling(Level level, const PreRollOutCallback& preRollOutCallback) {
    base::threading::ScopedLock scopedLock(lock());
    return unsafeValidateFileRolling(level, preRollOutCallback);
  }
};
class HitCounter {
 public:
  HitCounter(void) :
    m_filename(""),
    m_lineNumber(0),
    m_hitCounts(0) {
  }

  HitCounter(const char* filename, base::type::LineNumber lineNumber) :
    m_filename(filename),
    m_lineNumber(lineNumber),
    m_hitCounts(0) {
  }

  HitCounter(const HitCounter& hitCounter) :
    m_filename(hitCounter.m_filename),
    m_lineNumber(hitCounter.m_lineNumber),
    m_hitCounts(hitCounter.m_hitCounts) {
  }

  HitCounter& operator=(const HitCounter& hitCounter) {
    if (&hitCounter != this) {
      m_filename = hitCounter.m_filename;
      m_lineNumber = hitCounter.m_lineNumber;
      m_hitCounts = hitCounter.m_hitCounts;
    }
    return *this;
  }

  virtual ~HitCounter(void) {
  }

  inline void resetLocation(const char* filename, base::type::LineNumber lineNumber) {
    m_filename = filename;
    m_lineNumber = lineNumber;
  }

  inline void validateHitCounts(std::size_t n) {
    if (m_hitCounts >= base::consts::kMaxLogPerCounter) {
      m_hitCounts = (n >= 1 ? base::consts::kMaxLogPerCounter % n : 0);
    }
    ++m_hitCounts;
  }

  inline const char* filename(void) const {
    return m_filename;
  }

  inline base::type::LineNumber lineNumber(void) const {
    return m_lineNumber;
  }

  inline std::size_t hitCounts(void) const {
    return m_hitCounts;
  }

  inline void increment(void) {
    ++m_hitCounts;
  }

  class Predicate {
   public:
    Predicate(const char* filename, base::type::LineNumber lineNumber)
      : m_filename(filename),
        m_lineNumber(lineNumber) {
    }
    inline bool operator()(const HitCounter* counter) {
      return ((counter != nullptr) &&
              (strcmp(counter->m_filename, m_filename) == 0) &&
              (counter->m_lineNumber == m_lineNumber));
    }

   private:
    const char* m_filename;
    base::type::LineNumber m_lineNumber;
  };

 private:
  const char* m_filename;
  base::type::LineNumber m_lineNumber;
  std::size_t m_hitCounts;
};
class RegisteredHitCounters : public base::utils::RegistryWithPred<base::HitCounter,
base::HitCounter::Predicate> {
 public:
  bool validateEveryN(const char* filename, base::type::LineNumber lineNumber, std::size_t n);

  bool validateAfterN(const char* filename, base::type::LineNumber lineNumber, std::size_t n);

  bool validateNTimes(const char* filename, base::type::LineNumber lineNumber, std::size_t n);

  inline const base::HitCounter* getCounter(const char* filename, base::type::LineNumber lineNumber) {
    base::threading::ScopedLock scopedLock(lock());
    return get(filename, lineNumber);
  }
};
enum class DispatchAction : base::type::EnumType {
  None = 1, NormalLog = 2, SysLog = 4
};
}  // namespace base
template <typename T>
class Callback : protected base::threading::ThreadSafe {
 public:
  Callback(void) : m_enabled(true) {}
  inline bool enabled(void) const {
    return m_enabled;
  }
  inline void setEnabled(bool enabled) {
    base::threading::ScopedLock scopedLock(lock());
    m_enabled = enabled;
  }
 protected:
  virtual void handle(const T* handlePtr) = 0;
 private:
  bool m_enabled;
};
class LogDispatchData {
 public:
  LogDispatchData() : m_logMessage(nullptr), m_dispatchAction(base::DispatchAction::None) {}
  inline const LogMessage* logMessage(void) const {
    return m_logMessage;
  }
  inline base::DispatchAction dispatchAction(void) const {
    return m_dispatchAction;
  }
  inline void setLogMessage(LogMessage* logMessage) {
    m_logMessage = logMessage;
  }
  inline void setDispatchAction(base::DispatchAction dispatchAction) {
    m_dispatchAction = dispatchAction;
  }
 private:
  LogMessage* m_logMessage;
  base::DispatchAction m_dispatchAction;
  friend class base::LogDispatcher;

};
class LogDispatchCallback : public Callback<LogDispatchData> {
 protected:
  virtual void handle(const LogDispatchData* data);
  base::threading::Mutex& fileHandle(const LogDispatchData* data);
 private:
  friend class base::LogDispatcher;
  std::unordered_map<std::string, std::unique_ptr<base::threading::Mutex» m_fileLocks;
  base::threading::Mutex m_fileLocksMapLock;
};
class PerformanceTrackingCallback : public Callback<PerformanceTrackingData> {
 private:
  friend class base::PerformanceTracker;
};
class LoggerRegistrationCallback : public Callback<Logger> {
 private:
  friend class base::RegisteredLoggers;
};
class LogBuilder : base::NoCopy {
 public:
  LogBuilder() : m_termSupportsColor(base::utils::OS::termSupportsColor()) {}
  virtual ~LogBuilder(void) {
    ELPP_INTERNAL_INFO(3, "Destroying log builder...")
  }
  virtual base::type::string_t build(const LogMessage* logMessage, bool appendNewLine) const = 0;
  void convertToColoredOutput(base::type::string_t* logLine, Level level);
 private:
  bool m_termSupportsColor;
  friend class el::base::DefaultLogDispatchCallback;
};
typedef std::shared_ptr<LogBuilder> LogBuilderPtr;
class Logger : public base::threading::ThreadSafe, public Loggable {
 public:
  Logger(const std::string& id, base::LogStreamsReferenceMapPtr logStreamsReference);
  Logger(const std::string& id, const Configurations& configurations, base::LogStreamsReferenceMapPtr
logStreamsReference);
  Logger(const Logger& logger);
  Logger& operator=(const Logger& logger);

  virtual ~Logger(void) {
    base::utils::safeDelete(m_typedConfigurations);
  }

  virtual inline void log(el::base::type::ostream_t& os) const {
    os « m_id.c_str();
  }

  void configure(const Configurations& configurations);

  void reconfigure(void);

  inline const std::string& id(void) const {
    return m_id;
  }

  inline const std::string& parentApplicationName(void) const {
    return m_parentApplicationName;
  }

  inline void setParentApplicationName(const std::string& parentApplicationName) {
    m_parentApplicationName = parentApplicationName;
  }

  inline Configurations* configurations(void) {
    return &m_configurations;
  }

  inline base::TypedConfigurations* typedConfigurations(void) {
    return m_typedConfigurations;
  }

  static bool isValidId(const std::string& id);

  void flush(void);

  void flush(Level level, base::type::fstream_t* fs);

  inline bool isFlushNeeded(Level level) {
    return ++m_unflushedCount.find(level)->second >= m_typedConfigurations->logFlushThreshold(level);
  }

  inline LogBuilder* logBuilder(void) const {
    return m_logBuilder.get();
  }

  inline void setLogBuilder(const LogBuilderPtr& logBuilder) {
    m_logBuilder = logBuilder;
  }

  inline bool enabled(Level level) const {
    return m_typedConfigurations->enabled(level);
  }

#if ELPP_VARIADIC_TEMPLATES_SUPPORTED
#  define LOGGER_LEVEL_WRITERS_SIGNATURES(FUNCTION_NAME)\
template <typename T, typename... Args>\
inline void FUNCTION_NAME(const char*, const T&, const Args&...);\
template <typename T>\
inline void FUNCTION_NAME(const T&);

  template <typename T, typename... Args>
  inline void verbose(int, const char*, const T&, const Args&...);

  template <typename T>
  inline void verbose(int, const T&);

  LOGGER_LEVEL_WRITERS_SIGNATURES(info)
  LOGGER_LEVEL_WRITERS_SIGNATURES(debug)
  LOGGER_LEVEL_WRITERS_SIGNATURES(warn)
  LOGGER_LEVEL_WRITERS_SIGNATURES(error)
  LOGGER_LEVEL_WRITERS_SIGNATURES(fatal)
  LOGGER_LEVEL_WRITERS_SIGNATURES(trace)
#  undef LOGGER_LEVEL_WRITERS_SIGNATURES
#endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
 private:
  std::string m_id;
  base::TypedConfigurations* m_typedConfigurations;
  base::type::stringstream_t m_stream;
  std::string m_parentApplicationName;
  bool m_isConfigured;
  Configurations m_configurations;
  std::unordered_map<Level, unsigned int> m_unflushedCount;
  base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;
  LogBuilderPtr m_logBuilder;

  friend class el::LogMessage;
  friend class el::Loggers;
  friend class el::Helpers;
  friend class el::base::RegisteredLoggers;
  friend class el::base::DefaultLogDispatchCallback;
  friend class el::base::MessageBuilder;
  friend class el::base::Writer;
  friend class el::base::PErrorWriter;
  friend class el::base::Storage;
  friend class el::base::PerformanceTracker;
  friend class el::base::LogDispatcher;

  Logger(void);

#if ELPP_VARIADIC_TEMPLATES_SUPPORTED
  template <typename T, typename... Args>
  void log_(Level, int, const char*, const T&, const Args&...);

  template <typename T>
  inline void log_(Level, int, const T&);

  template <typename T, typename... Args>
  void log(Level, const char*, const T&, const Args&...);

  template <typename T>
  inline void log(Level, const T&);
#endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED

  void initUnflushedCount(void);

  inline base::type::stringstream_t& stream(void) {
    return m_stream;
  }

  void resolveLoggerFormatSpec(void) const;
};
namespace base {
class RegisteredLoggers : public base::utils::Registry<Logger, std::string> {
 public:
  explicit RegisteredLoggers(const LogBuilderPtr& defaultLogBuilder);

  virtual ~RegisteredLoggers(void) {
    unsafeFlushAll();
  }

  inline void setDefaultConfigurations(const Configurations& configurations) {
    base::threading::ScopedLock scopedLock(lock());
    m_defaultConfigurations.setFromBase(const_cast<Configurations*>(&configurations));
  }

  inline Configurations* defaultConfigurations(void) {
    return &m_defaultConfigurations;
  }

  Logger* get(const std::string& id, bool forceCreation = true);

  template <typename T>
  inline bool installLoggerRegistrationCallback(const std::string& id) {
    return base::utils::Utils::installCallback<T, base::type::LoggerRegistrationCallbackPtr>(id,
           &m_loggerRegistrationCallbacks);
  }

  template <typename T>
  inline void uninstallLoggerRegistrationCallback(const std::string& id) {
    base::utils::Utils::uninstallCallback<T, base::type::LoggerRegistrationCallbackPtr>(id,
&m_loggerRegistrationCallbacks);
  }

  template <typename T>
  inline T* loggerRegistrationCallback(const std::string& id) {
    return base::utils::Utils::callback<T, base::type::LoggerRegistrationCallbackPtr>(id,
&m_loggerRegistrationCallbacks);
  }

  bool remove(const std::string& id);

  inline bool has(const std::string& id) {
    return get(id, false) != nullptr;
  }

  inline void unregister(Logger*& logger) {
    base::threading::ScopedLock scopedLock(lock());
    base::utils::Registry<Logger, std::string>::unregister(logger->id());
  }

  inline LogStreamsReferenceMapPtr logStreamsReference(void) {
    return m_logStreamsReference;
  }

  inline void flushAll(void) {
    base::threading::ScopedLock scopedLock(lock());
    unsafeFlushAll();
  }

  inline void setDefaultLogBuilder(LogBuilderPtr& logBuilderPtr) {
    base::threading::ScopedLock scopedLock(lock());
    m_defaultLogBuilder = logBuilderPtr;
  }

 private:
  LogBuilderPtr m_defaultLogBuilder;
  Configurations m_defaultConfigurations;
  base::LogStreamsReferenceMapPtr m_logStreamsReference = nullptr;
  std::unordered_map<std::string, base::type::LoggerRegistrationCallbackPtr>
m_loggerRegistrationCallbacks;
  friend class el::base::Storage;

  void unsafeFlushAll(void);
};
class VRegistry : base::NoCopy, public base::threading::ThreadSafe {
 public:
  explicit VRegistry(base::type::VerboseLevel level, base::type::EnumType* pFlags);

  void setLevel(base::type::VerboseLevel level);

  inline base::type::VerboseLevel level(void) const {
    return m_level;
  }

  inline void clearModules(void) {
    base::threading::ScopedLock scopedLock(lock());
    m_modules.clear();
  }

  void setModules(const char* modules);

  bool allowed(base::type::VerboseLevel vlevel, const char* file);

  inline const std::unordered_map<std::string, base::type::VerboseLevel>& modules(void) const {
    return m_modules;
  }

  void setFromArgs(const base::utils::CommandLineArgs* commandLineArgs);

  inline bool vModulesEnabled(void) {
    return !base::utils::hasFlag(LoggingFlag::DisableVModules, *m_pFlags);
  }

 private:
  base::type::VerboseLevel m_level;
  base::type::EnumType* m_pFlags;
  std::unordered_map<std::string, base::type::VerboseLevel> m_modules;
};
}  // namespace base
class LogMessage {
 public:
  LogMessage(Level level, const std::string& file, base::type::LineNumber line, const std::string&
func,
             base::type::VerboseLevel verboseLevel, Logger* logger) :
    m_level(level), m_file(file), m_line(line), m_func(func),
    m_verboseLevel(verboseLevel), m_logger(logger), m_message(logger->stream().str()) {
  }
  inline Level level(void) const {
    return m_level;
  }
  inline const std::string& file(void) const {
    return m_file;
  }
  inline base::type::LineNumber line(void) const {
    return m_line;
  }
  inline const std::string& func(void) const {
    return m_func;
  }
  inline base::type::VerboseLevel verboseLevel(void) const {
    return m_verboseLevel;
  }
  inline Logger* logger(void) const {
    return m_logger;
  }
  inline const base::type::string_t& message(void) const {
    return m_message;
  }
 private:
  Level m_level;
  std::string m_file;
  base::type::LineNumber m_line;
  std::string m_func;
  base::type::VerboseLevel m_verboseLevel;
  Logger* m_logger;
  base::type::string_t m_message;
};
namespace base {
#if ELPP_ASYNC_LOGGING
class AsyncLogItem {
 public:
  explicit AsyncLogItem(const LogMessage& logMessage, const LogDispatchData& data, const
base::type::string_t& logLine)
    : m_logMessage(logMessage), m_dispatchData(data), m_logLine(logLine) {}
  virtual ~AsyncLogItem() {}
  inline LogMessage* logMessage(void) {
    return &m_logMessage;
  }
  inline LogDispatchData* data(void) {
    return &m_dispatchData;
  }
  inline base::type::string_t logLine(void) {
    return m_logLine;
  }
 private:
  LogMessage m_logMessage;
  LogDispatchData m_dispatchData;
  base::type::string_t m_logLine;
};
class AsyncLogQueue : public base::threading::ThreadSafe {
 public:
  virtual ~AsyncLogQueue() {
    ELPP_INTERNAL_INFO(6, "~AsyncLogQueue");
  }

  inline AsyncLogItem next(void) {
    base::threading::ScopedLock scopedLock(lock());
    AsyncLogItem result = m_queue.front();
    m_queue.pop();
    return result;
  }

  inline void push(const AsyncLogItem& item) {
    base::threading::ScopedLock scopedLock(lock());
    m_queue.push(item);
  }
  inline void pop(void) {
    base::threading::ScopedLock scopedLock(lock());
    m_queue.pop();
  }
  inline AsyncLogItem front(void) {
    base::threading::ScopedLock scopedLock(lock());
    return m_queue.front();
  }
  inline bool empty(void) {
    base::threading::ScopedLock scopedLock(lock());
    return m_queue.empty();
  }
 private:
  std::queue<AsyncLogItem> m_queue;
};
class IWorker {
 public:
  virtual ~IWorker() {}
  virtual void start() = 0;
};
#endif // ELPP_ASYNC_LOGGING
class Storage : base::NoCopy, public base::threading::ThreadSafe {
 public:
#if ELPP_ASYNC_LOGGING
  Storage(const LogBuilderPtr& defaultLogBuilder, base::IWorker* asyncDispatchWorker);
#else
  explicit Storage(const LogBuilderPtr& defaultLogBuilder);
#endif  // ELPP_ASYNC_LOGGING

  virtual ~Storage(void);

  inline bool validateEveryNCounter(const char* filename, base::type::LineNumber lineNumber, std::size_t occasion) {
    return hitCounters()->validateEveryN(filename, lineNumber, occasion);
  }

  inline bool validateAfterNCounter(const char* filename, base::type::LineNumber lineNumber, std::size_t n) {
    return hitCounters()->validateAfterN(filename, lineNumber, n);
  }

  inline bool validateNTimesCounter(const char* filename, base::type::LineNumber lineNumber, std::size_t n) {
    return hitCounters()->validateNTimes(filename, lineNumber, n);
  }

  inline base::RegisteredHitCounters* hitCounters(void) const {
    return m_registeredHitCounters;
  }

  inline base::RegisteredLoggers* registeredLoggers(void) const {
    return m_registeredLoggers;
  }

  inline base::VRegistry* vRegistry(void) const {
    return m_vRegistry;
  }

#if ELPP_ASYNC_LOGGING
  inline base::AsyncLogQueue* asyncLogQueue(void) const {
    return m_asyncLogQueue;
  }
#endif  // ELPP_ASYNC_LOGGING

  inline const base::utils::CommandLineArgs* commandLineArgs(void) const {
    return &m_commandLineArgs;
  }

  inline void addFlag(LoggingFlag flag) {
    base::utils::addFlag(flag, &m_flags);
  }

  inline void removeFlag(LoggingFlag flag) {
    base::utils::removeFlag(flag, &m_flags);
  }

  inline bool hasFlag(LoggingFlag flag) const {
    return base::utils::hasFlag(flag, m_flags);
  }

  inline base::type::EnumType flags(void) const {
    return m_flags;
  }

  inline void setFlags(base::type::EnumType flags) {
    m_flags = flags;
  }

  inline void setPreRollOutCallback(const PreRollOutCallback& callback) {
    m_preRollOutCallback = callback;
  }

  inline void unsetPreRollOutCallback(void) {
    m_preRollOutCallback = base::defaultPreRollOutCallback;
  }

  inline PreRollOutCallback& preRollOutCallback(void) {
    return m_preRollOutCallback;
  }

  bool hasCustomFormatSpecifier(const char* formatSpecifier);
  void installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier);
  bool uninstallCustomFormatSpecifier(const char* formatSpecifier);

  const std::vector<CustomFormatSpecifier>* customFormatSpecifiers(void) const {
    return &m_customFormatSpecifiers;
  }

  base::threading::Mutex& customFormatSpecifiersLock() {
    return m_customFormatSpecifiersLock;
  }

  inline void setLoggingLevel(Level level) {
    m_loggingLevel = level;
  }

  template <typename T>
  inline bool installLogDispatchCallback(const std::string& id) {
    return base::utils::Utils::installCallback<T, base::type::LogDispatchCallbackPtr>(id,
&m_logDispatchCallbacks);
  }

  template <typename T>
  inline void uninstallLogDispatchCallback(const std::string& id) {
    base::utils::Utils::uninstallCallback<T, base::type::LogDispatchCallbackPtr>(id,
&m_logDispatchCallbacks);
  }
  template <typename T>
  inline T* logDispatchCallback(const std::string& id) {
    return base::utils::Utils::callback<T, base::type::LogDispatchCallbackPtr>(id,
&m_logDispatchCallbacks);
  }

#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  template <typename T>
  inline bool installPerformanceTrackingCallback(const std::string& id) {
    return base::utils::Utils::installCallback<T, base::type::PerformanceTrackingCallbackPtr>(id,
           &m_performanceTrackingCallbacks);
  }

  template <typename T>
  inline void uninstallPerformanceTrackingCallback(const std::string& id) {
    base::utils::Utils::uninstallCallback<T, base::type::PerformanceTrackingCallbackPtr>(id,
        &m_performanceTrackingCallbacks);
  }

  template <typename T>
  inline T* performanceTrackingCallback(const std::string& id) {
    return base::utils::Utils::callback<T, base::type::PerformanceTrackingCallbackPtr>(id,
&m_performanceTrackingCallbacks);
  }
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)

  inline void setThreadName(const std::string& name) {
    if (name.empty()) return;
    base::threading::ScopedLock scopedLock(m_threadNamesLock);
    m_threadNames[base::threading::getCurrentThreadId()] = name;
  }

  inline std::string getThreadName(const std::string& threadId) {
    base::threading::ScopedLock scopedLock(m_threadNamesLock);
    std::unordered_map<std::string, std::string>::const_iterator it = m_threadNames.find(threadId);
    if (it == m_threadNames.end()) {
      return threadId;
    }
    return it->second;
  }
 private:
  base::RegisteredHitCounters* m_registeredHitCounters;
  base::RegisteredLoggers* m_registeredLoggers;
  base::type::EnumType m_flags;
  base::VRegistry* m_vRegistry;
#if ELPP_ASYNC_LOGGING
  base::AsyncLogQueue* m_asyncLogQueue;
  base::IWorker* m_asyncDispatchWorker;
#endif  // ELPP_ASYNC_LOGGING
  base::utils::CommandLineArgs m_commandLineArgs;
  PreRollOutCallback m_preRollOutCallback;
  std::unordered_map<std::string, base::type::LogDispatchCallbackPtr> m_logDispatchCallbacks;
  std::unordered_map<std::string, base::type::PerformanceTrackingCallbackPtr>
m_performanceTrackingCallbacks;
  std::unordered_map<std::string, std::string> m_threadNames;
  std::vector<CustomFormatSpecifier> m_customFormatSpecifiers;
  base::threading::Mutex m_customFormatSpecifiersLock;
  base::threading::Mutex m_threadNamesLock;
  Level m_loggingLevel;

  friend class el::Helpers;
  friend class el::base::DefaultLogDispatchCallback;
  friend class el::LogBuilder;
  friend class el::base::MessageBuilder;
  friend class el::base::Writer;
  friend class el::base::PerformanceTracker;
  friend class el::base::LogDispatcher;

  void setApplicationArguments(int argc, char** argv);

  inline void setApplicationArguments(int argc, const char** argv) {
    setApplicationArguments(argc, const_cast<char**>(argv));
  }
};
extern ELPP_EXPORT base::type::StoragePointer elStorage;
#define ELPP el::base::elStorage
class DefaultLogDispatchCallback : public LogDispatchCallback {
 protected:
  void handle(const LogDispatchData* data);
 private:
  const LogDispatchData* m_data;
  void dispatch(base::type::string_t&& logLine);
};
#if ELPP_ASYNC_LOGGING
class AsyncLogDispatchCallback : public LogDispatchCallback {
 protected:
  void handle(const LogDispatchData* data);
};
class AsyncDispatchWorker : public base::IWorker, public base::threading::ThreadSafe {
 public:
  AsyncDispatchWorker();
  virtual ~AsyncDispatchWorker();

  bool clean(void);
  void emptyQueue(void);
  virtual void start(void);
  void handle(AsyncLogItem* logItem);
  void run(void);

  void setContinueRunning(bool value) {
    base::threading::ScopedLock scopedLock(m_continueRunningLock);
    m_continueRunning = value;
  }

  bool continueRunning(void) const {
    return m_continueRunning;
  }
 private:
  std::condition_variable cv;
  bool m_continueRunning;
  base::threading::Mutex m_continueRunningLock;
};
#endif  // ELPP_ASYNC_LOGGING
}  // namespace base
namespace base {
class DefaultLogBuilder : public LogBuilder {
 public:
  base::type::string_t build(const LogMessage* logMessage, bool appendNewLine) const;
};
```

```
class LogDispatcher : base::NoCopy {
 public:
  LogDispatcher(bool proceed, LogMessage* logMessage, base::DispatchAction dispatchAction) :
    m_proceed(proceed),
    m_logMessage(logMessage),
    m_dispatchAction(std::move(dispatchAction)) {
  }

  void dispatch(void);

 private:
  bool m_proceed;
  LogMessage* m_logMessage;
  base::DispatchAction m_dispatchAction;
};
#if defined(ELPP_STL_LOGGING)
namespace workarounds {
template <typename T, typename Container>
class IterableContainer {
 public:
  typedef typename Container::iterator iterator;
  typedef typename Container::const_iterator const_iterator;
  IterableContainer(void) {}
  virtual ~IterableContainer(void) {}
  iterator begin(void) {
    return getContainer().begin();
  }
  iterator end(void) {
    return getContainer().end();
  }
 private:
  virtual Container& getContainer(void) = 0;
};
template<typename T, typename Container = std::vector<T>, typename Comparator = std::less<typename
Container::value_type»
class IterablePriorityQueue : public IterableContainer<T, Container>,
  public std::priority_queue<T, Container, Comparator> {
 public:
  IterablePriorityQueue(std::priority_queue<T, Container, Comparator> queue_) {
    std::size_t count_ = 0;
    while (++count_ < base::consts::kMaxLogPerContainer && !queue_.empty()) {
      this->push(queue_.top());
      queue_.pop();
    }
  }
 private:
  inline Container& getContainer(void) {
    return this->c;
  }
};
template<typename T, typename Container = std::deque<T»
class IterableQueue : public IterableContainer<T, Container>, public std::queue<T, Container> {
 public:
  IterableQueue(std::queue<T, Container> queue_) {
    std::size_t count_ = 0;
    while (++count_ < base::consts::kMaxLogPerContainer && !queue_.empty()) {
      this->push(queue_.front());
      queue_.pop();
    }
  }
 private:
  inline Container& getContainer(void) {
    return this->c;
  }
};
template<typename T, typename Container = std::deque<T»
class IterableStack : public IterableContainer<T, Container>, public std::stack<T, Container> {
 public:
  IterableStack(std::stack<T, Container> stack_) {
    std::size_t count_ = 0;
    while (++count_ < base::consts::kMaxLogPerContainer && !stack_.empty()) {
      this->push(stack_.top());
      stack_.pop();
    }
  }
 private:
  inline Container& getContainer(void) {
    return this->c;
  }
};
}  // namespace workarounds
#endif  // defined(ELPP_STL_LOGGING)
// Log message builder
class MessageBuilder {
 public:
  MessageBuilder(void) : m_logger(nullptr), m_containerLogSeparator(ELPP_LITERAL("")) {}
  void initialize(Logger* logger);
```

```
#  define ELPP_SIMPLE_LOG(LOG_TYPE)\
MessageBuilder& operator«(LOG_TYPE msg) {\
m_logger->stream() « msg;\
if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {\
m_logger->stream() « " ";\
}\
return *this;\
}

  inline MessageBuilder& operator«(const std::string& msg) {
    return operator«(msg.c_str());
  }
  ELPP_SIMPLE_LOG(char)
  ELPP_SIMPLE_LOG(bool)
  ELPP_SIMPLE_LOG(signed short)
  ELPP_SIMPLE_LOG(unsigned short)
  ELPP_SIMPLE_LOG(signed int)
  ELPP_SIMPLE_LOG(unsigned int)
  ELPP_SIMPLE_LOG(signed long)
  ELPP_SIMPLE_LOG(unsigned long)
  ELPP_SIMPLE_LOG(float)
  ELPP_SIMPLE_LOG(double)
  ELPP_SIMPLE_LOG(char*)
  ELPP_SIMPLE_LOG(const char*)
  ELPP_SIMPLE_LOG(const void*)
  ELPP_SIMPLE_LOG(long double)
  inline MessageBuilder& operator«(const std::wstring& msg) {
    return operator«(msg.c_str());
  }
  MessageBuilder& operator«(const wchar_t* msg);
  // ostream manipulators
  inline MessageBuilder& operator«(std::ostream& (*OStreamMani)(std::ostream&)) {
    m_logger->stream() « OStreamMani;
    return *this;
  }
#define ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(temp)                                              \
template <typename T>                                                                      \
inline MessageBuilder& operator«(const temp<T>& template_inst) {                           \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
#define ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(temp)                                              \
template <typename T1, typename T2>                                                        \
inline MessageBuilder& operator«(const temp<T1, T2>& template_inst) {                      \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
#define ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(temp)                                            \
template <typename T1, typename T2, typename T3>                                           \
inline MessageBuilder& operator«(const temp<T1, T2, T3>& template_inst) {                  \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
#define ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(temp)                                             \
template <typename T1, typename T2, typename T3, typename T4>                              \
inline MessageBuilder& operator«(const temp<T1, T2, T3, T4>& template_inst) {              \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}
#define ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(temp)                                             \
template <typename T1, typename T2, typename T3, typename T4, typename T5>                 \
inline MessageBuilder& operator«(const temp<T1, T2, T3, T4, T5>& template_inst) {          \
return writeIterator(template_inst.begin(), template_inst.end(), template_inst.size());      \
}

#if defined(ELPP_STL_LOGGING)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::vector)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::list)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(std::deque)
  ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(std::set)
  ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(std::multiset)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::map)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::multimap)
  template <class T, class Container>
  inline MessageBuilder& operator«(const std::queue<T, Container>& queue_) {
    base::workarounds::IterableQueue<T, Container> iterableQueue_ =
      static_cast<base::workarounds::IterableQueue<T, Container> >(queue_);
    return writeIterator(iterableQueue_.begin(), iterableQueue_.end(), iterableQueue_.size());
  }
  template <class T, class Container>
  inline MessageBuilder& operator«(const std::stack<T, Container>& stack_) {
    base::workarounds::IterableStack<T, Container> iterableStack_ =
      static_cast<base::workarounds::IterableStack<T, Container> >(stack_);
    return writeIterator(iterableStack_.begin(), iterableStack_.end(), iterableStack_.size());
  }
  template <class T, class Container, class Comparator>
  inline MessageBuilder& operator«(const std::priority_queue<T, Container, Comparator>&
priorityQueue_) {
    base::workarounds::IterablePriorityQueue<T, Container, Comparator> iterablePriorityQueue_ =
      static_cast<base::workarounds::IterablePriorityQueue<T, Container, Comparator>
```

```
>(priorityQueue_);
    return writeIterator(iterablePriorityQueue_.begin(), iterablePriorityQueue_.end(),
iterablePriorityQueue_.size());
  }
  template <class First, class Second>
  MessageBuilder& operator«(const std::pair<First, Second>& pair_) {
    m_logger->stream() « ELPP_LITERAL("(");
    operator « (static_cast<First>(pair_.first));
    m_logger->stream() « ELPP_LITERAL(", ");
    operator « (static_cast<Second>(pair_.second));
    m_logger->stream() « ELPP_LITERAL(")");
    return *this;
  }
  template <std::size_t Size>
  MessageBuilder& operator«(const std::bitset<Size>& bitset_) {
    m_logger->stream() « ELPP_LITERAL("[");
    operator « (bitset_.to_string());
    m_logger->stream() « ELPP_LITERAL("]");
    return *this;
  }
#  if defined(ELPP_LOG_STD_ARRAY)
  template <class T, std::size_t Size>
  inline MessageBuilder& operator«(const std::array<T, Size>& array) {
    return writeIterator(array.begin(), array.end(), array.size());
  }
#  endif  // defined(ELPP_LOG_STD_ARRAY)
#  if defined(ELPP_LOG_UNORDERED_MAP)
  ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(std::unordered_map)
  ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG(std::unordered_multimap)
#  endif  // defined(ELPP_LOG_UNORDERED_MAP)
#  if defined(ELPP_LOG_UNORDERED_SET)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::unordered_set)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(std::unordered_multiset)
#  endif  // defined(ELPP_LOG_UNORDERED_SET)
#endif  // defined(ELPP_STL_LOGGING)
#if defined(ELPP_QT_LOGGING)
  inline MessageBuilder& operator«(const QString& msg) {
#  if defined(ELPP_UNICODE)
    m_logger->stream() « msg.toStdWString();
#  else
    m_logger->stream() « msg.toStdString();
#  endif  // defined(ELPP_UNICODE)
    return *this;
  }
  inline MessageBuilder& operator«(const QByteArray& msg) {
    return operator « (QString(msg));
  }
  inline MessageBuilder& operator«(const QStringRef& msg) {
    return operator«(msg.toString());
  }
  inline MessageBuilder& operator«(qint64 msg) {
#  if defined(ELPP_UNICODE)
    m_logger->stream() « QString::number(msg).toStdWString();
#  else
    m_logger->stream() « QString::number(msg).toStdString();
#  endif  // defined(ELPP_UNICODE)
    return *this;
  }
  inline MessageBuilder& operator«(quint64 msg) {
#  if defined(ELPP_UNICODE)
    m_logger->stream() « QString::number(msg).toStdWString();
#  else
    m_logger->stream() « QString::number(msg).toStdString();
#  endif  // defined(ELPP_UNICODE)
    return *this;
  }
  inline MessageBuilder& operator«(QChar msg) {
    m_logger->stream() « msg.toLatin1();
    return *this;
  }
  inline MessageBuilder& operator«(const QLatin1String& msg) {
    m_logger->stream() « msg.latin1();
    return *this;
  }
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QList)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QVector)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QQueue)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QSet)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QLinkedList)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(QStack)
  template <typename First, typename Second>
  MessageBuilder& operator«(const QPair<First, Second>& pair_) {
    m_logger->stream() « ELPP_LITERAL("(");
    operator « (static_cast<First>(pair_.first));
    m_logger->stream() « ELPP_LITERAL(", ");
    operator « (static_cast<Second>(pair_.second));
    m_logger->stream() « ELPP_LITERAL(")");
    return *this;
  }
  template <typename K, typename V>
  MessageBuilder& operator«(const QMap<K, V>& map_) {
    m_logger->stream() « ELPP_LITERAL("[");
    QList<K> keys = map_.keys();
    typename QList<K>::const_iterator begin = keys.begin();
    typename QList<K>::const_iterator end = keys.end();
    int max_ = static_cast<int>(base::consts::kMaxLogPerContainer);  // to prevent warning
    for (int index_ = 0; begin != end && index_ < max_; ++index_, ++begin) {
      m_logger->stream() « ELPP_LITERAL("(");
      operator « (static_cast<K>(*begin));
      m_logger->stream() « ELPP_LITERAL(", ");
      operator « (static_cast<V>(map_.value(*begin)));
      m_logger->stream() « ELPP_LITERAL(")");
      m_logger->stream() « ((index_ < keys.size() -1) ? m_containerLogSeparator : ELPP_LITERAL(""));
    }
    if (begin != end) {
      m_logger->stream() « ELPP_LITERAL("...");
    }
    m_logger->stream() « ELPP_LITERAL("]");
    return *this;
  }
  template <typename K, typename V>
  inline MessageBuilder& operator«(const QMultiMap<K, V>& map_) {
    operator « (static_cast<QMap<K, V>>(map_));
    return *this;
  }
  template <typename K, typename V>
  MessageBuilder& operator«(const QHash<K, V>& hash_) {
    m_logger->stream() « ELPP_LITERAL("[");
    QList<K> keys = hash_.keys();
    typename QList<K>::const_iterator begin = keys.begin();
    typename QList<K>::const_iterator end = keys.end();
    int max_ = static_cast<int>(base::consts::kMaxLogPerContainer);  // prevent type warning
    for (int index_ = 0; begin != end && index_ < max_; ++index_, ++begin) {
      m_logger->stream() « ELPP_LITERAL("(");
      operator « (static_cast<K>(*begin));
      m_logger->stream() « ELPP_LITERAL(", ");
      operator « (static_cast<V>(hash_.value(*begin)));
      m_logger->stream() « ELPP_LITERAL(")");
      m_logger->stream() « ((index_ < keys.size() -1) ? m_containerLogSeparator : ELPP_LITERAL(""));
    }
    if (begin != end) {
      m_logger->stream() « ELPP_LITERAL("...");
    }
    m_logger->stream() « ELPP_LITERAL("]");
    return *this;
  }
  template <typename K, typename V>
  inline MessageBuilder& operator«(const QMultiHash<K, V>& multiHash_) {
    operator « (static_cast<QHash<K, V>>(multiHash_));
    return *this;
  }
#endif  // defined(ELPP_QT_LOGGING)
#if defined(ELPP_BOOST_LOGGING)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::vector)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::stable_vector)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::list)
  ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG(boost::container::deque)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(boost::container::map)
  ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG(boost::container::flat_map)
  ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(boost::container::set)
  ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG(boost::container::flat_set)
#endif  // defined(ELPP_BOOST_LOGGING)

#define MAKE_CONTAINERELPP_FRIENDLY(ContainerType, SizeMethod, ElementInstance) \
el::base::type::ostream_t& operator«(el::base::type::ostream_t& ss, const ContainerType& container) {\
const el::base::type::char_t* sep = ELPP->hasFlag(el::LoggingFlag::NewLineForContainer) ? \
ELPP_LITERAL("\n    ") : ELPP_LITERAL(", ");\
ContainerType::const_iterator elem = container.begin();\
ContainerType::const_iterator endElem = container.end();\
std::size_t size_ = container.SizeMethod; \
ss « ELPP_LITERAL("[");\
for (std::size_t i = 0; elem != endElem && i < el::base::consts::kMaxLogPerContainer; ++i, ++elem) { \
ss « ElementInstance;\
ss « ((i < size_ - 1) ? sep : ELPP_LITERAL(""));\
}\
if (elem != endElem) {\
ss « ELPP_LITERAL("...");\
}\
ss « ELPP_LITERAL("]");\
return ss;\
}
#if defined(ELPP_WXWIDGETS_LOGGING)
  ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG(wxVector)
#  define ELPP_WX_PTR_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(),
*(*elem))
#  define ELPP_WX_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(), (*elem))
#  define ELPP_WX_HASH_MAP_ENABLED(ContainerType) MAKE_CONTAINERELPP_FRIENDLY(ContainerType, size(), \
ELPP_LITERAL("(") « elem->first « ELPP_LITERAL(", ") « elem->second « ELPP_LITERAL(")")
#else
#  define ELPP_WX_PTR_ENABLED(ContainerType)
#  define ELPP_WX_ENABLED(ContainerType)
#  define ELPP_WX_HASH_MAP_ENABLED(ContainerType)
#endif  // defined(ELPP_WXWIDGETS_LOGGING)
  // Other classes
  template <class Class>
  ELPP_SIMPLE_LOG(const Class&)
#undef ELPP_SIMPLE_LOG
#undef ELPP_ITERATOR_CONTAINER_LOG_ONE_ARG
#undef ELPP_ITERATOR_CONTAINER_LOG_TWO_ARG
#undef ELPP_ITERATOR_CONTAINER_LOG_THREE_ARG
#undef ELPP_ITERATOR_CONTAINER_LOG_FOUR_ARG
#undef ELPP_ITERATOR_CONTAINER_LOG_FIVE_ARG
 private:
  Logger* m_logger;
  const base::type::char_t* m_containerLogSeparator;

  template<class Iterator>
  MessageBuilder& writeIterator(Iterator begin_, Iterator end_, std::size_t size_) {
    m_logger->stream() « ELPP_LITERAL("[");
    for (std::size_t i = 0; begin_ != end_ && i < base::consts::kMaxLogPerContainer; ++i, ++begin_) {
      operator « (*begin_);
      m_logger->stream() « ((i < size_ - 1) ? m_containerLogSeparator : ELPP_LITERAL(""));
    }
    if (begin_ != end_) {
      m_logger->stream() « ELPP_LITERAL("...");
    }
    m_logger->stream() « ELPP_LITERAL("]");
    if (ELPP->hasFlag(LoggingFlag::AutoSpacing)) {
      m_logger->stream() « " ";
    }
    return *this;
  }
};
class NullWriter : base::NoCopy {
 public:
  NullWriter(void) {}

  // Null manipulator
  inline NullWriter& operator«(std::ostream& (*)(std::ostream&)) {
    return *this;
  }

  template <typename T>
  inline NullWriter& operator«(const T&) {
    return *this;
  }

  inline operator bool() {
    return true;
  }
};
class Writer : base::NoCopy {
 public:
  Writer(Level level, const char* file, base::type::LineNumber line,
         const char* func, base::DispatchAction dispatchAction = base::DispatchAction::NormalLog,
         base::type::VerboseLevel verboseLevel = 0) :
    m_msg(nullptr), m_level(level), m_file(file), m_line(line), m_func(func),
m_verboseLevel(verboseLevel),
    m_logger(nullptr), m_proceed(false), m_dispatchAction(dispatchAction) {
  }

  Writer(LogMessage* msg, base::DispatchAction dispatchAction = base::DispatchAction::NormalLog) :
    m_msg(msg), m_level(msg != nullptr ? msg->level() : Level::Unknown),
    m_line(0), m_logger(nullptr), m_proceed(false), m_dispatchAction(dispatchAction) {
  }

  virtual ~Writer(void) {
    processDispatch();
  }

  template <typename T>
  inline Writer& operator«(const T& log) {
#if ELPP_LOGGING_ENABLED
    if (m_proceed) {
      m_messageBuilder « log;
    }
#endif  // ELPP_LOGGING_ENABLED
    return *this;
  }

  inline Writer& operator«(std::ostream& (*log)(std::ostream&)) {
```

```
#if ELPP_LOGGING_ENABLED
    if (m_proceed) {
      m_messageBuilder « log;
    }
#endif  // ELPP_LOGGING_ENABLED
    return *this;
  }

  inline operator bool() {
    return true;
  }

  Writer& construct(Logger* logger, bool needLock = true);
  Writer& construct(int count, const char* loggerIds, ...);
 protected:
  LogMessage* m_msg;
  Level m_level;
  const char* m_file;
  const base::type::LineNumber m_line;
  const char* m_func;
  base::type::VerboseLevel m_verboseLevel;
  Logger* m_logger;
  bool m_proceed;
  base::MessageBuilder m_messageBuilder;
  base::DispatchAction m_dispatchAction;
  std::vector<std::string> m_loggerIds;
  friend class el::Helpers;

  void initializeLogger(const std::string& loggerId, bool lookup = true, bool needLock = true);
  void processDispatch();
  void triggerDispatch(void);
};
class PErrorWriter : public base::Writer {
 public:
  PErrorWriter(Level level, const char* file, base::type::LineNumber line,
               const char* func, base::DispatchAction dispatchAction =
base::DispatchAction::NormalLog,
               base::type::VerboseLevel verboseLevel = 0) :
    base::Writer(level, file, line, func, dispatchAction, verboseLevel) {
  }

  virtual ~PErrorWriter(void);
};
}  // namespace base
// Logging from Logger class. Why this is here? Because we have Storage and Writer class available
#if ELPP_VARIADIC_TEMPLATES_SUPPORTED
template <typename T, typename... Args>
void Logger::log_(Level level, int vlevel, const char* s, const T& value, const Args&... args) {
  base::MessageBuilder b;
  b.initialize(this);
  while (*s) {
    if (*s == base::consts::kFormatSpecifierChar) {
      if (*(s + 1) == base::consts::kFormatSpecifierChar) {
        ++s;
      } else {
        if (*(s + 1) == base::consts::kFormatSpecifierCharValue) {
          ++s;
          b « value;
          log_(level, vlevel, ++s, args...);
          return;
        }
      }
    }
    b « *s++;
  }
  ELPP_INTERNAL_ERROR("Too many arguments provided. Unable to handle. Please provide more format
specifiers", false);
}
template <typename T>
void Logger::log_(Level level, int vlevel, const T& log) {
  if (level == Level::Verbose) {
    if (ELPP->vRegistry()->allowed(vlevel, __FILE__)) {
      base::Writer(Level::Verbose, "FILE", 0, "FUNCTION",
                   base::DispatchAction::NormalLog, vlevel).construct(this, false) « log;
    } else {
      stream().str(ELPP_LITERAL(""));
      releaseLock();
    }
  } else {
    base::Writer(level, "FILE", 0, "FUNCTION").construct(this, false) « log;
  }
}
template <typename T, typename... Args>
inline void Logger::log(Level level, const char* s, const T& value, const Args&... args) {
  acquireLock(); // released in Writer!
  log_(level, 0, s, value, args...);
}
```

```
template <typename T>
inline void Logger::log(Level level, const T& log) {
  acquireLock(); // released in Writer!
  log_(level, 0, log);
}
#  if ELPP_VERBOSE_LOG
template <typename T, typename... Args>
inline void Logger::verbose(int vlevel, const char* s, const T& value, const Args&... args) {
  acquireLock(); // released in Writer!
  log_(el::Level::Verbose, vlevel, s, value, args...);
}
template <typename T>
inline void Logger::verbose(int vlevel, const T& log) {
  acquireLock(); // released in Writer!
  log_(el::Level::Verbose, vlevel, log);
}
#  else
template <typename T, typename... Args>
inline void Logger::verbose(int, const char*, const T&, const Args&...) {
  return;
}
template <typename T>
inline void Logger::verbose(int, const T&) {
  return;
}
#  endif  // ELPP_VERBOSE_LOG
#  define LOGGER_LEVEL_WRITERS(FUNCTION_NAME, LOG_LEVEL)\
template <typename T, typename... Args>\
inline void Logger::FUNCTION_NAME(const char* s, const T& value, const Args&... args) {\
log(LOG_LEVEL, s, value, args...);\
}\
template <typename T>\
inline void Logger::FUNCTION_NAME(const T& value) {\
log(LOG_LEVEL, value);\
}
#  define LOGGER_LEVEL_WRITERS_DISABLED(FUNCTION_NAME, LOG_LEVEL)\
template <typename T, typename... Args>\
inline void Logger::FUNCTION_NAME(const char*, const T&, const Args&...) {\
return;\
}\
template <typename T>\
inline void Logger::FUNCTION_NAME(const T&) {\
return;\
}

#  if ELPP_INFO_LOG
LOGGER_LEVEL_WRITERS(info, Level::Info)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(info, Level::Info)
#  endif // ELPP_INFO_LOG
#  if ELPP_DEBUG_LOG
LOGGER_LEVEL_WRITERS(debug, Level::Debug)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(debug, Level::Debug)
#  endif // ELPP_DEBUG_LOG
#  if ELPP_WARNING_LOG
LOGGER_LEVEL_WRITERS(warn, Level::Warning)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(warn, Level::Warning)
#  endif // ELPP_WARNING_LOG
#  if ELPP_ERROR_LOG
LOGGER_LEVEL_WRITERS(error, Level::Error)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(error, Level::Error)
#  endif // ELPP_ERROR_LOG
#  if ELPP_FATAL_LOG
LOGGER_LEVEL_WRITERS(fatal, Level::Fatal)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(fatal, Level::Fatal)
#  endif // ELPP_FATAL_LOG
#  if ELPP_TRACE_LOG
LOGGER_LEVEL_WRITERS(trace, Level::Trace)
#  else
LOGGER_LEVEL_WRITERS_DISABLED(trace, Level::Trace)
#  endif // ELPP_TRACE_LOG
#  undef LOGGER_LEVEL_WRITERS
#  undef LOGGER_LEVEL_WRITERS_DISABLED
#endif // ELPP_VARIADIC_TEMPLATES_SUPPORTED
#if ELPP_COMPILER_MSVC
#  define ELPP_VARIADIC_FUNC_MSVC(variadicFunction, variadicArgs) variadicFunction variadicArgs
#  define ELPP_VARIADIC_FUNC_MSVC_RUN(variadicFunction, ...) ELPP_VARIADIC_FUNC_MSVC(variadicFunction,
(__VA_ARGS__))
#  define el_getVALength(...) ELPP_VARIADIC_FUNC_MSVC_RUN(el_resolveVALength, 0, ## __VA_ARGS__,\
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
#else
#  if ELPP_COMPILER_CLANG
#    define el_getVALength(...) el_resolveVALength(0, __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
#  else
#    define el_getVALength(...) el_resolveVALength(0, ## __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
0)
#  endif // ELPP_COMPILER_CLANG
#endif // ELPP_COMPILER_MSVC
#define el_resolveVALength(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N
#define ELPP_WRITE_LOG(writer, level, dispatchAction, ...) \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
#define ELPP_WRITE_LOG_IF(writer, condition, level, dispatchAction, ...) if (condition) \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
#define ELPP_WRITE_LOG_EVERY_N(writer, occasion, level, dispatchAction, ...) \
ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
#define ELPP_WRITE_LOG_AFTER_N(writer, n, level, dispatchAction, ...) \
ELPP->validateAfterNCounter(__FILE__, __LINE__, n) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
#define ELPP_WRITE_LOG_N_TIMES(writer, n, level, dispatchAction, ...) \
ELPP->validateNTimesCounter(__FILE__, __LINE__, n) && \
writer(level, __FILE__, __LINE__, ELPP_FUNC, dispatchAction).construct(el_getVALength(__VA_ARGS__),
__VA_ARGS__)
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
class PerformanceTrackingData {
 public:
  enum class DataType : base::type::EnumType {
    Checkpoint = 1, Complete = 2
  };
  // Do not use constructor, will run into multiple definition error, use init(PerformanceTracker*)
  explicit PerformanceTrackingData(DataType dataType) : m_performanceTracker(nullptr),
    m_dataType(dataType), m_firstCheckpoint(false), m_file(""), m_line(0), m_func("") {}
  inline const std::string* blockName(void) const;
  inline const struct timeval* startTime(void) const;
  inline const struct timeval* endTime(void) const;
  inline const struct timeval* lastCheckpointTime(void) const;
  inline const base::PerformanceTracker* performanceTracker(void) const {
    return m_performanceTracker;
  }
  inline PerformanceTrackingData::DataType dataType(void) const {
    return m_dataType;
  }
  inline bool firstCheckpoint(void) const {
    return m_firstCheckpoint;
  }
  inline std::string checkpointId(void) const {
    return m_checkpointId;
  }
  inline const char* file(void) const {
    return m_file;
  }
  inline base::type::LineNumber line(void) const {
    return m_line;
  }
  inline const char* func(void) const {
    return m_func;
  }
  inline const base::type::string_t* formattedTimeTaken() const {
    return &m_formattedTimeTaken;
  }
  inline const std::string& loggerId(void) const;
 private:
  base::PerformanceTracker* m_performanceTracker;
  base::type::string_t m_formattedTimeTaken;
  PerformanceTrackingData::DataType m_dataType;
  bool m_firstCheckpoint;
  std::string m_checkpointId;
  const char* m_file;
  base::type::LineNumber m_line;
  const char* m_func;
  inline void init(base::PerformanceTracker* performanceTracker, bool firstCheckpoint = false) {
    m_performanceTracker = performanceTracker;
    m_firstCheckpoint = firstCheckpoint;
  }

  friend class el::base::PerformanceTracker;
};
namespace base {
class PerformanceTracker : public base::threading::ThreadSafe, public Loggable {
 public:
  PerformanceTracker(const std::string& blockName,
                     base::TimestampUnit timestampUnit = base::TimestampUnit::Millisecond,
                     const std::string& loggerId =
std::string(el::base::consts::kPerformanceLoggerId),
                     bool scopedLog = true, Level level =
base::consts::kPerformanceTrackerDefaultLevel);
```

```
  PerformanceTracker(const PerformanceTracker& t) :
    m_blockName(t.m_blockName), m_timestampUnit(t.m_timestampUnit), m_loggerId(t.m_loggerId), m_scopedLog(t.m_scopedLog),
    m_level(t.m_level), m_hasChecked(t.m_hasChecked), m_lastCheckpointId(t.m_lastCheckpointId), m_enabled(t.m_enabled),
    m_startTime(t.m_startTime), m_endTime(t.m_endTime), m_lastCheckpointTime(t.m_lastCheckpointTime) {
  }
  virtual ~PerformanceTracker(void);
  void checkpoint(const std::string& id = std::string(), const char* file = __FILE__,
                  base::type::LineNumber line = __LINE__,
                  const char* func = "");
  inline Level level(void) const {
    return m_level;
  }
 private:
  std::string m_blockName;
  base::TimestampUnit m_timestampUnit;
  std::string m_loggerId;
  bool m_scopedLog;
  Level m_level;
  bool m_hasChecked;
  std::string m_lastCheckpointId;
  bool m_enabled;
  struct timeval m_startTime, m_endTime, m_lastCheckpointTime;

  PerformanceTracker(void);

  friend class el::PerformanceTrackingData;
  friend class base::DefaultPerformanceTrackingCallback;

  const inline base::type::string_t getFormattedTimeTaken() const {
    return getFormattedTimeTaken(m_startTime);
  }

  const base::type::string_t getFormattedTimeTaken(struct timeval startTime) const;

  virtual inline void log(el::base::type::ostream_t& os) const {
    os « getFormattedTimeTaken();
  }
};
class DefaultPerformanceTrackingCallback : public PerformanceTrackingCallback {
 protected:
  void handle(const PerformanceTrackingData* data) {
    m_data = data;
    base::type::stringstream_t ss;
    if (m_data->dataType() == PerformanceTrackingData::DataType::Complete) {
      ss « ELPP_LITERAL("Executed [") « m_data->blockName()->c_str() « ELPP_LITERAL("] in [") «
         *m_data->formattedTimeTaken() « ELPP_LITERAL("]");
    } else {
      ss « ELPP_LITERAL("Performance checkpoint");
      if (!m_data->checkpointId().empty()) {
        ss « ELPP_LITERAL(" [") « m_data->checkpointId().c_str() « ELPP_LITERAL("]");
      }
      ss « ELPP_LITERAL(" for block [") « m_data->blockName()->c_str() « ELPP_LITERAL("] : [") «
         *m_data->performanceTracker();
      if (!ELPP->hasFlag(LoggingFlag::DisablePerformanceTrackingCheckpointComparison)
          && m_data->performanceTracker()->m_hasChecked) {
        ss « ELPP_LITERAL(" ([") « *m_data->formattedTimeTaken() « ELPP_LITERAL("] from ");
        if (m_data->performanceTracker()->m_lastCheckpointId.empty()) {
          ss « ELPP_LITERAL("last checkpoint");
        } else {
          ss « ELPP_LITERAL("checkpoint ’") « m_data->performanceTracker()->m_lastCheckpointId.c_str() « ELPP_LITERAL("’");
        }
        ss « ELPP_LITERAL(")]");
      } else {
        ss « ELPP_LITERAL("]");
      }
    }
    el::base::Writer(m_data->performanceTracker()->level(), m_data->file(), m_data->line(), m_data->func()).construct(1,
        m_data->loggerId().c_str()) « ss.str();
  }
 private:
  const PerformanceTrackingData* m_data;
};
}  // namespace base
inline const std::string* PerformanceTrackingData::blockName() const {
  return const_cast<const std::string*>(&m_performanceTracker->m_blockName);
}
inline const struct timeval* PerformanceTrackingData::startTime() const {
  return const_cast<const struct timeval*>(&m_performanceTracker->m_startTime);
}
inline const struct timeval* PerformanceTrackingData::endTime() const {
  return const_cast<const struct timeval*>(&m_performanceTracker->m_endTime);
}
inline const struct timeval* PerformanceTrackingData::lastCheckpointTime() const {
  return const_cast<const struct timeval*>(&m_performanceTracker->m_lastCheckpointTime);
}
inline const std::string& PerformanceTrackingData::loggerId(void) const {
  return m_performanceTracker->m_loggerId;
}
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
namespace base {
namespace debug {
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
class StackTrace : base::NoCopy {
 public:
  static const unsigned int kMaxStack = 64;
  static const unsigned int kStackStart = 2;  // We want to skip c'tor and StackTrace::generateNew()
  class StackTraceEntry {
   public:
    StackTraceEntry(std::size_t index, const std::string& loc, const std::string& demang, const
std::string& hex,
                    const std::string& addr);
    StackTraceEntry(std::size_t index, const std::string& loc) :
      m_index(index),
      m_location(loc) {
    }
    std::size_t m_index;
    std::string m_location;
    std::string m_demangled;
    std::string m_hex;
    std::string m_addr;
    friend std::ostream& operator«(std::ostream& ss, const StackTraceEntry& si);

   private:
    StackTraceEntry(void);
  };

  StackTrace(void) {
    generateNew();
  }

  virtual ~StackTrace(void) {
  }

  inline std::vector<StackTraceEntry>& getLatestStack(void) {
    return m_stack;
  }

  friend std::ostream& operator«(std::ostream& os, const StackTrace& st);

 private:
  std::vector<StackTraceEntry> m_stack;

  void generateNew(void);
};
class CrashHandler : base::NoCopy {
 public:
  typedef void (*Handler)(int);

  explicit CrashHandler(bool useDefault);
  explicit CrashHandler(const Handler& cHandler) {
    setHandler(cHandler);
  }
  void setHandler(const Handler& cHandler);

 private:
  Handler m_handler;
};
#else
class CrashHandler {
 public:
  explicit CrashHandler(bool) {}
};
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
}  // namespace debug
}  // namespace base
extern base::debug::CrashHandler elCrashHandler;
#define MAKE_LOGGABLE(ClassType, ClassInstance, OutputStreamInstance) \
el::base::type::ostream_t& operator«(el::base::type::ostream_t& OutputStreamInstance, const ClassType&
ClassInstance)
class SysLogInitializer {
 public:
  SysLogInitializer(const char* processIdent, int options = 0, int facility = 0) {
#if defined(ELPP_SYSLOG)
    (void)base::consts::kSysLogLoggerId;
    openlog(processIdent, options, facility);
#else
    ELPP_UNUSED(processIdent);
    ELPP_UNUSED(options);
    ELPP_UNUSED(facility);
#endif  // defined(ELPP_SYSLOG)
```

```
  }
  virtual ~SysLogInitializer(void) {
#if defined(ELPP_SYSLOG)
    closelog();
#endif  // defined(ELPP_SYSLOG)
  }
};
#define ELPP_INITIALIZE_SYSLOG(id, opt, fac) el::SysLogInitializer elSyslogInit(id, opt, fac)
class Helpers : base::StaticClass {
 public:
  static inline void setStorage(base::type::StoragePointer storage) {
    ELPP = storage;
  }
  static inline base::type::StoragePointer storage() {
    return ELPP;
  }
  static inline void setArgs(int argc, char** argv) {
    ELPP->setApplicationArguments(argc, argv);
  }
  static inline void setArgs(int argc, const char** argv) {
    ELPP->setApplicationArguments(argc, const_cast<char**>(argv));
  }
  static inline void setThreadName(const std::string& name) {
    ELPP->setThreadName(name);
  }
  static inline std::string getThreadName() {
    return ELPP->getThreadName(base::threading::getCurrentThreadId());
  }
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
  static inline void setCrashHandler(const el::base::debug::CrashHandler::Handler& crashHandler) {
    el::elCrashHandler.setHandler(crashHandler);
  }
  static void crashAbort(int sig, const char* sourceFile = "", unsigned int long line = 0);
  static void logCrashReason(int sig, bool stackTraceIfAvailable = false,
                             Level level = Level::Fatal, const char* logger =
base::consts::kDefaultLoggerId);
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_CRASH_LOG)
  static inline void installPreRollOutCallback(const PreRollOutCallback& callback) {
    ELPP->setPreRollOutCallback(callback);
  }
  static inline void uninstallPreRollOutCallback(void) {
    ELPP->unsetPreRollOutCallback();
  }
  template <typename T>
  static inline bool installLogDispatchCallback(const std::string& id) {
    return ELPP->installLogDispatchCallback<T>(id);
  }
  template <typename T>
  static inline void uninstallLogDispatchCallback(const std::string& id) {
    ELPP->uninstallLogDispatchCallback<T>(id);
  }
  template <typename T>
  static inline T* logDispatchCallback(const std::string& id) {
    return ELPP->logDispatchCallback<T>(id);
  }
#if defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  template <typename T>
  static inline bool installPerformanceTrackingCallback(const std::string& id) {
    return ELPP->installPerformanceTrackingCallback<T>(id);
  }
  template <typename T>
  static inline void uninstallPerformanceTrackingCallback(const std::string& id) {
    ELPP->uninstallPerformanceTrackingCallback<T>(id);
  }
  template <typename T>
  static inline T* performanceTrackingCallback(const std::string& id) {
    return ELPP->performanceTrackingCallback<T>(id);
  }
#endif // defined(ELPP_FEATURE_ALL) || defined(ELPP_FEATURE_PERFORMANCE_TRACKING)
  template <typename T>
  static std::string convertTemplateToStdString(const T& templ) {
    el::Logger* logger =
      ELPP->registeredLoggers()->get(el::base::consts::kDefaultLoggerId);
    if (logger == nullptr) {
      return std::string();
    }
    base::MessageBuilder b;
    b.initialize(logger);
    logger->acquireLock();
    b « templ;
#if defined(ELPP_UNICODE)
    std::string s = std::string(logger->stream().str().begin(), logger->stream().str().end());
#else
    std::string s = logger->stream().str();
#endif  // defined(ELPP_UNICODE)
    logger->stream().str(ELPP_LITERAL(""));
    logger->releaseLock();
    return s;
  }
  static inline const el::base::utils::CommandLineArgs* commandLineArgs(void) {
    return ELPP->commandLineArgs();
  }
  static inline void reserveCustomFormatSpecifiers(std::size_t size) {
    ELPP->m_customFormatSpecifiers.reserve(size);
  }
  static inline void installCustomFormatSpecifier(const CustomFormatSpecifier& customFormatSpecifier)
{
    ELPP->installCustomFormatSpecifier(customFormatSpecifier);
  }
  static inline bool uninstallCustomFormatSpecifier(const char* formatSpecifier) {
    return ELPP->uninstallCustomFormatSpecifier(formatSpecifier);
  }
  static inline bool hasCustomFormatSpecifier(const char* formatSpecifier) {
    return ELPP->hasCustomFormatSpecifier(formatSpecifier);
  }
  static inline void validateFileRolling(Logger* logger, Level level) {
    if (ELPP == nullptr || logger == nullptr) return;
    logger->m_typedConfigurations->validateFileRolling(level, ELPP->preRollOutCallback());
  }
};
class Loggers : base::StaticClass {
 public:
  static Logger* getLogger(const std::string& identity, bool registerIfNotAvailable = true);
  static void setDefaultLogBuilder(el::LogBuilderPtr& logBuilderPtr);
  template <typename T>
  static inline bool installLoggerRegistrationCallback(const std::string& id) {
    return ELPP->registeredLoggers()->installLoggerRegistrationCallback<T>(id);
  }
  template <typename T>
  static inline void uninstallLoggerRegistrationCallback(const std::string& id) {
    ELPP->registeredLoggers()->uninstallLoggerRegistrationCallback<T>(id);
  }
  template <typename T>
  static inline T* loggerRegistrationCallback(const std::string& id) {
    return ELPP->registeredLoggers()->loggerRegistrationCallback<T>(id);
  }
  static bool unregisterLogger(const std::string& identity);
  static bool hasLogger(const std::string& identity);
  static Logger* reconfigureLogger(Logger* logger, const Configurations& configurations);
  static Logger* reconfigureLogger(const std::string& identity, const Configurations& configurations);
  static Logger* reconfigureLogger(const std::string& identity, ConfigurationType configurationType,
                                   const std::string& value);
  static void reconfigureAllLoggers(const Configurations& configurations);
  static inline void reconfigureAllLoggers(ConfigurationType configurationType, const std::string&
value) {
    reconfigureAllLoggers(Level::Global, configurationType, value);
  }
  static void reconfigureAllLoggers(Level level, ConfigurationType configurationType,
                                    const std::string& value);
  static void setDefaultConfigurations(const Configurations& configurations,
                                       bool reconfigureExistingLoggers = false);
  static const Configurations* defaultConfigurations(void);
  static const base::LogStreamsReferenceMapPtr logStreamsReference(void);
  static base::TypedConfigurations defaultTypedConfigurations(void);
  static std::vector<std::string>* populateAllLoggerIds(std::vector<std::string>* targetList);
  static void configureFromGlobal(const char* globalConfigurationFilePath);
  static bool configureFromArg(const char* argKey);
  static void flushAll(void);
  static inline void addFlag(LoggingFlag flag) {
    ELPP->addFlag(flag);
  }
  static inline void removeFlag(LoggingFlag flag) {
    ELPP->removeFlag(flag);
  }
  static inline bool hasFlag(LoggingFlag flag) {
    return ELPP->hasFlag(flag);
  }
  class ScopedAddFlag {
   public:
    ScopedAddFlag(LoggingFlag flag) : m_flag(flag) {
      Loggers::addFlag(m_flag);
    }
    ~ScopedAddFlag(void) {
      Loggers::removeFlag(m_flag);
    }
   private:
    LoggingFlag m_flag;
  };
  class ScopedRemoveFlag {
   public:
    ScopedRemoveFlag(LoggingFlag flag) : m_flag(flag) {
      Loggers::removeFlag(m_flag);
    }
    ~ScopedRemoveFlag(void) {
      Loggers::addFlag(m_flag);
    }
   private:
    LoggingFlag m_flag;
  };
  static void setLoggingLevel(Level level) {
    ELPP->setLoggingLevel(level);
  }
  static void setVerboseLevel(base::type::VerboseLevel level);
  static base::type::VerboseLevel verboseLevel(void);
  static void setVModules(const char* modules);
  static void clearVModules(void);
};
class VersionInfo : base::StaticClass {
 public:
  static const std::string version(void);

  static const std::string releaseDate(void);
};
}  // namespace el
#undef VLOG_IS_ON
#define VLOG_IS_ON(verboseLevel) (ELPP->vRegistry()->allowed(verboseLevel, __FILE__))
#undef TIMED_BLOCK
#undef TIMED_SCOPE
#undef TIMED_SCOPE_IF
#undef TIMED_FUNC
#undef TIMED_FUNC_IF
#undef ELPP_MIN_UNIT
#if defined(ELPP_PERFORMANCE_MICROSECONDS)
#  define ELPP_MIN_UNIT el::base::TimestampUnit::Microsecond
#else
#  define ELPP_MIN_UNIT el::base::TimestampUnit::Millisecond
#endif  // (defined(ELPP_PERFORMANCE_MICROSECONDS))
// Note: Do not surround this definition with null macro because of obj instance
#define TIMED_SCOPE_IF(obj, blockname, condition) el::base::type::PerformanceTrackerPtr obj( condition
? \
  new el::base::PerformanceTracker(blockname, ELPP_MIN_UNIT) : nullptr )
#define TIMED_SCOPE(obj, blockname) TIMED_SCOPE_IF(obj, blockname, true)
#define TIMED_BLOCK(obj, blockName) for (struct { int i; el::base::type::PerformanceTrackerPtr timer;
} obj = { 0, \
  el::base::type::PerformanceTrackerPtr(new el::base::PerformanceTracker(blockName, ELPP_MIN_UNIT)) };
obj.i < 1; ++obj.i)
#define TIMED_FUNC_IF(obj,condition) TIMED_SCOPE_IF(obj, ELPP_FUNC, condition)
#define TIMED_FUNC(obj) TIMED_SCOPE(obj, ELPP_FUNC)
#undef PERFORMANCE_CHECKPOINT
#undef PERFORMANCE_CHECKPOINT_WITH_ID
#define PERFORMANCE_CHECKPOINT(obj) obj->checkpoint(std::string(), __FILE__, __LINE__, ELPP_FUNC)
#define PERFORMANCE_CHECKPOINT_WITH_ID(obj, id) obj->checkpoint(id, __FILE__, __LINE__, ELPP_FUNC)
#undef ELPP_COUNTER
#undef ELPP_COUNTER_POS
#define ELPP_COUNTER (ELPP->hitCounters()->getCounter(__FILE__, __LINE__))
#define ELPP_COUNTER_POS (ELPP_COUNTER == nullptr ? -1 : ELPP_COUNTER->hitCounts())
// Undef levels to support LOG(LEVEL)
#undef INFO
#undef WARNING
#undef DEBUG
#undef ERROR
#undef FATAL
#undef TRACE
#undef VERBOSE
// Undef existing
#undef CINFO
#undef CWARNING
#undef CDEBUG
#undef CFATAL
#undef CERROR
#undef CTRACE
#undef CVERBOSE
#undef CINFO_IF
#undef CWARNING_IF
#undef CDEBUG_IF
#undef CERROR_IF
#undef CFATAL_IF
#undef CTRACE_IF
#undef CVERBOSE_IF
#undef CINFO_EVERY_N
#undef CWARNING_EVERY_N
#undef CDEBUG_EVERY_N
#undef CERROR_EVERY_N
#undef CFATAL_EVERY_N
#undef CTRACE_EVERY_N
#undef CVERBOSE_EVERY_N
#undef CINFO_AFTER_N
#undef CWARNING_AFTER_N
#undef CDEBUG_AFTER_N
#undef CERROR_AFTER_N
#undef CFATAL_AFTER_N
```

```
#undef CTRACE_AFTER_N
#undef CVERBOSE_AFTER_N
#undef CINFO_N_TIMES
#undef CWARNING_N_TIMES
#undef CDEBUG_N_TIMES
#undef CERROR_N_TIMES
#undef CFATAL_N_TIMES
#undef CTRACE_N_TIMES
#undef CVERBOSE_N_TIMES
// Normal logs
#if ELPP_INFO_LOG
#  define CINFO(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Info, dispatchAction,
__VA_ARGS__)
#else
#  define CINFO(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_INFO_LOG
#if ELPP_WARNING_LOG
#  define CWARNING(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Warning,
dispatchAction, __VA_ARGS__)
#else
#  define CWARNING(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_WARNING_LOG
#if ELPP_DEBUG_LOG
#  define CDEBUG(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Debug, dispatchAction,
__VA_ARGS__)
#else
#  define CDEBUG(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_DEBUG_LOG
#if ELPP_ERROR_LOG
#  define CERROR(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Error, dispatchAction,
__VA_ARGS__)
#else
#  define CERROR(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_ERROR_LOG
#if ELPP_FATAL_LOG
#  define CFATAL(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Fatal, dispatchAction,
__VA_ARGS__)
#else
#  define CFATAL(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_FATAL_LOG
#if ELPP_TRACE_LOG
#  define CTRACE(writer, dispatchAction, ...) ELPP_WRITE_LOG(writer, el::Level::Trace, dispatchAction,
__VA_ARGS__)
#else
#  define CTRACE(writer, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_TRACE_LOG
#if ELPP_VERBOSE_LOG
#  define CVERBOSE(writer, vlevel, dispatchAction, ...) if (VLOG_IS_ON(vlevel)) writer(\
el::Level::Verbose, __FILE__, __LINE__, ELPP_FUNC, dispatchAction,
vlevel).construct(el_getVALength(__VA_ARGS__), __VA_ARGS__)
#else
#  define CVERBOSE(writer, vlevel, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_VERBOSE_LOG
// Conditional logs
#if ELPP_INFO_LOG
#  define CINFO_IF(writer, condition_, dispatchAction, ...) \
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Info, dispatchAction, __VA_ARGS__)
#else
#  define CINFO_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_INFO_LOG
#if ELPP_WARNING_LOG
#  define CWARNING_IF(writer, condition_, dispatchAction, ...)\
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Warning, dispatchAction, __VA_ARGS__)
#else
#  define CWARNING_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_WARNING_LOG
#if ELPP_DEBUG_LOG
#  define CDEBUG_IF(writer, condition_, dispatchAction, ...)\
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Debug, dispatchAction, __VA_ARGS__)
#else
#  define CDEBUG_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_DEBUG_LOG
#if ELPP_ERROR_LOG
#  define CERROR_IF(writer, condition_, dispatchAction, ...)\
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Error, dispatchAction, __VA_ARGS__)
#else
#  define CERROR_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_ERROR_LOG
#if ELPP_FATAL_LOG
#  define CFATAL_IF(writer, condition_, dispatchAction, ...)\
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Fatal, dispatchAction, __VA_ARGS__)
#else
#  define CFATAL_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_FATAL_LOG
#if ELPP_TRACE_LOG
#  define CTRACE_IF(writer, condition_, dispatchAction, ...)\
ELPP_WRITE_LOG_IF(writer, (condition_), el::Level::Trace, dispatchAction, __VA_ARGS__)
```

```
#else
#  define CTRACE_IF(writer, condition_, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_TRACE_LOG
#if ELPP_VERBOSE_LOG
#  define CVERBOSE_IF(writer, condition_, vlevel, dispatchAction, ...) if (VLOG_IS_ON(vlevel) &&
(condition_)) writer( \
el::Level::Verbose, __FILE__, __LINE__, ELPP_FUNC, dispatchAction,
vlevel).construct(el_getVALength(__VA_ARGS__), __VA_ARGS__)
#else
#  define CVERBOSE_IF(writer, condition_, vlevel, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_VERBOSE_LOG
// Occasional logs
#if ELPP_INFO_LOG
#  define CINFO_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Info, dispatchAction, __VA_ARGS__)
#else
#  define CINFO_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_INFO_LOG
#if ELPP_WARNING_LOG
#  define CWARNING_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Warning, dispatchAction, __VA_ARGS__)
#else
#  define CWARNING_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_WARNING_LOG
#if ELPP_DEBUG_LOG
#  define CDEBUG_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Debug, dispatchAction, __VA_ARGS__)
#else
#  define CDEBUG_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_DEBUG_LOG
#if ELPP_ERROR_LOG
#  define CERROR_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Error, dispatchAction, __VA_ARGS__)
#else
#  define CERROR_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_ERROR_LOG
#if ELPP_FATAL_LOG
#  define CFATAL_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Fatal, dispatchAction, __VA_ARGS__)
#else
#  define CFATAL_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_FATAL_LOG
#if ELPP_TRACE_LOG
#  define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...)\
ELPP_WRITE_LOG_EVERY_N(writer, occasion, el::Level::Trace, dispatchAction, __VA_ARGS__)
#else
#  define CTRACE_EVERY_N(writer, occasion, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_TRACE_LOG
#if ELPP_VERBOSE_LOG
#  define CVERBOSE_EVERY_N(writer, occasion, vlevel, dispatchAction, ...)\
CVERBOSE_IF(writer, ELPP->validateEveryNCounter(__FILE__, __LINE__, occasion), vlevel, dispatchAction,
__VA_ARGS__)
#else
#  define CVERBOSE_EVERY_N(writer, occasion, vlevel, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_VERBOSE_LOG
// After N logs
#if ELPP_INFO_LOG
#  define CINFO_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
#else
#  define CINFO_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_INFO_LOG
#if ELPP_WARNING_LOG
#  define CWARNING_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
#else
#  define CWARNING_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_WARNING_LOG
#if ELPP_DEBUG_LOG
#  define CDEBUG_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
#else
#  define CDEBUG_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_DEBUG_LOG
#if ELPP_ERROR_LOG
#  define CERROR_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
#else
#  define CERROR_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_ERROR_LOG
#if ELPP_FATAL_LOG
#  define CFATAL_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
#else
#  define CFATAL_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_FATAL_LOG
#if ELPP_TRACE_LOG
```

```
#  define CTRACE_AFTER_N(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_AFTER_N(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)
#else
#  define CTRACE_AFTER_N(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_TRACE_LOG
#if ELPP_VERBOSE_LOG
#  define CVERBOSE_AFTER_N(writer, n, vlevel, dispatchAction, ...)\
CVERBOSE_IF(writer, ELPP->validateAfterNCounter(__FILE__, __LINE__, n), vlevel, dispatchAction,
__VA_ARGS__)
#else
#  define CVERBOSE_AFTER_N(writer, n, vlevel, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_VERBOSE_LOG
// N Times logs
#if ELPP_INFO_LOG
#  define CINFO_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Info, dispatchAction, __VA_ARGS__)
#else
#  define CINFO_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_INFO_LOG
#if ELPP_WARNING_LOG
#  define CWARNING_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Warning, dispatchAction, __VA_ARGS__)
#else
#  define CWARNING_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_WARNING_LOG
#if ELPP_DEBUG_LOG
#  define CDEBUG_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Debug, dispatchAction, __VA_ARGS__)
#else
#  define CDEBUG_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_DEBUG_LOG
#if ELPP_ERROR_LOG
#  define CERROR_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Error, dispatchAction, __VA_ARGS__)
#else
#  define CERROR_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_ERROR_LOG
#if ELPP_FATAL_LOG
#  define CFATAL_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Fatal, dispatchAction, __VA_ARGS__)
#else
#  define CFATAL_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_FATAL_LOG
#if ELPP_TRACE_LOG
#  define CTRACE_N_TIMES(writer, n, dispatchAction, ...)\
ELPP_WRITE_LOG_N_TIMES(writer, n, el::Level::Trace, dispatchAction, __VA_ARGS__)
#else
#  define CTRACE_N_TIMES(writer, n, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_TRACE_LOG
#if ELPP_VERBOSE_LOG
#  define CVERBOSE_N_TIMES(writer, n, vlevel, dispatchAction, ...)\
CVERBOSE_IF(writer, ELPP->validateNTimesCounter(__FILE__, __LINE__, n), vlevel, dispatchAction,
__VA_ARGS__)
#else
#  define CVERBOSE_N_TIMES(writer, n, vlevel, dispatchAction, ...) el::base::NullWriter()
#endif  // ELPP_VERBOSE_LOG
//
// Custom Loggers - Requires (level, dispatchAction, loggerId/s)
//
// undef existing
#undef CLOG
#undef CLOG_VERBOSE
#undef CVLOG
#undef CLOG_IF
#undef CLOG_VERBOSE_IF
#undef CVLOG_IF
#undef CLOG_EVERY_N
#undef CVLOG_EVERY_N
#undef CLOG_AFTER_N
#undef CVLOG_AFTER_N
#undef CLOG_N_TIMES
#undef CVLOG_N_TIMES
// Normal logs
#define CLOG(LEVEL, ...)\
C##LEVEL(el::base::Writer, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CVLOG(vlevel, ...) CVERBOSE(el::base::Writer, vlevel, el::base::DispatchAction::NormalLog,
__VA_ARGS__)
// Conditional logs
#define CLOG_IF(condition, LEVEL, ...)\
C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CVLOG_IF(condition, vlevel, ...)\
CVERBOSE_IF(el::base::Writer, condition, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)
// Hit counts based logs
#define CLOG_EVERY_N(n, LEVEL, ...)\
C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CVLOG_EVERY_N(n, vlevel, ...)\
CVERBOSE_EVERY_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)
```

```
#define CLOG_AFTER_N(n, LEVEL, ...)\
C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CVLOG_AFTER_N(n, vlevel, ...)\
CVERBOSE_AFTER_N(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CLOG_N_TIMES(n, LEVEL, ...)\
C##LEVEL##_N_TIMES(el::base::Writer, n, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CVLOG_N_TIMES(n, vlevel, ...)\
CVERBOSE_N_TIMES(el::base::Writer, n, vlevel, el::base::DispatchAction::NormalLog, __VA_ARGS__)
//
// Default Loggers macro using CLOG(), CLOG_VERBOSE() and CVLOG() macros
//
// undef existing
#undef LOG
#undef VLOG
#undef LOG_IF
#undef VLOG_IF
#undef LOG_EVERY_N
#undef VLOG_EVERY_N
#undef LOG_AFTER_N
#undef VLOG_AFTER_N
#undef LOG_N_TIMES
#undef VLOG_N_TIMES
#undef ELPP_CURR_FILE_LOGGER_ID
#if defined(ELPP_DEFAULT_LOGGER)
#  define ELPP_CURR_FILE_LOGGER_ID ELPP_DEFAULT_LOGGER
#else
#  define ELPP_CURR_FILE_LOGGER_ID el::base::consts::kDefaultLoggerId
#endif
#undef ELPP_TRACE
#define ELPP_TRACE CLOG(TRACE, ELPP_CURR_FILE_LOGGER_ID)
// Normal logs
#define LOG(LEVEL) CLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define VLOG(vlevel) CVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
// Conditional logs
#define LOG_IF(condition, LEVEL) CLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define VLOG_IF(condition, vlevel) CVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
// Hit counts based logs
#define LOG_EVERY_N(n, LEVEL) CLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define VLOG_EVERY_N(n, vlevel) CVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
#define LOG_AFTER_N(n, LEVEL) CLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define VLOG_AFTER_N(n, vlevel) CVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
#define LOG_N_TIMES(n, LEVEL) CLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define VLOG_N_TIMES(n, vlevel) CVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
// Generic PLOG()
#undef CPLOG
#undef CPLOG_IF
#undef PLOG
#undef PLOG_IF
#undef DCPLOG
#undef DCPLOG_IF
#undef DPLOG
#undef DPLOG_IF
#define CPLOG(LEVEL, ...)\
C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define CPLOG_IF(condition, LEVEL, ...)\
C##LEVEL##_IF(el::base::PErrorWriter, condition, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define DCPLOG(LEVEL, ...)\
if (ELPP_DEBUG_LOG) C##LEVEL(el::base::PErrorWriter, el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define DCPLOG_IF(condition, LEVEL, ...)\
C##LEVEL##_IF(el::base::PErrorWriter, (ELPP_DEBUG_LOG) && (condition),
el::base::DispatchAction::NormalLog, __VA_ARGS__)
#define PLOG(LEVEL) CPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define PLOG_IF(condition, LEVEL) CPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DPLOG(LEVEL) DCPLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DPLOG_IF(condition, LEVEL) DCPLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
// Generic SYSLOG()
#undef CSYSLOG
#undef CSYSLOG_IF
#undef CSYSLOG_EVERY_N
#undef CSYSLOG_AFTER_N
#undef CSYSLOG_N_TIMES
#undef SYSLOG
#undef SYSLOG_IF
#undef SYSLOG_EVERY_N
#undef SYSLOG_AFTER_N
#undef SYSLOG_N_TIMES
#undef DCSYSLOG
#undef DCSYSLOG_IF
#undef DCSYSLOG_EVERY_N
#undef DCSYSLOG_AFTER_N
#undef DCSYSLOG_N_TIMES
#undef DSYSLOG
#undef DSYSLOG_IF
#undef DSYSLOG_EVERY_N
#undef DSYSLOG_AFTER_N
#undef DSYSLOG_N_TIMES
#if defined(ELPP_SYSLOG)
```

```
#  define CSYSLOG(LEVEL, ...)\
C##LEVEL(el::base::Writer, el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define CSYSLOG_IF(condition, LEVEL, ...)\
C##LEVEL##_IF(el::base::Writer, condition, el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define CSYSLOG_EVERY_N(n, LEVEL, ...) C##LEVEL##_EVERY_N(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define CSYSLOG_AFTER_N(n, LEVEL, ...) C##LEVEL##_AFTER_N(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define CSYSLOG_N_TIMES(n, LEVEL, ...) C##LEVEL##_N_TIMES(el::base::Writer, n,
el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define SYSLOG(LEVEL) CSYSLOG(LEVEL, el::base::consts::kSysLogLoggerId)
#  define SYSLOG_IF(condition, LEVEL) CSYSLOG_IF(condition, LEVEL, el::base::consts::kSysLogLoggerId)
#  define SYSLOG_EVERY_N(n, LEVEL) CSYSLOG_EVERY_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
#  define SYSLOG_AFTER_N(n, LEVEL) CSYSLOG_AFTER_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
#  define SYSLOG_N_TIMES(n, LEVEL) CSYSLOG_N_TIMES(n, LEVEL, el::base::consts::kSysLogLoggerId)
#  define DCSYSLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) C##LEVEL(el::base::Writer,
el::base::DispatchAction::SysLog, __VA_ARGS__)
#  define DCSYSLOG_IF(condition, LEVEL, ...)\
C##LEVEL##_IF(el::base::Writer, (ELPP_DEBUG_LOG) && (condition), el::base::DispatchAction::SysLog,
__VA_ARGS__)
#  define DCSYSLOG_EVERY_N(n, LEVEL, ...)\
if (ELPP_DEBUG_LOG) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
#  define DCSYSLOG_AFTER_N(n, LEVEL, ...)\
if (ELPP_DEBUG_LOG) C##LEVEL##_AFTER_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
#  define DCSYSLOG_N_TIMES(n, LEVEL, ...)\
if (ELPP_DEBUG_LOG) C##LEVEL##_EVERY_N(el::base::Writer, n, el::base::DispatchAction::SysLog,
__VA_ARGS__)
#  define DSYSLOG(LEVEL) DCSYSLOG(LEVEL, el::base::consts::kSysLogLoggerId)
#  define DSYSLOG_IF(condition, LEVEL) DCSYSLOG_IF(condition, LEVEL,
el::base::consts::kSysLogLoggerId)
#  define DSYSLOG_EVERY_N(n, LEVEL) DCSYSLOG_EVERY_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
#  define DSYSLOG_AFTER_N(n, LEVEL) DCSYSLOG_AFTER_N(n, LEVEL, el::base::consts::kSysLogLoggerId)
#  define DSYSLOG_N_TIMES(n, LEVEL) DCSYSLOG_N_TIMES(n, LEVEL, el::base::consts::kSysLogLoggerId)
#else
#  define CSYSLOG(LEVEL, ...) el::base::NullWriter()
#  define CSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
#  define CSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
#  define CSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
#  define CSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
#  define SYSLOG(LEVEL) el::base::NullWriter()
#  define SYSLOG_IF(condition, LEVEL) el::base::NullWriter()
#  define SYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
#  define SYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()
#  define SYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
#  define DCSYSLOG(LEVEL, ...) el::base::NullWriter()
#  define DCSYSLOG_IF(condition, LEVEL, ...) el::base::NullWriter()
#  define DCSYSLOG_EVERY_N(n, LEVEL, ...) el::base::NullWriter()
#  define DCSYSLOG_AFTER_N(n, LEVEL, ...) el::base::NullWriter()
#  define DCSYSLOG_N_TIMES(n, LEVEL, ...) el::base::NullWriter()
#  define DSYSLOG(LEVEL) el::base::NullWriter()
#  define DSYSLOG_IF(condition, LEVEL) el::base::NullWriter()
#  define DSYSLOG_EVERY_N(n, LEVEL) el::base::NullWriter()
#  define DSYSLOG_AFTER_N(n, LEVEL) el::base::NullWriter()
#  define DSYSLOG_N_TIMES(n, LEVEL) el::base::NullWriter()
#endif  // defined(ELPP_SYSLOG)
//
// Custom Debug Only Loggers - Requires (level, loggerId/s)
//
// undef existing
#undef DCLOG
#undef DCVLOG
#undef DCLOG_IF
#undef DCVLOG_IF
#undef DCLOG_EVERY_N
#undef DCVLOG_EVERY_N
#undef DCLOG_AFTER_N
#undef DCVLOG_AFTER_N
#undef DCLOG_N_TIMES
#undef DCVLOG_N_TIMES
// Normal logs
#define DCLOG(LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG(LEVEL, __VA_ARGS__)
#define DCLOG_VERBOSE(vlevel, ...) if (ELPP_DEBUG_LOG) CLOG_VERBOSE(vlevel, __VA_ARGS__)
#define DCVLOG(vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG(vlevel, __VA_ARGS__)
// Conditional logs
#define DCLOG_IF(condition, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_IF(condition, LEVEL, __VA_ARGS__)
#define DCVLOG_IF(condition, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_IF(condition, vlevel, __VA_ARGS__)
// Hit counts based logs
#define DCLOG_EVERY_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_EVERY_N(n, LEVEL, __VA_ARGS__)
#define DCVLOG_EVERY_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_EVERY_N(n, vlevel, __VA_ARGS__)
#define DCLOG_AFTER_N(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_AFTER_N(n, LEVEL, __VA_ARGS__)
#define DCVLOG_AFTER_N(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_AFTER_N(n, vlevel, __VA_ARGS__)
#define DCLOG_N_TIMES(n, LEVEL, ...) if (ELPP_DEBUG_LOG) CLOG_N_TIMES(n, LEVEL, __VA_ARGS__)
#define DCVLOG_N_TIMES(n, vlevel, ...) if (ELPP_DEBUG_LOG) CVLOG_N_TIMES(n, vlevel, __VA_ARGS__)
//
// Default Debug Only Loggers macro using CLOG(), CLOG_VERBOSE() and CVLOG() macros
```

```
//
#if !defined(ELPP_NO_DEBUG_MACROS)
// undef existing
#undef DLOG
#undef DVLOG
#undef DLOG_IF
#undef DVLOG_IF
#undef DLOG_EVERY_N
#undef DVLOG_EVERY_N
#undef DLOG_AFTER_N
#undef DVLOG_AFTER_N
#undef DLOG_N_TIMES
#undef DVLOG_N_TIMES
// Normal logs
#define DLOG(LEVEL) DCLOG(LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DVLOG(vlevel) DCVLOG(vlevel, ELPP_CURR_FILE_LOGGER_ID)
// Conditional logs
#define DLOG_IF(condition, LEVEL) DCLOG_IF(condition, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DVLOG_IF(condition, vlevel) DCVLOG_IF(condition, vlevel, ELPP_CURR_FILE_LOGGER_ID)
// Hit counts based logs
#define DLOG_EVERY_N(n, LEVEL) DCLOG_EVERY_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DVLOG_EVERY_N(n, vlevel) DCVLOG_EVERY_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
#define DLOG_AFTER_N(n, LEVEL) DCLOG_AFTER_N(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DVLOG_AFTER_N(n, vlevel) DCVLOG_AFTER_N(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
#define DLOG_N_TIMES(n, LEVEL) DCLOG_N_TIMES(n, LEVEL, ELPP_CURR_FILE_LOGGER_ID)
#define DVLOG_N_TIMES(n, vlevel) DCVLOG_N_TIMES(n, vlevel, ELPP_CURR_FILE_LOGGER_ID)
#endif // defined(ELPP_NO_DEBUG_MACROS)
#if !defined(ELPP_NO_CHECK_MACROS)
// Check macros
#undef CCHECK
#undef CPCHECK
#undef CCHECK_EQ
#undef CCHECK_NE
#undef CCHECK_LT
#undef CCHECK_GT
#undef CCHECK_LE
#undef CCHECK_GE
#undef CCHECK_BOUNDS
#undef CCHECK_NOTNULL
#undef CCHECK_STRCASEEQ
#undef CCHECK_STRCASENE
#undef CHECK
#undef PCHECK
#undef CHECK_EQ
#undef CHECK_NE
#undef CHECK_LT
#undef CHECK_GT
#undef CHECK_LE
#undef CHECK_GE
#undef CHECK_BOUNDS
#undef CHECK_NOTNULL
#undef CHECK_STRCASEEQ
#undef CHECK_STRCASENE
#define CCHECK(condition, ...) CLOG_IF(!(condition), FATAL, __VA_ARGS__) « "Check failed: [" «
   #condition « "] "
#define CPCHECK(condition, ...) CPLOG_IF(!(condition), FATAL, __VA_ARGS__) « "Check failed: [" «
   #condition « "] "
#define CHECK(condition) CCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
#define PCHECK(condition) CPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
#define CCHECK_EQ(a, b, ...) CCHECK(a == b, __VA_ARGS__)
#define CCHECK_NE(a, b, ...) CCHECK(a != b, __VA_ARGS__)
#define CCHECK_LT(a, b, ...) CCHECK(a < b, __VA_ARGS__)
#define CCHECK_GT(a, b, ...) CCHECK(a > b, __VA_ARGS__)
#define CCHECK_LE(a, b, ...) CCHECK(a <= b, __VA_ARGS__)
#define CCHECK_GE(a, b, ...) CCHECK(a >= b, __VA_ARGS__)
#define CCHECK_BOUNDS(val, min, max, ...) CCHECK(val >= min && val <= max, __VA_ARGS__)
#define CHECK_EQ(a, b) CCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_NE(a, b) CCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_LT(a, b) CCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_GT(a, b) CCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_LE(a, b) CCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_GE(a, b) CCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_BOUNDS(val, min, max) CCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
#define CCHECK_NOTNULL(ptr, ...) CCHECK((ptr) != nullptr, __VA_ARGS__)
#define CCHECK_STREQ(str1, str2, ...) CLOG_IF(!el::base::utils::Str::cStringEq(str1, str2), FATAL,
   __VA_ARGS__) \
« "Check failed: [" « #str1 « " == " « #str2 « "] "
#define CCHECK_STRNE(str1, str2, ...) CLOG_IF(el::base::utils::Str::cStringEq(str1, str2), FATAL,
   __VA_ARGS__) \
« "Check failed: [" « #str1 « " != " « #str2 « "] "
#define CCHECK_STRCASEEQ(str1, str2, ...) CLOG_IF(!el::base::utils::Str::cStringCaseEq(str1, str2),
   FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " == " « #str2 « "] "
#define CCHECK_STRCASENE(str1, str2, ...) CLOG_IF(el::base::utils::Str::cStringCaseEq(str1, str2),
   FATAL, __VA_ARGS__) \
« "Check failed: [" « #str1 « " != " « #str2 « "] "
#define CHECK_NOTNULL(ptr) CCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
```

```
#define CHECK_STREQ(str1, str2) CCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_STRNE(str1, str2) CCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_STRCASEEQ(str1, str2) CCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define CHECK_STRCASENE(str1, str2) CCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#undef DCCHECK
#undef DCCHECK_EQ
#undef DCCHECK_NE
#undef DCCHECK_LT
#undef DCCHECK_GT
#undef DCCHECK_LE
#undef DCCHECK_GE
#undef DCCHECK_BOUNDS
#undef DCCHECK_NOTNULL
#undef DCCHECK_STRCASEEQ
#undef DCCHECK_STRCASENE
#undef DCPCHECK
#undef DCHECK
#undef DCHECK_EQ
#undef DCHECK_NE
#undef DCHECK_LT
#undef DCHECK_GT
#undef DCHECK_LE
#undef DCHECK_GE
#undef DCHECK_BOUNDS_
#undef DCHECK_NOTNULL
#undef DCHECK_STRCASEEQ
#undef DCHECK_STRCASENE
#undef DPCHECK
#define DCCHECK(condition, ...) if (ELPP_DEBUG_LOG) CCHECK(condition, __VA_ARGS__)
#define DCCHECK_EQ(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_EQ(a, b, __VA_ARGS__)
#define DCCHECK_NE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_NE(a, b, __VA_ARGS__)
#define DCCHECK_LT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LT(a, b, __VA_ARGS__)
#define DCCHECK_GT(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GT(a, b, __VA_ARGS__)
#define DCCHECK_LE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_LE(a, b, __VA_ARGS__)
#define DCCHECK_GE(a, b, ...) if (ELPP_DEBUG_LOG) CCHECK_GE(a, b, __VA_ARGS__)
#define DCCHECK_BOUNDS(val, min, max, ...) if (ELPP_DEBUG_LOG) CCHECK_BOUNDS(val, min, max,
__VA_ARGS__)
#define DCCHECK_NOTNULL(ptr, ...) if (ELPP_DEBUG_LOG) CCHECK_NOTNULL((ptr), __VA_ARGS__)
#define DCCHECK_STREQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STREQ(str1, str2, __VA_ARGS__)
#define DCCHECK_STRNE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRNE(str1, str2, __VA_ARGS__)
#define DCCHECK_STRCASEEQ(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASEEQ(str1, str2,
__VA_ARGS__)
#define DCCHECK_STRCASENE(str1, str2, ...) if (ELPP_DEBUG_LOG) CCHECK_STRCASENE(str1, str2,
__VA_ARGS__)
#define DCPCHECK(condition, ...) if (ELPP_DEBUG_LOG) CPCHECK(condition, __VA_ARGS__)
#define DCHECK(condition) DCCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_EQ(a, b) DCCHECK_EQ(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_NE(a, b) DCCHECK_NE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_LT(a, b) DCCHECK_LT(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_GT(a, b) DCCHECK_GT(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_LE(a, b) DCCHECK_LE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_GE(a, b) DCCHECK_GE(a, b, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_BOUNDS(val, min, max) DCCHECK_BOUNDS(val, min, max, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_NOTNULL(ptr) DCCHECK_NOTNULL((ptr), ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_STREQ(str1, str2) DCCHECK_STREQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_STRNE(str1, str2) DCCHECK_STRNE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_STRCASEEQ(str1, str2) DCCHECK_STRCASEEQ(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define DCHECK_STRCASENE(str1, str2) DCCHECK_STRCASENE(str1, str2, ELPP_CURR_FILE_LOGGER_ID)
#define DPCHECK(condition) DCPCHECK(condition, ELPP_CURR_FILE_LOGGER_ID)
#endif // defined(ELPP_NO_CHECK_MACROS)
#if defined(ELPP_DISABLE_DEFAULT_CRASH_HANDLING)
#  define ELPP_USE_DEF_CRASH_HANDLER false
#else
#  define ELPP_USE_DEF_CRASH_HANDLER true
#endif  // defined(ELPP_DISABLE_DEFAULT_CRASH_HANDLING)
#define ELPP_CRASH_HANDLER_INIT
#define ELPP_INIT_EASYLOGGINGPP(val) \
namespace el { \
namespace base { \
el::base::type::StoragePointer elStorage(val); \
} \
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER); \
}

#if ELPP_ASYNC_LOGGING
#  define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new
el::base::DefaultLogBuilder()),\
new el::base::AsyncDispatchWorker()))
#else
#  define INITIALIZE_EASYLOGGINGPP ELPP_INIT_EASYLOGGINGPP(new el::base::Storage(el::LogBuilderPtr(new
el::base::DefaultLogBuilder())))
#endif  // ELPP_ASYNC_LOGGING
#define INITIALIZE_NULL_EASYLOGGINGPP \
namespace el {\
namespace base {\
el::base::type::StoragePointer elStorage;\
}\
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
}
#define SHARE_EASYLOGGINGPP(initializedStorage)\
namespace el {\
namespace base {\
el::base::type::StoragePointer elStorage(initializedStorage);\
}\
el::base::debug::CrashHandler elCrashHandler(ELPP_USE_DEF_CRASH_HANDLER);\
}

#if defined(ELPP_UNICODE)
#  define START_EASYLOGGINGPP(argc, argv) el::Helpers::setArgs(argc, argv);
std::locale::global(std::locale(""))
#else
#  define START_EASYLOGGINGPP(argc, argv) el::Helpers::setArgs(argc, argv)
#endif  // defined(ELPP_UNICODE)
#endif // EASYLOGGINGPP_H
```

9.5 README.md File Reference

9.6 src/main.cpp File Reference

```
#include <easylogging++.h>
#include <iostream>
```

Namespaces

- namespace WIP

  Namespace for work in progress.

Functions

- void WIP::exampleEasyLogging ()

  Example of how to use easylogging with a configuration file.
- int main (int argc, char ∗argv[ ])

  Main function.

9.6.1 Function Documentation

9.6.1.1 main()

```
int main (
              int argc,
              char ∗ argv[ ] )
```

Main function.

Codeconvention:

- Formatter: astyle

Definition at line 26 of file main.cpp.

References WIP::exampleEasyLogging().

9.7 main.cpp

Go to the documentation of this file.

```
#include <easylogging++.h>
#include <iostream>

INITIALIZE_EASYLOGGINGPP

namespace WIP {
void exampleEasyLogging();
}

int main(int argc, char *argv[]) {
  WIP::exampleEasyLogging();
  std::cout « "Hello, World!" « std::endl;
  return 0;
}

namespace WIP {
void exampleEasyLogging() {
  el::Configurations conf("conf/easylogging.conf");
  el::Loggers::reconfigureLogger("default", conf);
  el::Loggers::reconfigureAllLoggers(conf);

  LOG(INFO) « "My first info log using default logger";
}
} // namespace WIP
```

Index